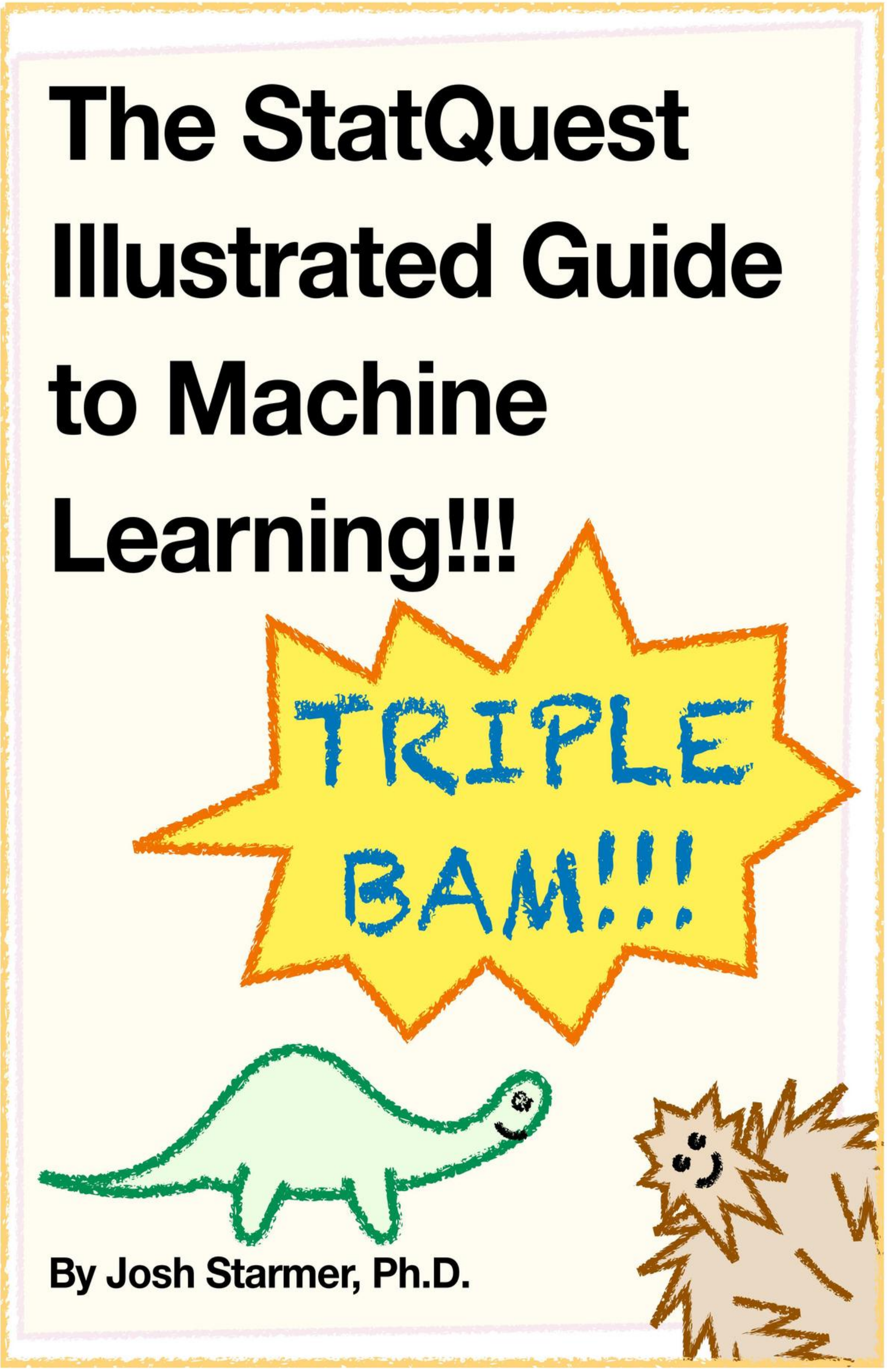
The StatQuest Illustrated Guide to Machine Learning!!!

By Josh Starmer, Ph.D.



The StatQuest Illustrated Guide to Machine Learning!!!

By Josh Starmer, Ph.D.

The StatQuest Illustrated Guide to Machine Learning!!!

www.statquest.org

Thank you for buying
The StatQuest Illustrated Guide to Machine Learning!!!

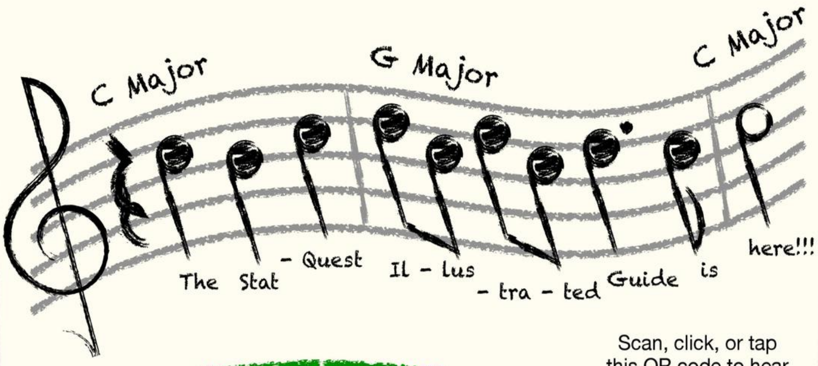
Every penny of your purchase goes to supporting **StatQuest** and helps create new videos, books, and webinars. Thanks to you, **StatQuest** videos and webinars are free on YouTube for everyone in the whole wide world to enjoy.

NOTE: If you're borrowing this copy of **The StatQuest Illustrated Guide to Machine Learning** from a friend, feel free to enjoy it. However, if you find it helpful, consider buying your own copy or supporting **StatQuest** however you can at https://statquest.org/support-statquest/.

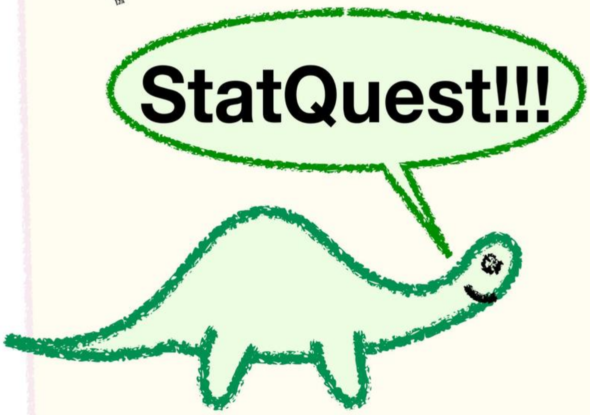
Scan, click, or tap this QR code to visit StatQuest.org!!!

For **F** and **B**, for teaching me to think differently,
T and **D**, for making it possible for me to think differently,
and for **A**, for everything.

Since every **StatQuest** starts with a **Silly Song**…



Scan, click, or tap
this QR code to hear
the **Silly Song**!!!



Hello!!!

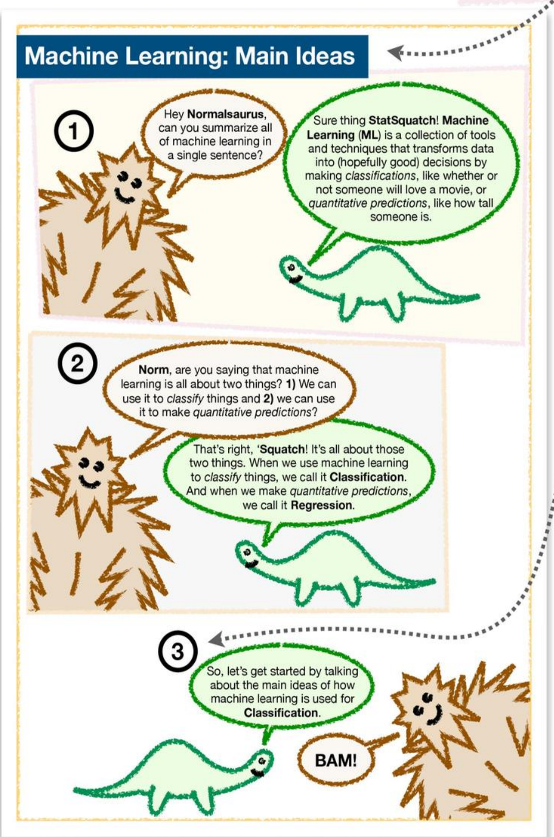
I'm Josh Starmer, and welcome to **The StatQuest Illustrated Guide to Machine Learning!!!** In this book, we'll talk about everything, from the very basics to advanced topics like **Neural Networks**. All concepts will be clearly illustrated, and we'll go through them one step at a time.

Table of Contents

How This Book Works

① **NOTE:** Before we get started, let's talk a little bit about how this book works by looking at a sample page.

② Each page starts with a header that tells you exactly what concept we're focusing on.



Machine Learning: Main Ideas

① Hey **Normalsaurus**, can you summarize all of machine learning in a single sentence?

Sure thing **StatSquatch**! **Machine Learning** (**ML**) is a collection of tools and techniques that transforms data into (hopefully good) decisions by making *classifications*, like whether or not someone will love a movie, or *quantitative predictions*, like how tall someone is.

② **Norm**, are you saying that machine learning is all about two things? **1)** We can use it to *classify* things and **2)** we can use it to make *quantitative predictions*?

That's right, '**Squatch**! It's all about those two things. When we use machine learning to *classify* things, we call it **Classification**. And when we make *quantitative predictions*, we call it **Regression**.

③ So, let's get started by talking about the main ideas of how machine learning is used for **Classification**.

BAM!

③ Throughout each page, you'll see circled numbers like these…

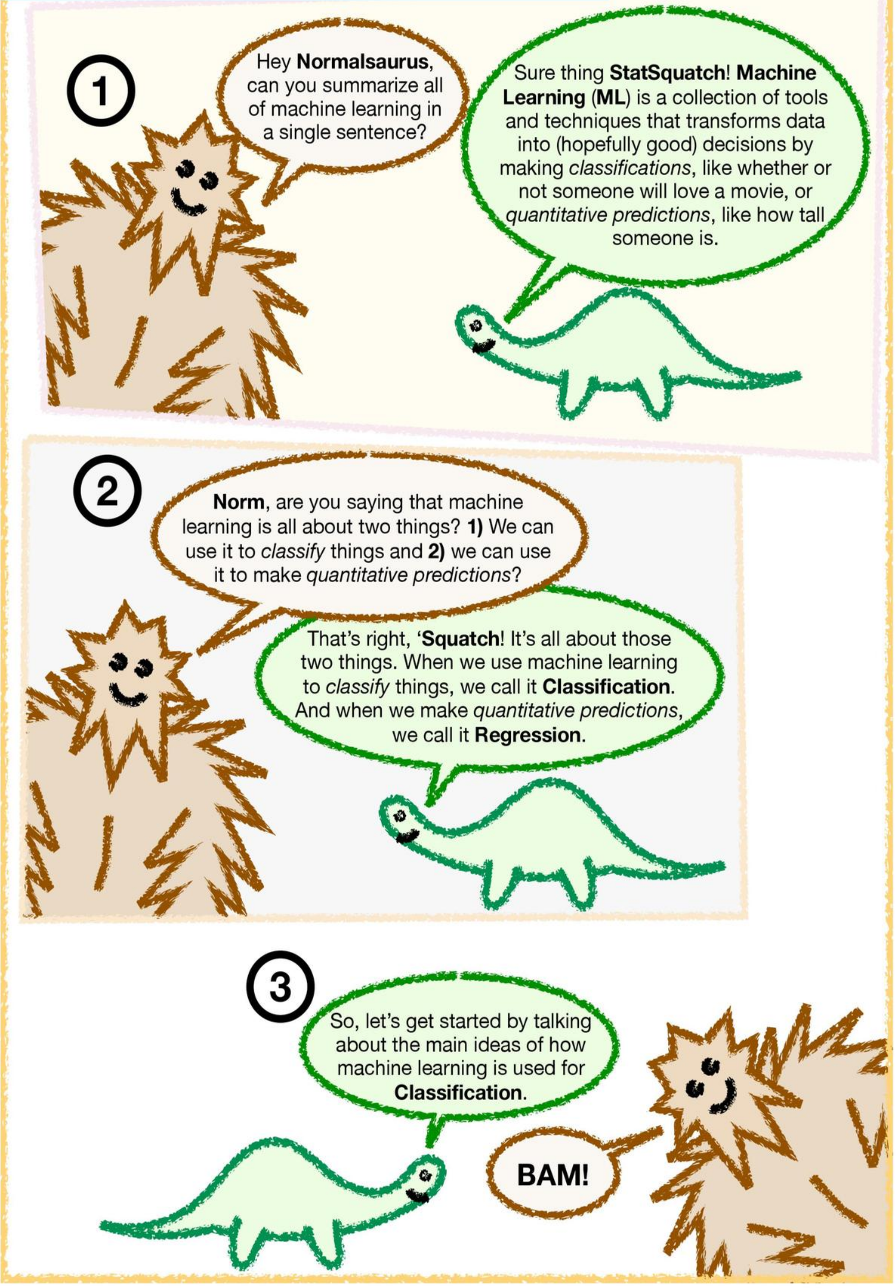
…that go from low to high, and all you have to do is follow them in order for each concept to be clearly explained.

④ **BAM!!** Now that you know how this book works, let's get started!!!

Chapter 01

Fundamental Concepts in Machine Learning!!!

Machine Learning: Main Ideas



Machine Learning Classification: Main Ideas

1 **The Problem:** We have a big pile of data, and we want to use it to make *classifications*.

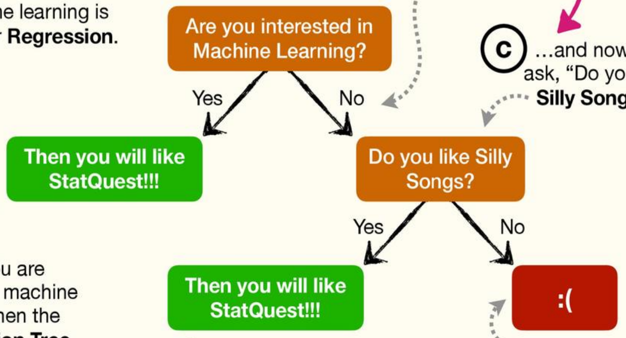
For example, we meet this person and want to **Classify** them as someone who will like **StatQuest** or not.

2 **A Solution:** We can use our data to build a **Classification Tree** (for details, see **Chapter 10**) to **Classify** a person as someone who will like **StatQuest** or not.

a Once the **Classification Tree** is built, we can use it to make **Classifications** by starting at the top and asking the question, "Are you interested in machine learning?"

b If you're *not* interested in machine learning, go to the ***right***…

c …and now we ask, "Do you like **Silly Songs**?"



d If you're not interested in machine learning and don't like **Silly Songs**, then **bummer!**

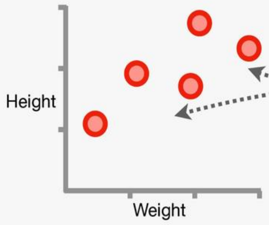
e On the other hand, if you like **Silly Songs**, then the **Classification Tree** predicts that you will like **StatQuest**!!!

f And if you are interested in machine learning, then the **Classification Tree** predicts that you will like **StatQuest**!!!

g **BAM!!!** Now let's learn the main ideas of how machine learning is used for **Regression**.

Machine Learning Regression: Main Ideas

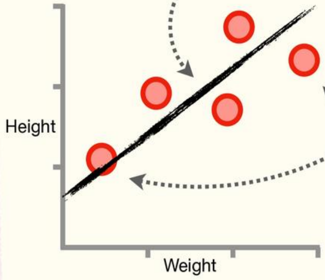
① **The Problem:** We have another pile of data, and we want to use it to make *quantitative predictions*, which means that we want to use machine learning to do **Regression**.



For example, here we measured the **Heights** and **Weights** of **5** different people.

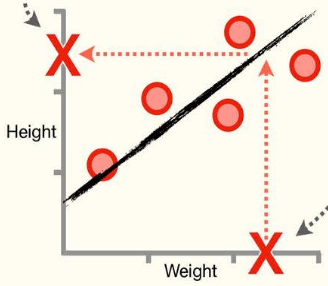
Because we can see a trend in the data—the larger the value for Weight, the taller the person—it seems reasonable to predict Height using Weight. Thus, when someone new shows up and tells us their Weight, we would like to use that information to predict their Height.

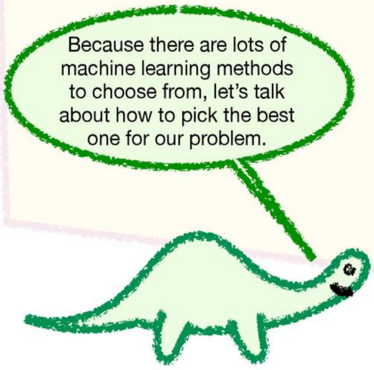
② **A Solution:** Using a method called **Linear Regression** (for details, see **Chapter 4**), we can fit a **line** to the original data we collected and use that line to make quantitative predictions.



The **line**, which goes up as the value for Weight increases, summarizes the trend we saw in the data: as a person's Weight increases, generally speaking, so does their Height.

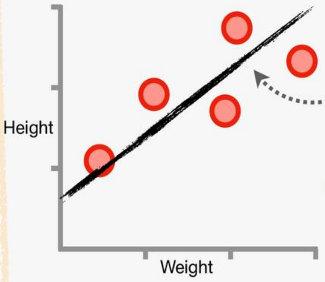
Now, if you told me that this was your Weight, then we could use the **line** to predict that this is your Height. **BAM!!!**





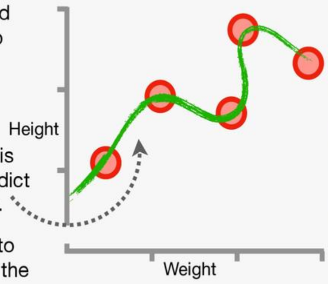
Comparing Machine Learning Methods: Main Ideas

① **The Problem:** As we'll learn in this book, machine learning consists of a lot of different methods that allow us to make **Classifications** or **Quantitative Predictions**. How do we choose which one to use?



For example, we could use this **black line** to predict Height from Weight…

…or we could use this **green squiggle** to predict Height from Weight.

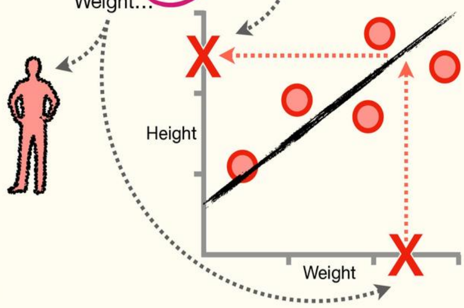


How do we decide to use the **black line** or the **green squiggle**?

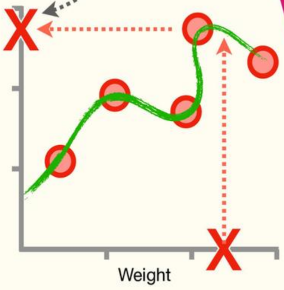
② **A Solution:** In machine learning, deciding which method to use often means just trying it and seeing how well it performs.

For example, given this person's Weight…

…the **black line** predicts this Height.

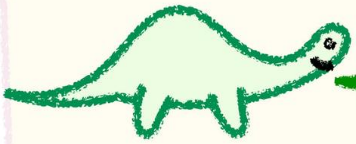


In contrast, the **green squiggle** predicts that the person will be *slightly taller*.



We can compare those two predictions to the person's *actual* Height to determine the quality of each prediction.

BAM!!!

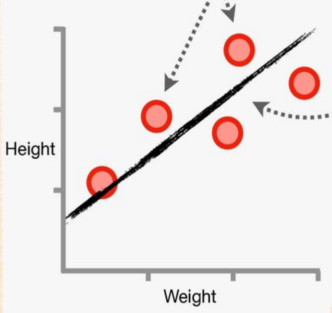


Now that we understand the **Main Ideas** of how to compare machine learning methods, let's get a better sense of how we do this in practice.

Comparing Machine Learning Methods: Intuition Part 1

1 The original data that we use to observe the trend and fit the **line** is called **Training Data**.

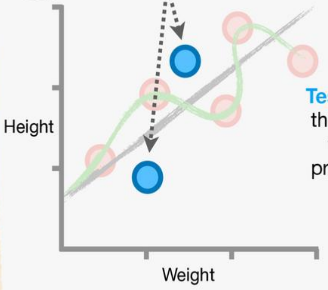
In other words, the **black line** is ***fit*** to the **Training Data**.



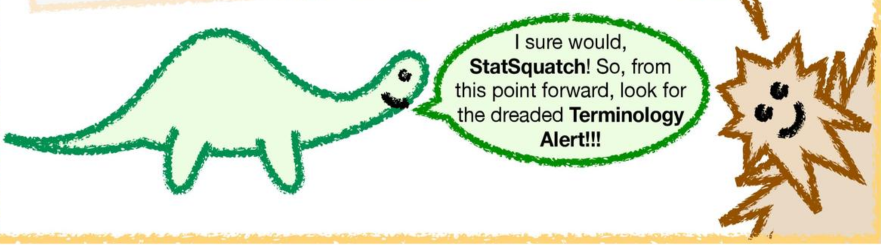
2 Alternatively, we could have fit a **green squiggle** to the **Training Data**.

The **green squiggle** fits the **Training Data** better than the **black line**, but remember the goal of machine learning is to make ***predictions***, so we need a way to determine if the **black line** or the **green squiggle** makes better predictions.

3 So, we collect more data, called **Testing Data**…

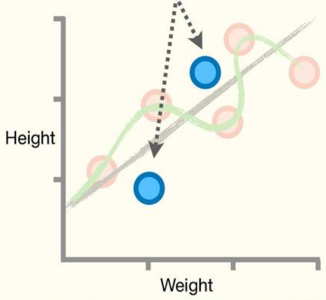


…and we use the **Testing Data** to compare the predictions made by the **black line** to the predictions made by the **green squiggle**.

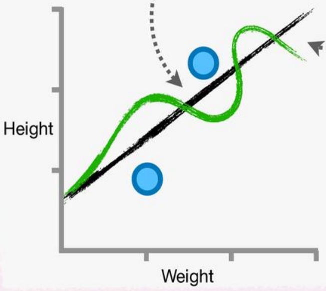


Comparing Machine Learning Methods: Intuition Part 2

4 Now, if these **blue dots** are the **Testing Data**…

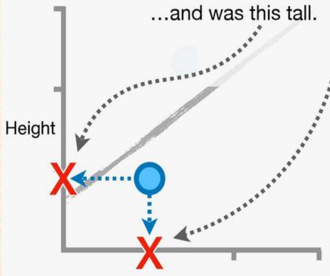


5 …then we can compare their **Observed** Heights to the Heights **Predicted** by the **black line** and the **green squiggle**.

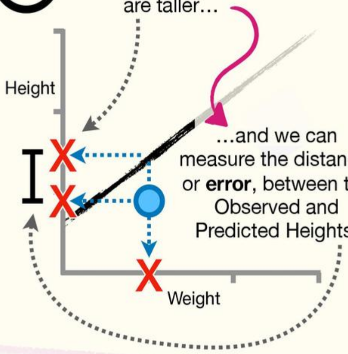


6 The first person in the **Testing Data** had this Weight…

…and was this tall.

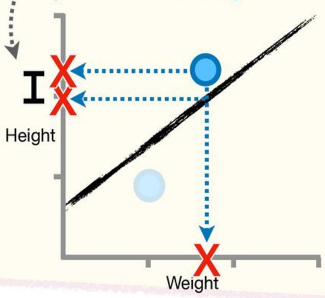


7 However, the **black line** predicts that they are taller…

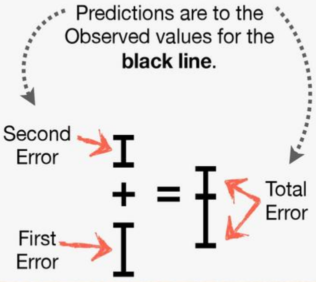


…and we can measure the distance, or **error**, between the Observed and Predicted Heights.

8 Likewise, we measure the **error** between the Observed and Predicted values for the second person in the **Testing Data**.

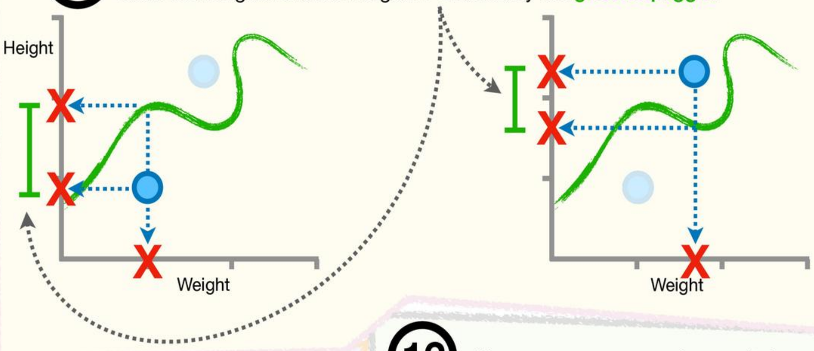


9 We can then add the two **errors** together to get a sense of how close the two Predictions are to the Observed values for the **black line**.

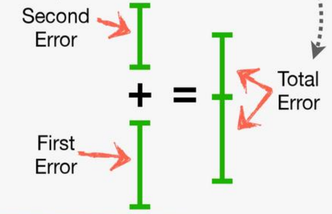


Comparing Machine Learning Methods: Intuition Part 3

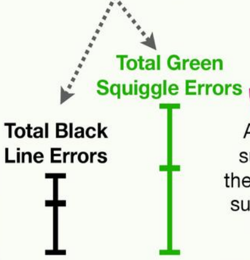
⑩ Likewise, we can measure the distances, or **errors**, between the Observed Heights and the Heights Predicted by the **green squiggle**.



⑪ We can then add the two errors together to get a sense of how close the Predictions are to the Observed values for the **green squiggle**.

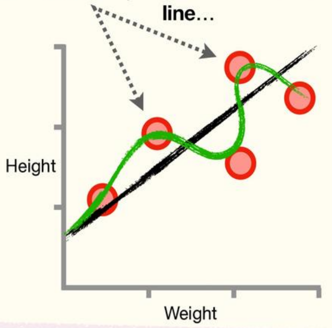


⑫ Now we can compare the predictions made by the **black line** to the predictions made by the **green squiggle** by comparing the sums of the **errors**.

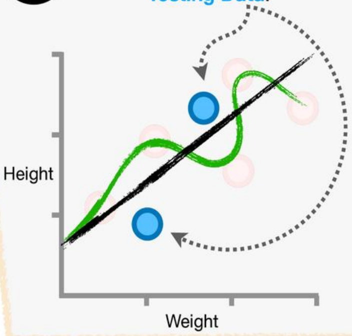


And we see that the sum of the **errors** for the **black line** is shorter, suggesting that it did a better job making predictions.

⑬ In other words, even though the **green squiggle** fit the **Training Data** way better than the **black line**…

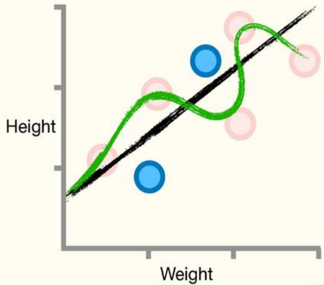


⑭ …the **black line** did a better job predicting **Height** with the **Testing Data**.

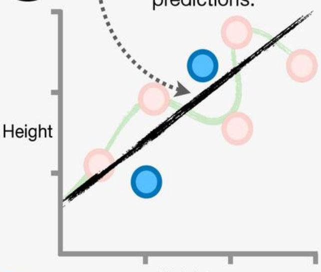


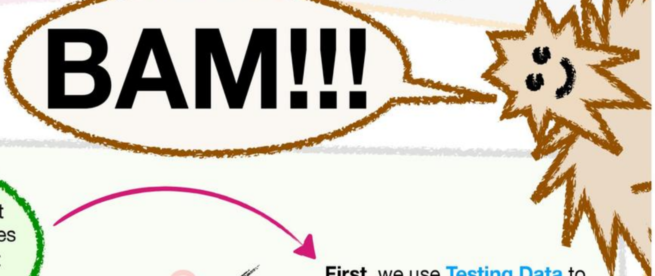
Comparing Machine Learning Methods: Intuition Part 4

⑮ So, if we had to choose between using the **black line** or the **green squiggle** to make predictions…



⑯ …we would choose the **black line** because it makes better predictions.





The example we just went through illustrates **2 Main Ideas** about machine learning.

First, we use **Testing Data** to evaluate machine learning methods.

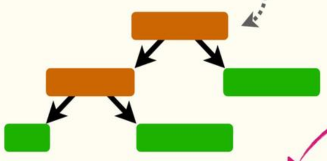
Second, just because a machine learning method fits the **Training Data** well, it doesn't mean it will perform well with the **Testing Data**.

TERMINOLOGY ALERT!!!

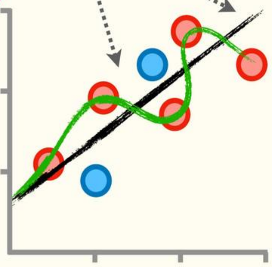
When a machine learning method fits the **Training Data** *really well* but makes *poor predictions*, we say that it is **Overfit** to the **Training Data**. **Overfitting** a machine learning method is related to something called the **Bias-Variance Tradeoff**, and we'll talk more about that later.

The Main Ideas of Machine Learning: Summary

① Now, you may be wondering why we started this book with a super simple **Decision Tree**…



…and a simple **black line** and a silly **green squiggle**…



…instead of a **Deep Learning Convolutional Neural Network** or a **[insert newest, fanciest machine learning method here]**.

② There are tons of fancy-sounding machine learning methods, like **Deep Learning Convolutional Neural Networks**, and each year something new and exciting comes along, but regardless of what you use, the most important thing is how it performs with the **Testing Data**.

There are lots of cool machine learning methods. In this book, we'll learn about…

Regression

Logistic Regression

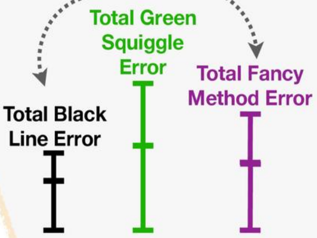
Naive Bayes

Classification Trees

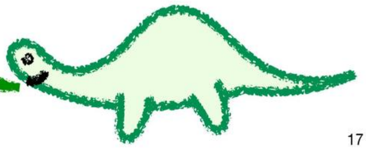
Regression Trees

Support Vector Machines

Neural Networks



BAM!!!

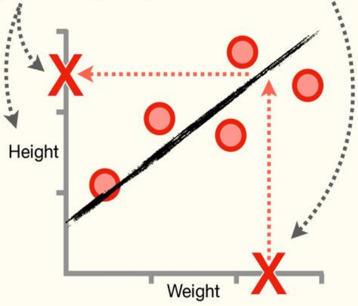


Now that we understand some of the main ideas of machine learning, let's learn some fancy terminology so we can sound smart when we talk about this stuff at dance parties.

Terminology Alert!!! Independent and Dependent Variables

① So far, we've been predicting Height…

…from Weight measurements…



…and the data have all been displayed on a nice graph. However, we can also organize the data in a nice table.

Weight	Height
0.4	1.1
1.2	1.9
1.9	1.7
2.0	2.8
2.8	2.3

Now, regardless of whether we look at the data in the graph or in the table, we can see that

Weight *varies* from person to person, and thus, Weight is called a **Variable**.

Likewise, Height *varies* from person to person, so Height is also called a **Variable**.

② That being said, we can be more specific about the types of **Variables** that Height and Weight represent.

Because our Height predictions *depend* on Weight measurements, we call Height a ***Dependent Variable***.

In contrast, because we're not predicting Weight, and thus, Weight does not depend on Height, we call Weight an ***Independent Variable***. Alternatively, Weight can be called a ***Feature***.

③ So far in our examples, we have only used Weight, a single **Independent Variable**, or **Feature**, to predict Height. However, it's very common to use multiple **Independent Variables**, or **Features**, to make predictions. For example, we might use Weight, Shoe Size and Favorite Color to predict Height.

Weight	Shoe Size	Favorite Color	Height
0.4	3	Blue	1.1
1.2	3.5	Green	1.9
1.9	4	Green	1.7
2.0	4	Pink	2.8
2.8	4.5	Blue	2.3

Bam.

Now, as we can see in the table, Weight is a *numeric measurement* and Favorite Color is a *discrete category*, so we have different types of data. Read on to learn more about these types!!!

Terminology Alert!!! Discrete and Continuous Data

① **Discrete Data…**

…is ***countable*** and only takes *specific* values.

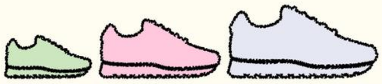
② For example, we can count the number of people that love the color **green** or love the color **blue**.

4 people love **green**

3 people love **blue**

Because we are counting individual people, and the totals can only be whole numbers, the data are **Discrete.**

③ American shoe sizes are **Discrete** because even though there are half sizes, like **8 1/2**, shoes sizes are never **8 7/36** or **9 5/18**.



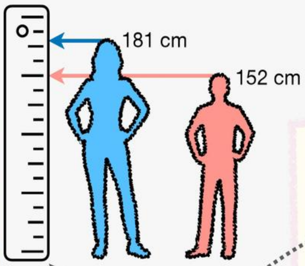
④ Rankings and other orderings are also **Discrete**. There is no award for coming in **1.68** place. Total bummer!

⑤ **Continuous Data…**

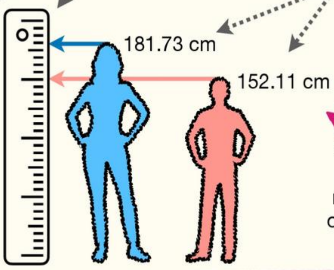
…is ***measurable*** and can take *any* numeric value within a range.

⑥ For example, Height measurements are **Continuous** data.

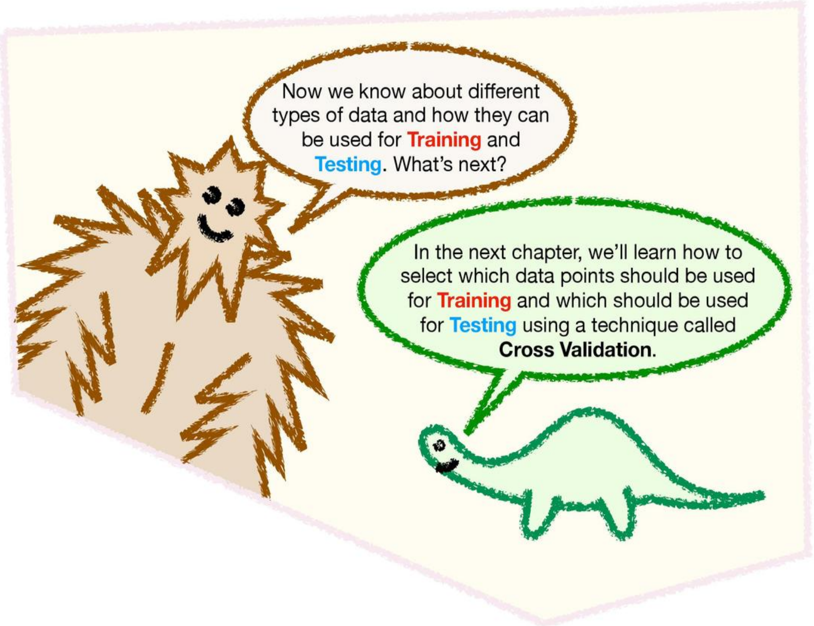
Height measurements can be any number between **0** and the height of the tallest person on the planet.



⑦ **NOTE:** If we get a more precise ruler… …then the measurements get more precise.



So the precision of **Continuous** measurements is only limited by the tools we use.


Now we know about different types of data and how they can be used for **Training** and **Testing**. What's next?
In the next chapter, we'll learn how to select which data points should be used for **Training** and which should be used for **Testing** using a technique called **Cross Validation**.

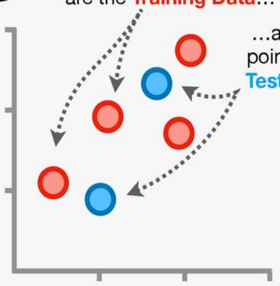
Chapter 02

Cross Validation!!!

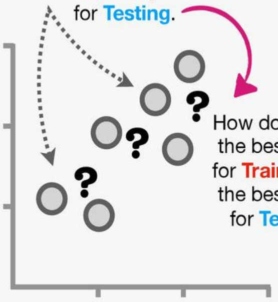
Cross Validation: Main Ideas

① **The Problem:** So far, we've simply been told which points are the Training Data…

…and which points are the Testing Data.

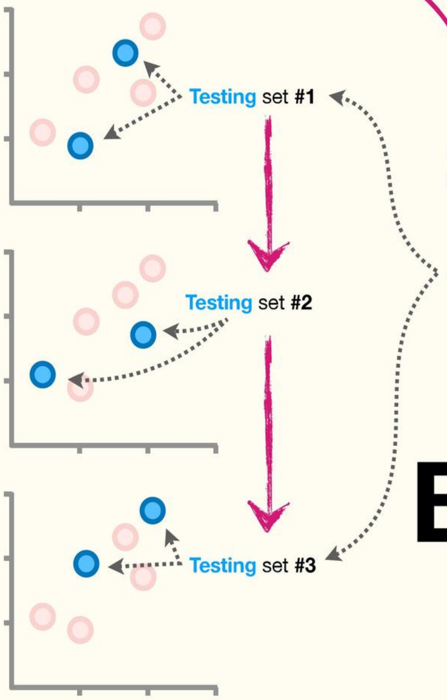


However, usually no one tells us what is for Training and what is for Testing.



How do we pick the best points for Training and the best points for Testing?

② **A Solution:** When we're not told which data should be used for Training and for Testing, we can use **Cross Validation** to figure out which is which in an unbiased way.



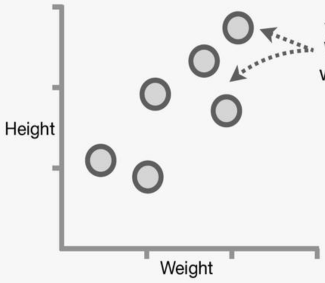
Rather than worry too much about which specific points are best for Training and best for Testing, **Cross Validation** uses *all* points for both in an *iterative* way, meaning that we use them in steps.

BAM!!!

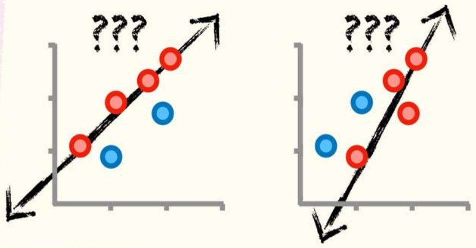
Cross Validation: Details Part 1

① Imagine we gathered these **6** pairs of Weight and Height measurements…

…and because we see a trend that people with larger Weights tend to be taller, we want to use Weight to predict Height…

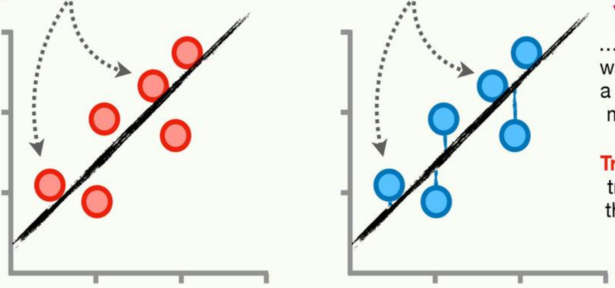


② …so we decide to fit a **line** to the data with **Linear Regression** (for details, see **Chapter 4**). However, we don't know which points to use for **Training** and which to use for **Testing**.



③ A terrible idea would be to use ***all*** of the data for **Training**…

…and then reuse the exact same data for **Testing**…



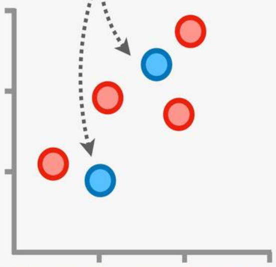
…because the only way to determine if a machine learning method has been **Overfit** to the **Training Data** is to try it on new data that it hasn't seen before.

TERMINOLOGY ALERT!!!

Reusing the same data for **Training** and **Testing** is called **Data Leakage**, and it usually results in you believing the machine learning method will perform better than it actually does because it is **Overfit**.

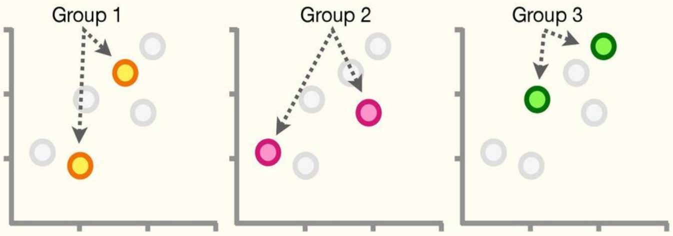
Cross Validation: Details Part 2

④ A slightly better idea is to randomly select some data to use *only* for **Testing** and select and use the rest for **Training**.



This avoids **Data Leakage**, but how do we know we selected the best data for **Testing**?

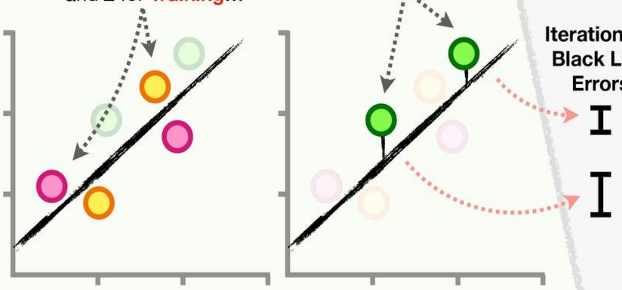
⑤ **Cross Validation** solves the problem of not knowing which points are the best for **Testing** by using them *all* in an *iterative* way. The first step is to randomly assign the data to different groups. In this example, we divide the data into **3** groups, where each group consists of **2** points.



⑥ Now, in the first **Cross Validation** iteration, we'll use **Groups 1** and **2** for **Training**…

…and **Group 3** for **Testing**.

⑦ Then, just like before, we can measure the errors for each point in the **Testing Data**…



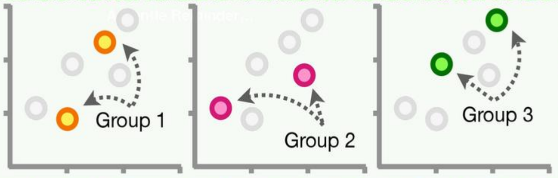
Iteration #1: Black Line Errors

…however, unlike before, we don't stop here. Instead, we keep iterating so that **Groups 1** and **2** can each get a turn to be used for **Testing**.

Cross Validation: Details Part 3

8 Because we have **3** groups of data points, we'll do **3** iterations, which ensures that each group is used for Testing. The number of iterations are also called **Folds**, so this is called **3-Fold Cross Validation**.

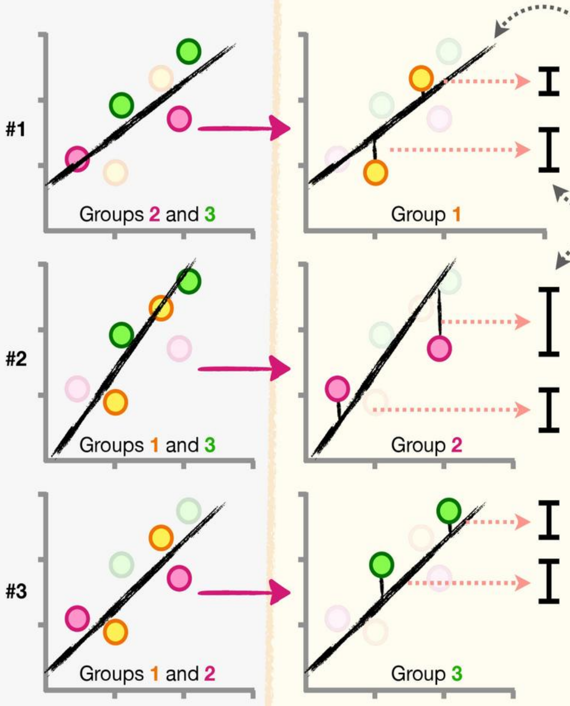
Gentle Reminder: These are the original **3** groups.



9 So, these are the **3** iterations of Training...

10 ...and these are the **3** iterations of Testing.

NOTE: Because each iteration uses a different combination of data for Training, each iteration results in a slightly different fitted **line**.



A different fitted line combined with using different data for Testing results in each iteration giving us different prediction errors.

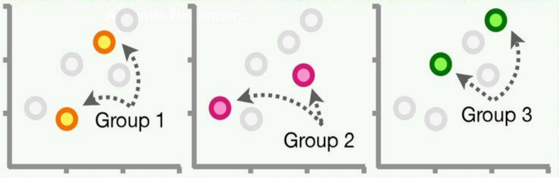
We can average these errors to get a general sense of how well this model will perform with future data...

...or we can compare these errors to errors made by another method.

Cross Validation: Details Part 4

⑪ For example, we could use **3-Fold Cross Validation** to compare the errors from the **black line** to the errors from the **green squiggle**.

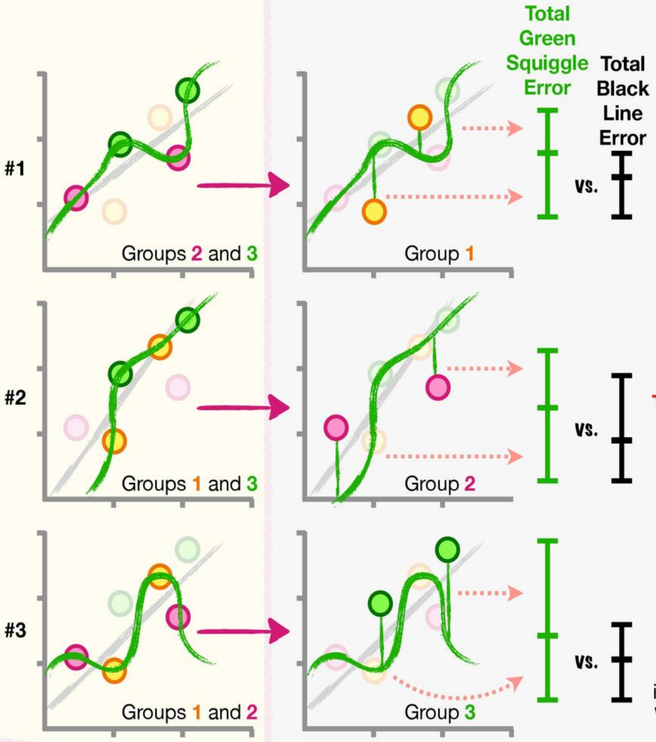
Gentle Reminder: These are the original **3** groups.



⑫ Training

⑬ Testing

In this case, all **3** iterations of the **3-Fold Cross Validation** show that the **black line** does a better job making predictions than the **green squiggle**.



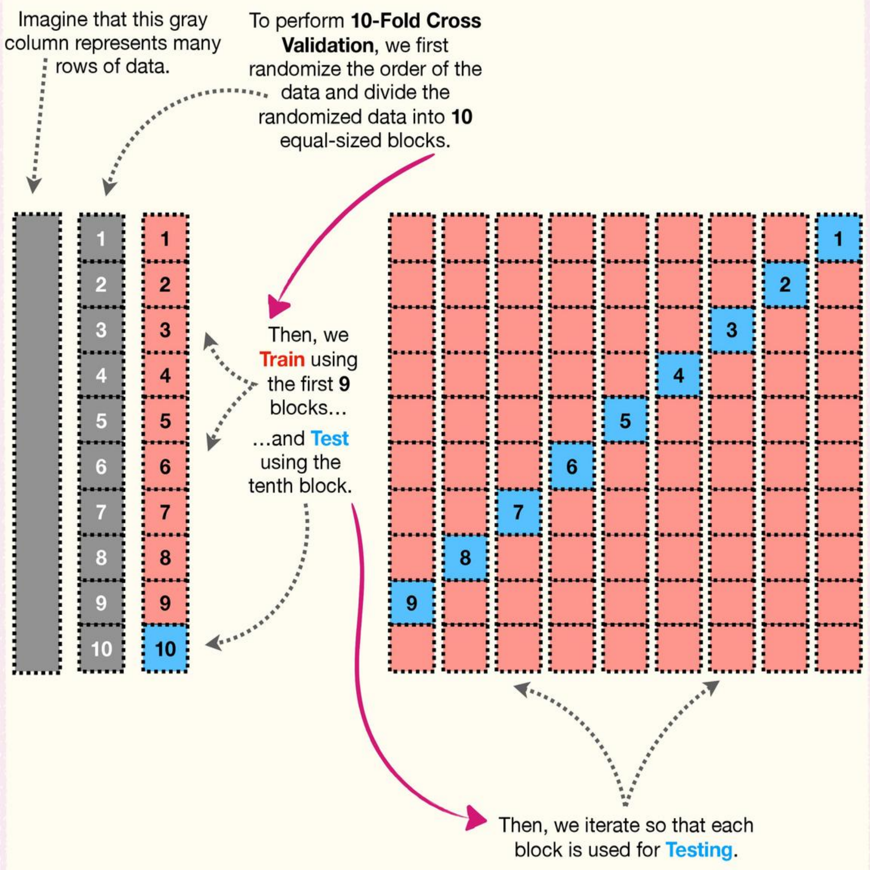
By using **Cross Validation**, we can be more confident that the **black line** will perform better with new data without having to worry about whether or not we selected the best data for Training and the best data for Testing.

BAM!!!

NOTE: In this example, the **black line** consistently performed better than the **green squiggle**, but this isn't usually the case. We'll talk more about this later.

Cross Validation: Details Part 5

⑭ When we have a lot of data, **10-Fold Cross Validation** is commonly used.



Imagine that this gray column represents many rows of data.

To perform **10-Fold Cross Validation**, we first randomize the order of the data and divide the randomized data into **10** equal-sized blocks.

Then, we **Train** using the first **9** blocks…

…and **Test** using the tenth block.

Then, we iterate so that each block is used for **Testing**.

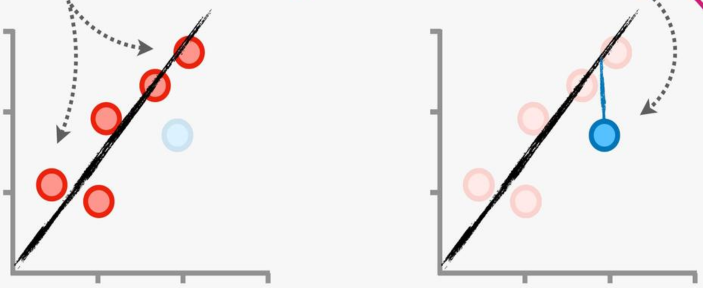
DOUBLE BAM!!!

Cross Validation: Details Part 6

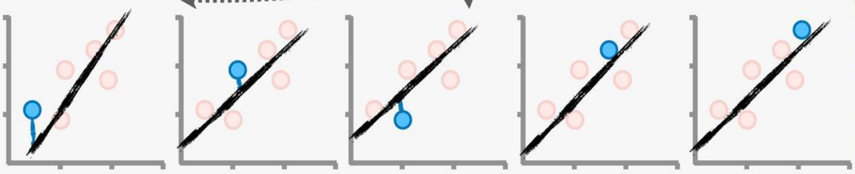
⑮ Another commonly used form of **Cross Validation** is called **Leave-One-Out**.

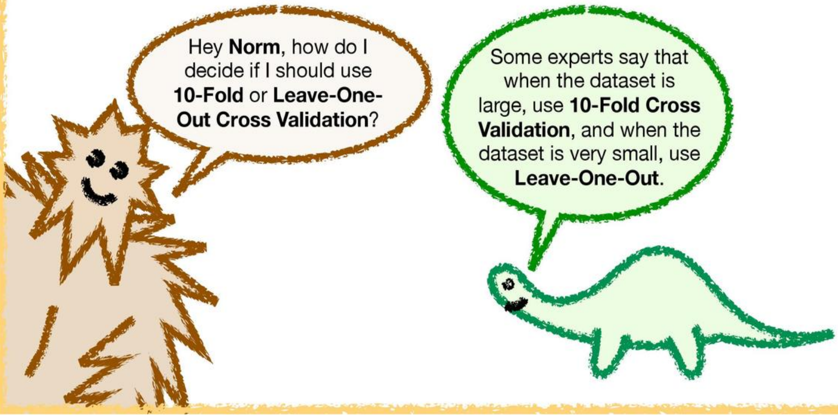
Leave-One-Out Cross Validation uses all but one point for **Training**…

…and uses the one remaining point for **Testing**…



…and then iterates until every single point has been used for **Testing**.

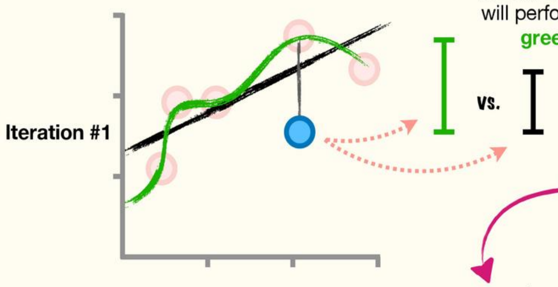




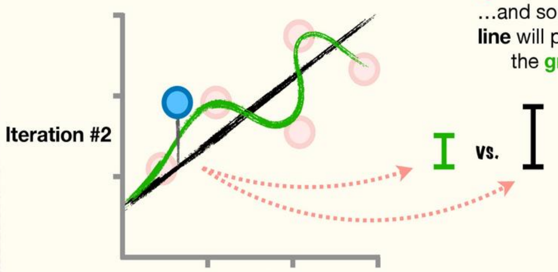
Cross Validation: Details Part 7

⑯ When we use **Cross Validation** to compare machine learning methods, for example, if we wanted to compare a **black line** to a **green squiggle**…

…sometimes the **black line** will perform *better* than the **green squiggle**…

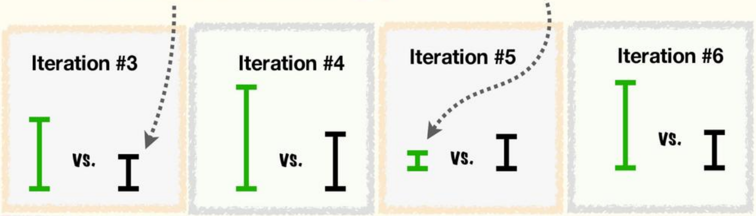


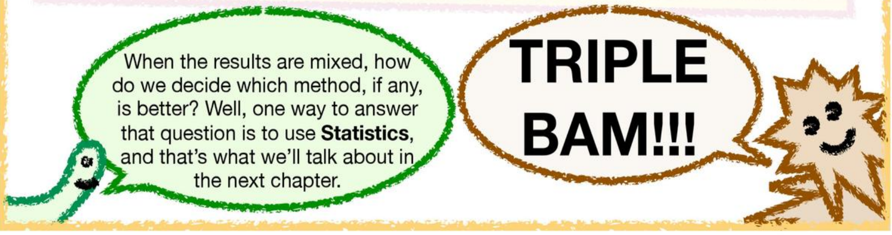
…and sometimes the **black line** will perform *worse* than the **green squiggle**.



And, after doing all of the iterations, we're left with a variety of results, some showing that the **black line** is better…

…and some showing that the **green squiggle** is better.





When the results are mixed, how do we decide which method, if any, is better? Well, one way to answer that question is to use **Statistics**, and that's what we'll talk about in the next chapter.

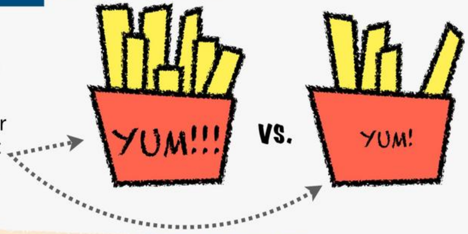
Chapter 03

Fundamental Concepts in Statistics!!!

Statistics: Main Ideas

1 **The Problem:** The world is an interesting place, and things are not always the same.

For example, every time we order french fries, we don't always get the exact same number of fries.



2 **A Solution: Statistics** provides us with a set of tools to quantify the variation that we find in everything and, for the purposes of machine learning, helps us make predictions and quantify how confident we should be in those predictions.

For example, once we notice that we don't always get the exact same number of fries, we can keep track of the number of fries we get each day…

Fry Diary

Monday: **21** fries

Tuesday: **24** fries

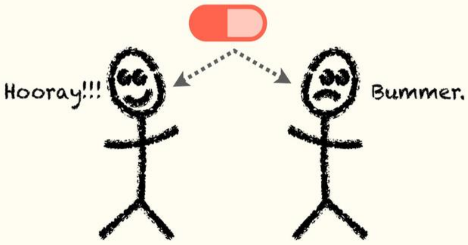
Wednesday: **19** fries

Thursday: ???

…and statistics can help us predict how many fries we'll get the next time we order them, and it tells us how confident we should be in that prediction.

Alternatively, if we have a new medicine that helps some people but hurts others…

…statistics can help us predict who will be helped by the medicine and who will be hurt, and it tells us how confident we should be in that prediction. This information can help us make decisions about how to treat people.



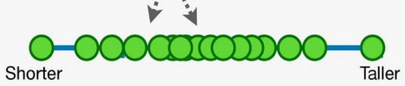
For example, if we predict that the medicine will help, but we're not very confident in that prediction, we might not recommend the medicine and use a different therapy to help the patient.

3 The first step in making predictions is to identify trends in the data that we've collected, so let's talk about how to do that with a **Histogram**.

Histograms: Main Ideas

① **The Problem:** We have a lot of measurements and want to gain insights into their hidden trends.

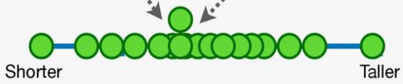
For example, imagine we measured the Heights of so many people that the data, represented by **green dots**, overlap, and some **green dots** are completely hidden.



We could try to make it easier to see the hidden measurements by stacking any that are exactly the same…

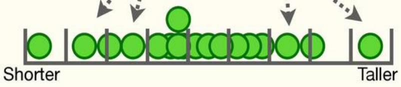
…but measurements that are *exactly* the same are rare, and a lot of the **green dots** are still hidden.

:(



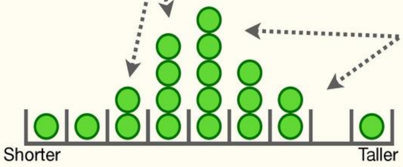
② **A Solution: Histograms** are one of the most basic, but surprisingly useful, statistical tools that we can use to gain insights into data.

Instead of stacking measurements that are *exactly* the same, we divide the range of values into bins…



…and stack the measurements that fall in the same bin…

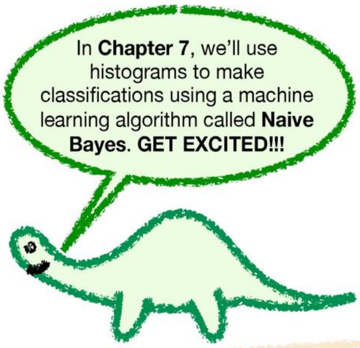
…and we end up with a **histogram!!!**



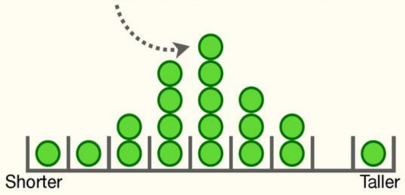
The **histogram** makes it easy to see trends in the data. In this case, we see that most people had close to average heights.

BAM!!!

Histograms: Details

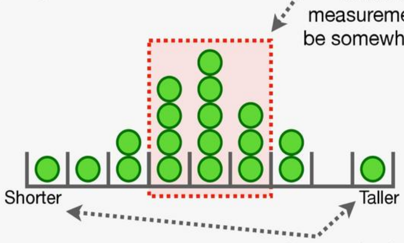


1 The taller the stack within a bin, the more measurements we made that fall into that bin.



2 We can use the histogram to estimate the probability of getting future measurements.

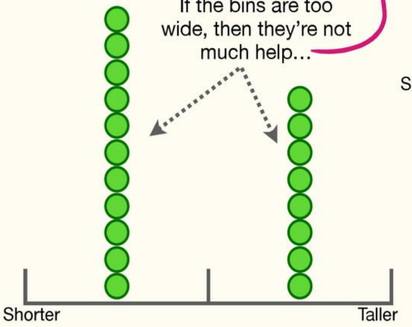
Because most of the measurements are inside this **red box**, we might be willing to bet that the next measurement we make will be somewhere in this range.



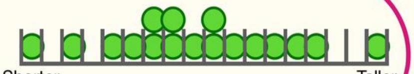
Extremely short or tall measurements are rarer and less likely to happen in the future.

3 **NOTE:** Figuring out how wide to make the bins can be tricky.

If the bins are too wide, then they're not much help…

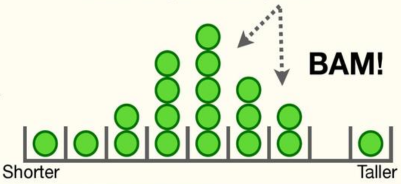


…and if the bins are too narrow, then they're not much help…



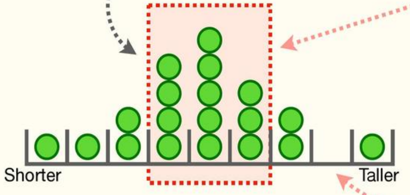
…so, sometimes you have to try a bunch of different bin widths to get a clear picture.

BAM!



Histograms: Calculating Probabilities Step-by-Step

① If we want to estimate the probability that the next measurement will be in this **red box**…



…we count the number of measurements, or observations, *in the box* and get **12**…

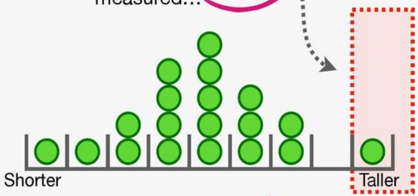
…and divide by the *total* number of measurements, **19**…

$$\frac{12}{19} = \boxed{0.63}$$

…and we get **0.63**. In theory, this means that **63%** of the time we'll get a measurement in the **red box**.

However, the confidence we have in this estimate depends on the number of measurements. Generally speaking, the more measurements you have, the more confidence you can have in the the estimate.

② To estimate the probability that the next measurement will be in this **red box**, which only contains the tallest person we measured…



…we count the number of measurements in the box and get **1**…

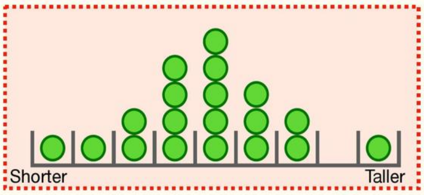
…and divide by the total number of measurements, **19**…

$$\frac{1}{19} = \boxed{0.05}$$

…and the result, **0.05**, tells us that, in theory, there's a **5%** chance that the next measurement will fall within the box. In other words, it's fairly rare to measure someone who is really tall.

Histograms: Calculating Probabilities Step-by-Step

3 To estimate the probability that the next measurement will be in a **red box** that spans all of the data…



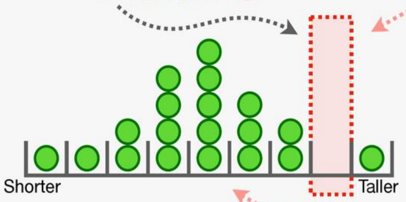
…we count the number of measurements in the box, **19**…

…and divide by the total number of measurements, **19**…

$$\frac{19}{19} = \boxed{1}$$

…and the result, **1**, tells us that there's a **100%** chance that the next measurement will fall within the box. In other words, the maximum probability is **1**.

4 If we want to estimate the probability that the next measurement will be in this **red box**…



…we count the number of measurements in the box, **0**…

…and divide by the total number of measurements, **19**…

$$\frac{0}{19} = \boxed{0}$$

…and we get **0**. This is the minimum probability and, in theory, it means that we'll ***never*** get a measurement in this box. However, it could be that the only reason the box was empty is that we simply did not measure enough people.

If we measure more people, we may either find someone who fits in this bin or become more confident that it should be empty. However, sometimes getting more measurements can be expensive, or take a lot of time, or both. This is a problem!!!

The good news is that we can solve this problem with a **Probability Distribution**. **Bam!**

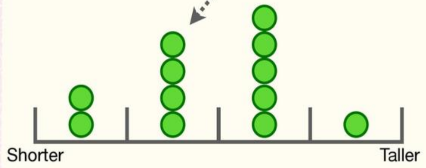
Probability Distributions: Main Ideas

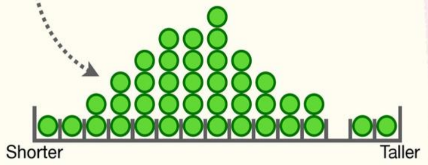
①

The Problem: If we don't have much data, then we can't make very precise probability estimates with a histogram…

…however, collecting tons of data to make precise estimates can be time-consuming and expensive. Is there another way?

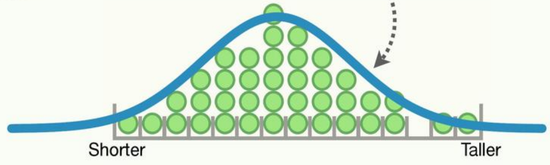
YES!!!

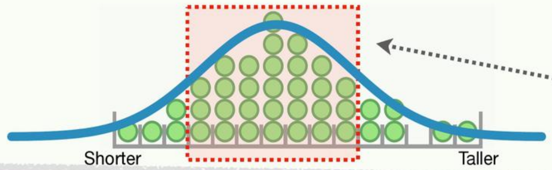




②

A Solution: We can use a **Probability Distribution**, which, in this example, is represented by a **blue, bell- shaped curve**, to approximate a histogram.





③

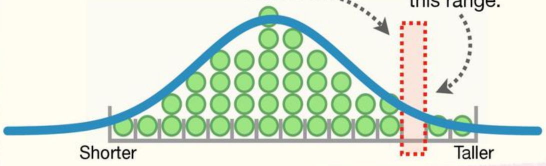
This **blue, bell-shaped curve** tells us the same types of things that the histogram tells us.

For example, the relatively large amount of area under the curve in this **red box** tells us that there's a relatively high probability that we will measure someone whose value falls in this region.

④

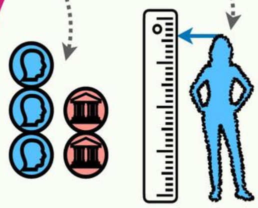
Now, even though we never measured someone who's value fell in this range…

…we can use the area under the curve to estimate the probability of measuring a value in this range.



⑤

NOTE: Because we have **Discrete** and **Continuous** data…



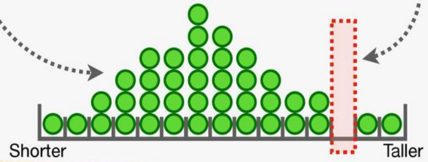
…there are **Discrete** and **Continuous Probability Distributions.**

So let's start by learning about **Discrete Probability Distributions.**

Discrete Probability Distributions: Main Ideas

① **The Problem:** Although, technically speaking, histograms are **Discrete Distributions**, meaning data can be put into discrete bins and we can use those to estimate probabilities…

…they require that we collect a lot of data, and it's not always clear what we should do with blank spaces in the histograms.



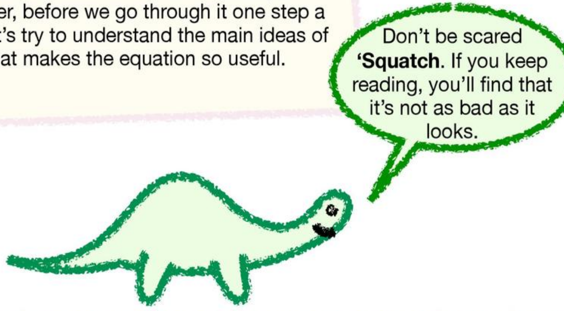
② **A Solution:** When we have discrete data, instead of collecting a ton of data to make a histogram and then worrying about blank spaces when calculating probabilities, we can let ***mathematical equations*** do all of the hard work for us.

③ One of the most commonly used **Discrete Probability Distributions** is the **Binomial Distribution**.

As you can see, it's a mathematical equation, so it doesn't depend on collecting tons of data, but, at least to **StatSquatch**, it looks *super scary!!!*

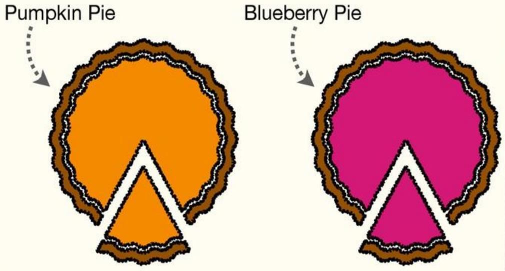
$$p(x \mid n, p) = \left(\frac{n!}{x!(n-x)!} \right) p^x (1-p)^{n-x}$$

④ The good news is that, deep down inside, the **Binomial Distribution** is really simple. However, before we go through it one step a time, let's try to understand the main ideas of what makes the equation so useful.



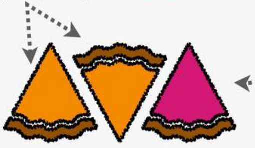
The Binomial Distribution: Main Ideas Part 1

1 First, let's imagine we're walking down the street in **StatLand** and we ask the first **3** people we meet if they prefer pumpkin pie or blueberry pie…



2 …and the first **2** people say they prefer pumpkin pie…

…and the last person says they prefer blueberry pie.



Based on our extensive experience judging pie contests in **StatLand**, we know that **70%** of people prefer pumpkin pie, while **30%** prefer blueberry pie. So now let's calculate the probability of observing that the first two people prefer pumpkin pie and the third person prefers blueberry.

3 The probability that the first person will prefer pumpkin pie is **0.7**…

…and the probability that the first two people will prefer pumpkin pie is **0.49**…

(Psst! If this math is blowing your mind, check out **Appendix A**.)

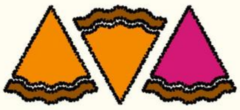
…and the probability that the first two people will prefer pumpkin pie and the third person prefers blueberry is **0.147**.

(Again, if this math is blowing your mind, check out **Appendix A**.)

0.7

$0.7 \times 0.7 = \mathbf{0.49}$

$0.7 \times 0.7 \times 0.3 = \mathbf{0.147}$



NOTE: 0.147 is the probability of observing that the first two people prefer pumpkin pie and the third person prefers blueberry…

…it is *not* the probability that **2** out of **3** people prefer pumpkin pie.

Let's find out why on the next page!!!

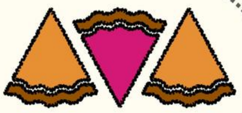
The Binomial Distribution: Main Ideas Part 2

④ It could have just as easily been the case that the first person said they prefer blueberry and the last two said they prefer pumpkin.

0.3 x 0.7 x 0.7 = 0.147

In this case, we would multiply the numbers together in a different order, but the probability would still be **0.147** (see **Appendix A** for details).

⑤ Likewise, if only the second person said they prefer blueberry, we would multiply the numbers together in a different order and still get **0.147**.



0.7 x 0.3 x 0.7 = 0.147

⑥ So, we see that all three combinations are equally probable…

0.3 x 0.7 x 0.7 = **0.147**

0.7 x 0.3 x 0.7 = **0.147**

0.7 x 0.7 x 0.3 = **0.147**

⑦ …and that means that the probability of observing that **2** out of **3** people prefer pumpkin pie is the ***sum*** of the **3** possible arrangements of people's pie preferences, **0.441**.

0.3 x 0.7 x 0.7 = **0.147**

+

0.7 x 0.3 x 0.7 = **0.147**

+

0.7 x 0.7 x 0.3 = **0.147**

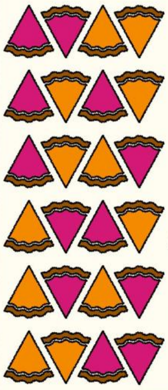
= 0.441

NOTE: Calculating by hand the probability of observing that **2** out of **3** people prefer pumpkin pie was not that bad. All we did was draw the **3** different ways **2** out of **3** people might prefer pumpkin pie, calculate the probability of each way, and add up the probabilities.

Bam.

The Binomial Distribution: Main Ideas Part 3

8 However, things quickly get tedious when we start asking more people which pie they prefer.

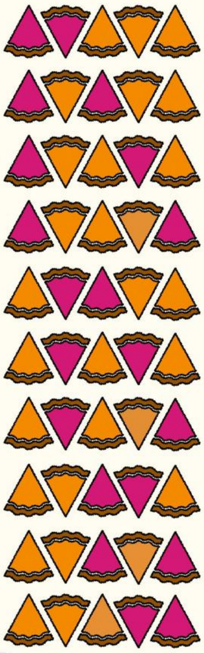


For example, if we wanted to calculate the probability of observing that **2** out of **4** people prefer pumpkin pie, we have to calculate and sum the individual probabilities from **6** different arrangements…

…and there are **10** ways to arrange **3** out of **5** people who prefer pumpkin pie.

UGH!!! Drawing all of these slices of delicious pie is super tedious.

:(



9 So, instead of drawing out different arrangements of pie slices, we can use the equation for the **Binomial Distribution** to calculate the probabilities directly.

$$p(x\,|\,n,\,p) = \left(\frac{n!}{x!(n-x)!}\right)p^{x}(1-p)^{n-x}$$

BAM!!!

In the next pages, we'll use the **Binomial Distribution** to calculate the probabilities of pie preference among **3** people, but it works in any situation that has binary outcomes, like wins and losses, yeses and noes, or successes and failures.

0.3 x 0.7 x 0.7 = **0.147**

+

0.7 x 0.3 x 0.7 = **0.147**

+

0.7 x 0.7 x 0.3 = **0.147**

= 0.441

Now that we understand why the equation for the **Binomial Distribution** is so useful, let's walk through, one step at a time, how the equation calculates the probability of observing **2** out of **3** people who prefer pumpkin pie.

The Binomial Distribution: Details Part 1

① First, let's focus on just the left-hand side of the equation.

In our pie example, ***x*** is the number of people who prefer pumpkin pie, so in this case, ***x* = 2**…

…***n*** is the number of people we ask. In this case, ***n* = 3**…

…and this ***p*** is the probability that someone prefers pumpkin pie. In this case, ***p* = 0.7**…

$$p(x\,|\,n,\,p) = \left(\frac{n!}{x!(n-x)!}\right)p^x(1-p)^{n-x}$$

② …this ***p*** means **probability**…

…the vertical bar or pipe symbol means **given** or **given that**…

…and the comma between ***n*** and ***p*** means **and**…

So, the left-hand side of the equation reads:

$$p(x\,|\,n,\,p) =$$

"The *probability* we meet ***x* = 2** people who prefer pumpkin pie, *given that* we ask ***n* = 3** people *and* the probability of someone preferring pumpkin pie is ***p* = 0.7**."

BAM!

Gentle Reminder: We're using the equation for the **Binomial Distribution** to calculate the probability that **2** out of **3** people prefer pumpkin pie…

$0.3 \times 0.7 \times 0.7 = \mathbf{0.147}$

+

$0.7 \times 0.3 \times 0.7 = \mathbf{0.147}$

+

$0.7 \times 0.7 \times 0.3 = \mathbf{0.147}$

= 0.441

The Binomial Distribution: Details Part 2

3 Now, let's look at the first part on the right-hand side of the equation. **StatSquatch** says it looks scary because it has factorials (the exclamation points; see below for details), but it's not that bad.

Despite the factorials, the first term simply represents the number of different ways we can meet **3** people, **2** of whom prefer pumpkin pie…

…and, as we saw earlier, there are **3** different ways that **2** out of **3** people we meet can prefer pumpkin pie.

$$p(x \mid n, p) = \left(\frac{n!}{x!(n-x)!}\right) p^x (1-p)^{n-x}$$

4 When we plug in ***x* = 2**, the number of people who prefer pumpkin pie…

…and ***n* = 3**, the number of people we asked, and then do the math…

…we get **3**, the same number we got when we did everything by hand.

$$\frac{n!}{x!\,(n-x)!} = \frac{3!}{2!\,(3-2)!} = \frac{3!}{2!\,(1)!} = \frac{3 \times 2 \times 1}{2 \times 1 \times 1} = 3$$

NOTE: If ***x*** is the number of people who prefer pumpkin pie, and ***n*** is the total number of people, then **(*n* - *x*)** = the number of people who prefer blueberry pie.

Gentle Reminder: We're using the equation for the **Binomial Distribution** to calculate the probability that **2** out of **3** people prefer pumpkin pie…

0.3 x 0.7 x 0.7 = **0.147**

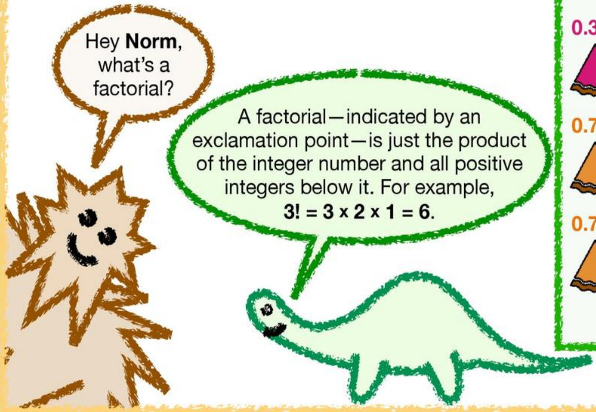
+

0.7 x 0.3 x 0.7 = **0.147**

+

0.7 x 0.7 x 0.3 = **0.147**

= 0.441



The Binomial Distribution: Details Part 3

⑤ Now let's look at the second term on the right hand side.

The second term is just the probability that **2** out of the **3** people prefer pumpkin pie.

In other words, since ***p***, the probability that someone prefers pumpkin pie, is **0.7**…

…and there are ***x*** **= 2** people who prefer pumpkin pie, the second term = 0.7^2 = 0.7 x 0.7.

$$p(x \mid n, p) = \left(\frac{n!}{x!(n-x)!}\right) p^x (1-p)^{n-x}$$

0.3 x 0.7 x 0.7 = 0.147

⑥ The third and final term is the probability that **1** out of **3** people prefers blueberry pie…

…because if ***p*** is the probability that someone prefers pumpkin pie, **(1 - *p*)** is the probability that someone prefers blueberry pie…

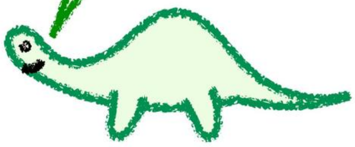
…and if ***x*** is the number of people who prefer pumpkin pie and ***n*** is the total number of people we asked, then ***n* - *x*** is the number of people who prefer blueberry pie.

$$p(x \mid n, p) = \left(\frac{n!}{x!(n-x)!}\right) p^x (1-p)^{n-x}$$

So, in this example, if we plug in ***p*** **= 0.7**, ***n*** **= 3**, and ***x*** **= 2**, we get **0.3**.

$$(1 - p)^{n - x} = (1 - 0.7)^{3-2} = 0.3^1 = 0.3$$

Just so you know, sometimes people let ***q*** **= (1 - *p*)**, and use ***q*** in the formula instead of **(1 - *p*)**.

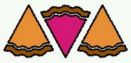


Gentle Reminder: We're using the equation for the **Binomial Distribution** to calculate the probability that **2** out of **3** people prefer pumpkin pie…

0.3 x 0.7 x 0.7 = **0.147**

+

0.7 x 0.3 x 0.7 = **0.147**



+

0.7 x 0.7 x 0.3 = **0.147**

= 0.441

The Binomial Distribution: Details Part 4

⑦ Now that we've looked at each part of the equation for the **Binomial Distribution**, let's put everything together and solve for the probability that **2** out of **3** people we meet prefer pumpkin pie.

We start by plugging in the number of people who prefer pumpkin pie, ***x* = 2**, the number of people we asked, ***n* = 3**, and the probability that someone prefers pumpkin pie, ***p* = 0.7**…

$$p(x = 2 \mid n = 3, p = 0.7) = \left(\frac{n!}{x!(n-x)!}\right)p^x(1-p)^{n-x}$$

…then we just do the math…

$$= \left(\frac{3!}{2!(3-2)!}\right)0.7^2(1-0.7)^{3-2}$$

(Psst! Remember: the first term is the number of ways we can arrange the pie preferences, the second term is the probability that **2** people prefer pumpkin pie, and the last term is the probability that **1** person prefers blueberry pie.)

$$= 3 \times 0.7^2 \times (0.3)^1$$

$$= 3 \times 0.7 \times 0.7 \times 0.3$$

$$\boxed{= 0.441}$$

…and the result is **0.441**, which is the same value we got when we drew pictures of the slices of pie.

Gentle Reminder: We're using the equation for the **Binomial Distribution** to calculate the probability that **2** out of **3** people prefer pumpkin pie…

0.3 x **0.7** x **0.7** = **0.147**

+

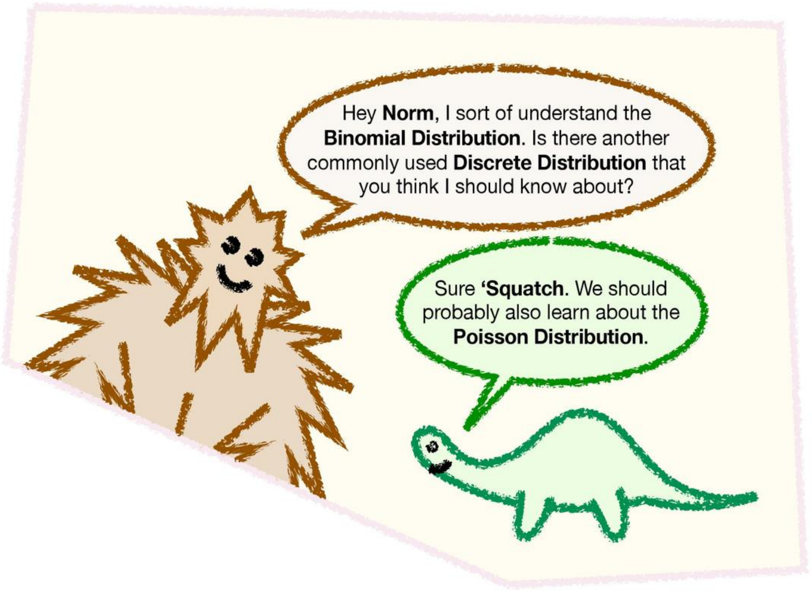
0.7 x **0.3** x **0.7** = **0.147**

+

0.7 x **0.7** x **0.3** = **0.147**

= 0.441

TRIPLE BAM!!!


Hey **Norm**, I sort of understand the **Binomial Distribution.** Is there another commonly used **Discrete Distribution** that you think I should know about?
Sure **'Squatch.** We should probably also learn about the **Poisson Distribution**.

The Poisson Distribution: Details

① So far, we've seen how the **Binomial Distribution** gives us probabilities for sequences of binary outcomes, like **2** out of **3** people preferring pumpkin pie, but there are lots of other **Discrete Probability Distributions** for lots of different situations.

② For example, if you can read, on average, **10** pages of this book in an hour, then you can use the **Poisson Distribution** to calculate the probability that in the next hour, you'll read exactly **8** pages.

The equation for the **Poisson Distribution** looks super fancy because it uses the Greek character **λ**, ***lambda***, but lambda is just the average. So, in this example, **λ** = **10** pages an hour.

$$p(x \mid \lambda) = \frac{e^{-\lambda}\lambda^{x}}{x!}$$

NOTE: This '**e**' is **Euler's number**, which is roughly **2.72**.

x is the number of pages we think we might read in the next hour. In this example, ***x*** = **8**.

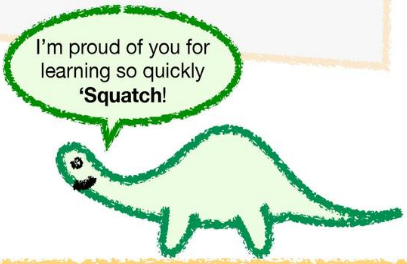
③ Now we just plug in the numbers and do the math…

$$p(x = 8 \mid \lambda = 10) = \frac{e^{-\lambda}\lambda^{x}}{x!} = \frac{e^{-10}10^{8}}{8!}$$

$$= \frac{e^{-10}10^{8}}{8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1} = 0.113$$

…and we get **0.113**. So the probability that you'll read exactly **8** pages in the next hour, given that, on average, you read **10** pages per hour, is **0.113**.

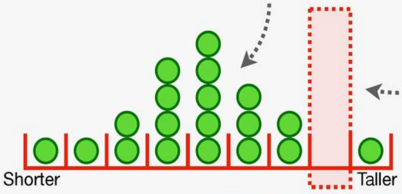
BAM!!!



Discrete Probability Distributions: Summary

1 To summarize, we've seen that **Discrete Probability Distributions** can be derived from histograms…

…and while these can be useful, they require a lot of data that can be expensive and time-consuming to get, and it's not always clear what to do about the blank spaces.



2 So, we usually use ***mathematical equations***, like the equation for the **Binomial Distribution**, instead.

$$p(x \mid n, p) = \left(\frac{n!}{x!(n-x)!} \right) p^x (1-p)^{n-x}$$

The **Binomial Distribution** is useful for anything that has binary outcomes (wins and losses, yeses and noes, etc.), but there are lots of other **Discrete Probability Distributions**.

3 For example, when we have **events** that happen in discrete units of time or space, like reading **10** pages an hour, we can use the **Poisson Distribution**.

$$p(x \mid \lambda) = \frac{e^{-\lambda} \lambda^x}{x!}$$

4 There are lots of other **Discrete Probability Distributions** for lots of other types of data. In general, their equations look intimidating, but looks are deceiving. Once you know what each symbol means, you just plug in the numbers and do the math.

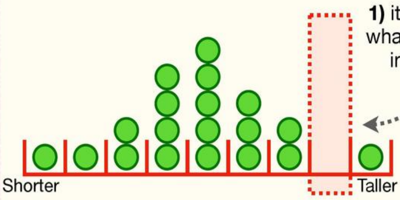
BAM!!!

Now let's talk about **Continuous Probability Distributions**.

Continuous Probability Distributions: Main Ideas

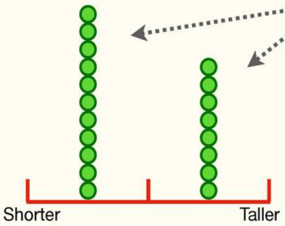
① **The Problem:** Although they can be super useful, beyond needing a lot of data, histograms have two problems when it comes to continuous data:

1) it's not always clear what to do about gaps in the data and…



…**2)** histograms can be very sensitive to the size of the bins.

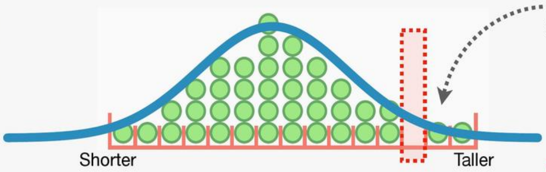
If the bins are too wide, then we lose all of the precision…



…and if the bins are too narrow, it's impossible to see trends.

② **A Solution:** When we have continuous data, a **Continuous Distribution** allows us to avoid all of these problems by using mathematical formulas just like we did with **Discrete Distributions**.

In this example, we can use a **Normal Distribution**, which creates a **bell-shaped curve**, instead of a histogram. It doesn't have a gap, and there's no need to fiddle with bin size.

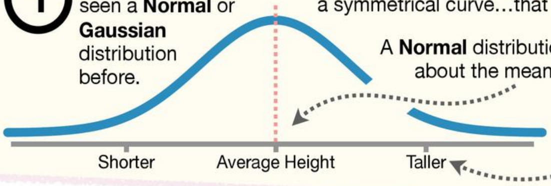


There are lots of commonly used **Continuous Distributions**. Now we'll talk about the most useful of all, the **Normal Distribution**.

The Normal (Gaussian) Distribution: Main Ideas Part 1

① Chances are you've seen a **Normal** or **Gaussian** distribution before.

It's also called a **Bell-Shaped Curve** because it's a symmetrical curve…that looks like a bell.



A **Normal** distribution is symmetrical about the mean, or average, value.

In this example, the curve represents human Height measurements.

② The y-axis represents the **Likelihood** of observing any specific Height.

NOTE: The **Normal Distribution**'s maximum likelihood value occurs at its mean.

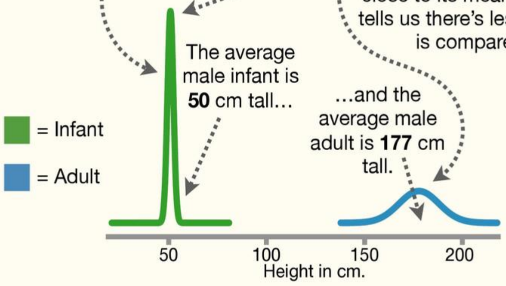
For example, it's relatively rare to see someone who is super short…

…relatively common to see someone who is close to the average height…

…and relatively rare to see someone who is super tall.

③ Here are two **Normal Distributions** of the heights of male infants and adults.

Because the normal distribution for infants has a higher peak than the one for adults, we can see that there's a higher likelihood that an infant will be close to its mean than an adult will be close to its mean. The difference in peak height tells us there's less variation in how tall an infant is compared to how tall an adult is.

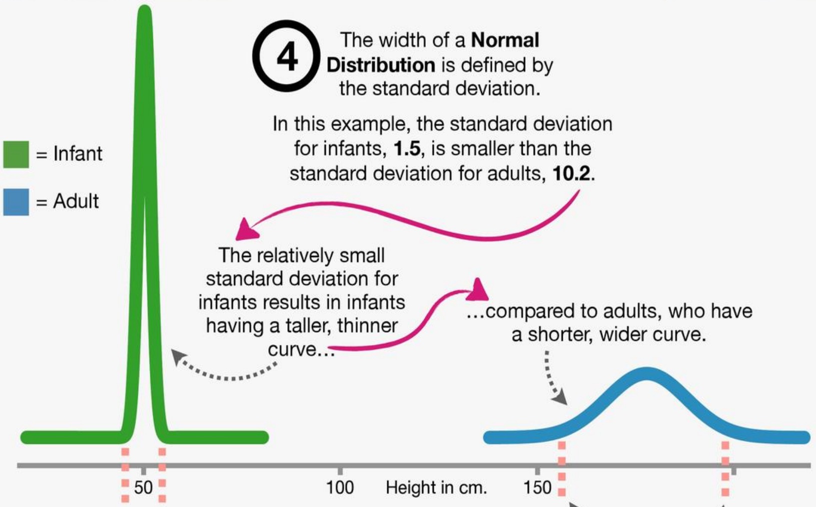


The average male infant is **50** cm tall…

…and the average male adult is **177** cm tall.

Lots of things can be approximated with **Normal Distributions**: Height, birth weight, blood pressure, job satisfaction, and many more!!!

The Normal (Gaussian) Distribution: Main Ideas Part 2



④ The width of a **Normal Distribution** is defined by the standard deviation.

In this example, the standard deviation for infants, **1.5**, is smaller than the standard deviation for adults, **10.2**.

The relatively small standard deviation for infants results in infants having a taller, thinner curve…

…compared to adults, who have a shorter, wider curve.

⑤ Knowing the standard deviation is helpful because normal curves are drawn such that about **95%** of the measurements fall between **+/- 2 Standard Deviations** around the **Mean**.

Because the mean measurement for infants is **50** cm, and **2 x the standard deviation = 2 x 1.5 = 3**, about **95%** of the infant measurements fall between **47** and **53** cm.

Because the mean adult measurement is **177** cm, and **2 x the standard deviation = 2 x 10.2 = 20.4**, about **95%** of the adult measurements fall between **156.6** and **197.4** cm.

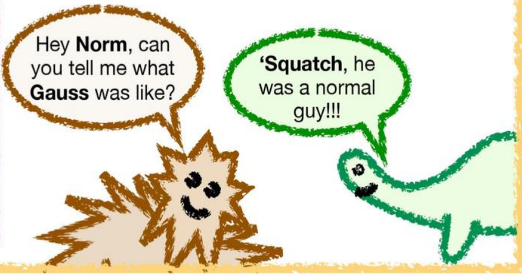
To draw a **Normal Distribution**, you need to know:

1) The **Mean** or average measurement. This tells you where the center of the curve goes.

2) The **Standard Deviation** of the measurements. This tells you how tall and skinny, or short and fat, the curve should be.

If you don't already know about the **Mean** and **Standard Deviation**, check out **Appendix B**.

BAM!!!



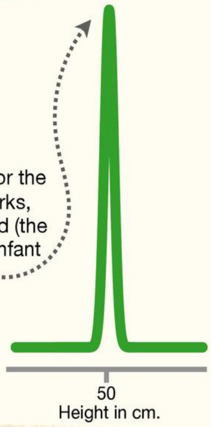
The Normal (Gaussian) Distribution: Details

① The equation for the **Normal Distribution** looks scary, but, just like every other equation, it's just a matter of plugging in numbers and doing the math.

$$f(x\,|\,\mu,\,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-(x-\mu)^2/2\sigma^2}$$

② To see how the equation for the **Normal Distribution** works, let's calculate the likelihood (the y-axis coordinate) for an infant that is **50** cm tall.

Since the mean of the distribution is also **50** cm, we'll calculate the y-axis coordinate for the highest part of the curve.



$$f(x\,|\,\mu,\,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-(x-\mu)^2/2\sigma^2}$$

③ ***x*** is the x-axis coordinate. So, in this example, the x-axis represents **Height** and ***x* = 50**.

The Greek character ***μ***, ***mu***, represents the mean of the distribution. In this case, ***μ* = 50**.

Lastly, the Greek character ***σ***, ***sigma***, represents the standard deviation of the distribution. In this case, ***σ* = 1.5**.

$$f(x = 50\,|\,\mu = 50,\,\sigma = 1.5) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-(x-\mu)^2/2\sigma^2}$$

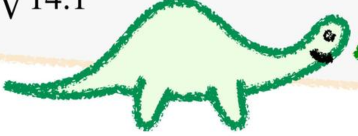
$$= \frac{1}{\sqrt{2\pi 1.5^2}}e^{-(50-50)^2/(2\times 1.5^2)}$$

Now, we just do the math…

$$= \frac{1}{\sqrt{14.1}}e^{-0^2/4.5}$$

…and we see that the likelihood, the y-axis coordinate, for the tallest point on the curve, is **0.27**.

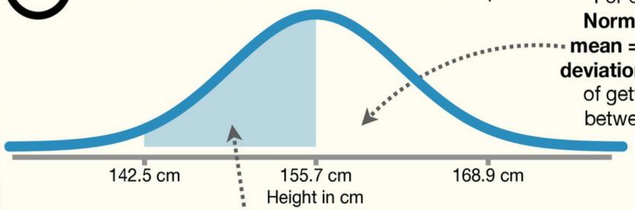
$$= \frac{1}{\sqrt{14.1}}e^{0} \quad = \frac{1}{\sqrt{14.1}} \quad \boxed{= 0.27}$$



Remember, the output from the equation, the y-axis coordinate, is a **likelihood**, ***not*** a **probability**. In **Chapter 7**, we'll see how likelihoods are used in **Naive Bayes**. To learn how to calculate probabilities with **Continuous Distributions**, read on…

Calculating Probabilities with Continuous Probability Distributions: Details

① For **Continuous Probability Distributions**, probabilities are the **area under the curve** between two points.

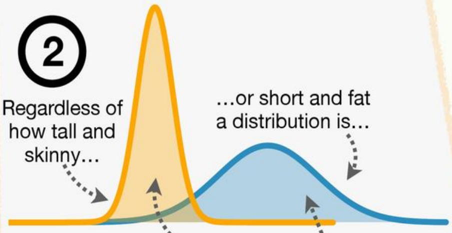


For example, given this **Normal Distribution** with **mean = 155.7** and **standard deviation = 6.6**, the probability of getting a measurement between **142.5** and **155.7** cm…

…is equal to this area under the curve, which in this example is **0.48**. So, the probability is **0.48** that we will measure someone in this range.

② Regardless of how tall and skinny…

…or short and fat a distribution is…



…the total area under its curve is **1**. Meaning, the probability of measuring anything in the range of possible values is **1**.

③ There are two ways to calculate the area under the curve between two points:

1) The hard way, by using calculus and integrating the equation between the two points ***a*** and ***b***.

$$\int_a^b f(x)\,dx$$

UGH!!! NO ONE ACTUALLY DOES THIS!!!

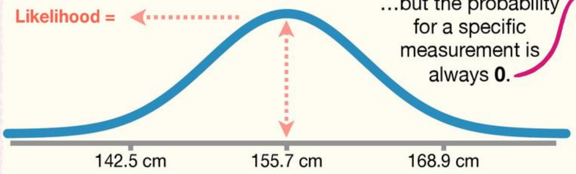
2) The easy way, by using a computer. See **Appendix C** for a list of commands.

Area = 0.48

BAM!!!

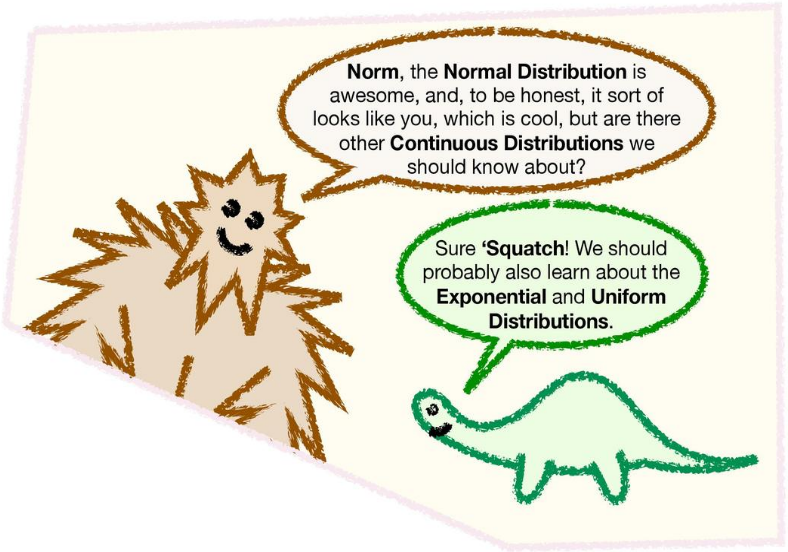
④ One confusing thing about **Continuous Distributions** is that the likelihood for a specific measurement, like **155.7**, is the y-axis coordinate and **> 0**…

…but the probability for a specific measurement is always **0**.



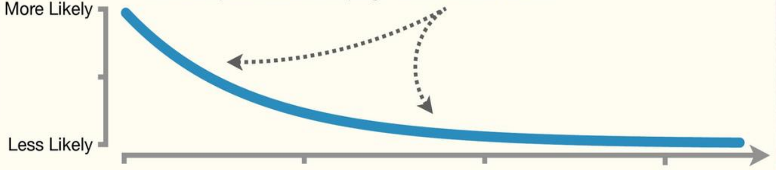
One way to understand why the probability is **0** is to remember that probabilities are areas, and the area of something with no width is **0**.

Another way to understand why the probability for a specific value, rather than a range of values, is **0** is to realize that a continuous distribution has infinite precision, thus, we're really asking the probability of measuring someone who is *exactly* **155.70000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000**… tall.


Norm, the **Normal Distribution** is awesome, and, to be honest, it sort of looks like you, which is cool, but are there other **Continuous Distributions** we should know about?
Sure **'Squatch**! We should probably also learn about the **Exponential** and **Uniform Distributions**.

Other Continuous Distributions: Main Ideas

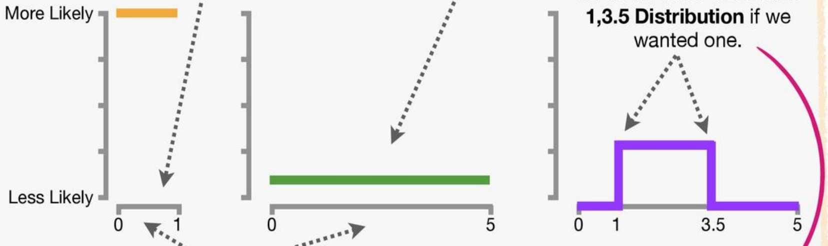
① **Exponential Distributions** are commonly used when we're interested in how much time passes between events. For example, we could measure how many minutes pass between page turns in this book.



② **Uniform Distributions** are commonly used to generate random numbers that are equally likely to occur. For example, if I want to select random numbers between **0** and **1**, then I would use a **Uniform Distribution** that goes from **0** to **1**, which is called a **Uniform 0,1 Distribution**, because it ensures that every value between **0** and **1** is equally likely to occur.

In contrast, if I wanted to generate random numbers between **0** and **5**, then I would use a **Uniform Distribution** that goes from **0** to **5**, which is called a **Uniform 0,5 Distribution**.

Uniform Distributions can span any **2** numbers, so we could have a **Uniform 1,3.5 Distribution** if we wanted one.



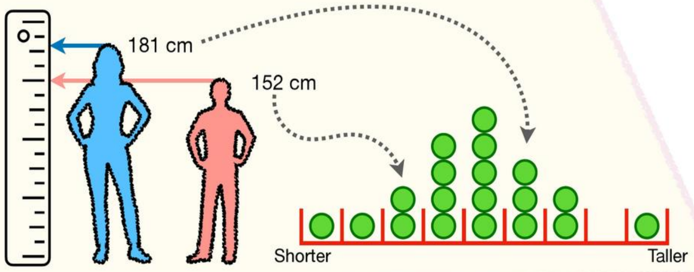
NOTE: Because the distance between **0** and **1** is shorter than between **0** and **5**, we see that the corresponding likelihood for any specific number is higher for the **Uniform 0,1 Distribution** than the **Uniform 0,5 Distribution**.

Using Distributions To Generate Random Numbers

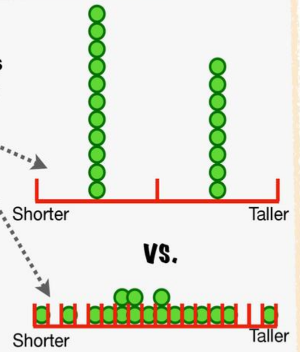
We can get a computer to generate numbers that reflect the likelihoods of any distribution. In machine learning, we usually need to generate random numbers to initialize algorithms before training them with **Training Data**. Random numbers are also useful for randomizing the order of our data, which is useful for the same reasons we shuffle a deck of cards before playing a game. We want to make sure everything is randomized.

Continuous Probability Distributions: Summary

① Just like **Discrete Distributions**, **Continuous Distributions** spare us from having to gather tons of data for a histogram…

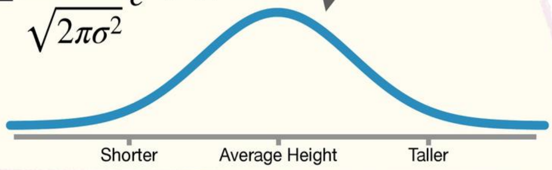


② …and, additionally, **Continuous Distributions** also spare us from having to decide how to bin the data.



③ Instead, **Continuous Distributions** use equations that represent smooth curves and can provide likelihoods and probabilities for all possible measurements.

$$f(x \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$$



④ Like **Discrete Distributions**, there are **Continuous Distributions** for all kinds of data, like the values we get from measuring people's height or timing how long it takes you to read this page.

In the context of machine learning, both types of distributions allow us to create **Models** that can predict what will happen next.

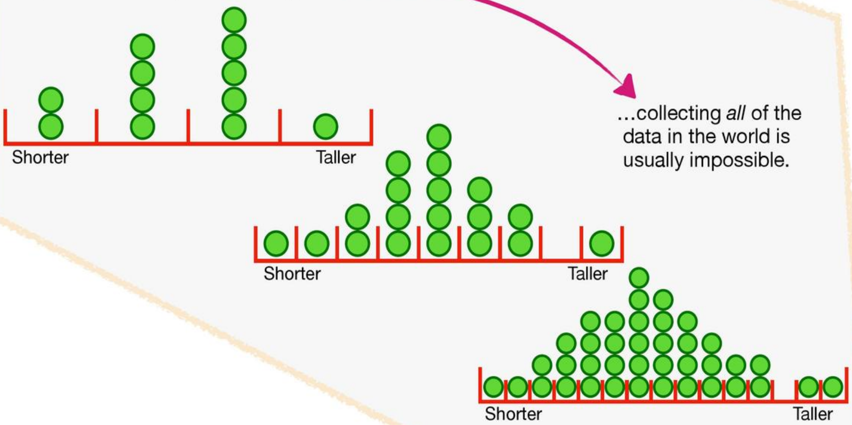
So, let's talk about what **Models** are and how to use them.

(small but mighty) **BAM!!!**

Models: Main Ideas Part 1

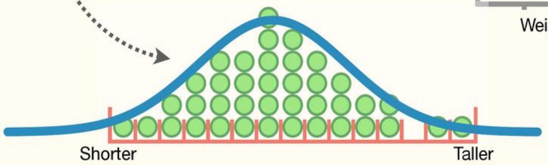
① **The Problem:** Although we could spend a lot of time and money to build a precise histogram…

…collecting *all* of the data in the world is usually impossible.

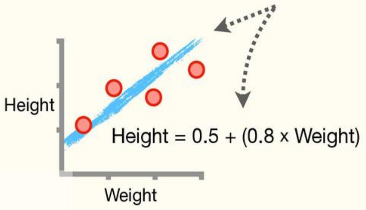


② **A Solution:** A statistical, mathematical, or machine learning **Model** provides an *approximation* of reality that we can use in a wide variety of ways.

A **Probability Distribution** is a type of model that approximates a histogram with an infinite amount of data.

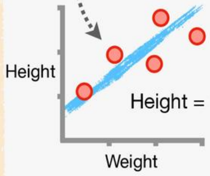


Another commonly used model is the equation for a straight line. Here, we're using a **blue line** to model a relationship between Weight and Height.



Models: Main Ideas Part 2

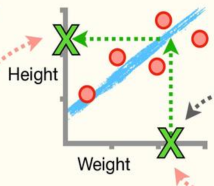
③ As we saw in **Chapter 1**, models need **Training Data**. Using machine learning lingo, we say that we *build models by training machine learning algorithms*.



$\text{Height} = 0.5 + (0.8 \times \text{Weight})$

④ Models, or equations, can tell us about people we haven't measured yet.

For example, if we want to know how tall someone is who weighs this much…



…we plug the Weight into the equation and solve for Height…

$\text{Height} = 0.5 + (0.8 \times \text{Weight})$

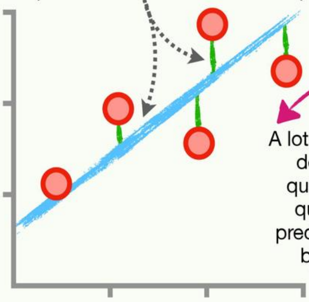
$\text{Height} = 0.5 + (0.8 \times 2.1)$

$\text{Height} = 2.18$

…and get **2.18**.

⑤ Because models are only approximations, it's important that we're able to measure the quality of their predictions.

These **green lines** show the distances from the model's predictions to the actual data points.



A lot of statistics is dedicated to quantifying the quality of the predictions made by a model.

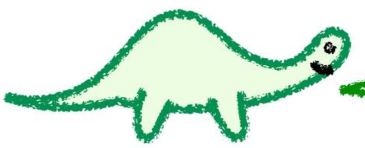
⑥ In summary:

1) Models approximate reality to let us explore relationships and make predictions.

2) In machine learning, we build models by training machine learning algorithms with **Training Data**.

3) Statistics can be used to determine if a model is useful or believable.

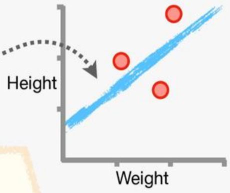
Bam!



Now let's talk about how statistics can quantify the quality of a model. The first step is to learn about the **Sum of the Squared Residuals**, which is something we'll use throughout this book.

The Sum of the Squared Residuals: Main Ideas Part 1

① **The Problem:** We have a model that makes predictions. In this case, we're using Weight to predict Height. However, we need to quantify the quality of the model and its predictions.

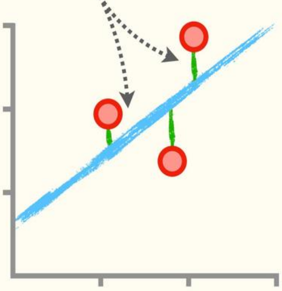


② **A Solution:** One way quantify the quality of a model and its predictions is to calculate the **Sum of the Squared Residuals**.

As the name implies, we start by calculating **Residuals**, the differences between the **Observed** values and the values **Predicted** by the model.

Visually, we can draw **Residuals** with these green lines.

Residual = Observed - Predicted



Since, in general, the smaller the **Residuals**, the better the model fits the data, it's tempting to compare models by comparing the sum of their **Residuals**, but the **Residuals** below the blue line would cancel out the ones above it!!!

So, instead of calculating the sum of the **Residuals**, we square the **Residuals** first and calculate the **Sum of the Squared Residuals** (**SSR**).

The Sum of Squared Residuals (SSR) is usually defined with fancy **Sigma** notation and the right-hand side reads: "The sum of all squared differences between the observed and predicted values."

$$\text{SSR} = \sum_{i=1}^{n} (\text{Observed}_i - \text{Predicted}_i)^2$$

n = the number of **Observations**.

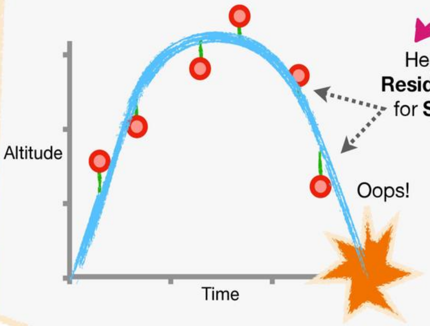
i = the index for each **Observation**. For example, ***i* = 1** refers to the first **Observation**.

The **Sigma** symbol, **Σ**, tells us to do a **summation**.

NOTE: ***Squaring***, as opposed to taking the ***absolute value***, makes it easy to take the derivative, which will come in handy when we do **Gradient Descent** in **Chapter 5**.

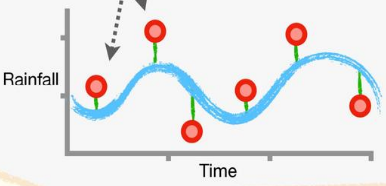
The Sum of the Squared Residuals: Main Ideas Part 2

③ So far, we've looked at the **SSR** only in terms of a simple straight line model, but we can calculate it for all kinds of models.



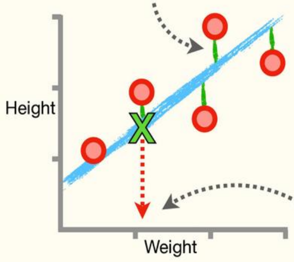
Here's an example of the **Residuals** for altitude vs. time for **SpaceX's SN9** rocket…

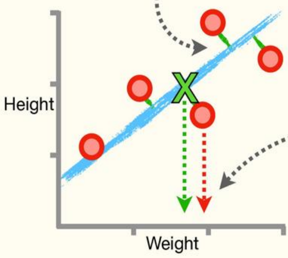
…and here's an example of the **Residuals** for a sinusoidal model of rainfall. Some months are more rainy than others, and the pattern is cyclical over time. If you can calculate the **Residuals**, you can square them and add them up!



④ **NOTE:** When we calculate the **Residuals**, we use the **vertical distance** to the **model**…

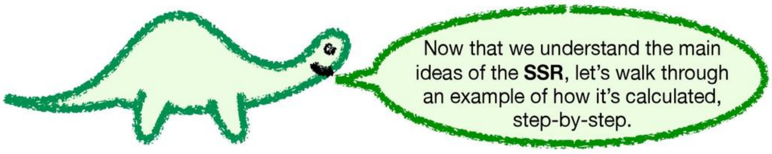
…instead of the shortest distance, the **perpendicular distance**…





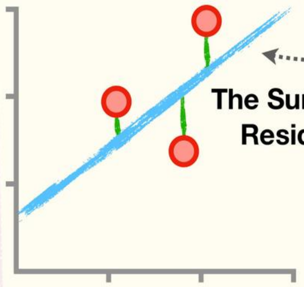
…because, in this example, perpendicular lines result in different Weights for the Observed and Predicted Heights.

In contrast, the vertical distance allows both the Observed and Predicted Heights to correspond to the same Weight.



Now that we understand the main ideas of the **SSR**, let's walk through an example of how it's calculated, step-by-step.

SSR: Step-by-Step



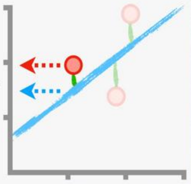
① In this example, we have **3** Observations, so ***n* = 3**, and we expand the summation into **3** terms.

$$\textbf{The Sum of Squared Residuals (SSR)} = \sum_{i=1}^{n}(\text{Observed}_i - \text{Predicted}_i)^2$$

Observed = ⇢

Predicted = ⇢

Residual = |



The residual for ***i* = 1**, the term for the first Observation is…

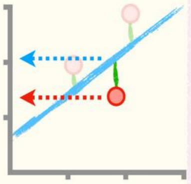
$(1.9 - 1.7)^2$

② Once we expand the summation, we plug in the **Residuals** for each Observation.

$$\textbf{SSR} = (\text{Observed}_1 - \text{Predicted}_1)^2$$

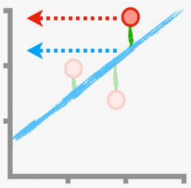
$$+ (\text{Observed}_2 - \text{Predicted}_2)^2$$

$$+ (\text{Observed}_3 - \text{Predicted}_3)^2$$



For ***i* = 2**, the term for the second Observation, the residual is…

$(1.6 - 2.0)^2$



For ***i* = 3**, the term for the third Observation…

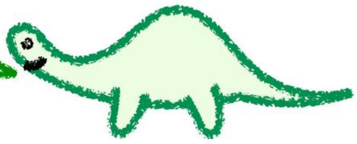
$(2.9 - 2.2)^2$

③ Now, we just do the math, and the final **Sum of Squared Residuals (SSR)** is **0.69**.

$$\textbf{SSR} = (1.9 - 1.7)^2 + (1.6 - 2.0)^2 + (2.9 - 2.2)^2$$

$$\boxed{= 0.69}$$

BAM!!!

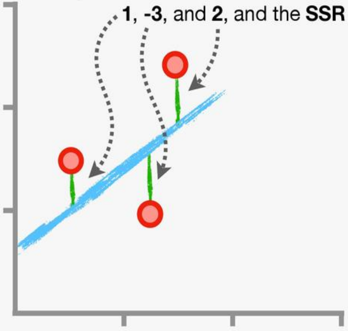


Don't get me wrong, the **SSR** is awesome, but it has a pretty big problem that we'll talk about on the next page.

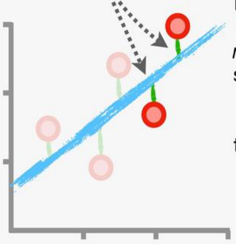
Mean Squared Error (MSE): Main Ideas

① **The Problem: Sum of the Squared Residuals** (**SSR**), although awesome, is not super easy to interpret because it depends, in part, on how much data you have.

For example, if we start with a simple dataset with **3** points, the **Residuals** are, from left to right, **1**, **-3**, and **2**, and the **SSR** = **14**.



Now, if we have a second dataset that includes **2** more data points added to the first one, and the **Residuals** are **-2** and **2**, then the **SSR** increases to **22**.



However, the increase in the **SSR** from **14** to **22** *does not* suggest that the second model, fit to the second, larger dataset, is worse than the first. It only tells us that the model with more data has more **Residuals**.

② **A Solution:** One way to compare the two models that may be fit to different-sized datasets is to calculate the **Mean Squared Error** (**MSE**), which is simply the average of the **SSR**.

$$\textbf{Mean Squared Error (MSE)} = \frac{\textbf{The Sum of Squared Residuals (SSR)}}{\textbf{Number of Observations, } \boldsymbol{n}}$$

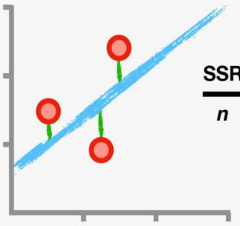
$$= \frac{\sum_{i=1}^{n} (\text{Observed}_i - \text{Predicted}_i)^2}{n}$$

Mean Squared Error (MSE): Step-by-Step

① Now let's see the **MSE** in action by calculating it for the two datasets!!!

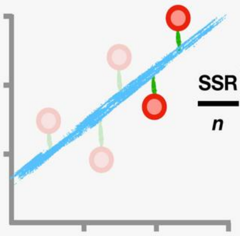
$$\textbf{Mean Squared Error (MSE)} = \frac{\textbf{SSR}}{n} = \sum_{i=1}^{n} \frac{(\text{Observed}_i - \text{Predicted}_i)^2}{n}$$

② The first dataset has only **3** points and the **SSR** = **14**, so the **Mean Squared Error** (**MSE**) is **14/3** = **4.7**.



$$\frac{\textbf{SSR}}{n} = \frac{14}{3} = 4.7$$

The second dataset has **5** points and the **SSR** increases to **22**. In contrast, the **MSE**, **22/5** = **4.4**, is now slightly lower.

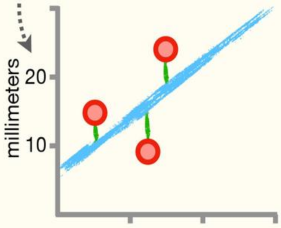


$$\frac{\textbf{SSR}}{n} = \frac{22}{5} = 4.4$$

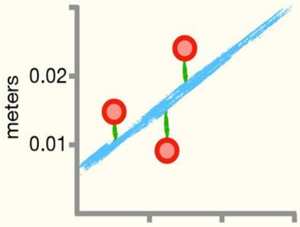
So, unlike the **SSR**, which increases when we add more data to the model, the **MSE** can increase or decrease depending on the average residual, which gives us a better sense of how the model is performing overall.

③ Unfortunately, **MSEs** are still difficult to interpret on their own because the maximum values depend on the scale of the data.

For example, if the y-axis is in *millimeters* and the **Residuals** are **1**, **-3**, and **2**, then the **MSE** = **4.7**.



However, if we change the y-axis to *meters*, then the **Residuals** for the exact same data shrink to **0.001**, **-0.003**, and **0.002**, and the **MSE** is now **0.0000047**. It's tiny!

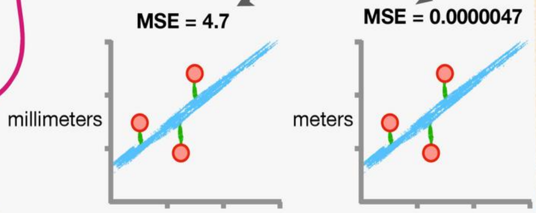


The good news, however, is that both the **SSR** and the **MSE** can be used to calculate something called ***R*²**, which is independent of both the size of the dataset and the scale, so keep reading!

R^2: Main Ideas

① **The Problem:** As we just saw, the **MSE**, although totally cool, can be difficult to interpret because it depends, in part, on the scale of the data.

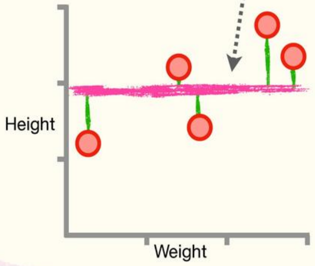
In this example, changing the units from millimeters to meters reduced the **MSE** by a lot!!!

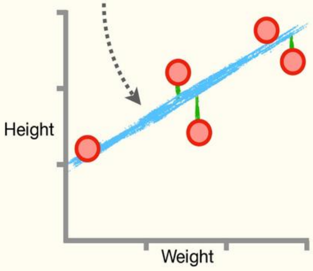


② **A Solution:** ***R*²**, pronounced **R squared**, is a simple, easy-to-interpret metric that does not depend on the size of the dataset or its scale.

Typically, ***R*²** is calculated by comparing the **SSR** or **MSE** around the **mean** y-axis value. In this example, we calculate the **SSR** or **MSE** around the **average** **Height**…

…and compare it to the **SSE** or **MSE** around the model we're interested in. In this case, that means we calculate the **SSR** or **MSE** around the **blue line** that uses **Weight** to predict **Height**.



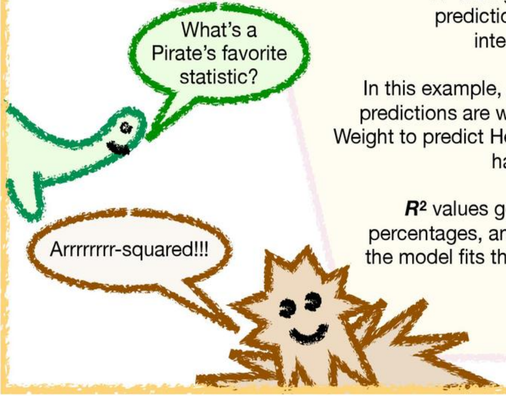


***R*²** then gives us a percentage of how much the predictions improved by using the model we're interested in instead of just the **mean**.

In this example, ***R*²** would tell us how much better our predictions are when we use the **blue line**, which uses Weight to predict Height, instead of predicting that everyone has the **average** Height.

***R*²** values go from **0** to **1** and are interpreted as percentages, and the closer the value is to **1**, the better the model fits the data relative to the mean y-axis value.

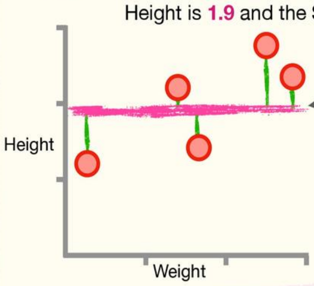
Now that we understand the main ideas, let's dive into the details!!!



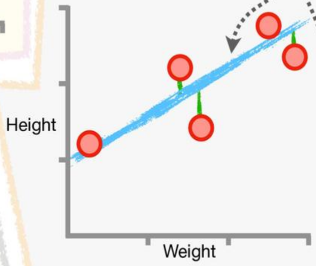
R^2: Details Part 1

① First, we calculate the **Sum of the Squared Residuals** for the mean. We'll call this **SSR** the **SSR(mean)**. In this example, the mean Height is 1.9 and the **SSR(mean) = 1.6**.

$$\text{SSR(mean)} = (2.3 - 1.9)^2 + (1.2 - 1.9)^2 + (2.7 - 1.9)^2 + (1.4 - 1.9)^2 + (2.2 - 1.9)^2 = \boxed{1.6}$$



② Then, we calculate the **SSR** for the fitted line, **SSR(fitted line)**, and get **0.5**.



NOTE: The smaller **Residuals** around the fitted line, and thus the smaller **SSR** given the same dataset, suggest the fitted line does a better job making predictions than the mean.

$$\text{SSR(fitted line)} = (1.2 - 1.1)^2 + (2.2 - 1.8)^2 + (1.4 - 1.9)^2 + (2.7 - 2.4)^2 + (2.3 - 2.5)^2 = \boxed{0.5}$$

③ Now we can calculate the R^2 value using a surprisingly simple formula…

$$\boxed{R^2 = \frac{\text{SSR(mean)} - \text{SSR(fitted line)}}{\text{SSR(mean)}}}$$

$$= \frac{1.6 - 0.5}{1.6}$$

$$\boxed{= 0.7}$$

…and the result, **0.7**, tells us that there was a **70%** reduction in the size of the **Residuals** between the mean and the fitted line.

④ In general, because the numerator for R^2…

SSR(mean) - SSR(fitted line)

…is the amount by which the **SSRs** shrank when we fitted the line, R^2 values tell us the percentage the **Residuals** around the mean shrank when we used the fitted line.

When **SSR(mean) = SSR(fitted line)**, then both models' predictions are equally good (or equally bad), and $R^2 = 0$

$$\frac{\text{SSR(mean)} - \text{SSR(fitted line)}}{\text{SSR(mean)}} = \frac{0}{\text{SSR(mean)}} = 0$$

When **SSR(fitted line) = 0**, meaning that the fitted line fits the data perfectly, then $R^2 = 1$.

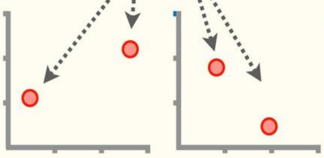
$$\frac{\text{SSR(mean)} - 0}{\text{SSR(mean)}} = \frac{\text{SSR(mean)}}{\text{SSR(mean)}} = 1$$

R^2: Details Part 2

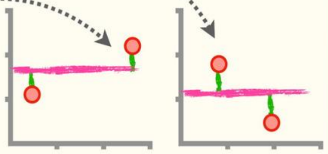
$$R^2 = \frac{\text{SSR(mean)} - \text{SSR(fitted line)}}{\text{SSR(mean)}}$$

5

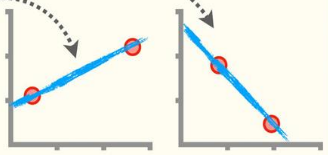
NOTE: Any **2** random data points have R^2 **= 1**…



…because regardless of the **Residuals** around the **means**…



…the **Residuals** around a **fitted line** will always be **0**…



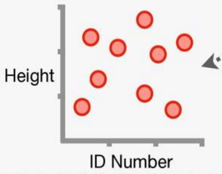
…and when we do the math, we are just dividing **SSR(mean)** by **SSR(mean)**, which is **1**.

$$\frac{\text{SSR(mean)} - 0}{\text{SSR(mean)}} = \frac{\text{SSR(mean)}}{\text{SSR(mean)}} = 1$$

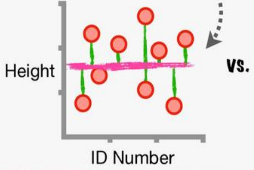
And because a small amount of random data can have a high (close to **1**) R^2, any time we see a trend in a small dataset, it's difficult to have confidence that a high R^2 value is not due to random chance.

6

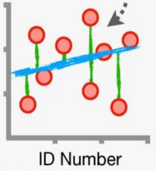
If we had a lot of data organized randomly using a random **ID Number**, we would expect the graph to look like this…



…and have a relatively small (close to **0**) R^2 because the **Residuals** around the **mean**…

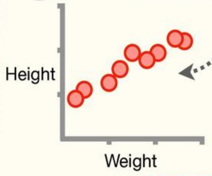


…would be similar to the residuals around the **fitted line**.

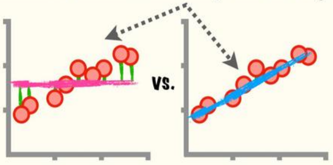


7

In contrast, when we see a trend in a large amount of data like this…



…we can, intuitively, have more confidence that a large R^2 is not due to random chance because the data are not scattered around randomly like we might expect.



Never satisfied with intuition, statisticians developed something called ***p*-values** to quantify how much confidence we should have in R^2 values and pretty much any other method that summarizes data. We'll talk about ***p*-values** in a bit, but first let's calculate R^2 using the **Mean Squared Error** (**MSE**).

Calculating R^2 with the Mean Squared Error (MSE): Details

So far, we've calculated R^2 using the **Sum of the Squared Residuals** (**SSR**), but we can just as easily calculate it using the **Mean Squared Error** (**MSE**).

$$\frac{\text{MSE(mean)} - \text{MSE(fitted line)}}{\text{MSE(mean)}}$$

First, we rewrite the **MSE** in terms of the **SSR** divided by the size of the dataset, ***n***…

$$= \frac{\dfrac{\text{SSR(mean)}}{n} - \dfrac{\text{SSR(fitted line)}}{n}}{\dfrac{\text{SSR(mean)}}{n}}$$

…then we consolidate all of the division by ***n*** into a single term…

$$= \frac{\text{SSR(mean)} - \text{SSR(fitted line)}}{\text{SSR(mean)}} \times \frac{n}{n}$$

…and since ***n*** divided by ***n*** is **1**…

$$= \frac{\text{SSR(mean)} - \text{SSR(fitted line)}}{\text{SSR(mean)}} \times 1$$

$$= R^2$$

…we end up with R^2 times **1**, which is just R^2. So, we can calculate R^2 with the **SSR** or **MSE**, whichever is readily available. Either way, we'll get the same value.

BAM!!!

Gentle Reminders:

Residual = Observed - Predicted

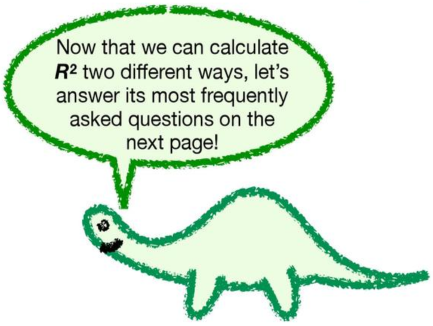
SSR = Sum of Squared Residuals

$$\text{SSR} = \sum_{i=1}^{n} (\text{Observed}_i - \text{Predicted}_i)^2$$

$$\textbf{Mean Squared Error (MSE)} = \frac{\textbf{SSR}}{n}$$

…where ***n*** is the sample size

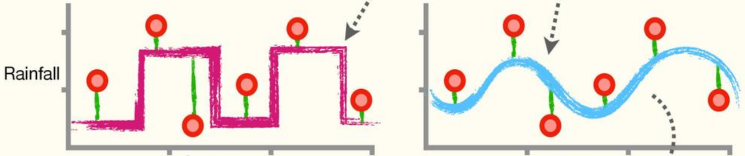
$$R^2 = \frac{\text{SSR(mean)} - \text{SSR(fitted line)}}{\text{SSR(mean)}}$$



R^2: FAQ

Does R^2 always compare the mean to a straight fitted line?

The most common way to calculate R^2 is to compare the **mean** to a **fitted line**. However, we can calculate it for anything we can calculate the **Sum of the Squared Residuals** for. For example, for rainfall data, we use R^2 to compare a **square wave** to a **sine wave**.



In this case, we calculate R^2 based on the **Sum of the Squared Residuals** around the **square** and **sine** waves.

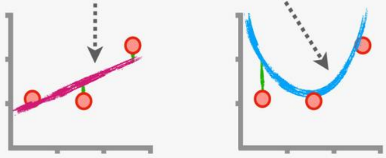
$$R^2 = \frac{\text{SSR(square)} - \text{SSR(sine)}}{\text{SSR(square)}}$$

Can R^2 be negative?

When we're only comparing the **mean** to a **fitted line**, R^2 is positive, but when we compare other types of models, anything can happen.

For example, if we use R^2 to compare a **straight line** to a **parabola**…

…we get a negative R^2 value, **-1.4**, and it tells us the **Residuals** *increased* by **140%**.



RSS(straight line) = 5 **RSS(parabola) = 12**

$$R^2 = \frac{\text{SSR(line)} - \text{SSR(parabola)}}{\text{SSR(line)}}$$

$$R^2 = \frac{5 - 12}{5} = \boxed{-1.4}$$

BAM!!!

Is R^2 related to Pearson's correlation coefficient?

Yes! If you can calculate **Pearson's correlation coefficient**, ρ (the Greek character **rho**) or r, for a relationship between two things, then the square of that coefficient is equal to R^2. In other words…

$$\rho^2 = r^2 = R^2$$

…and now we can see where R^2 got its name.

p-values: Main Ideas Part 1

① **The Problem:** We need to quantify how confident we should be in the results of our analysis.

② **A Solution: *p*-values** give us a measure of confidence in the results from a statistical analysis.

Imagine we had two antiviral drugs, **A** and **B**, and we wanted to know if they were *different*.

NOTE: Throughout the description of ***p*-values**, we'll only focus on determining whether or not Drug A is *different* from Drug B. If a ***p*-value** allows us to establish a difference, then we can worry about whether Drug A is better or worse than Drug B.

So we gave Drug A to **1** person and they were cured…

…and we gave Drug B to another person and they were not cured.

Can we conclude that Drug A is different from Drug B?

No!!! Drug B may have failed for a lot of reasons. Maybe this person is taking a medication that has a bad interaction with Drug B, or maybe they have a rare allergy to Drug B, or maybe they didn't take Drug B properly and missed a dose.

Or maybe Drug A doesn't actually work, and the placebo effect deserves all of the credit.

There are a lot of weird, random things that can happen when doing a test, and this means that we need to test each drug on more than just one person.

③ So, we redid the experiment with lots and lots of people, and these were the results: Drug A cured a lot of people compared to Drug B, which hardly cured anyone.

Cured!!!	Not Cured
1,043	3

Cured!!!	Not Cured
2	1,432

Now, it's pretty obvious that Drug A is different from Drug B because it would be unrealistic to suppose that these results were due to just random chance and that there's no real difference between Drug A and Drug B.

It's possible that some of the people taking Drug A were actually cured by placebo, and some of the people taking Drug B were not cured because they had a rare allergy, but there are just too many people cured by Drug A, and too few cured by Drug B, for us to seriously think that these results are just random and that Drug A is no different from Drug B.

p-values: Main Ideas Part 2

④ In contrast, let's say that these were the results…

Drug A

Cured!!!	Not Cured
73	125

Drug B

Cured!!!	Not Cured
59	131

…and **37%** of the people who took Drug A were cured compared to **31%** who took Drug B.

Drug A cured a larger percentage of people, but given that no study is perfect and there are always a few random things that happen, how confident can we be that Drug A is different from Drug B?

This is where ***p*-values** come in. ***p*-values** are numbers between **0** and **1** that, in this example, quantify how confident we should be that Drug A is different from Drug B. The closer a **p-value** is to **0**, the more confidence we have that Drug A and Drug B are different.

So, the question is, "how small does a ***p*-value** have to be before we're sufficiently confident that Drug A is different from Drug B?"

In other words, what threshold can we use to make a good decision about whether these drugs are different?

⑤ In practice, a commonly used threshold is **0.05**. It means that if there's no difference between Drug A and Drug B, and if we did this exact same experiment a bunch of times, then only **5%** of those experiments would result in the wrong decision.

Yes! This wording is awkward. So, let's go through an example and work this out, one step at a time.

⑥ Imagine we gave the *same* drug, Drug A, to two different groups.

Drug A vs. **Drug A**

Now, all of the differences in the results can definitely be attributed to weird, random things, like a rare allergy in one person or a strong placebo effect in another.

Drug A

Cured!!!	Not Cured
73	125

Drug A

Cured!!!	Not Cured
71	127

$p = 0.9$

When we calculate the ***p*-value** for these data using a **Statistical Test** (for example, **Fisher's Exact Test**, but we'll save those details for another day) we get **0.9**, which is larger than **0.05**. Thus, we would say that we fail to see a difference between these two groups. And that makes sense because both groups are taking Drug A and the only differences are weird, random things like rare allergies.

p-values: Main Ideas Part 3

7 If we repeated this same experiment over and over again, most of the time we would get similarly large ***p*-values**…

Drug A

Cured!!!	Not Cured
73	125

Drug A

Cured!!!	Not Cured
71	127

$p = 0.9$

Cured!!!	Not Cured
71	127

Cured!!!	Not Cured
72	126

$p = 1$

Cured!!!	Not Cured
75	123

Cured!!!	Not Cured
70	128

$p = 0.7$

etc. etc. etc.

etc. etc. etc.

etc. etc. etc.

Cured!!!	Not Cured
69	129

Cured!!!	Not Cured
71	127

$p = 0.9$

8 However, every once in a while, by random chance, all of the people with rare allergies might end up in the group on the left…

…and by random chance, all of the people with strong (positive) placebo reactions might end up in the group on the right…

Drug A

Cured!!!	Not Cured
60	138

30% Cured

Drug A

Cured!!!	Not Cured
84	114

42% Cured

…and, as a result, the ***p*-value** for this specific run of the experiment is **0.01** (calculated using **Fisher's Exact Test**, but we'll save those details for another day), since the results are pretty different.

Thus, because the ***p*-value** is **< 0.05** (the threshold we're using for making a decision), we would say that the two groups are different, even though they both took the same drug!

TERMINOLOGY ALERT!!!

Getting a small ***p*-value** when there is no difference is called a **False Positive.**

A **0.05** threshold for ***p*-values** means that **5%** of the experiments, where the only differences come from weird, random things, will generate a ***p*-value** smaller than **0.05**.

In other words, if there's no difference between Drug A and Drug B, in **5%** of the times we do the experiment, we'll get a ***p*-value** less than **0.05**, and that would be a **False Positive**.

p-values: Main Ideas Part 4

9 If it's extremely important that we're correct when we say the drugs are different, then we can use a smaller threshold, like **0.01** or **0.001** or even smaller.

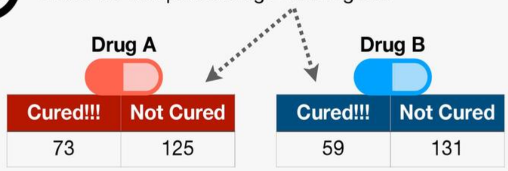
Using a threshold of **0.001** would get a **False Positive** only once in every **1,000** experiments.

Likewise, if it's not that important (for example, if we're trying to decide if the ice-cream truck will arrive on time), then we can use a larger threshold, like **0.2**.

Using a threshold of **0.2** means we're willing to get a **False Positive 2** times out of **10**.

That said, the most common threshold is **0.05** because trying to reduce the number of **False Positives** below **5%** often costs more than it's worth.

10 Now, going back to the original experiment, where we compared Drug A to Drug B…



Cured!!!	Not Cured
73	125

Cured!!!	Not Cured
59	131

…if we calculate a ***p*-value** for this experiment and the ***p*-value < 0.05**, then we'll decide that **Drug A** is different from **Drug B**.

That said, the ***p*-value = 0.24**, (again calculated using **Fisher's Exact Test**), so we're not confident that **Drug A** is different from **Drug B**.

TERMINOLOGY ALERT!!!

In fancy statistical lingo, the idea of trying to determine if these drugs are the same or not is called **Hypothesis Testing**.

The **Null Hypothesis** is that the drugs are the same, and the ***p*-value** helps us decide if we should *reject* the **Null Hypothesis**.

p-values: Main Ideas Part 5

⑪ While a small ***p*-value** helps us decide if Drug A is different from Drug B, it does not tell us *how different* they are.

In other words, you can have a small ***p*-value** regardless of the size of the difference between Drug A and Drug B.

The difference can be tiny or huge.
For example, this experiment gives us a relatively large ***p*-value**, **0.24**, even though there's a **6**-point difference between Drug A and Drug B.

Drug A

Cured!!!	Not Cured
73	125

37% Cured

Drug B

Cured!!!	Not Cured
59	131

31% Cured

In contrast, this experiment involving a lot more people gives us a smaller ***p*-value**, **0.04**, even though there's only a **1**-point difference between Drug A and Drug B.

Drug A

Cured!!!	Not Cured
5,005	9,868

34% Cured

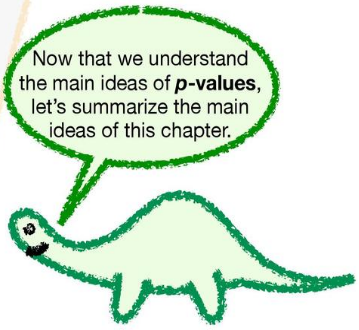
Drug B

Cured!!!	Not Cured
4,800	9,000

35% Cured

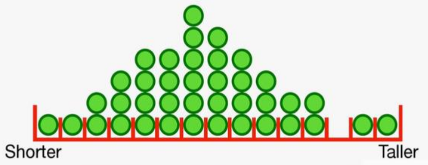
⑫ In summary, a small ***p*-value** *does not imply* that the effect size, or difference between Drug A and Drug B, is large.

DOUBLE BAM!!!

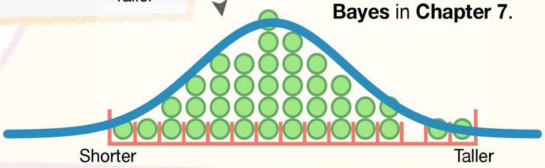


The Fundamental Concepts of Statistics: Summary

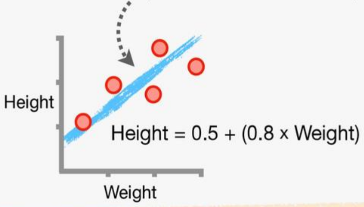
① We can see trends in data with **histograms**. We'll learn how to use **histograms** to make classifications with **Naive Bayes** in **Chapter 7**.



② However, **histograms** have limitations (they need a lot of data and can have gaps), so we also use **probability distributions** to represent trends. We'll learn how to use **probability distributions** to make classifications with **Naive Bayes** in **Chapter 7**.



③ Rather than collect all of the data in the whole wide world, which would take forever and be way too expensive, we use **models** to approximate reality. **Histograms** and **probability distributions** are examples of **models** that we can use to make predictions. We can also use a **mathematical formula**, like the equation for the blue line, as a **model** that makes predictions.



Throughout this book, we'll create machine learning **models** to make predictions.

④ We can evaluate how well a model reflects the data using the **Sum of the Squared Residuals** (**SSR**), the **Mean Squared Error** (**MSE**), and R^2. We'll use these metrics throughout the book.

Residual = Observed - Predicted

SSR = Sum of Squared Residuals

$$\textbf{SSR} = \sum_{i=1}^{n} (\text{Observed}_i - \text{Predicted}_i)^2$$

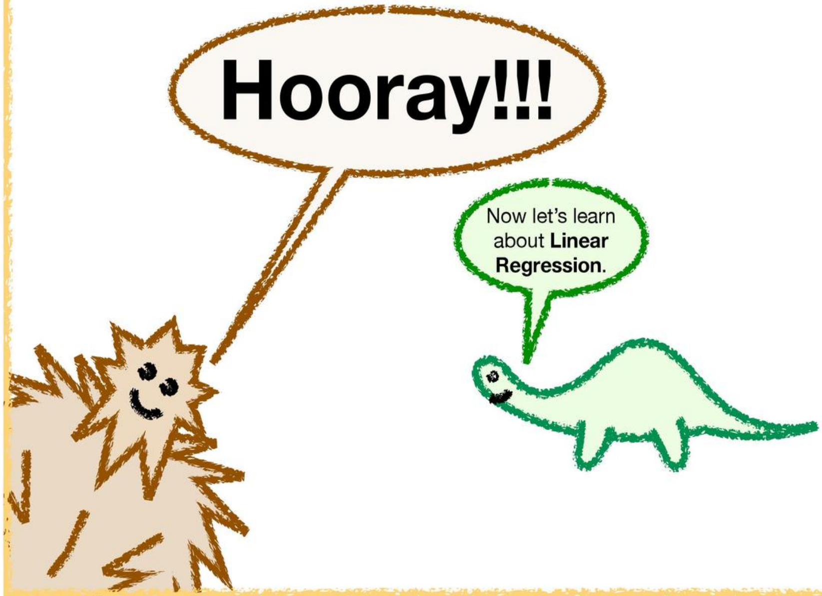
$$\textbf{Mean Squared Error (MSE)} = \frac{\textbf{SSR}}{n}$$

...where n is the sample size

$$R^2 = \frac{\text{SSR(mean)} - \text{SSR(fitted line)}}{\text{SSR(mean)}}$$

⑤ Lastly, we use ***p*-values** to give us a sense of how much confidence we should put in the predictions that our **models** make. We'll use ***p*-values** in **Chapter 4** when we do **Linear Regression**.

TRIPLE BAM!!!



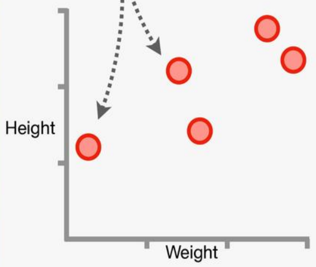
Chapter 04

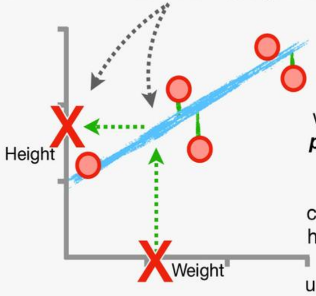
Linear Regression!!!

Linear Regression: Main Ideas

① **The Problem:** We've collected Weight and Height measurements from **5** people, and we want to use Weight to predict Height, which is ***continuous***…

…and in **Chapter 3**, we learned that we could fit a **line** to the data and use it to make predictions.

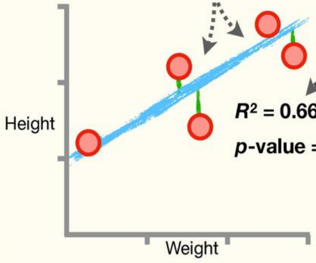




However, **1)** we didn't talk about how we fit a **line** to the data and **2)** we didn't calculate a ***p*-value** for the **fitted line**, which would quantify how much confidence we should have in its predictions compared to just using the **mean** y-axis value.

② **A Solution: Linear Regression** fits a **line** to the data that *minimizes* the **Sum of the Squared Residuals** (SSR)…

…and once we fit the line to the data, we can easily calculate R^2, which gives us a sense of how accurate our predictions will be…

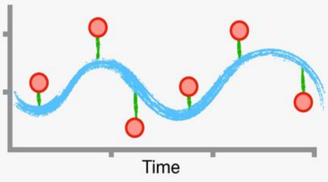


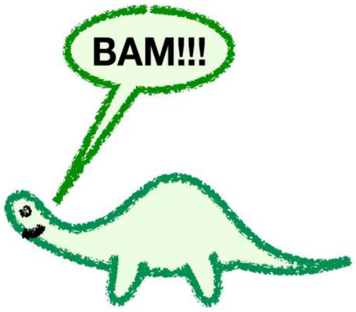
R^2 = 0.66

p-value = 0.1

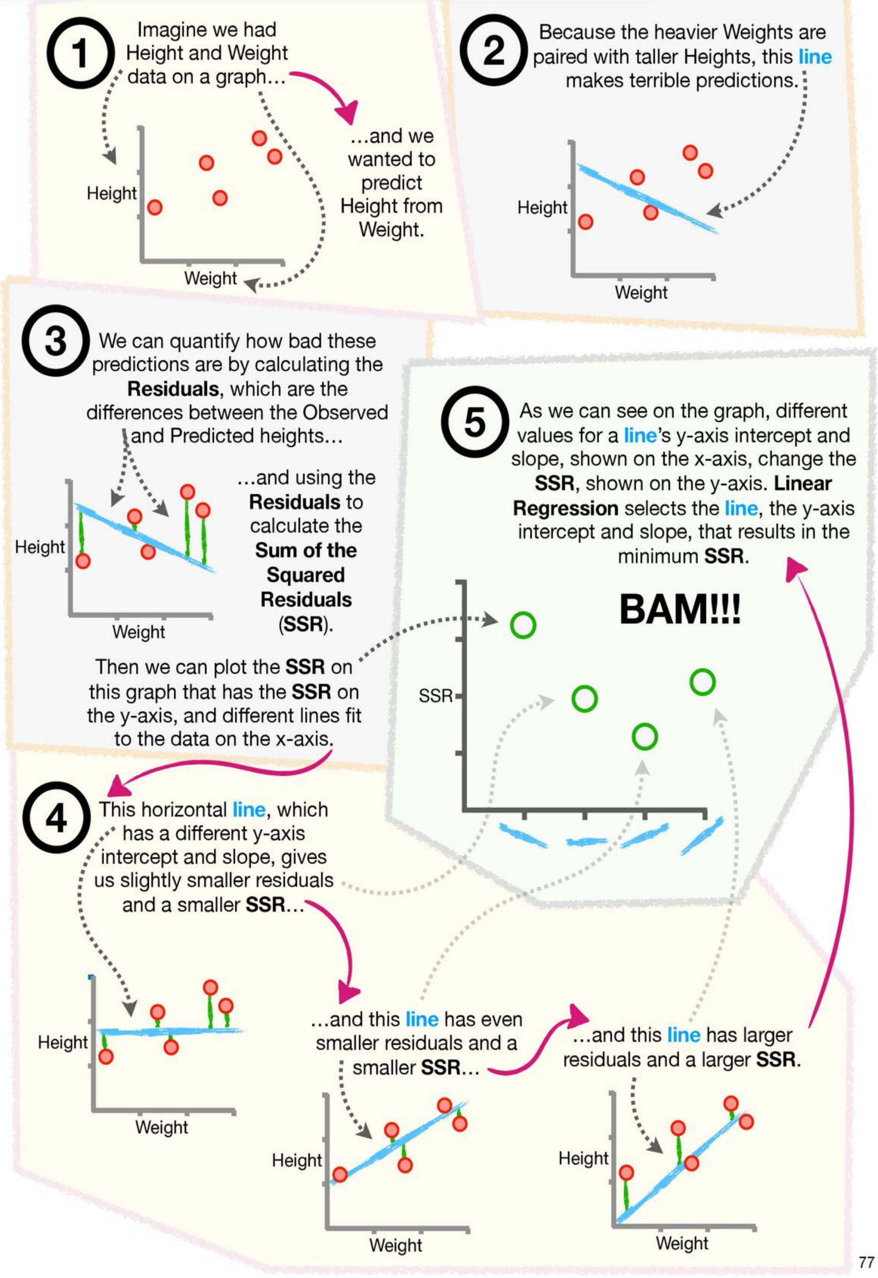
…and **Linear Regression** provides us with a ***p*-value** for the R^2 value, so we can get a sense of how confident we should be that the predictions made with the **fitted line** are better than predictions made with the **mean** of the y-axis coordinates for the data.

NOTE: Linear Regression is the gateway to a general technique called **Linear Models**, which can be used to create and evaluate models that go way beyond fitting simple straight lines to data!!!





Fitting a Line to Data: Main Ideas



① Imagine we had Height and Weight data on a graph…

…and we wanted to predict Height from Weight.

② Because the heavier Weights are paired with taller Heights, this **line** makes terrible predictions.

③ We can quantify how bad these predictions are by calculating the **Residuals**, which are the differences between the Observed and Predicted heights…

…and using the **Residuals** to calculate the **Sum of the Squared Residuals** (**SSR**).

Then we can plot the **SSR** on this graph that has the **SSR** on the y-axis, and different lines fit to the data on the x-axis.

④ This horizontal **line**, which has a different y-axis intercept and slope, gives us slightly smaller residuals and a smaller **SSR**…

…and this **line** has even smaller residuals and a smaller **SSR**…

…and this **line** has larger residuals and a larger **SSR**.

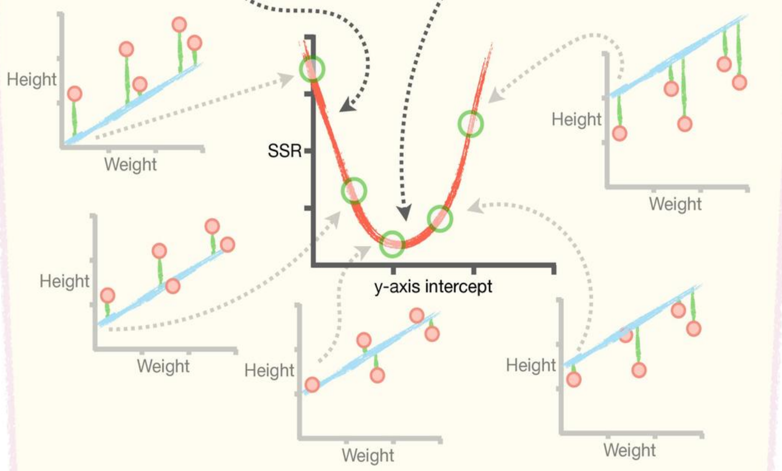
⑤ As we can see on the graph, different values for a **line**'s y-axis intercept and slope, shown on the x-axis, change the **SSR**, shown on the y-axis. **Linear Regression** selects the **line**, the y-axis intercept and slope, that results in the minimum **SSR**.

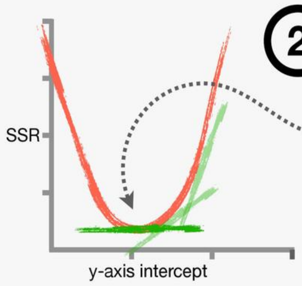
BAM!!!

Fitting a Line to Data: Intuition

① If we don't change the slope, we can see how the **SSR** changes for different y-axis intercept values…

…and, in this case, the goal of **Linear Regression** would be to find the y-axis intercept that results in the lowest **SSR** at the bottom of this curve.





② One way to find the lowest point in the curve is to calculate the derivative of the curve (**NOTE:** If you're not familiar with derivatives, see **Appendix D**) and solve for where the derivative is equal to **0**, at the bottom of the curve.

Solving this equation results in an **Analytical Solution**, meaning, we end up with a formula that we can plug our data into, and the output is the optimal value. Analytical solutions are awesome when you can find them (like for **Linear Regression**), but they're rare and only work in very specific situations.

③ Another way to find an optimal slope and y-axis intercept is to use an **Iterative Method** called **Gradient Descent**. In contrast to an **Analytical Solution**, an **Iterative Method** starts with a guess for the value and then goes into a loop that improves the guess one small step at a time. Although **Gradient Descent** takes longer than an analytical solution, it's one of the most important tools in machine learning because it can be used in a wide variety of situations where there are no analytical solutions, including **Logistic Regression**, **Neural Networks**, and many more.

Because **Gradient Descent** is so important, we'll spend all of **Chapter 5** on it. **GET EXCITED!!!**

p-values for Linear Regression and R^2: Main Ideas

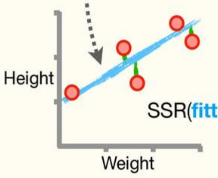
Gentle Reminder:

$$R^2 = \frac{\text{SSR(mean)} - \text{SSR(fitted line)}}{\text{SSR(mean)}}$$

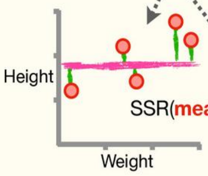
① Now, assuming that we've fit a line to the data that minimizes the **SSR** using an analytical solution or **Gradient Descent**, we calculate R^2 with the **SSR** for the fitted line…

…and the **SSR** for the mean height…

…and plug them into the equation for R^2 and get **0.66**.



SSR(fitted line) = 0.55



SSR(mean) = 1.61

$$R^2 = \frac{1.61 - 0.55}{1.61} = 0.66$$

② The R^2 value, **0.66**, suggests that using Weight to predict Height will be useful, but now we need to calculate the ***p*-value** to make sure that this result isn't due to random chance.

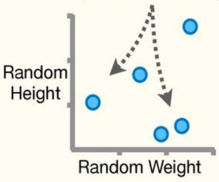
In this context, a ***p*-value** tells us the probability that random data could result in a similar R^2 value or a better one. In other words, the ***p*-value** will tell us the probability that random data could result in an $R^2 \geq 0.66$.

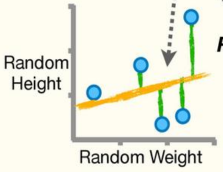
③ Because the original dataset has **5** pairs of measurements, one way* to calculate a ***p*-value** is to pair **5** *random* values for Height with **5** *random* values for Weight and plot them on a graph…

…then use **Linear Regression** to fit a line to the random data and calculate R^2…

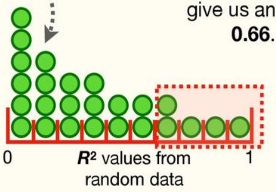
…and then add that R^2 to a histogram…

…and then create **>10,000** more sets of random data and add their R^2 values to the histogram and use the histogram to calculate the probability that random data will give us an $R^2 \geq$ **0.66**.





R^2 **= 0.03**



* **NOTE:** Because **Linear Regression** was invented before computers could quickly generate random data, **this is not the traditional way to calculate *p*-values**, but it works!!!

④ In the end, we get ***p*-value = 0.1**, meaning there's a **10%** chance that random data could give us an $R^2 \geq$ **0.66**. That's a relatively *high* ***p*-value**, so we might not have a lot of confidence in the predictions, which makes sense because we didn't have much data to begin with.

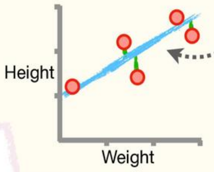
small bam.

Multiple Linear Regression: Main Ideas

1 So far, the example we've used demonstrates something called **Simple Linear Regression** because we use one variable, Weight, to predict Height…

Height = 1.1 + 0.5 × **Weight**

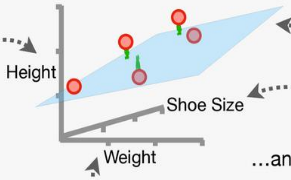
…and, as we've seen, **Simple Linear Regression** fits a line to the data that we can use to make predictions.



2 However, it's just as easy to use **2** or more variables, like Weight and Shoe Size, to predict Height.

Height = 1.1 + 0.5 × **Weight** + 0.3 × **Shoe Size**

This is called **Multiple Linear Regression**, and in this example, we end up with a **3**-dimensional graph of the data, which has **3** axes…



…one for Height…

…one for Weight…

…and one for Shoe Size…

…and instead of a fitting a **line** to the data, we fit a **plane**.

3 Just like for **Simple Linear Regression**, **Multiple Linear Regression** calculates R^2 and ***p*-values** from the **Sum of the Squared Residuals** (**SSR**). And the **Residuals** are still the difference between the **Observed Height** and the **Predicted Height**.

The only difference is that now we calculate **Residuals** around the **fitted plane** instead of a line.

$$R^2 = \frac{\text{SSR(mean) - SSR(fitted plane)}}{\text{SSR(mean)}}$$

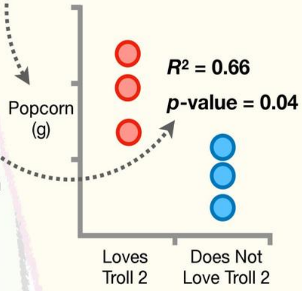
4 **NOTE:** When we use **3** or more variables to make a prediction, we can't draw the graph, but we can still do the math to calculate the **Residuals** for R^2 and its ***p*-value**.

Bam.

Beyond Linear Regression

① As mentioned at the start of this chapter, **Linear Regression** is the gateway to something called **Linear Models**, which are incredibly flexible and powerful.

② **Linear Models** allow us to use ***discrete*** data, like whether or not someone loves the movie Troll 2, to predict something ***continuous***, like how many grams of Popcorn they eat each day.

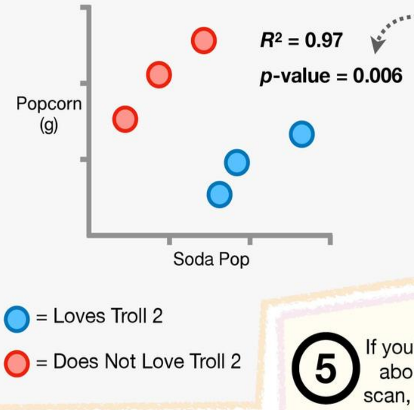


③ Just like when we used Weight to predict Height, **Linear Models** will give us an R^2 for this prediction, which gives us a sense of how accurate the predictions will be, and a ***p*-value** that lets us know how much confidence we should have in the predictions.

In this case, the ***p*-value** of **0.04** is relatively *small*, which suggests that it would be unlikely for random data to give us the same result or something more extreme. In other words, we can be confident that knowing whether or not someone loves Troll 2 will improve our prediction of how much Popcorn they will eat.

④ **Linear Models** also easily combine ***discrete*** data, like whether or not someone loves Troll 2, with ***continuous*** data, like how much Soda Pop they drink, to predict something ***continuous***, like how much Popcorn they will eat.

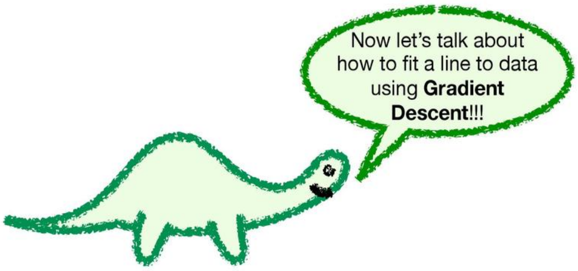
In this case, adding how much Soda Pop someone drinks to the model dramatically increased the R^2 value, which means the predictions will be more accurate, and reduced the ***p*-value**, suggesting we can have more confidence in the predictions.



DOUBLE BAM!!!

⑤ If you'd like to learn more about **Linear Models**, scan, click, or tap this QR code to check out the **'Quests** on YouTube!!!

bam.


Now let's talk about how to fit a line to data using **Gradient Descent**!!!

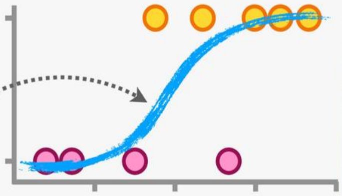
Chapter 05

Gradient Descent!!!

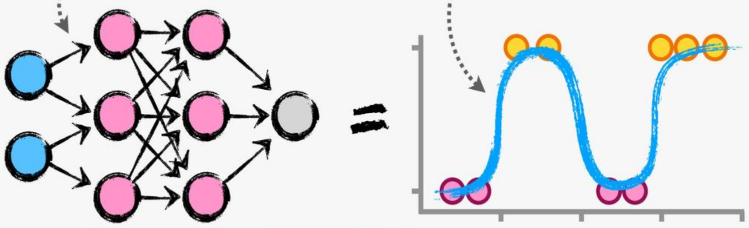
Gradient Descent: Main Ideas

① **The Problem:** A major part of machine learning is optimizing a model's fit to the data. Sometimes this can be done with an analytical solution, but it's not always possible.

For example, there is no analytical solution for **Logistic Regression** (**Chapter 6**), which fits an **s-shaped squiggle** to data.



Likewise, there is no analytical solution for **Neural Networks** (**Chapter 12**), which fit **fancy squiggles** to data.



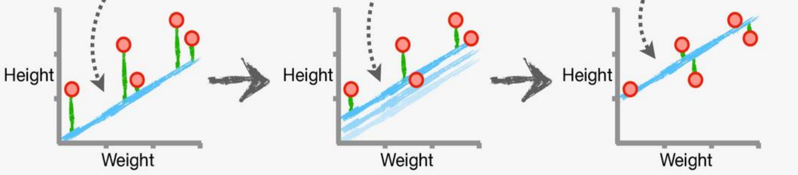
② **A Solution:** When there's no analytical solution, **Gradient Descent** can save the day!

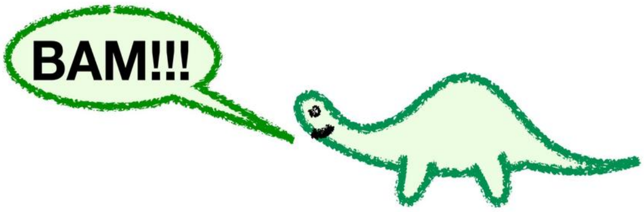
Gradient Descent is an ***iterative solution*** that incrementally steps toward an optimal solution and is used in a very wide variety of situations.

③ **Gradient Descent** starts with an initial guess…

…and then improves the guess, one step at a time…

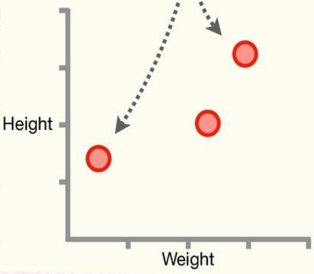
…until it finds an optimal solution or reaches a maximum number of steps.





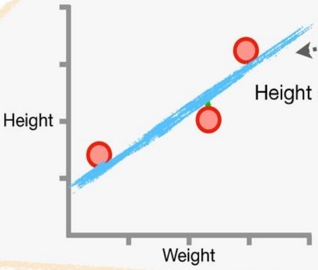
Gradient Descent: Details Part 1

① Let's show how **Gradient Descent** fits a line to these Height and Weight measurements.



NOTE: Even though there's an analytical solution for **Linear Regression**, we're using it to demonstrate how **Gradient Descent** works because we can compare the output from **Gradient Descent** to the known optimal values.

② Specifically, we'll show how **Gradient Descent** estimates the **intercept** and the **slope** of this line so that we minimize the **Sum of the Squared Residuals** (**SSR**).



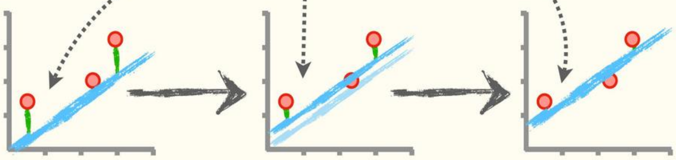
Height = **intercept** + **slope** x Weight

③ To keep things simple at the start, let's plug in the analytical solution for the **slope**, **0.64**...

...and show how **Gradient Descent** optimizes the **intercept** one step at a time.

Once we understand how **Gradient Descent** optimizes the **intercept**, we'll show how it optimizes the **intercept** *and* the **slope** at the same time.

Height = **intercept** + **0.64** x Weight



Gradient Descent: Details Part 2

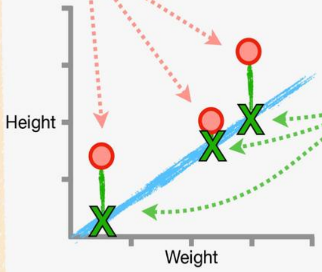
④ In this example, we're fitting a line to data, and we can evaluate how well that line fits with the **Sum of the Squared Residuals** (**SSR**).

Remember, **Residuals** are the difference between the Observed and Predicted values.

$$\text{Residual} = (\text{Observed Height} - \text{Predicted Height})$$

The Observed Heights are the values we originally measured…

…and the Predicted Heights come from the equation for the line…

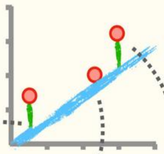


$$\text{Predicted Height} = \text{intercept} + 0.64 \times \text{Weight}$$

…so we can plug the equation for the line in for the Predicted value.

$$\begin{aligned}\text{Residual} &= (\text{Observed Height} - \text{Predicted Height})\\ &= (\text{Observed Height} - (\text{intercept} + 0.64 \times \text{Weight}))\end{aligned}$$

⑤ Now, because we have **3** data points, and thus, **3 Residuals**, the **SSR** has **3** terms.

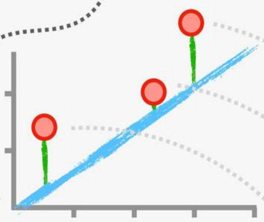


$$\begin{aligned}\text{SSR} = &(\text{Observed Height}_1 - (\text{intercept} + 0.64 \times \text{Weight}_1))^2\\ &+ (\text{Observed Height}_2 - (\text{intercept} + 0.64 \times \text{Weight}_2))^2\\ &+ (\text{Observed Height}_3 - (\text{intercept} + 0.64 \times \text{Weight}_3))^2\end{aligned}$$

Gradient Descent: Details Part 3

⑥ In this first example, since we're only optimizing the y-axis **intercept**, we'll start by assigning it a random value. In this case, we'll initialize the **intercept** by setting it to **0**.

Height = **0** + 0.64 x Weight



⑦ Now, to calculate the **SSR**, we first plug the value for the y-axis **intercept**, **0**, into the equation we derived in **Steps 4** and **5**…

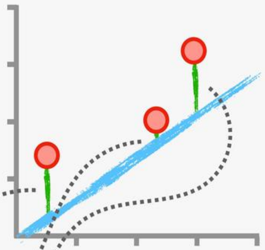
$$\begin{aligned}\text{SSR} &= (\text{Observed Height}_1 - (\textbf{intercept} + 0.64 \times \text{Weight}_1))^2 \\ &+ (\text{Observed Height}_2 - (\textbf{intercept} + 0.64 \times \text{Weight}_2))^2 \\ &+ (\text{Observed Height}_3 - (\textbf{intercept} + 0.64 \times \text{Weight}_3))^2\end{aligned}$$

$$\begin{aligned}\text{SSR} &= (\text{Observed Height}_1 - (\mathbf{0} + 0.64 \times \text{Weight}_1))^2 \\ &+ (\text{Observed Height}_2 - (\mathbf{0} + 0.64 \times \text{Weight}_2))^2 \\ &+ (\text{Observed Height}_3 - (\mathbf{0} + 0.64 \times \text{Weight}_3))^2\end{aligned}$$

⑧ …then we plug in the **Observed** values for Height and Weight for each data point.

$$\begin{aligned}\text{SSR} &= (\text{Observed Height}_1 - (\mathbf{0} + 0.64 \times \text{Weight}_1))^2 \\ &+ (\text{Observed Height}_2 - (\mathbf{0} + 0.64 \times \text{Weight}_2))^2 \\ &+ (\text{Observed Height}_3 - (\mathbf{0} + 0.64 \times \text{Weight}_3))^2\end{aligned}$$

$$\begin{aligned}\text{SSR} &= (\mathbf{1.4} - (\mathbf{0} + 0.64 \times \mathbf{0.5}))^2 \\ &+ (\mathbf{1.9} - (\mathbf{0} + 0.64 \times \mathbf{2.3}))^2 \\ &+ (\mathbf{3.2} - (\mathbf{0} + 0.64 \times \mathbf{2.9}))^2\end{aligned}$$

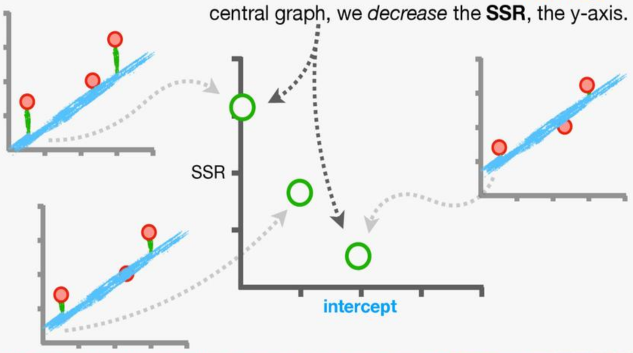


⑨ Lastly, we just do the math. The **SSR** for when the y-axis **intercept** is set to **0** is **3.1**. Bam!

$$\text{SSR} = 1.1^2 + 0.4^2 + 1.3^2 \boxed{= 3.1}$$

Gradient Descent: Details Part 4

⑩ Now, because the goal is to minimize the **SSR**, it's a type of **Loss** or **Cost Function** (see **Terminology Alert** below). In **Gradient Descent**, we minimize the **Loss** or **Cost Function** by taking steps away from the initial guess toward the optimal value. In this case, we see that as we *increase* the **intercept**, the x-axis of the central graph, we *decrease* the **SSR**, the y-axis.



TERMINOLOGY ALERT!!!

The terms **Loss Function** and **Cost Function** refer to anything we want to optimize when we fit a model to data. For example, we might want to optimize the **SSR** or the **Mean Squared Error** (**MSE**) when we fit a straight line with **Regression** or a squiggly line (in **Neural Networks**). That said, some people use the term **Loss Function** to specifically refer to a function (like the **SSR**) applied to *only one data point*, and use the term **Cost Function** to specifically refer to a function (like the **SSR**) applied to *all* of the data. Unfortunately, these specific meanings are not universal, so be aware of the context and be prepared to be flexible. In this book, we'll use them together and interchangeably, as in "The **Loss** or **Cost Function** is the **SSR**."

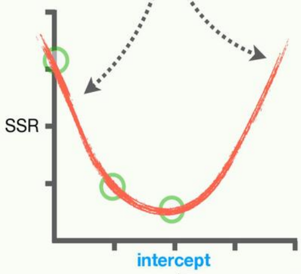
⑪ Now, instead of just randomly trying a bunch of values for the y-axis **intercept** and plotting the resulting **SSR** on a graph, we can plot the **SSR** as a function of the y-axis **intercept**. In other words, this equation for the **SSR**…

…corresponds to this **curve** on a graph that has the **SSR** on the y-axis and the **intercept** on the x-axis.

$$\text{SSR} = (1.4 - (\text{intercept} + 0.64 \times 0.5))^2$$
$$+ (1.9 - (\text{intercept} + 0.64 \times 2.3))^2$$
$$+ (3.2 - (\text{intercept} + 0.64 \times 2.9))^2$$

Psst! Remember: these are the **Observed Heights**…

…and these are the **Observed Weights**.

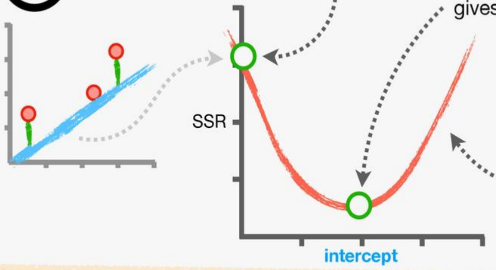


Gradient Descent: Details Part 5

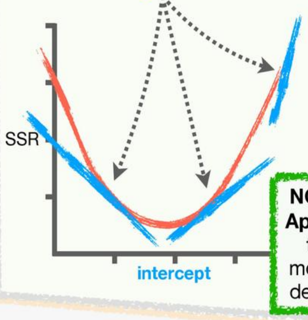
12 Now, when we started with the y-axis **intercept** = **0**, we got this **SSR**…

…so how do we take steps toward this y-axis intercept that gives us the lowest **SSR**…

…and how do we know when to stop or if we've gone too far?



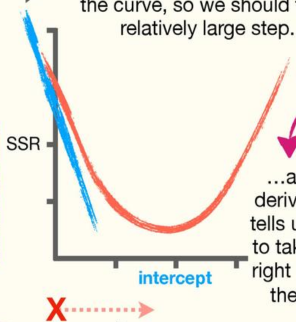
13 The answers to those questions come from the derivative of the curve, which tells us the slope of any **tangent line** that touches it.



NOTE: See **Appendix D** to learn more about derivatives.

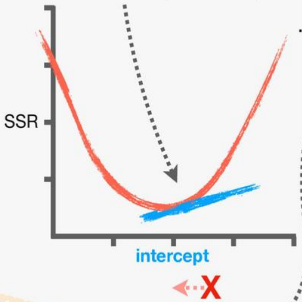
14 A relatively large value for the derivative, which corresponds to a relatively steep slope for the **tangent line**, suggests we're relatively far from the bottom of the curve, so we should take a relatively large step…

…and a *negative* derivative, or slope, tells us that we need to take a step to the right to get closer to the lowest **SSR**.



15 A relatively small value for the derivative suggests we're relatively close to the bottom of the curve, so we should take a relatively small step…

…and a *positive* derivative tells us that we need to take a step to the left to get closer to the lowest **SSR**.



In summary, the derivative tells us in which direction to take a step and how large that step should be, so let's learn how to take the derivative of the **SSR**!!!

Gradient Descent: Details Part 6

16 Because a single term of the **SSR** consists of a **Residual**… …wrapped in parentheses and squared… …one way to take the derivative of the **SSR** is to use **The Chain Rule** (see **Appendix F** if you need to refresh your memory about how **The Chain Rule** works).

$$\text{SSR} = (\ \text{Height} - (\text{intercept} + 0.64 \times \text{Weight})\ )^2$$

Step 1: Create a *link* between the **intercept** and the **SSR** by rewriting the **SSR** as the function of the **Residual**.

$$\text{SSR} = (\text{Residual})^2 \qquad \text{Residual} = \text{Height} - (\text{intercept} + 0.64 \times \text{Weight})$$

Step 2: Because the **Residual** links the **intercept** to the **SSR**, **The Chain Rule** tells us that the derivative of the **SSR** with respect to the **intercept** is…

$$\frac{d\ \text{SSR}}{d\ \text{intercept}} = \frac{d\ \text{SSR}}{d\ \text{Residual}} \times \frac{d\ \text{Residual}}{d\ \text{intercept}}$$

Step 3: Use **The Power Rule** (**Appendix E**) to solve for the two derivatives.

Because of the subtraction, we can remove the parentheses by multiplying everything inside by **-1**.

$$\frac{d\ \text{Residual}}{d\ \text{intercept}} = \frac{d}{d\ \text{intercept}} \text{Height} - (\text{intercept} + 0.64 \times \text{Weight})$$

$$= \frac{d}{d\ \text{intercept}} \text{Height} - \text{intercept} - 0.64 \times \text{Weight}$$

$$= 0 - 1 - 0 = -1$$

Because the first and last terms do not include the **intercept**, their derivatives, with respect to the **intercept**, are both **0**. However, the second term is the negative **intercept**, so its derivative is **-1**.

$$\frac{d\ \text{SSR}}{d\ \text{Residual}} = \frac{d}{d\ \text{Residual}} (\text{Residual})^2 = 2 \times \text{Residual}$$

Step 4: Plug the derivatives into **The Chain Rule** to get the final derivative of the **SSR** with respect to the **intercept**.

$$\frac{d\ \text{SSR}}{d\ \text{intercept}} = \frac{d\ \text{SSR}}{d\ \text{Residual}} \times \frac{d\ \text{Residual}}{d\ \text{intercept}} = 2 \times \text{Residual} \times -1$$

$$= 2 \times (\ \text{Height} - (\text{intercept} + 0.64 \times \text{Weight})\ ) \times -1$$

$$= -2 \times (\ \text{Height} - (\text{intercept} + 0.64 \times \text{Weight})\ )$$

Multiply this **-1** on the right by the **2** on the left to get **-2**.

BAM!!!

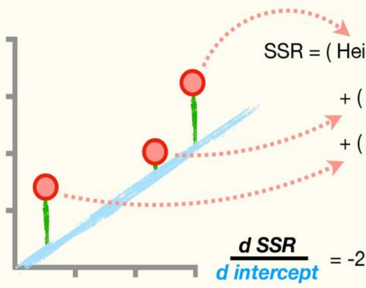
Gradient Descent: Details Part 7

⑰ So far, we've calculated the derivative of the **SSR** for a single observation.

$$SSR = (\text{Height} - (\text{intercept} + 0.64 \times \text{Weight}))^2$$

$$\frac{d\,SSR}{d\,\text{intercept}} = -2 \times (\text{Height} - (\text{intercept} + 0.64 \times \text{Weight}))$$

⑱ However, we have three observations in the dataset, so the **SSR** and its derivative both have three terms.

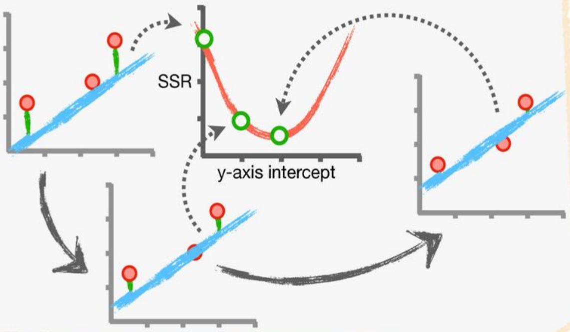


$$\begin{aligned} SSR = {} & (\text{Height} - (\text{intercept} + 0.64 \times \text{Weight}))^2 \\ & + (\text{Height} - (\text{intercept} + 0.64 \times \text{Weight}))^2 \\ & + (\text{Height} - (\text{intercept} + 0.64 \times \text{Weight}))^2 \end{aligned}$$

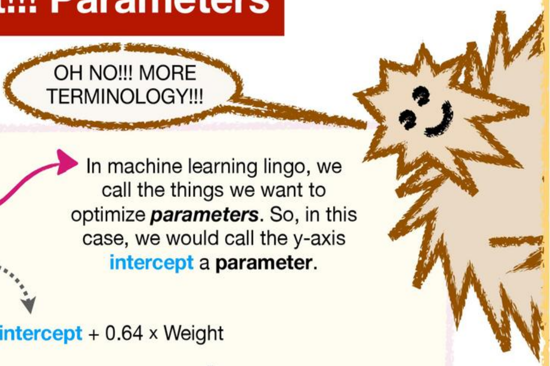
$$\begin{aligned} \frac{d\,SSR}{d\,\text{intercept}} = {} & -2 \times (\text{Height} - (\text{intercept} + 0.64 \times \text{Weight})) \\ & + -2 \times (\text{Height} - (\text{intercept} + 0.64 \times \text{Weight})) \\ & + -2 \times (\text{Height} - (\text{intercept} + 0.64 \times \text{Weight})) \end{aligned}$$

Gentle Reminder: Because we're using **Linear Regression** as our example, we don't actually ***need*** to use **Gradient Descent** to find the optimal value for the intercept. Instead, we could just set the derivative equal to **0** and solve for the **intercept**. This would be an analytical solution. However, by applying **Gradient Descent** to this problem, we can compare the optimal value that it gives us to the analytical solution and evaluate how well **Gradient Descent** performs. This will give us more confidence in **Gradient Descent** when we use it in situations without analytical solutions like **Logistic Regression** and **Neural Networks**.

⑲ Now that we have the derivative of the **SSR** for all **3** data points, we can go through, step-by-step, how **Gradient Descent** uses this derivative to find the **intercept** value that minimizes the **SSR**. However, before we get started, it's time for the dreaded **Terminology Alert!!!**



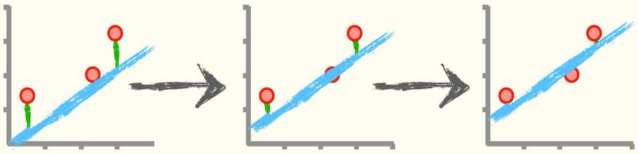
Terminology Alert!!! Parameters



① In the current example, we're trying to optimize the y-axis **intercept**.

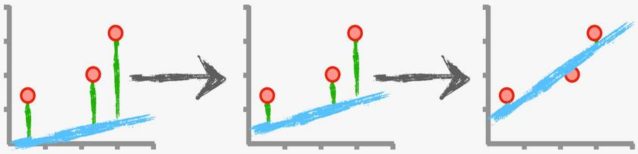
In machine learning lingo, we call the things we want to optimize ***parameters***. So, in this case, we would call the y-axis **intercept** a **parameter**.

Predicted Height = **intercept** + 0.64 x Weight

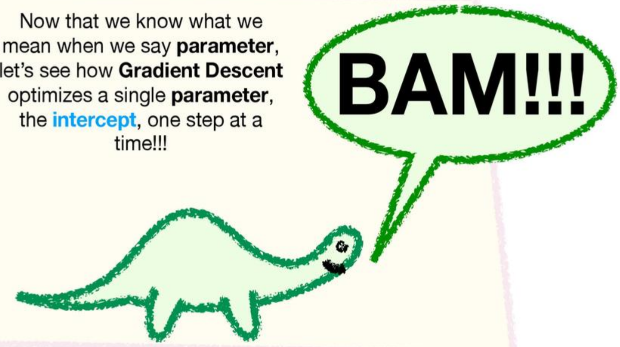


② If we wanted to optimize both the y-axis **intercept** and the **slope**, then we would need to optimize two **parameters**. tiny bam.

Predicted Height = **intercept** + **slope** x Weight



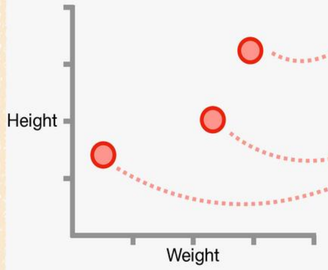
③ Now that we know what we mean when we say **parameter**, let's see how **Gradient Descent** optimizes a single **parameter**, the **intercept**, one step at a time!!!



Gradient Descent for One Parameter: Step-by-Step

① First, plug the **Observed** values into the derivative of the **Loss** or **Cost Function**. In this example, the **SSR** is the **Loss** or **Cost Function**...

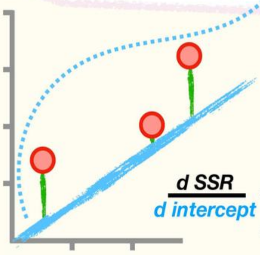
...so that means plugging the **Observed Weight** and **Height** measurements into the derivative of the **SSR**.



$$\frac{d\ SSR}{d\ intercept} = -2 \times (\text{Height} - (\text{intercept} + 0.64 \times \text{Weight}))$$
$$+ -2 \times (\text{Height} - (\text{intercept} + 0.64 \times \text{Weight}))$$
$$+ -2 \times (\text{Height} - (\text{intercept} + 0.64 \times \text{Weight}))$$

$$\frac{d\ SSR}{d\ intercept} = -2 \times (3.2 - (\text{intercept} + 0.64 \times 2.9))$$
$$+ -2 \times (1.9 - (\text{intercept} + 0.64 \times 2.3))$$
$$+ -2 \times (1.4 - (\text{intercept} + 0.64 \times 0.5))$$

② Now we initialize the parameter we want to optimize with a random value. In this example, where we just want to optimize the y-axis **intercept**, we start by setting it to **0**.



$$\frac{d\ SSR}{d\ intercept} = -2 \times (3.2 - (0 + 0.64 \times 2.9))$$
$$+ -2 \times (1.9 - (0 + 0.64 \times 2.3))$$
$$+ -2 \times (1.4 - (0 + 0.64 \times 0.5))$$

= **intercept** + 0.64 x Weight

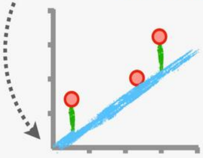
Height = **0** + 0.64 x Weight

Gradient Descent for One Parameter: Step-by-Step

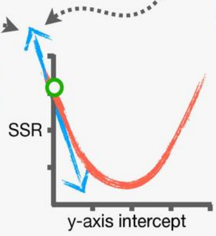
③ Now evaluate the derivative at the current value for the **intercept**. In this case, the current value is **0**.

When we do the math, we get **-5.7**…

…thus, when the **intercept** = **0**, the slope of this **tangent line** is **-5.7**.



$$\frac{d\ SSR}{d\ intercept} = -2 \times (\,3.2 - \ (0 \ + 0.64 \times 2.9)\,) + -2 \times (\,1.9 - \ (0 \ + 0.64 \times 2.3)\,) + -2 \times (\,1.4 - \ (0 \ + 0.64 \times 0.5)\,) \ \boxed{= -5.7}$$



④ Now calculate the **Step Size** with the following equation:

Gentle Reminder: The magnitude of the derivative is proportional to how big of a step we should take toward the minimum. The sign (+/-) tells us which direction.

Step Size = **Derivative** × **Learning Rate**

$$= -5.7 \times 0.1$$

$$\boxed{= -0.57}$$

NOTE: The **Learning Rate** prevents us from taking steps that are too big and skipping past the lowest point in the curve. Typically, for **Gradient Descent**, the **Learning Rate** is determined automatically: it starts relatively large and gets smaller with every step taken. However, you can also use **Cross Validation** to determine a good value for the **Learning Rate**. In this case, we're setting the **Learning Rate** to **0.1**.

⑤ Take a step from the **current intercept** to get closer to the optimal value with the following equation:

New intercept = **Current intercept** - **Step Size**

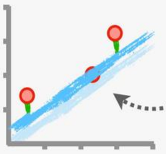
$$= 0 - (-0.57)$$

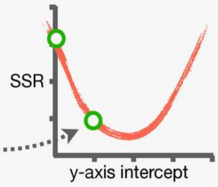
$$\boxed{= 0.57}$$

Remember, in this case, the **current intercept** is **0**.

The **new intercept**, **0.57**, moves the line up a little closer to the data…

…and it results in a lower **SSR**. Bam!



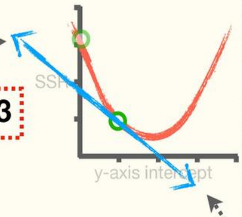


Gradient Descent for One Parameter: Step-by-Step

6 Now repeat the previous three steps, updating the **intercept** after each iteration until the **Step Size** is close to **0** or we take the maximum number of steps, which is often set to **1,000** iterations.

a Evaluate the derivative at the current value for the intercept…

$$\begin{aligned}\frac{d\ SSR}{d\ intercept} &= -2 \times (3.2 - (0.57 + 0.64 \times 2.9))\\ &+ -2 \times (1.9 - (0.57 + 0.64 \times 2.3)) \quad \boxed{= -2.3}\\ &+ -2 \times (1.4 - (0.57 + 0.64 \times 0.5))\end{aligned}$$



b Calculate the **Step Size**…

Step Size = **Derivative** x **Learning Rate**

$$= -2.3 \times 0.1$$

$$\boxed{= -0.23}$$

NOTE: The **Step Size** is smaller than before because the slope of the **tangent line** is not as steep as before. The smaller slope means we're getting closer to the optimal value.

c Calculate the **new intercept** value…

New intercept = **Current intercept** - **Step Size**

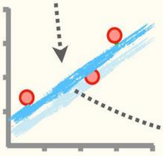
$$= 0.57 - (-0.23)$$

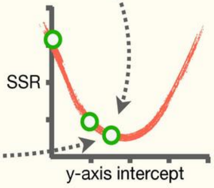
$$\boxed{= 0.8}$$

The **new intercept**, **0.8**, moves the line up a little closer to the data…

…and it results in a lower **SSR**.

Double Bam!





Gradient Descent for One Parameter: Step-by-Step

7 After **7** iterations of **Gradient Descent**…

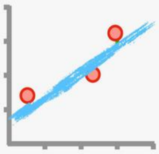
a Evaluate the derivative at the current value…

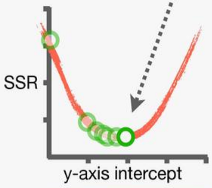
b Calculate the **Step Size**…

c Calculate the new value…

…the **Step Size** was very close to **0,** so we stopped with the **current intercept** = **0.95**…

…and we made it to the lowest **SSR**.





8 If, earlier on, instead of using **Gradient Descent**, we simply set the derivative to **0** and solved for the **intercept**, we would have gotten **0.95**, which is the same value that **Gradient Descent** gave us. Thus, **Gradient Descent** did a decent job.

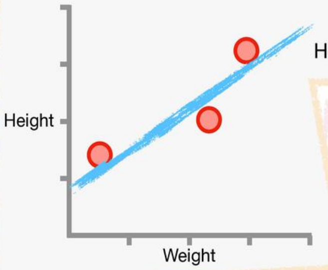
BAM???

Not yet! Now let's see how well **Gradient Descent** optimizes the **intercept** *and* the **slope**!

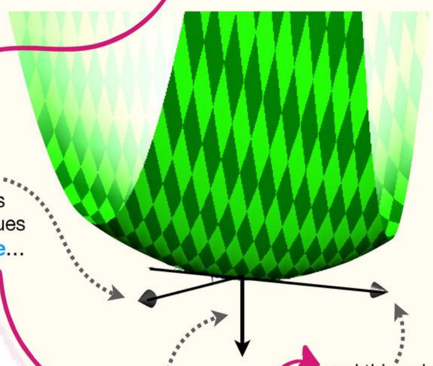
Optimizing Two or More Parameters: Details

① Now that we know how to optimize the **intercept** of the line that minimizes the **SSR**, let's optimize both the **intercept** and the **slope**.

Height = **intercept** + **slope** x Weight



② When we optimize two parameters, we get a 3-dimensional graph of the **SSR**.

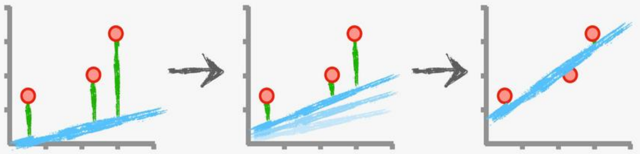


This axis represents different values for the **slope**…

…the vertical axis is for the **SSR**…

…and this axis represents different values for the **intercept**.

③ Just like before, the goal is to find the parameter values that give us the lowest **SSR**. And just like before, **Gradient Descent** initializes the parameters with random values and then uses derivatives to update those parameters, one step at a time, until they're optimal.



④ So, now let's learn how to take derivatives of the **SSR** with respect to both the **intercept** and the **slope**.

SSR = (Height - (**intercept** + **slope** x Weight))^2

Taking Multiple (Partial) Derivatives of the SSR: Part 1

① The good news is that taking the derivative of the **SSR** with respect to the **intercept** is exactly the same as before.

We can use **The Chain Rule** to tell us how the **SSR** changes with respect to the **intercept**.

$$\text{SSR} = (\text{ Height - (intercept + slope × Weight) })^2$$

Step 1: Create a *link* between the **intercept** and the **SSR** by rewriting the **SSR** as the function of the **Residual**.

$$\text{SSR} = (\textbf{Residual})^2 \qquad \textbf{Residual} = \text{Observed Height - (intercept + slope × Weight)}$$

Step 2: Because the **Residual** links the **intercept** to the **SSR**, **The Chain Rule** tells us that the derivative of the **SSR** with respect to the **intercept** is...

$$\frac{d\ SSR}{d\ intercept} = \frac{d\ SSR}{d\ Residual} \times \frac{d\ Residual}{d\ intercept}$$

Step 3: Use **The Power Rule** to solve for the two derivatives.

Because of the subtraction, we can remove the parentheses by multiplying everything inside by **-1**.

$$\frac{d\ Residual}{d\ intercept} = \frac{d}{d\ intercept}\ \text{Height - (intercept + slope × Weight)}$$

$$= \frac{d}{d\ intercept}\ \text{Height - intercept - slope × Weight}$$

$$= 0 - \mathbf{1} - 0 = -1$$

Because the first and last terms do not include the **intercept**, their derivatives, with respect to the **intercept**, are both **0**. However, the second term is the negative **intercept**, so its derivative is **-1**.

$$\frac{d\ SSR}{d\ Residual} = \frac{d}{d\ Residual}(\textbf{Residual})^2 = 2 \times \textbf{Residual}$$

Step 4: Plug the derivatives into **The Chain Rule** to get the final derivative of the **SSR** with respect to the **intercept**.

$$\frac{d\ SSR}{d\ intercept} = \frac{d\ SSR}{d\ Residual} \times \frac{d\ Residual}{d\ intercept} = 2 \times \textbf{Residual} \times -1$$

$$= 2 \times \textbf{(}\text{ Height - (intercept + slope × Weight) }\textbf{)} \times -1$$

$$= -2 \times \textbf{(}\text{ Height - (intercept + slope × Weight) }\textbf{)}$$

Multiply this **-1** on the right by the **2** on the left to get **-2**.

BAM!!!

Taking Multiple (Partial) Derivatives of the SSR: Part 2

② The other good news is that taking the derivative of the **SSR** with respect to the **slope** is very similar to what we just did for the **intercept**.

We can use **The Chain Rule** to tell us how the **SSR** changes with respect to the **slope**.

SSR = (Height - (**intercept** + **slope** x Weight))²

Step 1: Create a *link* between the **slope** and the **SSR** by rewriting the **SSR** as the function of the **Residual**.

SSR = (**Residual**)²

Residual = Observed Height - (**intercept** + **slope** x Weight)

Step 2: Because the **Residual** links the **slope** to the **SSR**, **The Chain Rule** tells us that the derivative of the **SSR** with respect to the **slope** is…

$$\frac{d\ SSR}{d\ slope} = \frac{d\ SSR}{d\ Residual} \times \frac{d\ Residual}{d\ slope}$$

Step 3: Use **The Power Rule** to solve for the two derivatives.

Because of the subtraction, we can remove the parentheses by multiplying everything inside by **-1**.

$$\frac{d\ Residual}{d\ slope} = \frac{d}{d\ slope} \text{Height - (intercept + slope x Weight)}$$

$$= \frac{d}{d\ slope} \text{Height - intercept - slope x Weight}$$

= 0 - 0 - Weight = -Weight

Because the first and second terms do not include the **slope**, their derivatives, with respect to the **slope**, are both **0**. However, the last term is the negative **slope** times **Weight**, so its derivative is **-Weight**.

$$\frac{d\ SSR}{d\ Residual} = \frac{d}{d\ Residual}(\text{Residual})^2 = 2 \times \text{Residual}$$

Step 4: Plug the derivatives into **The Chain Rule** to get the final derivative of the **SSR** with respect to the **slope**.

$$\frac{d\ SSR}{d\ slope} = \frac{d\ SSR}{d\ Residual} \times \frac{d\ Residual}{d\ slope} = 2 \times \text{Residual} \times \text{-Weight}$$

= 2 x **(** Height - (**intercept** + **slope** x Weight) **)** x -Weight

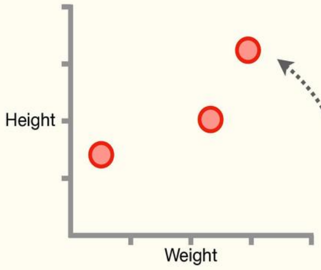
= -2 x Weight x **(** Height - (**intercept** + **slope** x Weight) **)**

Multiply this **-Weight** on the right by the **2** on the left to get **-2 x Weight**.

DOUBLE BAM!!!

Gradient Descent for Two Parameters: Step-by-Step

① Plug the **Observed** values into the derivatives of the **Loss** or **Cost Function**. In this example, the **SSR** is the **Loss** or **Cost Function**, so we'll plug the **Observed Weight** and **Height** measurements into the two derivatives of the **SSR**, one with respect to the **intercept**…



Gentle Reminder: The Weight and Height values that we're plugging into the derivatives come from the raw data in the graph.

$$\begin{aligned}\frac{d\,\text{SSR}}{d\,\text{intercept}} &= -2 \times (\text{Height}_1 - (\text{intercept} + \text{slope} \times \text{Weight}_1))\\ &+ -2 \times (\text{Height}_2 - (\text{intercept} + \text{slope} \times \text{Weight}_2))\\ &+ -2 \times (\text{Height}_3 - (\text{intercept} + \text{slope} \times \text{Weight}_3))\end{aligned}$$

$$\begin{aligned}\frac{d\,\text{SSR}}{d\,\text{intercept}} &= -2 \times (3.2 - (\text{intercept} + \text{slope} \times 2.9))\\ &+ -2 \times (1.9 - (\text{intercept} + \text{slope} \times 2.3))\\ &+ -2 \times (1.4 - (\text{intercept} + \text{slope} \times 0.5))\end{aligned}$$

….and one with respect to the **slope**.

$$\begin{aligned}\frac{d\,\text{SSR}}{d\,\text{slope}} &= -2 \times \text{Weight}_1 \times (\text{Height}_1 - (\text{intercept} + \text{slope} \times \text{Weight}_1))\\ &+ -2 \times \text{Weight}_2 \times (\text{Height}_2 - (\text{intercept} + \text{slope} \times \text{Weight}_2))\\ &+ -2 \times \text{Weight}_3 \times (\text{Height}_3 - (\text{intercept} + \text{slope} \times \text{Weight}_3))\end{aligned}$$

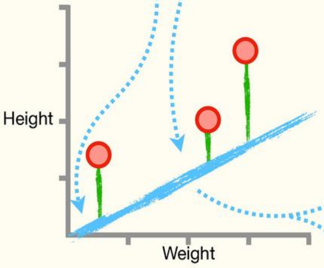
$$\begin{aligned}\frac{d\,\text{SSR}}{d\,\text{slope}} &= -2 \times 2.9 (3.2 - (\text{intercept} + \text{slope} \times 2.9))\\ &+ -2 \times 2.3 (1.9 - (\text{intercept} + \text{slope} \times 2.3))\\ &+ -2 \times 0.5 (1.4 - (\text{intercept} + \text{slope} \times 0.5))\end{aligned}$$

Gradient Descent for Two Parameters: Step-by-Step

② Now initialize the parameter, or parameters, that we want to optimize with random values. In this example, we'll set the **intercept** to **0** and the **slope** to **0.5**.

Height = **intercept** + **slope** x Weight

Height = **0** + **0.5** x Weight



$$\frac{d\ SSR}{d\ intercept} = -2 \times (3.2 - (intercept + slope \times 2.9))$$
$$+ -2 \times (1.9 - (intercept + slope \times 2.3))$$
$$+ -2 \times (1.4 - (intercept + slope \times 0.5))$$

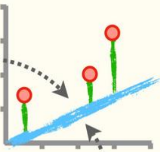
$$\frac{d\ SSR}{d\ intercept} = -2 \times (3.2 - (0 + 0.5 \times 2.9))$$
$$+ -2 \times (1.9 - (0 + 0.5 \times 2.3))$$
$$+ -2 \times (1.4 - (0 + 0.5 \times 0.5))$$

$$\frac{d\ SSR}{d\ slope} = -2 \times 2.9 \times (3.2 - (intercept + slope \times 2.9))$$
$$+ -2 \times 2.3 \times (1.9 - (intercept + slope \times 2.3))$$
$$+ -2 \times 0.5 \times (1.4 - (intercept + slope \times 0.5))$$

$$\frac{d\ SSR}{d\ slope} = -2 \times 2.9 \times (3.2 - (0 + 0.5 \times 2.9))$$
$$+ -2 \times 2.3 \times (1.9 - (0 + 0.5 \times 2.3))$$
$$+ -2 \times 0.5 \times (1.4 - (0 + 0.5 \times 0.5))$$

Gradient Descent for Two Parameters: Step-by-Step

3 Evaluate the derivatives at the current values for the **intercept**, **0**, and **slope**, **0.5**.



$$\frac{d\,SSR}{d\,intercept} = -2 \times (3.2 - (0 + 0.5 \times 2.9)) + -2 \times (1.9 - (0 + 0.5 \times 2.3)) + -2 \times (1.4 - (0 + 0.5 \times 0.5)) \boxed{= -7.3}$$

$$\frac{d\,SSR}{d\,slope} = -2 \times 2.9 \times (3.2 - (0 + 0.5 \times 2.9)) + -2 \times 1.9 \times (2.3 - (0 + 0.5 \times 1.9)) + -2 \times 0.5 \times (1.4 - (0 + 0.5 \times 0.5)) \boxed{= -14.8}$$

4 Calculate the **Step Sizes**: one for the **intercept**…

$$\textbf{Step Size}_{\text{Intercept}} = \textbf{Derivative} \times \textbf{Learning Rate}$$

$$= -7.3 \times 0.01$$

$$\boxed{= -0.073}$$

…and one for the **slope**.

$$\textbf{Step Size}_{\text{Slope}} = \textbf{Derivative} \times \textbf{Learning Rate}$$

$$= -14.8 \times 0.01$$

$$\boxed{= -0.148}$$

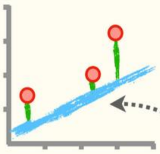
NOTE: We're using a smaller **Learning Rate** now (**0.01**) than before (**0.1**) because **Gradient Descent** can be very sensitive to it. However, as we said earlier, usually the **Learning Rate** is determined automatically.

5 Take a step from the **current intercept**, **0**, and **slope**, **0.5**, to get closer to the optimal values…

$$\textbf{New intercept} = \textbf{Current intercept} - \textbf{Step Size}_{\text{Intercept}}$$

$$= 0 - (-0.073)$$

$$\boxed{= 0.073}$$



…and the **intercept** increases from **0** to **0.073**, the **slope** increases from **0.5** to **0.648**, and the **SSR** decreases. **BAM!**

$$\textbf{New slope} = \textbf{Current slope} - \textbf{Step Size}_{\text{Slope}}$$

$$= 0.5 - (-0.148)$$

$$\boxed{= 0.648}$$

Gradient Descent for Two Parameters: Step-by-Step

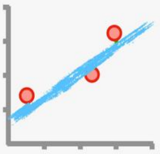
6 And after **475** iterations…

a Evaluate the derivatives at their current values…

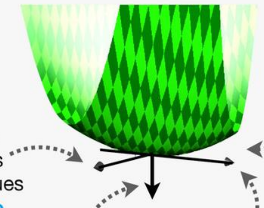
b Calculate the **Step Sizes**…

c Calculate the new values…

…the **Step Size** was very close to **0**, so we stopped with the **current intercept** = **0.95** and the **current slope** = **0.64**…



…and we made it to the lowest **SSR**.



This axis represents different values for the **slope**…

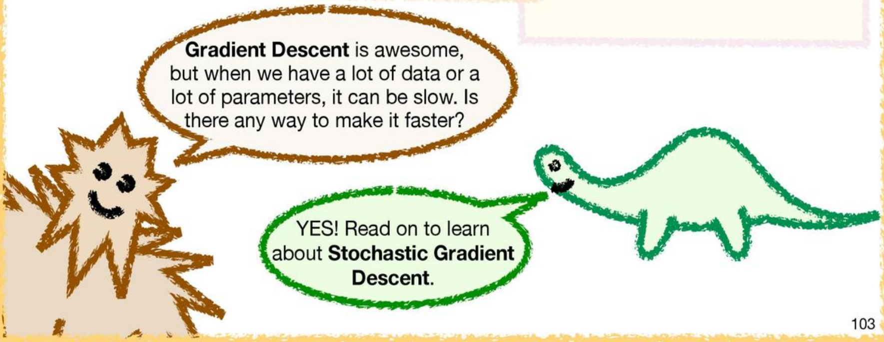
…this axis is for the **SSR**…

…and this axis represents different values for the **intercept**.

7

If, earlier on, instead of using **Gradient Descent**, we simply set the derivatives to **0** and solved for the **intercept** and **slope**, we would have gotten **0.95** and **0.64**, which are the same values **Gradient Descent** gave us. Thus, **Gradient Descent** did a great job, and we can confidently use it in situations where there are no analytical solutions, like **Logistic Regression** and **Neural Networks**.

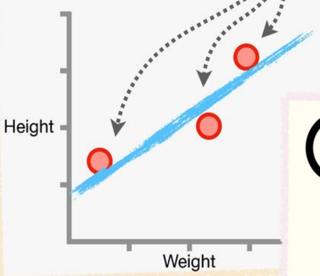
TRIPLE BAM!!!



Stochastic Gradient Descent: Main Ideas

① So far, things have been pretty simple. We started out with a tiny dataset that only had **3** points…

…and we fit a straight **line**, which only has **2** parameters, the **intercept** and the **slope**.



Height = **intercept** + **slope** x Weight

② Because there were only **2** parameters, we only had **2** derivatives that we had to calculate in each step…

$$\frac{d\ SSR}{d\ intercept} \qquad \frac{d\ SSR}{d\ slope}$$

…and because we only had **3** data points, each derivative only needed to compute **3** terms per derivative.

$$\frac{d\ SSR}{d\ intercept} = -2 \times (3.2 - (\text{intercept} + \text{slope} \times 2.9)) + -2 \times (1.9 - (\text{intercept} + \text{slope} \times 2.3)) + -2 \times (1.4 - (\text{intercept} + \text{slope} \times 0.5))$$

$$\frac{d\ SSR}{d\ slope} = -2 \times 2.9 \times (3.2 - (\text{intercept} + \text{slope} \times 2.9)) + -2 \times 2.3 \times (1.9 - (\text{intercept} + \text{slope} \times 2.3)) + -2 \times 0.5 \times (1.4 - (\text{intercept} + \text{slope} \times 0.5))$$

③ However, what if we had **1,000,000** data points? Then we would have to compute **1,000,000** terms per derivative.

Ugh!

And what if we had a more complicated model with **10,000** parameters? Then we would have **10,000** derivatives to compute.

Double Ugh!

Taking **10,000** derivatives, each with **1,000,000** terms to compute, is a lot of work, and all of that work only gets us one step into the process that can take **1,000s** of steps!!!

TRIPLE UGH!

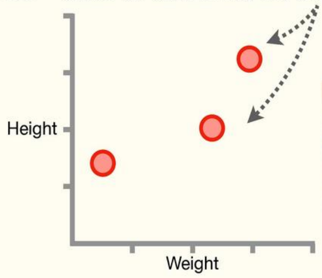
Thus, for **BIG DATA**, **Gradient Descent** requires a lot of computation and can be slow.

④ The good news is that **Stochastic Gradient Descent** can drastically reduce the amount of computation required to optimize parameters. Although it sounds fancy, the word **Stochastic** just means **Randomly Determined** and all **Stochastic Gradient Descent** does is randomly select one data point per step. So, regardless of how large your dataset is, only one term is computed per derivative for each iteration.

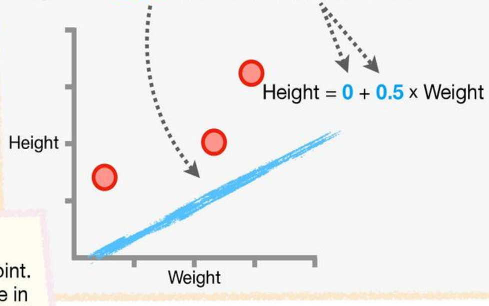
BAM!

Stochastic Gradient Descent: Details Part 1

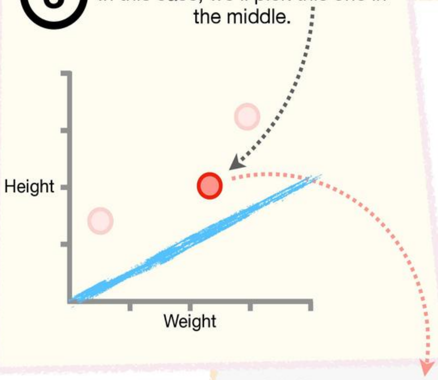
① To see how **Stochastic Gradient Descent** works, let's go back to our simple example, where we want to fit a line to **3** data points.



② And just like with normal **Gradient Descent**, we start by initializing the **intercept** and **slope** of the line with random values.



③ Now we randomly pick one point. In this case, we'll pick this one in the middle.



④ Then, we evaluate the derivatives using just that single point...

$$\frac{d\ \mathit{SSR}}{d\ \mathit{intercept}} = -2 \times (\ \text{Height} - (\text{intercept} + \text{slope} \times \text{Weight})\ )$$

$$\frac{d\ \mathit{SSR}}{d\ \mathit{slope}} = -2 \times \text{Weight} \times (\ \text{Height} - (\text{intercept} + \text{slope} \times \text{Weight})\ )$$

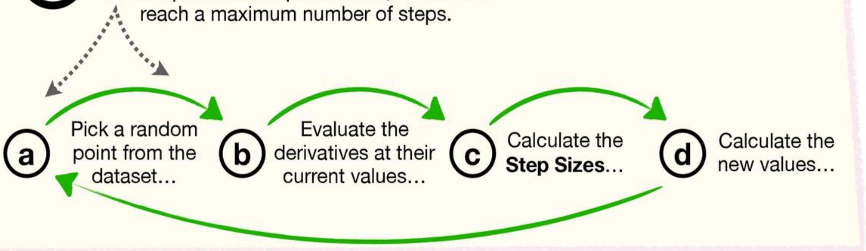
⑤ ...and then we calculate the **Step Sizes**...

⑥ ...and then we calculate the new values.

Stochastic Gradient Descent: Details Part 2

7 Then we just repeat the last **4** steps until the **Step Sizes** are super small, which suggests we've optimized the parameters, or until we reach a maximum number of steps.

BAM!



a Pick a random point from the dataset…

b Evaluate the derivatives at their current values…

c Calculate the **Step Sizes**…

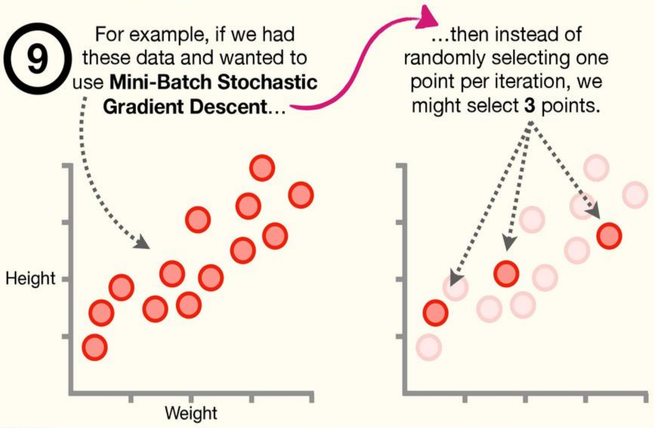
d Calculate the new values…

8 **TERMINOLOGY ALERT!!!**

Although a strict definition of **Stochastic Gradient Descent** says that we only select a single point per iteration, it's much more common to randomly select a small subset of the observations. This is called **Mini-Batch Stochastic Gradient Descent**. Using a small subset, rather than a single point, usually converges on the optimal values in fewer steps and takes much less time than using all of the data.

9 For example, if we had these data and wanted to use **Mini-Batch Stochastic Gradient Descent**…

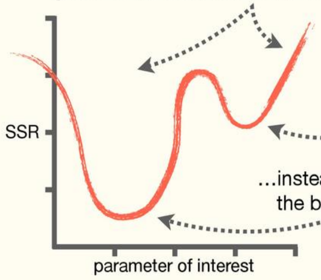
…then instead of randomly selecting one point per iteration, we might select **3** points.



Gradient Descent: FAQ

Will Gradient Descent always find the best parameter values?

Unfortunately, **Gradient Descent** does not always find the best parameter values. For example, if the graph of the **SSR** looked like this….



…then it's possible that we might get stuck at the bottom of this **local minimum**…

…instead of finding our way to the bottom and the **global minimum**.

When this happens (when we get stuck in a local minimum instead of finding the global minimum), it's a bummer. Even worse, usually it's not possible to graph the **SSR**, so we might not even know we're in one, of potentially many, local minimums. However, there are a few things we can do about it. We can:

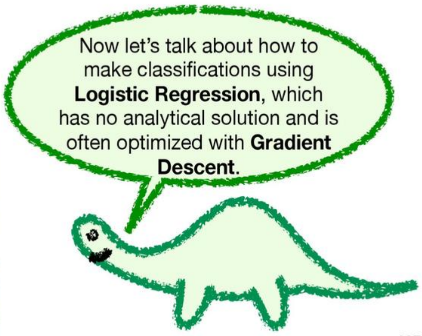
1) Try again using different random numbers to initialize the parameters that we want to optimize. Starting with different values may avoid a local minimum.

2) Fiddle around with the **Step Size**. Making it a little larger may help avoid getting stuck in a local minimum.

3) Use **Stochastic Gradient Descent**, because the extra randomness helps avoid getting trapped in a local minimum.

How do you choose the size of a Mini-Batch for Stochastic Gradient Descent?

The answer to this question really depends on the computer hardware you're using to train (optimize) your model. For example, because one of the main reasons we use **Mini-Batch Stochastic Gradient Descent** is to train our model as quickly as possible, one major consideration is how much high-speed memory we have access to. The more high-speed memory we have, the larger the **Mini-Batch** can be.



Chapter 06

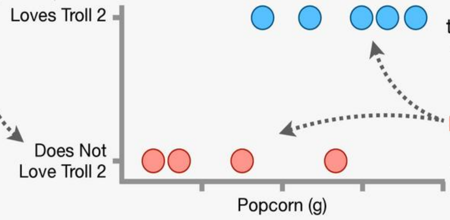
Logistic Regression!!!

Logistic Regression: Main Ideas Part 1

① **The Problem: Linear Regression** and **Linear Models** are great when we want to predict something that is ***continuous***, like **Height**, but what if we want to classify something ***discrete*** that only has two possibilities, like whether or not someone loves the movie Troll 2?

🔵 = Loves Troll 2

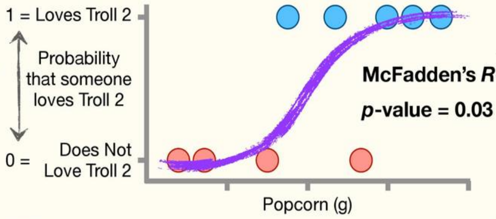
🔴 = Does Not Love Troll 2



In this example, we measured the amount of Popcorn a bunch of people ate (in grams), which is ***continuous***, and whether they **Love Troll 2** or **Do Not Love Troll 2**, which is ***discrete***.

The goal is to make a ***classifier*** that uses the amount of Popcorn someone eats to classify whether or not they love Troll 2.

② **A Solution: Logistic Regression**, which probably should have been named **Logistic Classification** since it's used to classify things, fits a **squiggle** to data that tells us the predicted probability (between **0** and **1**) for ***discrete*** variables, like whether or not someone loves Troll 2.



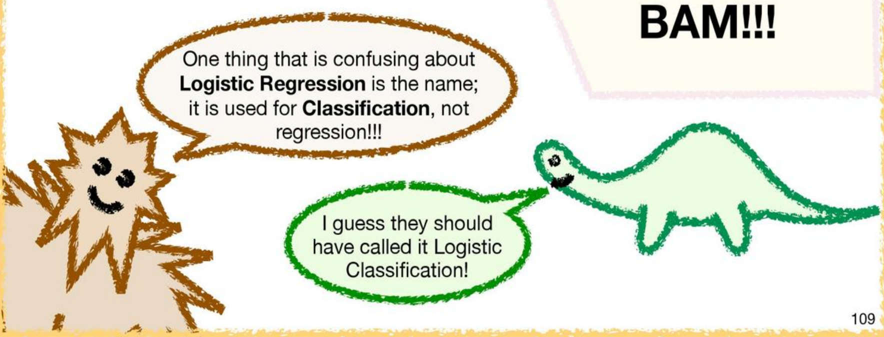
McFadden's R^2 = 0.4

***p*-value = 0.03**

Like **Linear Regression**, **Logistic Regression** has metrics that are similar to ***R^2*** to give us a sense of how accurate our predictions will be, and it also calculates ***p*-values**.

Even better, all of the tricks we can do with **Linear Models** also apply to **Logistic Regression**, so we can mix and match ***discrete*** and ***continuous*** features to make ***discrete*** classifications.

BAM!!!



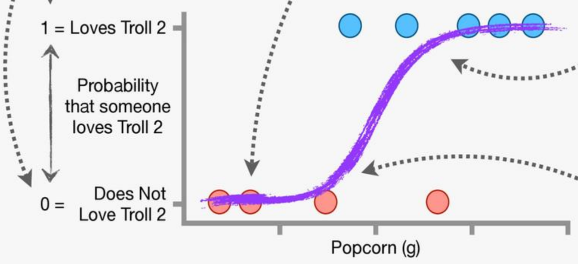
Logistic Regression: Main Ideas Part 2

3 The y-axis on a **Logistic Regression** graph represents probability and goes from **0** to **1**. In this case, it's the probability that someone loves Troll 2.

The colored dots are the **Training Data**, which we used to fit the **squiggle**. The data consist of **4** people who **Do Not Love Troll 2** and **5** people who **Love Troll 2**.

The **squiggle** tells us the predicted probability that someone loves Troll 2, which means that when the **squiggle** is close to the top of the graph, there's a high probability (a probability close to **1**) that someone will love Troll 2…

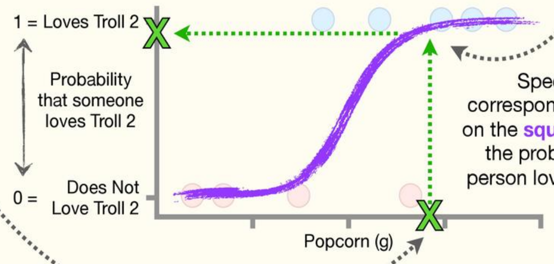
…and when the **squiggle** is close to the bottom of the graph, there's a low probability (a probability close to **0**) that someone will love Troll 2.



4 If someone new comes along and tells us that they ate this much Popcorn…

…then the **squiggle** tells us that there's a relatively high probability that that person will love Troll 2.

Specifically, the corresponding y-axis value on the **squiggle** tells us that the probability that this person loves Troll 2 is **0.96**.



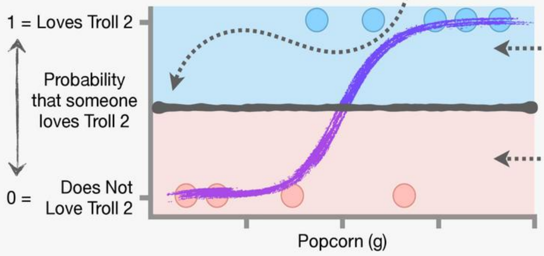
🔵 = Loves Troll 2

🔴 = Does Not Love Troll 2

DOUBLE BAM!!!

Logistic Regression: Main Ideas Part 3

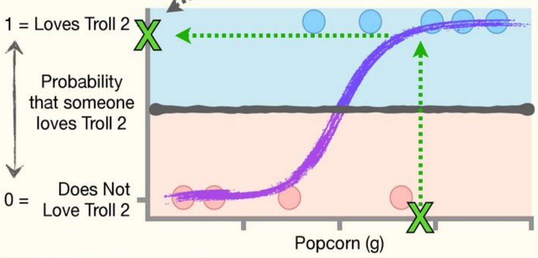
5 Now that we know the *probability* that this person will love Troll 2, we can *classify* them as someone who either **Loves Troll 2** or **Does Not Love Troll 2**. Usually the threshold for classification is **0.5**…



…and in this case, that means anyone with a probability of loving Troll 2 > **0.5** will be classified as someone who **Loves Troll 2**…

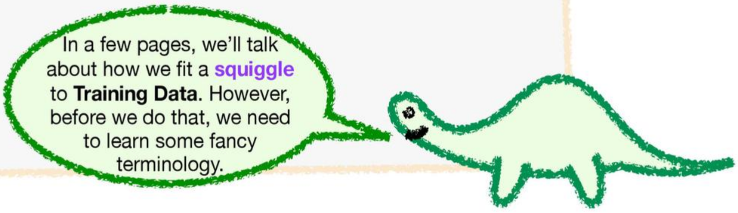
….and anyone with a probability ≤ **0.5** will be classified as someone who **Does Not Love Troll 2**.

6 Thus, in this example, since **0.96 > 0.5**, we'll classify this person as someone who **Loves Troll 2**.

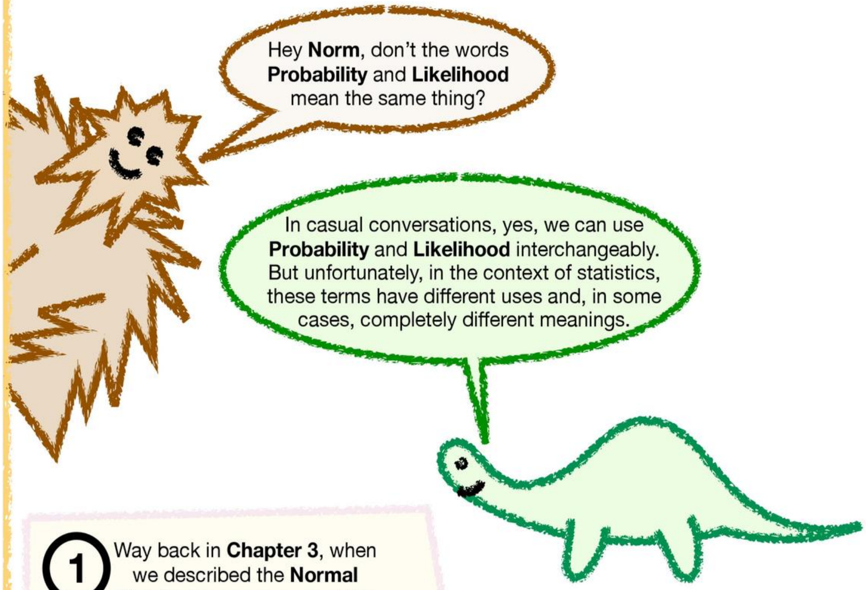


7 One last thing before we go: In this example, the classification threshold was **50%**. However, when we talk about **Receiver Operator Curves** (**ROCs**) in **Chapter 8**, we'll see examples that use different classification thresholds. So get excited!!!

TRIPLE BAM!!!

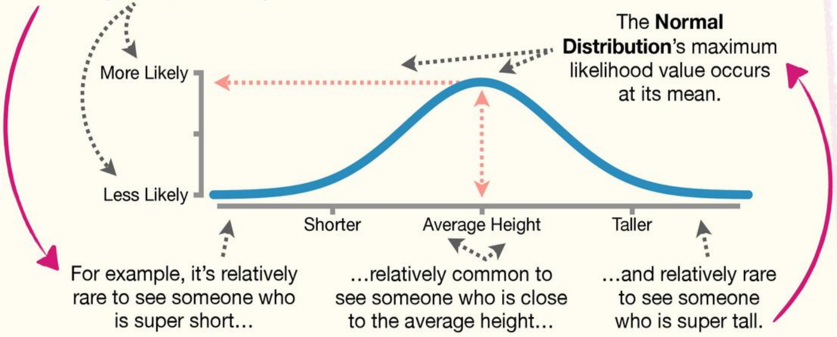


Terminology Alert!!! Probability vs. Likelihood: Part 1



① Way back in **Chapter 3**, when we described the **Normal Distribution**, we saw that the y-axis represented **Likelihood**.

In this specific example, the y-axis represents the **Likelihood** of observing any specific Height.



The **Normal Distribution**'s maximum likelihood value occurs at its mean.

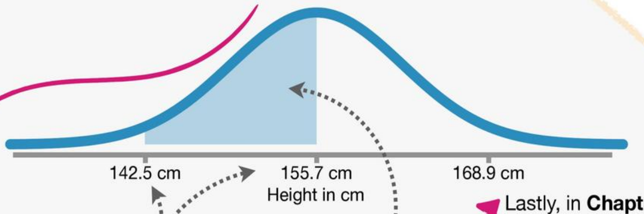
For example, it's relatively rare to see someone who is super short…

…relatively common to see someone who is close to the average height…

…and relatively rare to see someone who is super tall.

Terminology Alert!!! Probability vs. Likelihood: Part 2

2 In contrast, later in **Chapter 3**, we saw that **Probabilities** are derived from a **Normal Distribution** by calculating the **area under the curve** between two points.



For example, given this **Normal Distribution** with **mean = 155.7** and **standard deviation = 6.6**, the probability of getting a measurement between **142.5** and **155.7** cm…

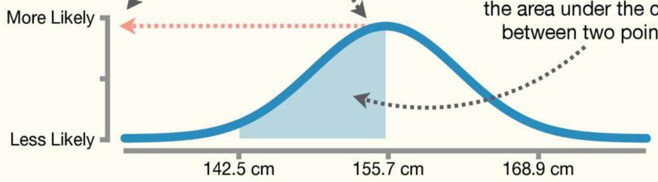
…is equal to this area under the curve, which, in this example, is **0.48**. So, the probability is **0.48** that we will measure someone in this range.

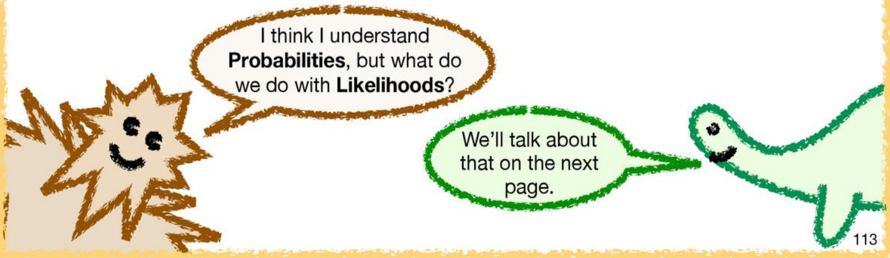
Lastly, in **Chapter 3** we mentioned that when we use a **Continuous Distribution**, like the **Normal Distribution**, the probability of getting any specific measurement is always **0** because the area of something with no width is **0**.

3 So, in the case of the **Normal Distribution**…

…**Likelihoods** are the y-axis coordinates for specific points on the curve…

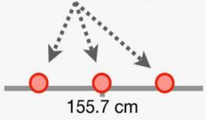
…whereas **Probabilities** are the area under the curve between two points.



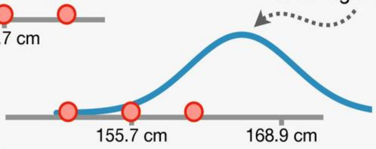


Terminology Alert!!! Probability vs. Likelihood: Part 3

4 **Likelihoods** are often used to evaluate how well a statistical distribution fits a dataset. For example, imagine we collected these three Height measurements…



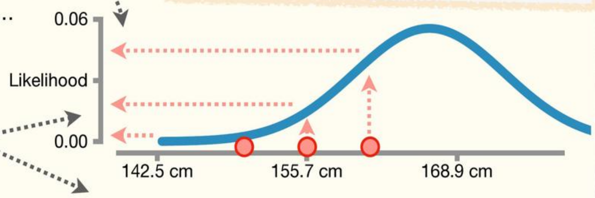
…and we wanted to compare the fit of a **Normal Curve** that has its peak to the *right* of the data…



…to the fit of a **Normal Curve** that is *centered* over the data.

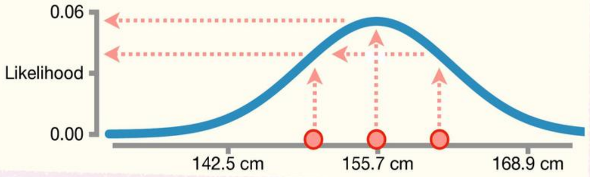
5 First, we determine the **Likelihoods**, the y-axis coordinates on the curves, for each data point…

…and, by eye, we can see that, overall, the **Likelihoods** are larger when we center the curve over the data.



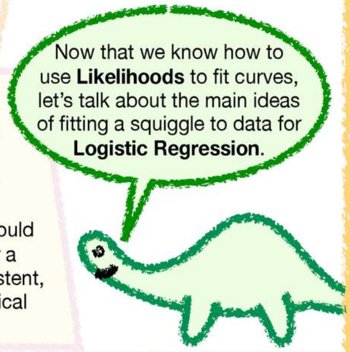
And larger likelihoods suggest that the centered curve fits the data better than the one shifted to the right.

BAM!!!



6 **NOTE:** When we try to fit a **Normal Curve** to data, we can't use **Probabilities** because, as we saw in **Chapter 3**, the probability for a specific point under a **Normal Curve** is always **0**.

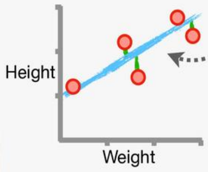
7 Lastly, as we just saw with the **Normal Distribution**, **Probabilities** and **Likelihoods** can be different. However, as we'll soon see, this is not always the case. In other words, sometimes **Probabilities** and **Likelihoods** are the same. When **Probabilities** and **Likelihoods** are the same, we could use either **Probabilities** or **Likelihoods** to fit a curve or a squiggle to data. However, to make the terminology consistent, when we're fitting a curve or squiggle to data in a statistical context, we almost always use **Likelihoods**.

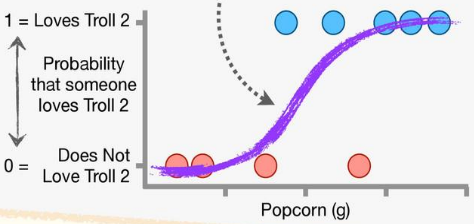


Fitting A Squiggle To Data: Main Ideas Part 1

1 When we use **Linear Regression**, we fit a **line** to the data by minimizing the **Sum of the Squared Residuals** (**SSR**).

In contrast, **Logistic Regression** swaps out the **Residuals** for **Likelihoods** (y-axis coordinates) and fits a **squiggle** that represents the **Maximum Likelihood**.





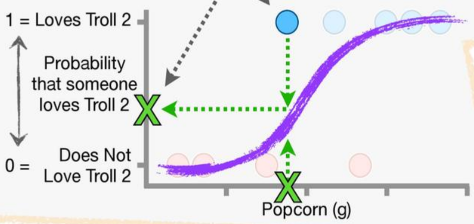
2 However, because we have two classes of people, one that **Loves Troll 2** and one that **Does Not Love Troll 2**, there are two ways to calculate **Likelihoods**, one for each class.

● = Loves Troll 2

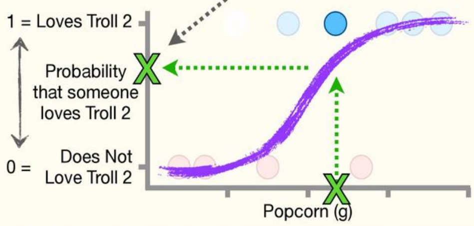
● = Does Not Love Troll 2

3 For example, to calculate the **Likelihood** for this person, who **Loves Troll 2**, we use the **squiggle** to find the y-axis coordinate that corresponds to the amount of Popcorn they ate…

…and this y-axis coordinate, **0.4**, is both the predicted probability that they **Love Troll 2** and the **Likelihood**.



4 Likewise, the **Likelihood** for this person, who also **Loves Troll 2**, is the y-axis coordinate for the **squiggle**, **0.6**, that corresponds to the amount of Popcorn they ate.

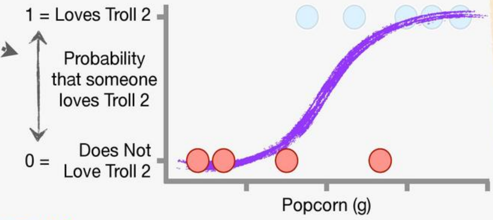


Fitting A Squiggle To Data: Main Ideas Part 2

5 In contrast, calculating the **Likelihoods** is different for the people who **Do Not Love Troll 2** because the y-axis is the probability that they **Loves Troll 2**.

🔵 = Loves Troll 2

🔴 = Does Not Love Troll 2



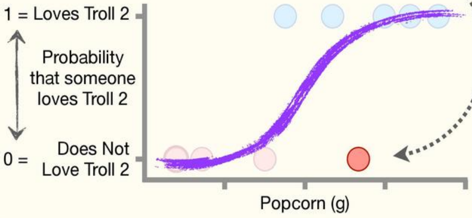
The good news is, because someone either loves Troll 2 or they don't, the probability that someone does not love Troll 2 is just **1** minus the probability that they love Troll 2…

p(**Does Not Love Troll 2**) = 1 - *p*(**Loves Troll 2**)

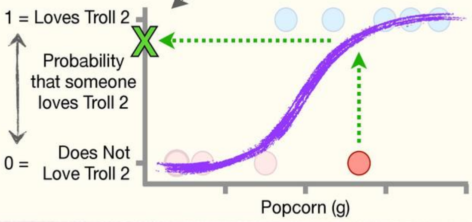
…and since the y-axis is both probability and likelihood, we can calculate the **Likelihoods** with this equation.

L(**Does Not Love Troll 2**) = 1 - *L*(**Loves Troll 2**)

6 For example, to calculate the **Likelihood** for this person, who **Does Not Love Troll 2**…



…we first calculate the **Likelihood** that they **Love Troll 2**, **0.8**…



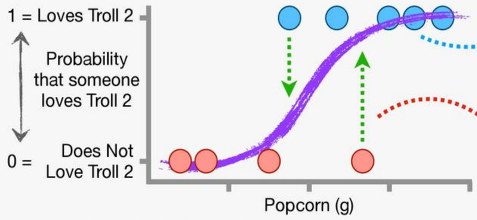
…and then use that value to calculate the **Likelihood** that they **Do Not Love Troll 2** = **1 - 0.8 = 0.2**.

Bam!

Fitting A Squiggle To Data: Main Ideas Part 3

7 Now that we know how to calculate the **Likelihoods** for people who **Love Troll 2** and people who **Do Not Love Troll 2**, we can calculate the **Likelihood** for the entire **squiggle** by multiplying the individual **Likelihoods** together…

…and when we do the math, we get **0.02**.



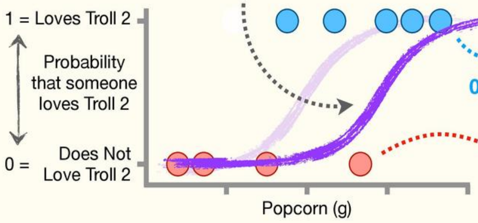
0.4 x **0.6** x **0.8** x **0.9** x **0.9**

x **0.9** x **0.9** x **0.7** x **0.2** = **0.02**

VS.

8 Now we calculate the **Likelihood** for a different **squiggle**…

…and compare the total **Likelihoods** for both **squiggles**.



0.1 x **0.2** x **0.6** x **0.7** x **0.9**

x **0.9** x **0.9** x **0.9** x **0.8** = **0.004**

9 The goal is to find the **squiggle** with the **Maximum Likelihood**.

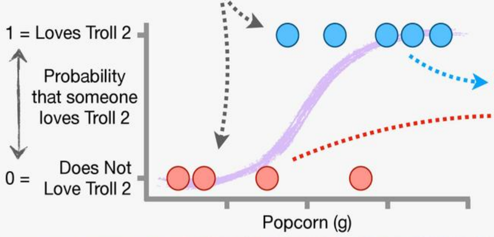
In practice, we usually find the optimal **squiggle** using **Gradient Descent**.

TRIPLE BAM!!!

Fitting A Squiggle To Data: Details

① The example we've used so far has a relatively small **Training Dataset** of only **9** points total…

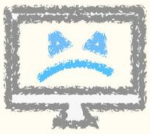
…so when we multiplied the **9 Likelihoods** together, it was easy, and we got **0.02**.



$0.4 \times 0.6 \times 0.8 \times 0.9 \times 0.9$

$\times\ 0.9 \times 0.9 \times 0.7 \times 0.2 = \mathbf{0.02}$

② However, if the **Training Dataset** was much larger, then we might run into a computational problem, called **Underflow**, that happens when you try to multiply a lot of small numbers between **0** and **1**.



Technically, **Underflow** happens when a mathematical operation, like multiplication, results in a number that's smaller than the computer is capable of storing.

Underflow can result in errors, which are bad, or it can result in weird, unpredictable results, which are worse.

If you'd like to learn more details about **Logistic Regression**, scan, click, or tap this QR code to check out the **'Quests** on YouTube!!!

③ A very common way to avoid **Underflow** errors is to just take the **log** (usually the **natural log**, or **log base *e***), which turns the multiplication into addition…

$$\log(0.4 \times 0.6 \times 0.8 \times 0.9 \times 0.9 \times 0.9 \times 0.9 \times 0.7 \times 0.2)$$

$$= \log(0.4) + \log(0.6) + \log(0.8) + \log(0.9) + \log(0.9)$$

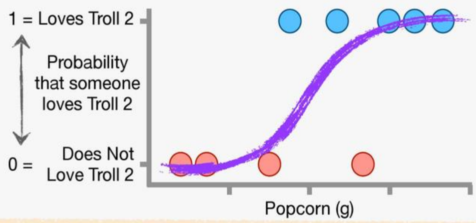
$$+ \log(0.9) + \log(0.9) + \log(0.7) + \log(0.2)$$

$$= -4.0$$

…and, ultimately, turns a number that was relatively close to **0**, like **0.02**, into a number relatively far from **0**, **-4.0**.

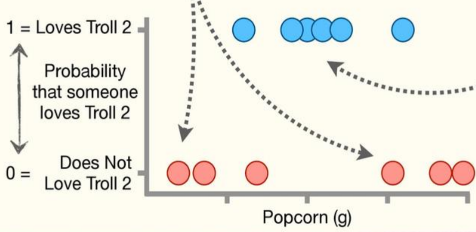
Logistic Regression: Weaknesses

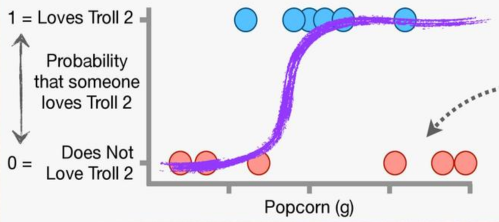
① When we use **Logistic Regression**, we assume that an **s-shaped squiggle** (or, if we have more than one independent variable, an s-shaped surface) will fit the data well. In other words, we assume that there's a relatively straightforward relationship between the Popcorn and Loves Troll 2 variables: if someone eats very little Popcorn, then there's a relatively low probability that they love Troll 2, and if someone eats a lot of Popcorn, then there's a relatively high probability that they love Troll 2.



② In contrast, if our data had people who ate a little or a lot of Popcorn and both groups do not love Troll 2…

…and people who ate an intermediate amount do, then we would violate the assumption that **Logistic Regression** makes about the data.





③ And if we used **Logistic Regression** to fit an **s-shaped squiggle** to the data, we would get a horrible model that misclassified everyone who ate a lot of Popcorn as someone who loves Troll 2.

④ Thus, one of the limitations of **Logistic Regression** is that it assumes that we can fit an **s-shaped squiggle** to the data. If that's not a valid assumption, then we need a **Decision Tree** (**Chapter 10**), or a **Support Vector Machine** (**Chapter 11**), or a **Neural Network** (**Chapter 12**), or some other method that can handle more complicated relationships among the data.

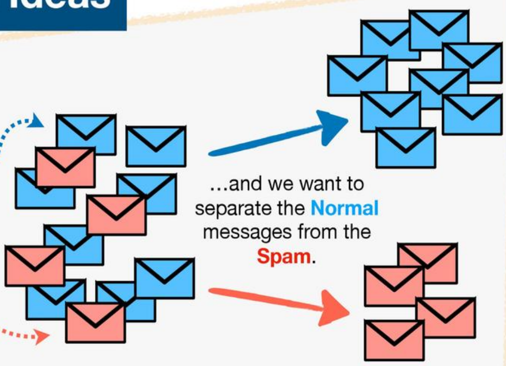
Chapter 07

Naive Bayes!!!

Naive Bayes: Main Ideas

1 **The Problem:** We start with a jumble of messages: some of them are Normal messages from friends and family…

…and some of them are Spam, unwanted messages that are usually scams or unsolicited advertisements…



…and we want to separate the Normal messages from the Spam.

2 **A Solution:** We can use a **Naive Bayes** classifier, one of the simplest, but surprisingly effective machine learning methods.

3 Let's say we get a message that says **Dear Friend**, and we want to classify it as either Normal or Spam. We'll need two different equations: one for Normal and one for Spam.

Dear Friend

4 First we multiply the **Prior Probability**, which is an initial guess, that the message is Normal…

…by the probabilities of seeing the words **Dear** and **Friend**, *given* (**NOTE:** the vertical bar, |, stands for **given**) that the message is Normal.

$$p(\,\mathrm{N}\,) \times p(\,\textbf{Dear} \mid \mathrm{N}\,) \times p(\,\textbf{Friend} \mid \mathrm{N}\,)$$

$$p(\,\mathrm{S}\,) \times p(\,\textbf{Dear} \mid \mathrm{S}\,) \times p(\,\textbf{Friend} \mid \mathrm{S}\,)$$

5 Then we compare that to the **Prior Probability**, another initial guess, that the message is Spam…

…multiplied by the probabilities of seeing the words **Dear** and **Friend**, given that the message is Spam.

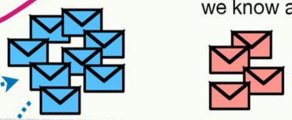
6 Whichever equation gives us the larger value, which we'll call a score, tells us the final classification.

Multinomial Naive Bayes: Details Part 1

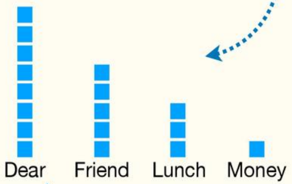
① There are several types of **Naive Bayes** algorithms, but the most commonly used version is called **Multinomial Naive Bayes**.

② We start with **Training Data**: In this case we have **8** messages that we know are **Normal**…

…and **4** messages that we know are **Spam**.



③ Then we make a histogram for all of the words in the **Normal** messages…



④ …and a histogram for all of the words in the **Spam** messages.

⑤ Now we calculate the probabilities of seeing each word given that they came from **Normal** messages.

For example, the probability that we see the word **Dear** *given* (remember, the vertical bar, |, stands for **given**) that we see it in **Normal** (**N**) messages…

$$p(\textbf{Dear} \mid \textbf{N}) = \frac{8}{17} \boxed{= 0.47}$$

…is **8**, the number of times **Dear** occurs in **Normal** messages, divided by the total number of words in the **Normal** messages, **17**…

…and we get **0.47**.

Then we do the same thing for all the other words in the **Normal** messages.

$p(\textbf{Dear} \mid \textbf{N}) = 0.47$

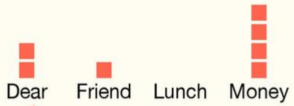
$p(\textbf{Friend} \mid \textbf{N}) = 0.29$

$p(\textbf{Lunch} \mid \textbf{N}) = 0.18$

$p(\textbf{Money} \mid \textbf{N}) = 0.06$

Multinomial Naive Bayes: Details Part 2

⑥ Then we calculate the probabilities of seeing each word given that they came from **Spam** messages.



For example, the probability that we see the word **Dear** given that we see it in **Spam** (**S**) messages…

$$p(\textbf{Dear} \mid \textbf{S}) = \frac{2}{7} \boxed{= 0.29}$$

…is **2**, the number of times **Dear** occurs in **Spam** messages divided by the total number of words in the **Spam** messages, **7**…

…and we get **0.29**.

Then we do the same thing for all the other words in the **Spam** messages.

$\boxed{p(\textbf{Dear} \mid \textbf{S}) = 0.29}$

$p(\textbf{Friend} \mid \textbf{S}) = 0.14$

$p(\textbf{Lunch} \mid \textbf{S}) = 0.00$

$p(\textbf{Money} \mid \textbf{S}) = 0.57$

⑦ Now we calculate **Prior Probabilities.** In this context, a **Prior Probability** is simply a guess that we make without looking at the words in the message about whether or not a message is **Normal** or **Spam**.

NOTE: The **Prior Probabilities** can be any pair of probabilities we want, but we usually derive them from the **Training Data**.

⑧ For example, because **8** of the **12** messages are **Normal**, we let the **Prior Probability** for **Normal** messages, $p(\text{N})$, be **8/12 = 0.67**…

$$p(\textbf{N}) = \frac{\text{\# of Normal Messages}}{\text{Total \# of Messages}} = \frac{8}{12} \boxed{= 0.67}$$

$$p(\textbf{S}) = \frac{\text{\# of Spam Messages}}{\text{Total \# of Messages}} = \frac{4}{12} \boxed{= 0.33}$$

⑨ …and because **4** of the **12** messages are **Spam**, we let the **Prior Probability** for **Spam** messages, $p(\text{S})$, be **4/12 = 0.33**.

Multinomial Naive Bayes: Details Part 3

10 Now that we have the **Prior Probability** for **Normal** messages…

…and the probabilities for each word occurring in **Normal** messages…

…we can calculate the overall score that the message, **Dear Friend**, is **Normal**…

$p(\text{N}) = 0.67$

$p(\textbf{Dear} \mid \text{N}) = 0.47$

$p(\textbf{Friend} \mid \text{N}) = 0.29$

$p(\textbf{Lunch} \mid \text{N}) = 0.18$

$p(\textbf{Money} \mid \text{N}) = 0.06$

$$p(\text{N}) \times p(\textbf{Dear} \mid \text{N}) \times p(\textbf{Friend} \mid \text{N}) = 0.67 \times 0.47 \times 0.29 \boxed{= 0.09}$$

11 …by multiplying the **Prior Probability** that the message is **Normal**…

…by the probabilities of seeing the words **Dear** and **Friend** in **Normal** messages…

…and when we do the math, we get **0.09**.

12 Likewise, using the **Prior Probability** for **Spam** messages…

…and the probabilities for each word occurring in **Spam** messages…

…we can calculate the overall score that the message, **Dear Friend**, is **Spam**, and when we do the math, we get **0.01**.

$p(\text{S}) = 0.33$

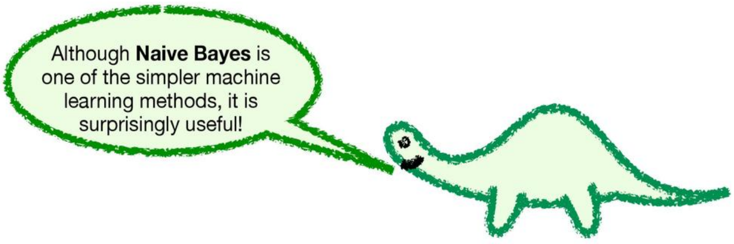
$p(\textbf{Dear} \mid \text{S}) = 0.29$

$p(\textbf{Friend} \mid \text{S}) = 0.14$

$p(\textbf{Lunch} \mid \text{S}) = 0.00$

$p(\textbf{Money} \mid \text{S}) = 0.57$

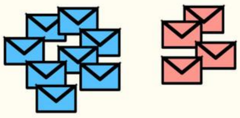
$$p(\text{S}) \times p(\textbf{Dear} \mid \text{S}) \times p(\textbf{Friend} \mid \text{S}) = 0.33 \times 0.29 \times 0.14 \boxed{= 0.01}$$



Multinomial Naive Bayes: Details Part 4

13 Now, remember the goal was to classify the message **Dear Friend** as either **Normal** or **Spam**…

…and we started with **Training Data**, **8 Normal** and **4 Spam** messages…

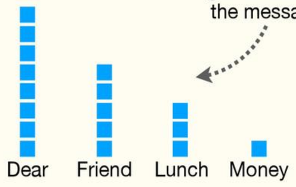


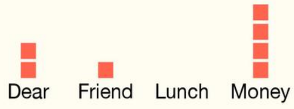
…and we calculated **Prior Probabilities**, which are just guesses that we make without looking at the contents of a message, that a message is either **Normal** or **Spam**…

$$p(\textbf{N}) = \frac{\text{\# of Normal Messages}}{\text{Total \# of Messages}} = 0.67$$

$$p(\textbf{S}) = \frac{\text{\# of Spam Messages}}{\text{Total \# of Messages}} = 0.33$$

…and we created histograms of the words in the messages…





…and we used the histograms to calculate probabilities…

$p($ **Dear** $|$ **N** $) = 0.47$

$p($ **Friend** $|$ **N** $) = 0.29$

$p($ **Lunch** $|$ **N** $) = 0.18$

$p($ **Money** $|$ **N** $) = 0.06$

$p($ **Dear** $|$ **S** $) = 0.29$

$p($ **Friend** $|$ **S** $) = 0.14$

$p($ **Lunch** $|$ **S** $) = 0.00$

$p($ **Money** $|$ **S** $) = 0.57$

…then, using the **Prior Probabilities** and the probabilities for each word, given that it came from either a **Normal** message or **Spam**, we calculated scores for **Dear Friend**.…

$p(\textbf{N}) \times p(\textbf{Dear} \mid \textbf{N}) \times p(\textbf{Friend} \mid \textbf{N}) = 0.67 \times 0.47 \times 0.29 = 0.09$

$p(\textbf{S}) \times p(\textbf{Dear} \mid \textbf{S}) \times p(\textbf{Friend} \mid \textbf{S}) = 0.33 \times 0.29 \times 0.14 = 0.01$

…and now we can finally classify **Dear Friend**!!! Because the score for **Normal** (**0.09**) is greater than **Spam** (**0.01**), we classify **Dear Friend** as a **Normal** message.

Dear Friend ✉

BAM!!!

Multinomial Naive Bayes: Details Part 5

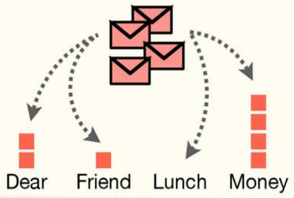
⑭ The reason **Naive Bayes** is ***naive***, is that it's simple: it treats all of the words independently, and this means it ignores word order and phrasing.

So, both **Dear Friend** and **Friend Dear** get the same score, **0.09**.

p(**N**) x *p*(**Friend** | **N**) x *p*(**Dear** | **N**) = 0.67 x 0.29 x 0.47 = 0.09

p(**N**) x *p*(**Dear** | **N**) x *p*(**Friend** | **N**) = 0.67 x 0.47 x 0.29 = 0.09

⑮ Missing data can, in theory, be a problem. Remember, in the last example, the word **Lunch** didn't occur in any of the **Spam**…



…and that means that the probability of seeing **Lunch** in **Spam** is **0**…

p(**Dear** | **S**) = 0.29

p(**Friend** | **S**) = 0.14

p(**Lunch** | **S**) = 0.00

p(**Money** | **S**) = 0.57

…and this means that any message that contains the word **Lunch** will be classified as **Normal** because the score for **Spam** will always be **0**.

⑯ For example, if we see this message… → **Money Money Money Money Lunch**

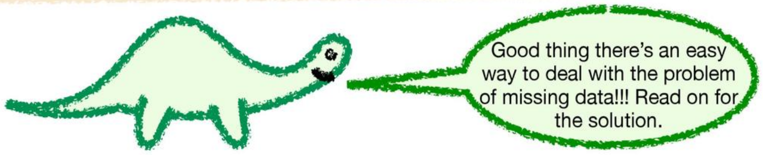
…which looks like it could be **Spam** because it has the word **Money** in it a bunch of times.

However, when we calculate this score…

…because *p*(**Lunch** | **S**) = **0**, and anything times **0** is **0**, we get **0**.

p(**S**) x *p*(**Money** | **S**) x *p*(**Money** | **S**) x *p*(**Money** | **S**) x *p*(**Money** | **S**) x *p*(**Lunch** | **S**)

= 0.33 x 0.57 x 0.57 x 0.57 x 0.57 x 0 = 0



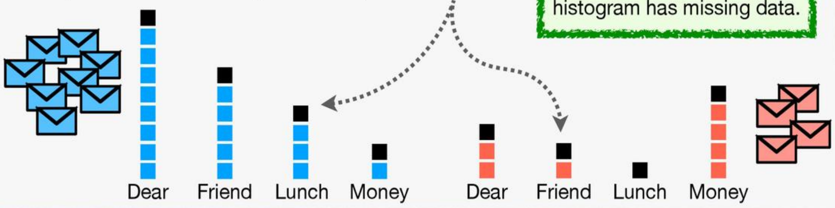
Multinomial Naive Bayes: Dealing With Missing Data

1 Missing data can pose a real problem for **Naive Bayes** or anything else based on histograms. As we saw in **Chapter 3**, we can easily have missing data if the **Training Dataset** is not large enough.

So, **Naive Bayes** eliminates the problem of missing data by adding something called a ***pseudocount*** to each word.

2 A ***pseudocount*** is just an extra value added to each word, and usually adding ***pseudocounts*** means adding **1** count to each word. Here the ***pseudocounts*** are represented by **black boxes**.

NOTE: ***pseudocounts*** are added to every word in *both* histograms, even if only one histogram has missing data.



3 After we add the ***pseudocounts*** to the histograms, we calculate the probabilities just like before, only this time we include the ***pseudocounts*** in the calculations.

$$p(\textbf{Dear} \mid \textbf{N}) = \frac{8 + 1}{17 + 4} \boxed{= 0.43}$$

$p(\textbf{Dear} \mid \textbf{N}) = 0.43$ $p(\textbf{Dear} \mid \textbf{S}) = 0.27$

$p(\textbf{Friend} \mid \textbf{N}) = 0.29$ $p(\textbf{Friend} \mid \textbf{S}) = 0.18$

$p(\textbf{Lunch} \mid \textbf{N}) = 0.19$ $p(\textbf{Lunch} \mid \textbf{S}) = 0.09$

$p(\textbf{Money} \mid \textbf{N}) = 0.10$ $p(\textbf{Money} \mid \textbf{S}) = 0.45$

4 Now the scores for this message are… **Money Money Money Money Lunch**

$$p(\textbf{N}) \times p(\textbf{Money} \mid \textbf{N})^4 \times p(\textbf{Lunch} \mid \textbf{N}) = 0.67 \times 0.10^4 \times 0.19 = 0.00001$$

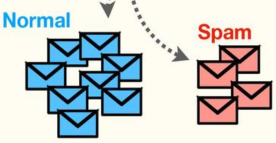
$$p(\textbf{S}) \times p(\textbf{Money} \mid \textbf{S})^4 \times p(\textbf{Lunch} \mid \textbf{S}) = 0.33 \times 0.45^4 \times 0.09 = 0.00122$$

5 Because **Money Money Money Money Lunch** has a higher score for **Spam** (**0.00122**) than **Normal** (**0.00001**), we classify it as **Spam**.

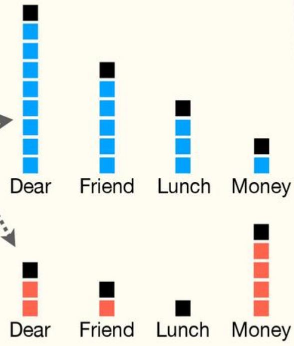
SPAM!!!

Multinomial Naive Bayes vs. Gaussian Naive Bayes

① So far, we've shown how **Naive Bayes** works with ***discrete*** data, like individual words in a message…



…by building histograms…



…and using the histograms to calculate probabilities that are used to make classifications.

p(**Dear** | **N**) = 0.43
p(**Friend** | **N**) = 0.29
p(**Lunch** | **N**) = 0.19
p(**Money** | **N**) = 0.10

p(**Dear** | **S**) = 0.27
p(**Friend** | **S**) = 0.18
p(**Lunch** | **S**) = 0.09
p(**Money** | **S**) = 0.45

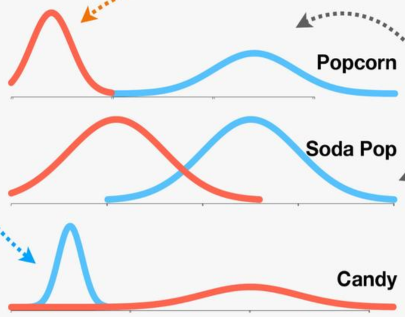
② In contrast, when we have ***continuous*** data, like these measurements for Popcorn, Soda Pop, and Candy consumption taken from **4** people who **Love Troll 2**…

…and from **3** people who **Do Not Love Troll 2**…

Popcorn (grams)	Soda Pop (ml)	Candy (grams)
24.3	750.7	0.2
28.2	533.2	50.5
etc.	etc.	etc.

Popcorn (grams)	Soda Pop (ml)	Candy (grams)
2.1	120.5	90.7
4.8	110.9	102.3
etc.	etc.	etc.

…we can calculate the **means** and **standard deviations** of each column to draw **Gaussian** (**Normal**) curves…



…and use these curves to make classifications. Read on to learn how!

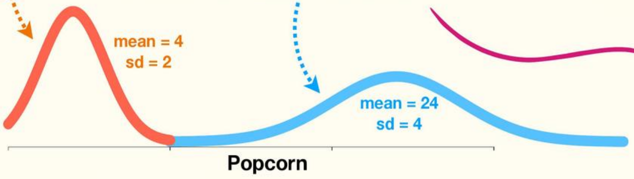
Gaussian Naive Bayes: Details Part 1

① First we create **Gaussian** (**Normal**) curves for each **Feature** (each column in the **Training Data** that we're using to make predictions).

Popcorn (grams)	Soda Pop (ml)	Candy (grams)
24.3	750.7	0.2
28.2	533.2	50.5
etc.	etc.	etc.
2.1	120.5	90.7
4.8	110.9	102.3
etc.	etc.	etc.

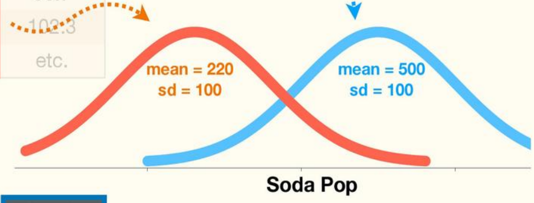
Starting with Popcorn, for the people who **Do Not Love Troll 2**, we calculate their mean, **4**, and standard deviation (**sd**), **2**, and then use those values to draw a **Gaussian** curve.

Then we draw a Gaussian curve for the people who **Love Troll 2** using their mean, **24**, and their standard deviation, **4**.



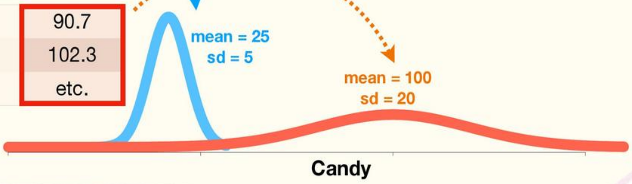
Popcorn (grams)	Soda Pop (ml)	Candy (grams)
24.3	750.7	0.2
28.2	533.2	50.5
etc.	etc.	etc.
2.1	120.5	90.7
4.8	110.9	102.3
etc.	etc.	etc.

Likewise, we draw curves for Soda Pop…



Popcorn (grams)	Soda Pop (ml)	Candy (grams)
24.3	750.7	0.2
28.2	533.2	50.5
etc.	etc.	etc.
2.1	120.5	90.7
4.8	110.9	102.3
etc.	etc.	etc.

…and for Candy.



Gaussian Naive Bayes: Details Part 2

② Now we calculate the **Prior Probability** that someone Loves Troll 2.

Just like for **Multinomial Naive Bayes**, this **Prior Probability** is just a guess and can be any probability we want. However, we usually estimate it from the number of people in the **Training Data**.

$$p(\text{Loves Troll 2}) = \frac{\text{\# of People Who Love Troll 2}}{\text{Total \# of People}} = \frac{4}{4+3} = 0.6$$

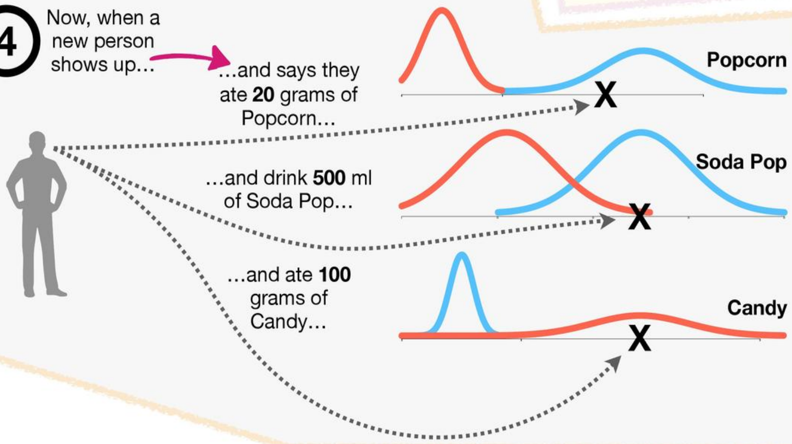
In this case, the **Training Data** came from 4 people who Love Troll 2 and 3 people who Do Not Love Troll 2.

③ Then we calculate the **Prior Probability** that someone Does Not Love Troll 2...

$$p(\text{Does Not Love Troll 2}) = \frac{\text{\# of People Who Do Not Love Troll 2}}{\text{Total \# of People}}$$

$$= \frac{3}{4+3} = 0.4$$

④ Now, when a new person shows up...



...and says they ate **20** grams of Popcorn...

...and drink **500** ml of Soda Pop...

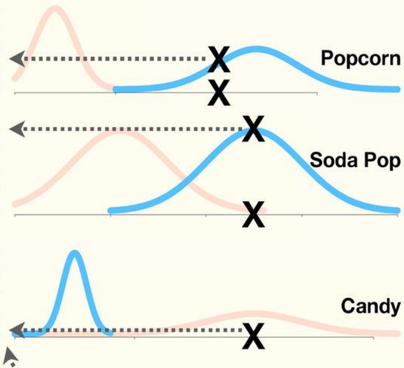
...and ate **100** grams of Candy...

Gaussian Naive Bayes: Details Part 3

5 …we calculate the score for Loves Troll 2 by multiplying the **Prior Probability** that they Love Troll 2…

p(Loves Troll 2)
x *L*(**Popcorn** = 20 | Loves)
x *L*(**Soda Pop** = 500 | Loves)
x *L*(**Candy** = 100 | Loves)

…by the **Likelihoods**, the y-axis coordinates, that correspond to **20** grams of Popcorn, **500** ml of Soda Pop, and **100** grams of Candy, given that they Love Troll 2.



6 **NOTE:** When we plug in the actual numbers…

…we end up plugging in a tiny number for the **Likelihood** of **Candy**, because the **y-axis coordinate** is super close to **0**…

p(Loves Troll 2) → 0.6
x *L*(**Popcorn** = 20 | Loves) → x 0.06
x *L*(**Soda Pop** = 500 | Loves) → x 0.004
x *L*(**Candy** = 100 | Loves) → x 0.000000000…001

…and computers can have trouble doing multiplication with numbers very close to **0**. This problem is called **Underflow**.

7 To avoid problems associated with numbers close to **0**, we take the **log** (usually the **natural log**, or **log base *e***), which turns the multiplication into addition…

log(0.6 x 0.06 x 0.004 x a tiny number) = **log**(0.6) + **log**(0.06) + **log**(0.004) + **log**(a tiny number)

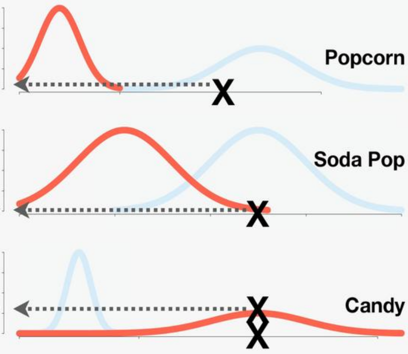
…and turns numbers close to **0** into numbers far from **0**…

= -0.51 + -2.8 + -5.52 + -115 = -124

…and when we do the math, the score for Loves Troll 2 = **-124**.

Gaussian Naive Bayes: Details Part 4

⑧ Likewise, we calculate the score for **Does Not Love Troll 2** by multiplying the **Prior Probability**…



p(**No Love**)
× *L*(**Popcorn** = 20 | **No Love**)
× *L*(**Soda Pop** = 500 | **No Love**)
× *L*(**Candy** = 100 | **No Love**)

…by the **Likelihoods** from the **Does Not Love Troll 2** distributions for Popcorn, Soda Pop, and Candy. But before we do the math, we first take the **log**…

log(*p*(**No Love**))
+ **log**(*L*(**Popcorn** = 20 | **No Love**))
+ **log**(*L*(**Soda Pop** = 500 | **No Love**))
+ **log**(*L*(**Candy** = 100 | **No Love**))

…then we plug in the numbers, do the math, and get **-48**.

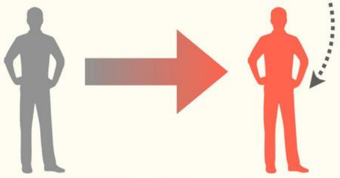
log(0.4) + **log**(a tiny number) + **log**(0.00008) + **log**(0.02)

= -0.92 + -33.61 + -9.44 + -3.91 = -48

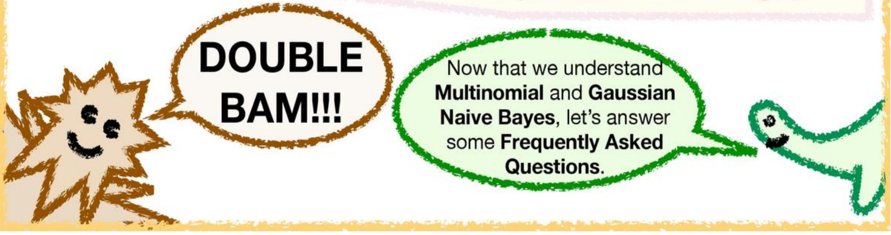
⑨ Lastly, because the score for **Does Not Love Troll 2** (**-48**) is greater than the score for **Loves Troll 2** (**-124**), we classify this person…

Log(**Loves Troll 2 Score**) = -124

Log(**Does Not Love Troll 2 Score**) = -48



…as someone who **Does Not Love Troll 2**.



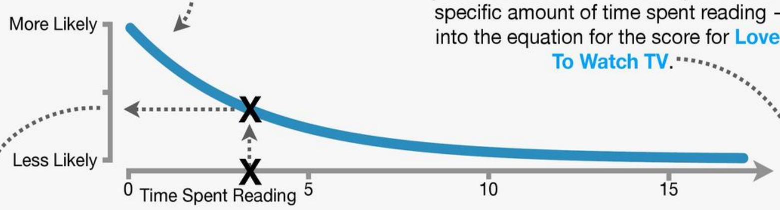
Naive Bayes: FAQ Part 1

If my continuous data are not *Gaussian*, can I still use *Gaussian Naive Bayes*?

Although the **Gaussian** (**Normal**) distribution is the most commonly used, we can use *any* statistical distribution in combination with **Naive Bayes**.

For example, if this **Exponential** distribution represents the amount of time spent reading books among people who **Love to Watch TV**…

…then we can plug the **Likelihood** — the y-axis coordinate that corresponds to the specific amount of time spent reading — into the equation for the score for **Loves To Watch TV**.



p(**Loves TV Prior Probability**)

× *L*(Reading = 3 | **Loves TV**)

However, there's one problem with using an **Exponential** distribution…

…we can't call it **Gaussian Naive Bayes** anymore!!! Now we'll call it **Exponential Naive Bayes**.

Silly Bam. :)

Naive Bayes: FAQ Part 2

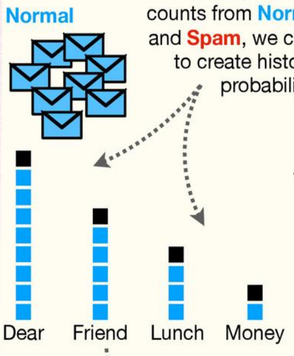
What if some of my data are *discrete* and some of my data are *continuous*? Can I still use Naive Bayes?

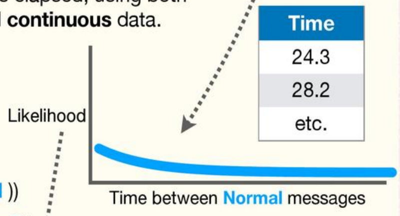
Yes! We can combine the histogram approach from **Multinomial Naive Bayes** with the distribution approach from **Gaussian Naive Bayes**.

For example, if we had word counts from **Normal** messages and **Spam**, we could use them to create histograms and probabilities…

…and combine those with **Likelihoods** from **Exponential** distributions that represent the amount of time that elapses between receiving **Normal** messages and **Spam**…

…to calculate **Normal** and **Spam** scores for the message **Dear Friend**, received after **25** seconds elapsed, using both **discrete** and **continuous** data.





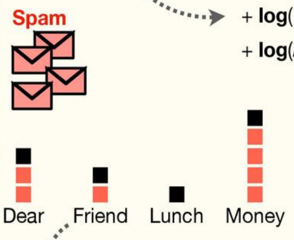
Time
24.3
28.2
etc.

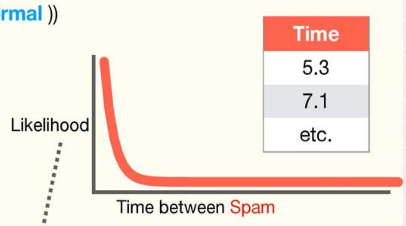
log(*p*(**Normal**))

+ **log**(*p*(**Dear** | **Normal**))

+ **log**(*p*(**Friend** | **Normal**))

+ **log**(*L*(Time = 25 | **Normal**))



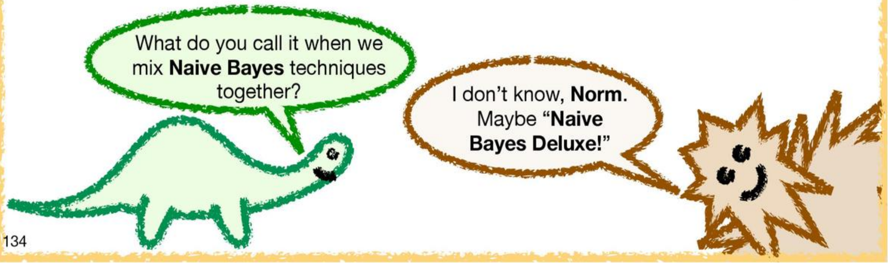


Time
5.3
7.1
etc.

log(*p*(**Spam**)) + **log**(*p*(**Dear** | **Spam**))

+ **log**(*p*(**Friend** | **Spam**)) + **log**(*L*(**Time = 25** | **Spam**))

TRIPLE SPAM!!!



Naive Bayes: FAQ Part 3

How is Naive Bayes related to Bayes' Theorem?

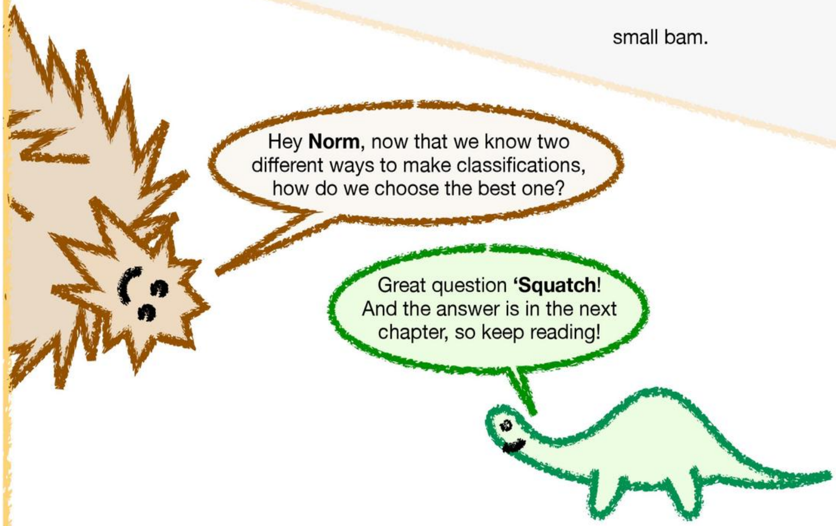
$$p(\text{N} \mid \textbf{Dear Friend}) = \frac{p(\text{S}) \times p(\textbf{Dear} \mid \text{S}) \times p(\textbf{Friend} \mid \text{S})}{p(\text{N}) \times p(\textbf{Dear} \mid \text{N}) \times p(\textbf{Friend} \mid \text{N}) + p(\text{S}) \times p(\textbf{Dear} \mid \text{S}) \times p(\textbf{Friend} \mid \text{S})}$$

$$p(\text{S} \mid \textbf{Dear Friend}) = \frac{p(\text{N}) \times p(\textbf{Dear} \mid \text{N}) \times p(\textbf{Friend} \mid \text{N})}{p(\text{N}) \times p(\textbf{Dear} \mid \text{N}) \times p(\textbf{Friend} \mid \text{N}) + p(\text{S}) \times p(\textbf{Dear} \mid \text{S}) \times p(\textbf{Friend} \mid \text{S})}$$

If we wanted to do extra math that wouldn't change the result, we could divide each score by the sum of the scores, and that would give us **Bayes' Theorem.**

Because the denominators are the same in both equations, the results are determined entirely by the numerators. So, to save time and trouble, often the denominators are omitted.

small bam.



Chapter 08

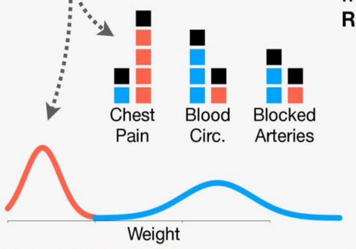
Assessing Model Performance!!!

Assessing Model Performance: Main Ideas

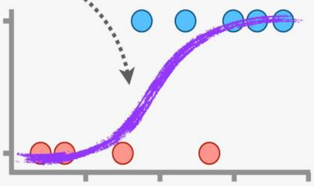
1 **The Problem:** We've got this dataset, and we want to use it to predict if someone has Heart Disease, but we don't know in advance which model will make the best predictions.

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
...	...	...	...	...

Should we choose **Naive Bayes**…



…or **Logistic Regression**?



2 **A Solution:** Rather than worry too much upfront about which model will be best, we can try both and assess their performance using a wide variety of tools…

…from **Confusion Matrices**, simple grids that tell us what each model did right and what each one did wrong…

		Predicted	
		Has Heart Disease	Does Not Have Heart Disease
Actual	Has Heart Disease	**142**	22
	Does Not Have Heart Disease	29	**110**

…to **Receiver Operator Curves** (**ROCs**), which give us an easy way to evaluate how each model performs with different classification thresholds.

BAM!!!

Let's start by learning about the **Confusion Matrix**.

The Confusion Matrix: Main Ideas

① So we have this dataset that we can use to predict if someone has Heart Disease, and we want to decide between using **Naive Bayes** and **Logistic Regression**.

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
...	...	...	...	...

② We start by dividing the data into two blocks...

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
...	...	...	...	...

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	No	210	No
...	...	...	...	...

...and we use the first block to optimize both models using **Cross Validation.**

③ Then we apply the optimized **Naive Bayes** model to the second block and build a **Confusion Matrix** that helps us keep track of **4** things: **1)** the number of people with Heart Disease who were *correctly* predicted to have Heart Disease...

...**2)** the number of people with Heart Disease who were *incorrectly* predicted to not have Heart Disease...

...**3)** the number of people without Heart Disease who were *correctly* predicted to not have Heart Disease...

...and, lastly, **4)** the number of people without Heart Disease who were *incorrectly* predicted to have Heart Disease.

		Predicted	
		Has Heart Disease	Does Not Have Heart Disease
Actual	Has Heart Disease	142	22
	Does Not Have Heart Disease	29	110

④ We then build a **Confusion Matrix** for the optimized **Logistic Regression** model...

		Predicted	
		Yes	No
Actual	Yes	137	22
	No	29	115

...and, at a glance, we can see that **Logistic Regression** is better at predicting people ***without*** Heart Disease and **Naive Bayes** is better at predicting people ***with*** it.

Now we can pick the model based on whether we want to identify someone ***with*** or ***without*** Heart Disease.

BAM!!

The Confusion Matrix: Details Part 1

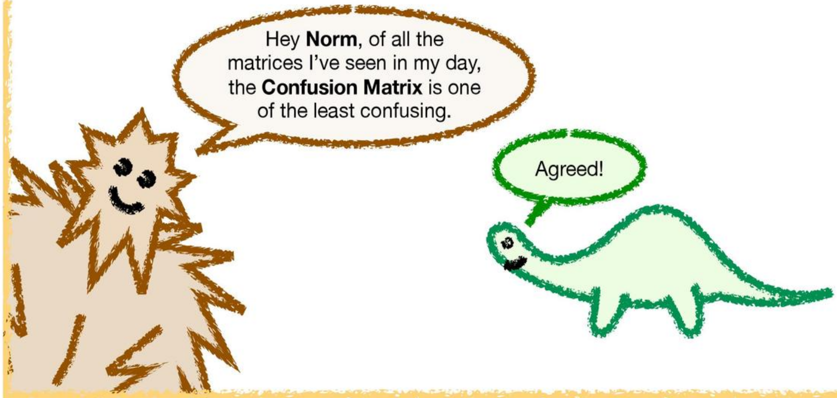
① When the actual and predicted values are both **YES**, then we call that a **True Positive**…

…when the actual value is **YES**, but the predicted value is **NO**, then we call that a **False Negative**…

…when the actual and predicted values are both **NO**, then we call that a **True Negative**…

…and when the actual value is **NO**, but the predicted value is **YES**, then we call that a **False Positive**.

		Predicted	
		Yes	No
Actual	Yes	**True Positives**	False Negatives
	No	False Positives	**True Negatives**



The Confusion Matrix: Details Part 2

② When there are only two possible outcomes, like **Yes** and **No**…

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
…	…	…	…	…

…then the corresponding **Confusion Matrix** has **2** rows and **2** columns: one each for **Yes** and **No**.

		Predicted	
		Has Heart Disease	Does Not Have Heart Disease
Actual	Has Heart Disease	**142**	22
	Does Not Have Heart Disease	29	**110**

③ When there are **3** possible outcomes, like in this dataset that has **3** choices for favorite movie, **Troll 2**, **Gore Police**, and **Cool as Ice**…

Jurassic Park III	Run for Your Wife	Out Kold	Howard the Duck	Favorite Movie
Yes	No	Yes	Yes	Troll 2
No	No	Yes	No	Gore Police
No	Yes	Yes	Yes	Cool as Ice
…	…	…	…	…

…then the corresponding **Confusion Matrix** has **3** rows and **3** columns.

		Predicted		
		Troll 2	Gore Police	Cool As Ice
Actual	Troll 2	**142**	22	..
	Gore Police	29	**110**	..
	Cool as Ice	..	..	**…**

④ In general, the size of the matrix corresponds to the number of classifications we want to predict.

Bam.

The Confusion Matrix: Details Part 3

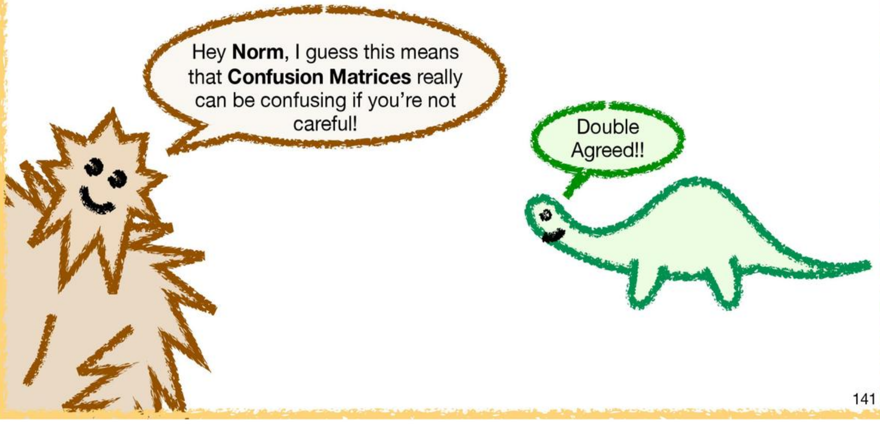
⑤ Unfortunately, there's no standard for how a **Confusion Matrix** is oriented. In many cases, the rows reflect the actual, or known, classifications and the columns represent the predictions.

		Predicted	
		Has Heart Disease	Does Not Have Heart Disease
Actual	Has Heart Disease	**True Positives**	False Negatives
	Does Not Have Heart Disease	False Positives	**True Negatives**

In other cases, the rows reflect the predictions and the columns represent the actual, or known, classifications.

		Actual	
		Has Heart Disease	Does Not Have Heart Disease
Predicted	Has Heart Disease	**True Positives**	False Positives
	Does Not Have Heart Disease	False Negatives	**True Negatives**

So, make sure you read the labels before you interpret a **Confusion Matrix**!



The Confusion Matrix: Details Part 4

Gentle Reminder:

		Predicted	
		Yes	No
Actual	Yes	TP	FN
	No	FP	TN

True Positive → TP
False Negative → FN
False Positive → FP
True Negative → TN

6 Lastly, in the original example, when we compared the **Confusion Matrices** for **Naive Bayes** and **Logistic Regression**…

Naive Bayes		Predicted	
		Yes	No
Actual	Yes	142	22
	No	29	110

Logistic Regression		Predicted	
		Yes	No
Actual	Yes	137	22
	No	29	115

…because both matrices contained the same number of **False Negatives** (**22** each) and **False Positives** (**29** each)…

…all we needed to do was compare the **True Positives** (**142** vs. **137**) to quantify how much better **Naive Bayes** was at predicting people *with* Heart Disease. Likewise, to quantify how much better **Logistic Regression** was at predicting people *without* Heart Disease, all we needed to do was compare the **True Negatives** (**110** vs. **115**).

7 However, what if we had ended up with different numbers of **False Negatives** and **False Positives**?

Naive Bayes		Predicted	
		Yes	No
Actual	Yes	142	29
	No	22	110

Logistic Regression		Predicted	
		Yes	No
Actual	Yes	139	32
	No	20	112

We can still see that **Naive Bayes** is better at predicting people *with* Heart Disease, but quantifying how much better is a little more complicated because now we have to compare **True Positives** (**142** vs. **139**) and **False Negatives** (**29** vs. **32**).

And we still see that **Logistic Regression** is better at predicting people *without* Heart Disease, but quantifying how much better has to take **True Negatives** (**110** vs. **112**) *and* **False Positives** (**22** vs. **20**) into account.

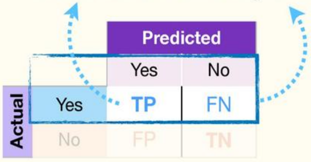
8 The good news is that we have metrics that include various combinations of **True** and **False Positives** with **True** and **False Negatives** and allow us to easily quantify all kinds of differences in algorithm performance. The first of these metrics are **Sensitivity** and **Specificity**, and we'll talk about those next.

BAM!!!

Sensitivity and Specificity: Main Ideas

① When we want to quantify how well an algorithm (like **Naive Bayes**) correctly classifies the *actual* **Positives**, in this case, the known people *with* Heart Disease, we calculate **Sensitivity**, which is the percentage of the actual **Positives** that were *correctly* classified.

$$\textbf{Sensitivity} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$



		Predicted	
		Yes	No
Actual	Yes	**TP**	FN
	No	FP	TN

For example, using the Heart Disease data and **Confusion Matrix**, the **Sensitivity** for **Naive Bayes** is **0.83**…

Naive Bayes

		Predicted	
		Yes	No
Actual	Yes	**142**	29
	No	22	110

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP + FN}} = \frac{142}{142 + 29} = 0.83$$

…which means that **83%** of the people *with* Heart Disease were correctly classified.

BAM!!!

② When we want to quantify how well an algorithm (like **Logistic Regression**) correctly classifies the *actual* **Negatives**, in this case, the known people *without* Heart Disease, we calculate **Specificity**, which is the percentage of the actual **Negatives** that were *correctly* classified.

$$\textbf{Specificity} = \frac{\text{True Negatives}}{\text{True Negatives + False Positives}}$$

		Predicted	
		Yes	No
Actual	Yes	TP	FN
	No	FP	**TN**

For example, using the **Heart Disease** data and **Confusion Matrix**, the **Specificity** for **Logistic Regression** is **0.85**…

Logistic Regression

		Predicted	
		Yes	No
Actual	Yes	139	32
	No	20	**112**

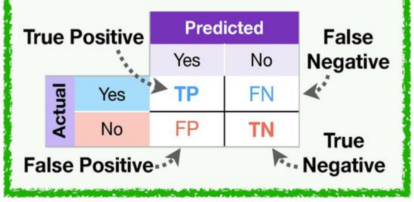
$$\text{Specificity} = \frac{\text{TN}}{\text{TN + FP}} = \frac{112}{112 + 20} = 0.85$$

…which means that **85%** of the people *without* Heart Disease were correctly classified.

DOUBLE BAM!!!

Now let's talk about **Precision** and **Recall**.

Gentle Reminder:

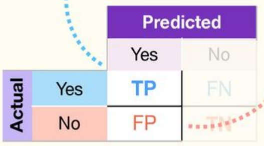


		Predicted	
		Yes	No
Actual	Yes	**TP**	FN
	No	FP	**TN**

Precision and Recall: Main Ideas

① **Precision** is another metric that can summarize a **Confusion Matrix**. It tells us the percentage of the *predicted* **Positive** results (so, both **True** and **False Positives**) that were *correctly* classified.

$$\textbf{Precision} = \frac{\text{True Positives}}{\text{True Positives + False Positives}}$$



		Predicted	
		Yes	No
Actual	Yes	TP	FN
	No	FP	TN

For example, using the Heart Disease data and **Confusion Matrix**, the **Precision** for **Naive Bayes** is **0.87**…

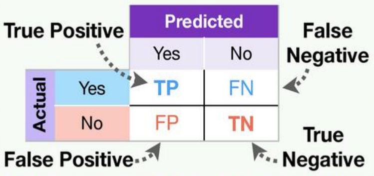
Naive Bayes

		Predicted	
		Yes	No
Actual	Yes	142	29
	No	22	110

$$\text{Precision} = \frac{\text{TP}}{\text{TP + FP}} = \frac{142}{142 + 22} = 0.87$$

…which means that of the **164** people that we predicted to have Heart Disease, **87%** actually have it. In other words, **Precision** gives us a sense of the *quality* of the positive results. When we have high **Precision**, we have high-quality positive results.

Gentle Reminder:

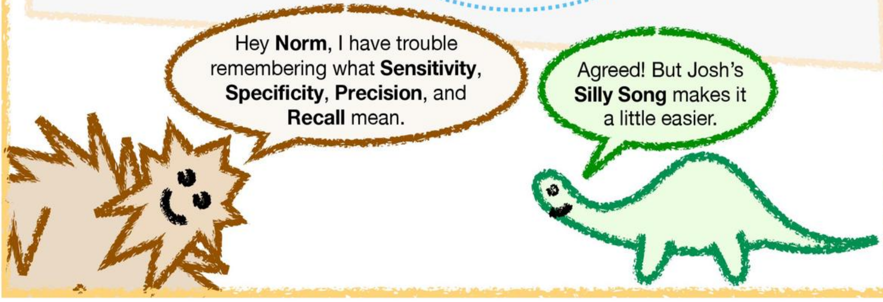


		Predicted	
		Yes	No
Actual	Yes	TP	FN
	No	FP	TN

② **Recall** is just another name for **Sensitivity**, which is the percentage of the *actual* **Positives** that were *correctly* classified. Why do we need two different names for the same thing? I don't know.

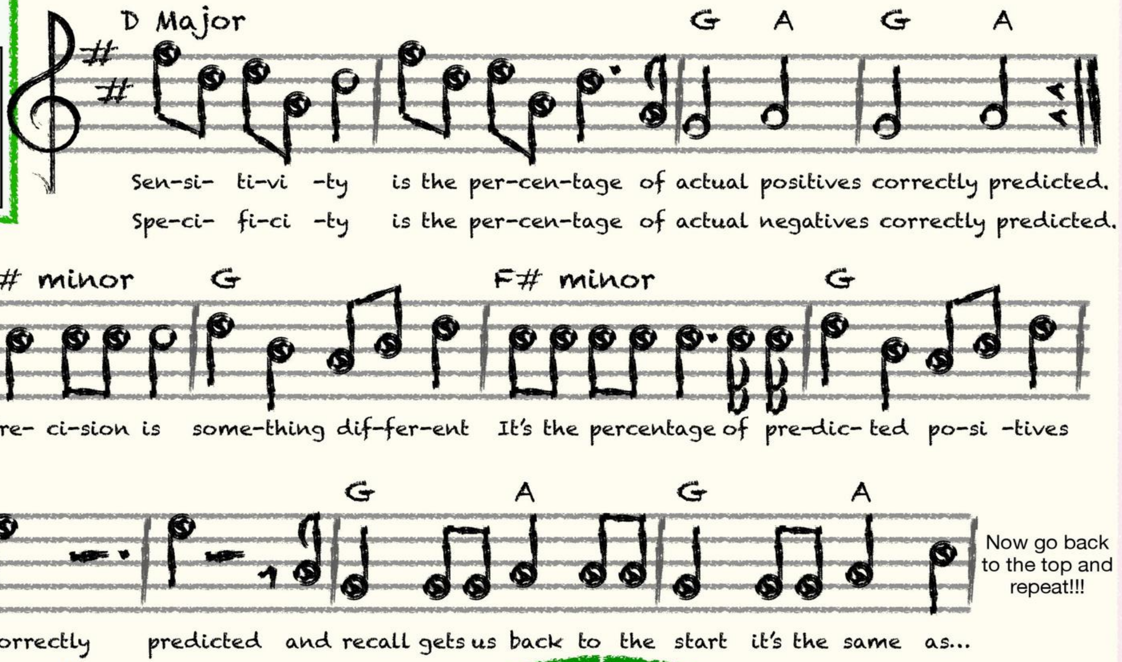
$$\textbf{Recall} = \textbf{Sensitivity} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$

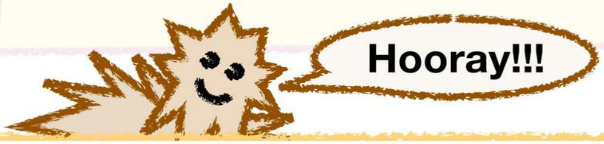
		Predicted	
		Yes	No
Actual	Yes	TP	FN
	No	FP	TN

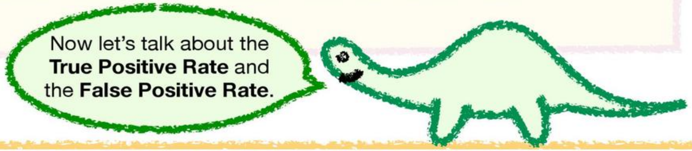


Scan, click, or tap this QR code to hear the **Silly Song**!!!

The Sensitivity, Specificity, Precision, Recall Song!!!



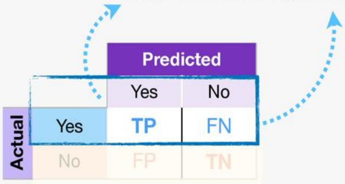




True Positive Rate and False Positive Rate: Main Ideas

① The **True Positive Rate** is the same thing as **Recall**, which is the same thing as **Sensitivity**. I kid you not. We have **3** names for the exact same thing: the percentage of *actual* **Positives** that were *correctly* classified. **TRIPLE UGH!!!**

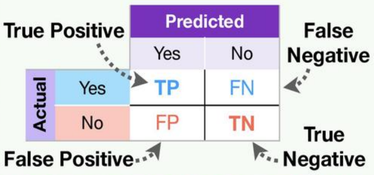
$$\textbf{True Positive Rate} = \textbf{Recall} = \textbf{Sensitivity} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$



		Predicted	
		Yes	No
Actual	Yes	TP	FN
	No	FP	TN

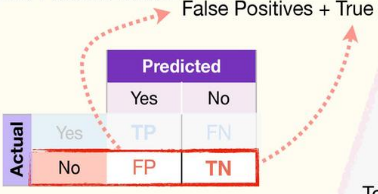
② The **False Positive Rate** tells you the percentage of *actual* **Negatives** that were *incorrectly* classified. In this case, it's the known people *without* Heart Disease who were *incorrectly* classified.

Gentle Reminder:



		Predicted	
		Yes	No
Actual	Yes	TP	FN
	No	FP	TN

$$\textbf{False Positive Rate} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$



		Predicted	
		Yes	No
Actual	Yes	TP	FN
	No	FP	TN

③ Okay, now we've got some fancy terminology that we can use to summarize individual confusion matrices. BAM?

To be honest, even with the **Silly Song**, I find it hard to remember all of this fancy terminology and so don't use it that much on its own. However, it's a stepping stone to something that I use all the time and find super useful: **ROC** and **Precision Recall Graphs**, so let's learn about those!

BAM!!!

NOTE: Remember, **Specificity** is the proportion of actual **Negatives** that were *correctly* classified, thus…

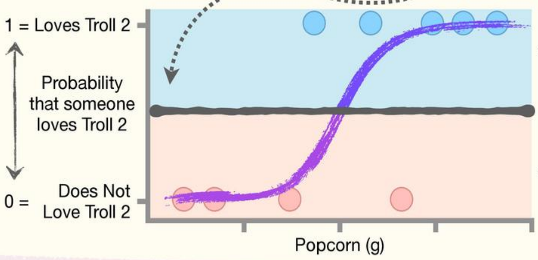
False Positive Rate = 1 - **Specificity**

…and…

Specificity = 1 - **False Positive Rate**

ROC: Main Ideas Part 1

① In **Chapter 6**, we used **Logistic Regression** to predict whether or not someone loves Troll 2, and we mentioned that usually the classification threshold is **50%**…

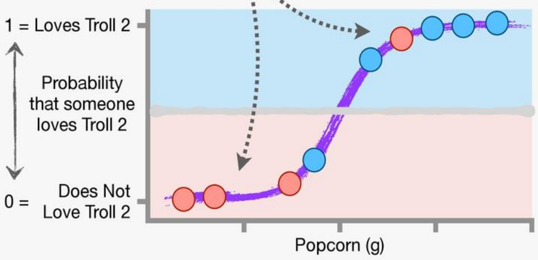


…which means that if the predicted probability is **> 50%**, then we classify them as someone who **Loves Troll 2**…

…and if the probability is **≤ 50%**, then we classify them as someone who **Does Not Love Troll 2**.

② Now, using a classification threshold of **50%**, we can classify the each person…

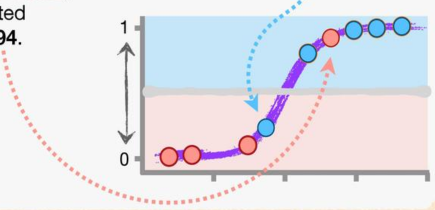
…and create this **Confusion Matrix**.



		Predicted	
		Yes	No
Actual	Yes	4	1
	No	1	3

NOTE: This **False Negative** comes from someone we know **Loves Troll 2**, but has a predicted probability of **0.27**.

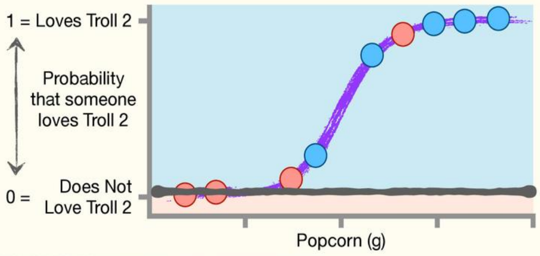
NOTE: This **False Positive** comes from someone we know **Does Not Love Troll 2**, but has a predicted probability of **0.94**.



ROC: Main Ideas Part 2

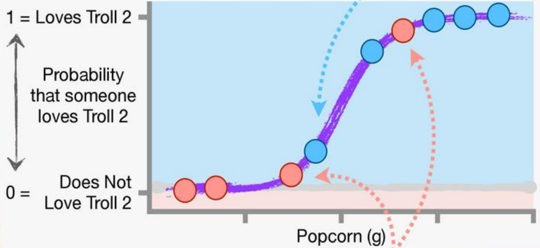
③ Now let's talk about what happens when we use a different classification threshold for deciding if someone **Loves Troll 2** or not.

For example, if it was super important to correctly classify every single person who **Loves Troll 2**, we could set the threshold to **0.01**.



NOTE: If the idea of using a classification threshold other than **0.5** is blowing your mind, imagine we're trying to classify people with the Ebola virus. In this case, it's absolutely essential that we correctly identify every single person with the virus to minimize the risk of an outbreak. And that means lowering the threshold, even if it results in more **False Positives**.

④ When the classification threshold is **0.01**, we correctly classify everyone who **Loves Troll 2**…



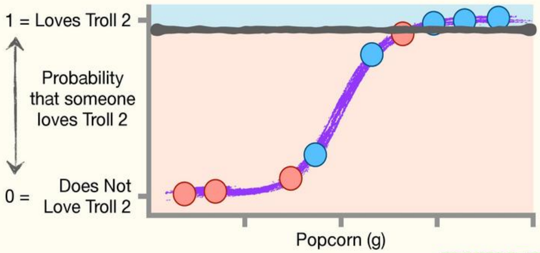
…and that means there are **0 False Negatives**…

		Predicted	
		Yes	No
Actual	Yes	5	0
	No	2	2

…but we also increase the number of **False Positives** to **2** because these two people we know **Do Not Love Troll 2** are now predicted to **Love Troll 2**.

ROC: Main Ideas Part 3

5 On the other hand, if it was super important to correctly classify everyone who **Does Not Love Troll 2**, we could set the classification threshold to **0.95**…



Gentle Reminder:

		Predicted	
		Yes	No
Actual	Yes	TP	FN
	No	FP	TN

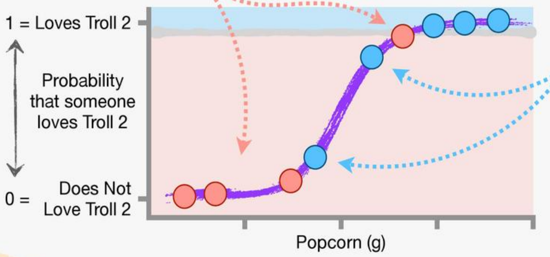
True Positive

False Negative

False Positive

True Negative

6 …and now we would have **0 False Positives** because all of the people that we know **Do Not Love Troll 2** would be correctly classified…



…but now we would have **2 False Negatives** because two people that we know **Love Troll 2** would now be incorrectly classified as people who **Do Not Love Troll 2**…

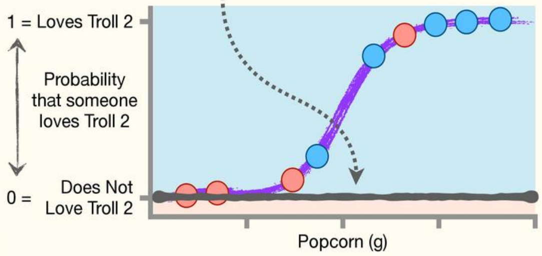
…and we would end up with this **Confusion Matrix**.

		Predicted	
		Yes	No
Actual	Yes	3	2
	No	0	4

ROC: Main Ideas Part 4

7 We could also set the classification threshold to **0**, and classify everyone as someone who **Loves Troll 2**...

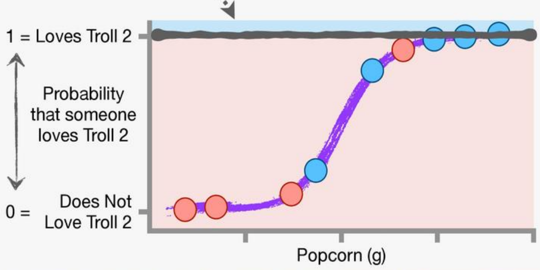
...and that would give us this **Confusion Matrix**.



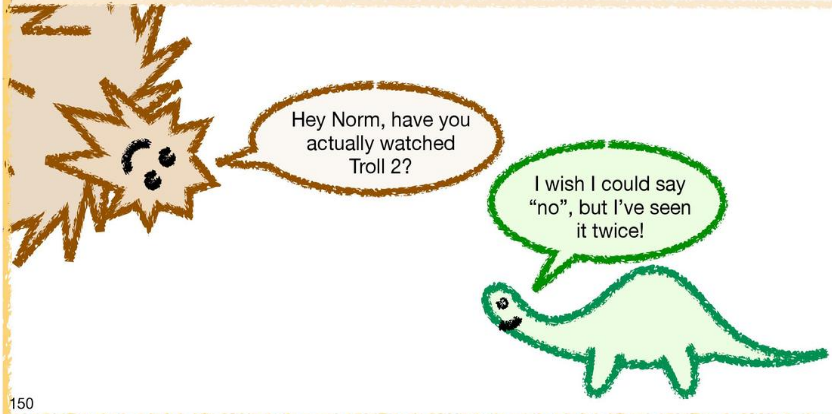
		Predicted	
		Yes	No
Actual	Yes	**5**	0
	No	4	**0**

8 Or, we could set the classification threshold to **1** and classify everyone as someone who **Does Not Love Troll 2**...

...and that would give us this **Confusion Matrix**.

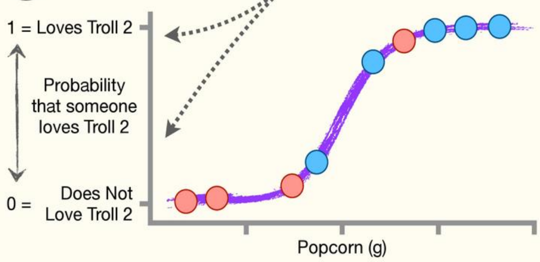


		Predicted	
		Yes	No
Actual	Yes	**0**	5
	No	0	**4**



ROC: Main Ideas Part 5

9 Ultimately, we can try any classification threshold from **0** to **1**…



…and when we do, we end up with **10** different **Confusion Matrices** that we can choose from. (**NOTE:** The threshold under each **Confusion Matrix** is just one of many that will result in the exact same **Confusion Matrix**).

		Predicted	
		Yes	No
Actual	Yes	5	0
	No	4	0

Threshold = 0

		Predicted	
		Yes	No
Actual	Yes	5	0
	No	2	2

Threshold = 0.0645

		Predicted	
		Yes	No
Actual	Yes	5	0
	No	3	1

Threshold = 0.007

		Predicted	
		Yes	No
Actual	Yes	5	0
	No	1	3

Threshold = 0.195

		Predicted	
		Yes	No
Actual	Yes	4	1
	No	1	3

Threshold = 0.5

		Predicted	
		Yes	No
Actual	Yes	3	2
	No	1	3

Threshold = 0.87

		Predicted	
		Yes	No
Actual	Yes	3	2
	No	0	4

Threshold = 0.95

		Predicted	
		Yes	No
Actual	Yes	2	3
	No	0	4

Threshold = 0.965

		Predicted	
		Yes	No
Actual	Yes	1	4
	No	0	4

Threshold = 0.975

		Predicted	
		Yes	No
Actual	Yes	0	5
	No	0	4

Threshold = 1

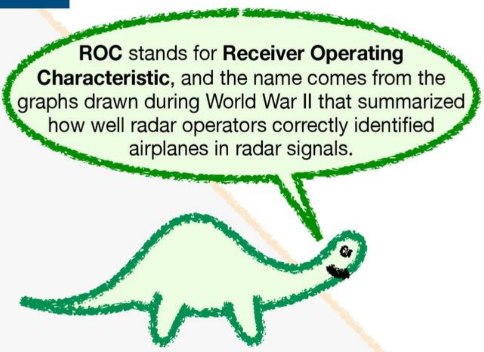
UGH!!! Trying to find the ideal classification threshold among all of these technicolor **Confusion Matrices** is tedious and annoying. Wouldn't it be awesome if we could consolidate them into one easy-to-interpret graph?

YES!!!

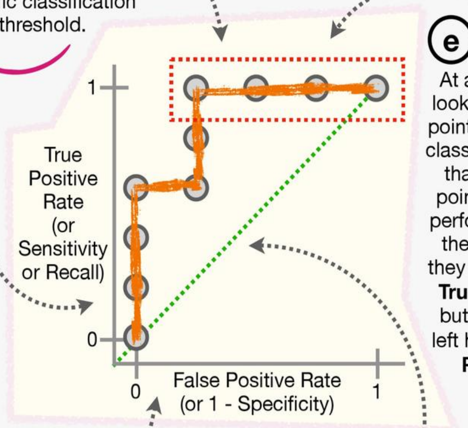
Well, we're in luck because that's what **ROC** graphs do!!!

ROC: Main Ideas Part 6

10 **ROC** graphs are super helpful when we need to identify a good classification threshold because they summarize how well each threshold performed in terms of the **True Positive Rate** and the **False Positive Rate**.



a Each **gray dot** on the **ROC** graph tells us the **True Positive Rate** and the **False Positive Rate** for a specific classification threshold.



b The higher the dot is along the y-axis, the higher the percentage of actual **Positives** were *correctly* classified...

c ...and the further to the left along the x-axis, the lower the percentage of actual **Negatives** that were *incorrectly* classified.

d The **diagonal line** shows where the **True Positive Rate** = **False Positive Rate**.

e At a glance, we can look at the top row of points and tell that the classification threshold that resulted in the point on the left side performed better than the others because they all have the same **True Positive Rate**, but the point on the left has a lower **False Positive Rate**.

11 Now that we understand the main ideas behind **ROC** graphs, let's dive into the details!!!

ROC: Details Part 1

Gentle Reminder:

		Predicted	
		Yes	No
Actual	Yes	**TP**	FN
	No	FP	**TN**

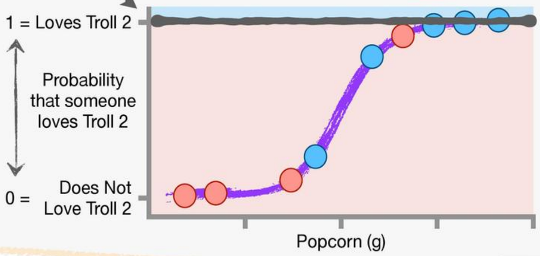
True Positive

False Negative

False Positive

True Negative

① To get a better sense of how an **ROC** graph works, let's draw one from start to finish. We'll start by using a classification threshold, **1**, that classifies everyone as someone who **Does Not Love Troll 2**…



…and when the classification threshold is set to **1**, we end up with this **Confusion Matrix**.

		Predicted	
		Yes	No
Actual	Yes	0	5
	No	0	4

Threshold = 1

② Using the values in the **Confusion Matrix**, we can calculate the **True Positive Rate**…

$$\textbf{True Positive Rate} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

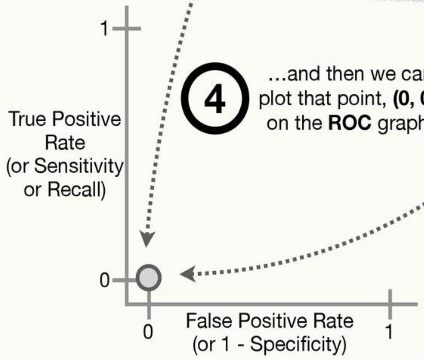
$$= \frac{0}{0 + 5} = 0$$

③ …and the **False Positive Rate**…

$$\textbf{False Positive Rate} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

$$= \frac{0}{0 + 4} = 0$$

④ …and then we can plot that point, **(0, 0)**, on the **ROC** graph.



ROC: Details Part 2

Gentle Reminder:

True Positive

False Negative

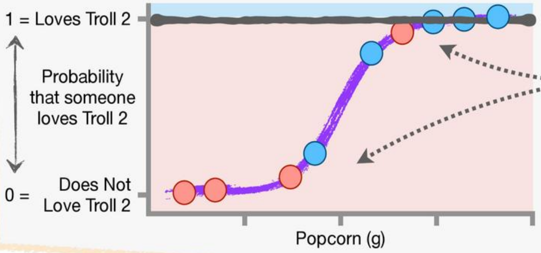
		Predicted	
		Yes	No
Actual	Yes	**TP**	FN
	No	FP	**TN**

False Positive

True Negative

⑤ Now let's lower the classification threshold to **0.975**…

…which is just enough to classify one person as someone who **Loves Troll 2**…



…and everyone else is classified as someone who **Does Not Love Troll 2**…

…and that gives us this **Confusion Matrix**.

		Predicted	
		Yes	No
Actual	Yes	**1**	4
	No	0	**4**

Threshold = 0.975

⑥ Using the values in the **Confusion Matrix**, we can calculate the **True Positive Rate**…

$$\textbf{True Positive Rate} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$

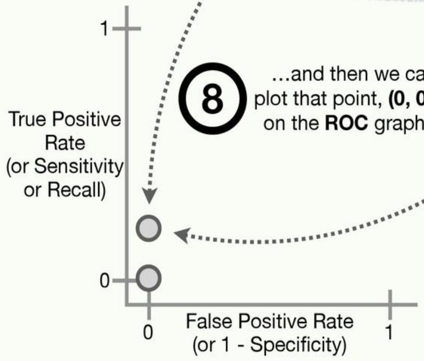
$$= \frac{1}{1 + 4} = 0.2$$

⑦ …and the **False Positive Rate**…

$$\textbf{False Positive Rate} = \frac{\text{False Positives}}{\text{False Positives + True Negatives}}$$

$$= \frac{0}{0 + 4} = 0$$

⑧ …and then we can plot that point, **(0, 0.2)**, on the **ROC** graph…



…and the new point is above the first point, showing that the new threshold increases the proportion of *actual* **Positives** that were *correctly* classified. **BAM!!!**

ROC: Details Part 3

Gentle Reminder:

		Predicted	
		Yes	No
Actual	Yes	**TP**	FN
	No	FP	**TN**

True Positive

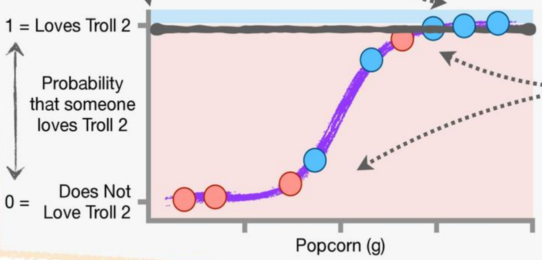
False Negative

False Positive

True Negative

⑨ Now let's lower the classification threshold to **0.965**…

…which is just enough to classify **2** people as people who **Love Troll 2**…



…and everyone else is classified as someone who **Does Not Love Troll 2**…

…and that gives us this **Confusion Matrix**.

		Predicted	
		Yes	No
Actual	Yes	2	3
	No	0	4

Threshold = 0.965

⑩ Using the values in the **Confusion Matrix**, we can calculate the **True Positive Rate**…

$$\textbf{True Positive Rate} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$

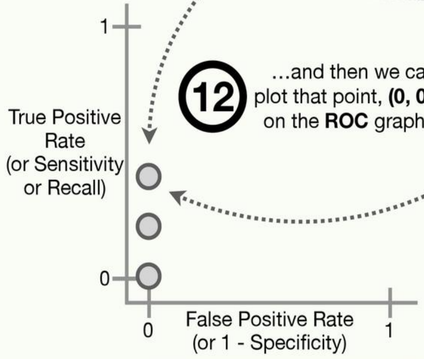
$$= \frac{2}{2 + 3} = 0.4$$

⑪ …and the **False Positive Rate**…

$$\textbf{False Positive Rate} = \frac{\text{False Positives}}{\text{False Positives + True Negatives}}$$

$$= \frac{0}{0 + 4} = 0$$

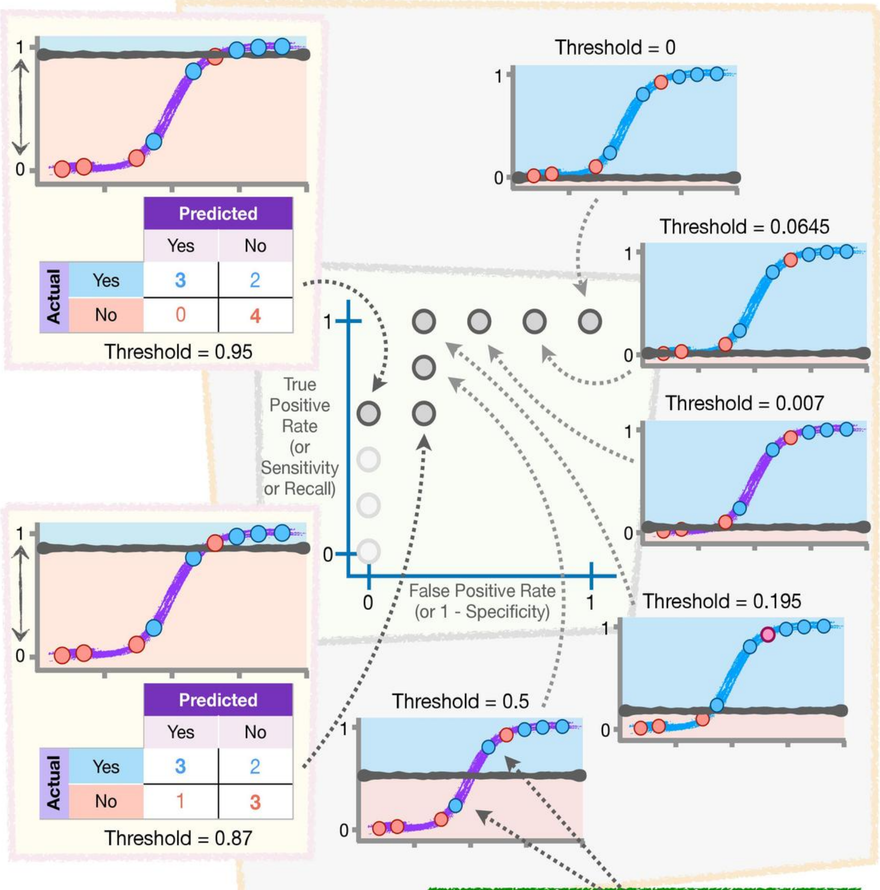
⑫ …and then we can plot that point, **(0, 0.4)**, on the **ROC** graph…



…and the new point is above the first two points, showing that the new threshold increases the proportion of *actual* **Positives** that were *correctly* classified.

ROC: Details Part 4

⑬ Likewise, for each threshold that increases the number of **Positive** classifications (in this example, that means classifying a person as someone who **Loves Troll 2**), we calculate the **True Positive Rate** and **False Positive Rate** until everyone is classified as **Positive**.



Threshold = 0.95

		Predicted: Yes	Predicted: No
Actual	Yes	3	2
	No	0	4

Threshold = 0.87

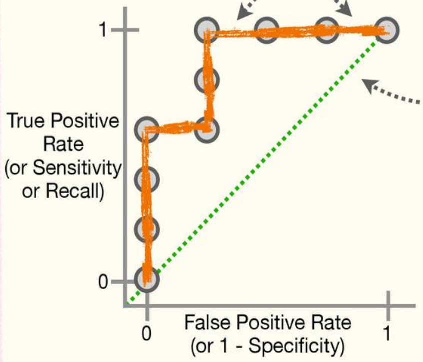
		Predicted: Yes	Predicted: No
Actual	Yes	3	2
	No	1	3

NOTE: Although there are a lot of potential thresholds between these two points, they all result in the same **True Positive Rate** and the same **False Positive Rate**, so it doesn't matter which one we pick, we just have to pick one of them.

ROC: Details Part 5

⑭ After we finish plotting the points from each possible **Confusion Matrix**, we usually **connect the dots**...

...and add a **diagonal line** that tells us when the **True Positive Rate = False Positive Rate.**

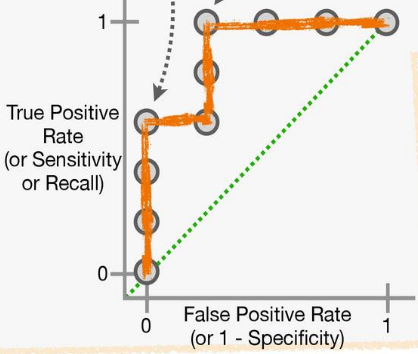


⑮ Now, without having to sort through a huge pile of **Confusion Matrices**, we can use the **ROC** graph to pick a classification threshold.

If we want to avoid all **False Positives**, but want to maximize the number of actual **Positives** correctly classified, we would pick this threshold...

...but if we can tolerate a few **False Positives**, we would pick this threshold because it *correctly* classifies all of the actual **Positives**.

BAM!!!



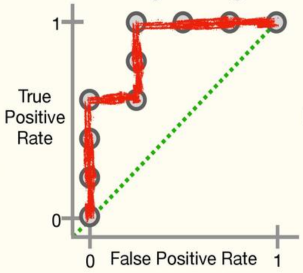
⑯ **ROC** graphs are great for selecting an optimal classification threshold for a model. But what if we want to compare how one model performs vs. another? This is where the **AUC**, which stands for **Area Under the Curve**, comes in handy. So read on!!!

AUC: Main Ideas

1 Now, imagine we created **Logistic Regression** and **Naive Bayes** models and tested them with the same data, and we wanted to know which model performed better.

In theory, we could compare the individual **ROC** graphs, and when we only have two models to compare, this is a pretty good option.

ROC for Logistic Regression



ROC for Naive Bayes

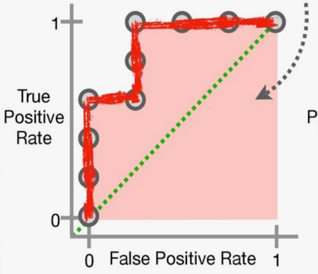
However, if we wanted to compare a bunch of models, this would be just as tedious as comparing a bunch of **Confusion Matrices**.

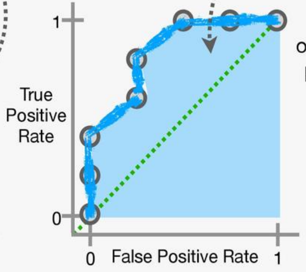
UGH!!!

2 So, instead of comparing a bunch of **ROC** graphs, one simple way to summarize them is to calculate and compare the **AUC:** the **Area Under** each **Curve**.

In this case, the **AUC** for **Logistic Regression** is **0.9**…

…and the **AUC** for **Naive Bayes** is **0.85**…





…and because **Logistic Regression** has a larger **AUC**, we can tell that, overall, **Logistic Regression** performs better than **Naive Bayes** with these data.

BAM!!!

AUC: FAQ

How do you calculate the AUC?

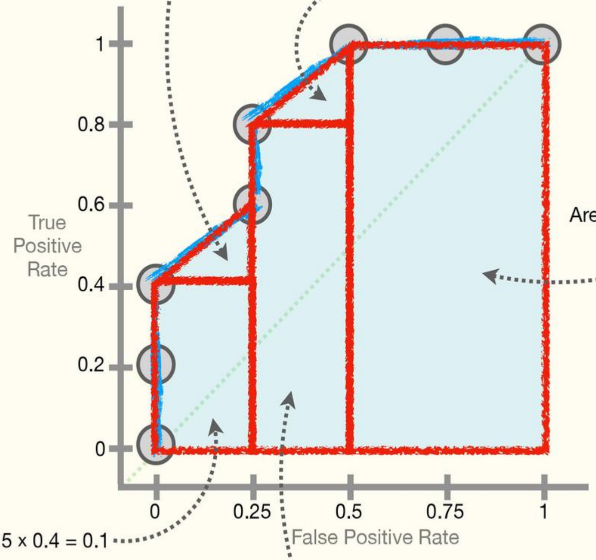
My favorite way is to get a computer to do it…

…but if you want to do it by hand, you simply divide the area into a bunch of **rectangles** and **triangles** and add the areas of each shape together.

$$
\begin{array}{r}
0.500 \\
0.200 \\
0.100 \\
0.025 \\
+\ 0.025 \\
\hline
\text{AUC = Total Area = } 0.850
\end{array}
$$

Area = $\frac{1}{2}$ 0.25 x 0.2 = 0.025

Area = $\frac{1}{2}$ 0.25 x 0.2 = 0.025



Area = 0.5 x 1 = 0.5

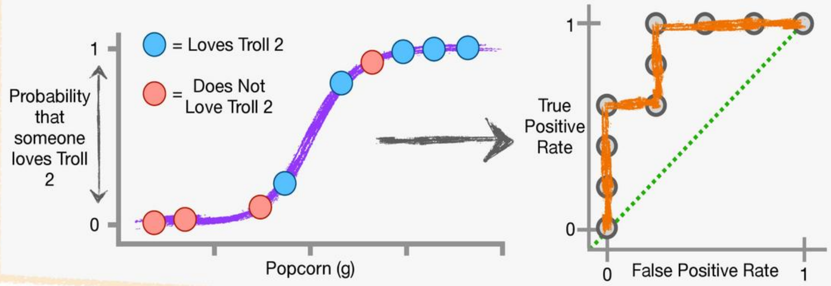
Area = 0.25 x 0.4 = 0.1

Area = 0.25 x 0.8 = 0.2

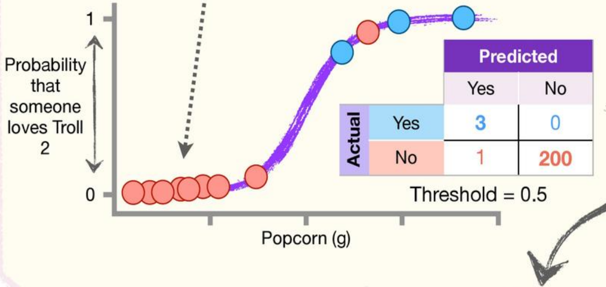

BUT WAIT!!! THERE'S MORE!!!

Precision Recall Graphs: Main Ideas Part 1

1 An **ROC** graph uses the **False Positive Rate** on the x-axis, and this is fine when the data are *balanced*, which in this case means there are similar numbers of people who **Love Troll 2** and who **Do Not Love Troll 2**.



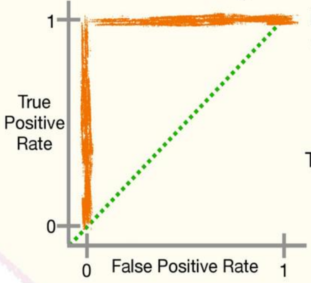
2 However, when the data are *imbalanced*, and we have way more people who **Do Not Love Troll 2** (which wouldn't be surprising since it consistently wins "worst movie ever" awards)...



		Predicted	
		Yes	No
Actual	Yes	3	0
	No	1	200

Threshold = 0.5

...then the **ROC** graph is hard to interpret because the **False Positive Rate** barely budges above **0** before we have a **100% True Positive Rate**.

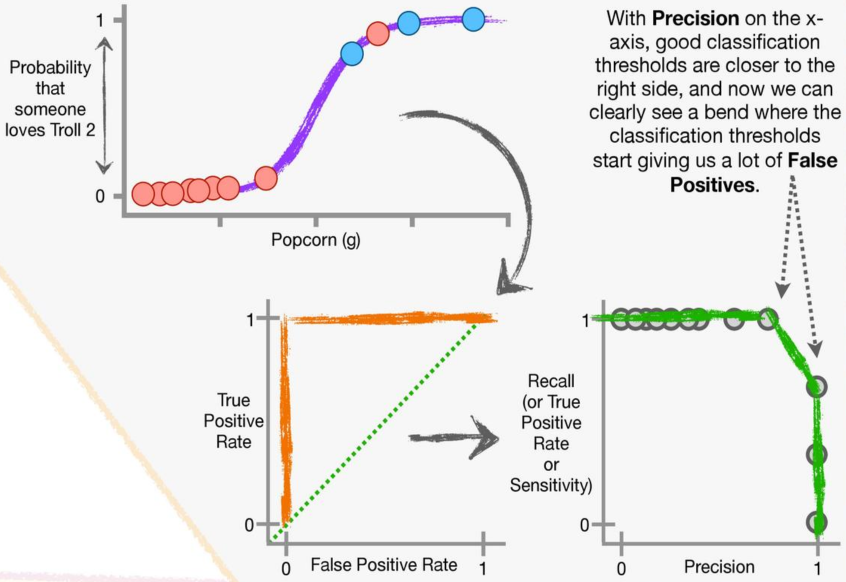


In other words, the **ROC** graph makes any model that simply predicts **No** **100%** of the time look really good.

The good news is that we have **Precision Recall** graphs that attempt to deal with this problem. Read on for details!!!

Precision Recall Graphs: Main Ideas Part 2

③ A **Precision Recall** graph simply replaces the **False Positive Rate** on the x-axis with **Precision** and renames the y-axis **Recall** since **Recall** is the same thing as the **True Positive Rate**.



With **Precision** on the x-axis, good classification thresholds are closer to the right side, and now we can clearly see a bend where the classification thresholds start giving us a lot of **False Positives**.

④ The reason **Precision** works better than the **False Positive Rate** when the data are highly imbalanced is that **Precision** does not include the number of **True Negatives**.

$$\textbf{Precision} = \frac{\text{True Positives}}{\text{True Positives + False Positives}}$$

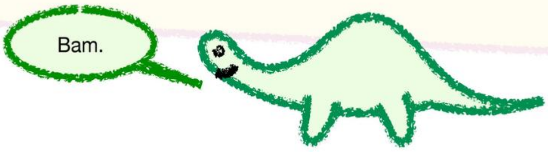
Gentle Reminder:

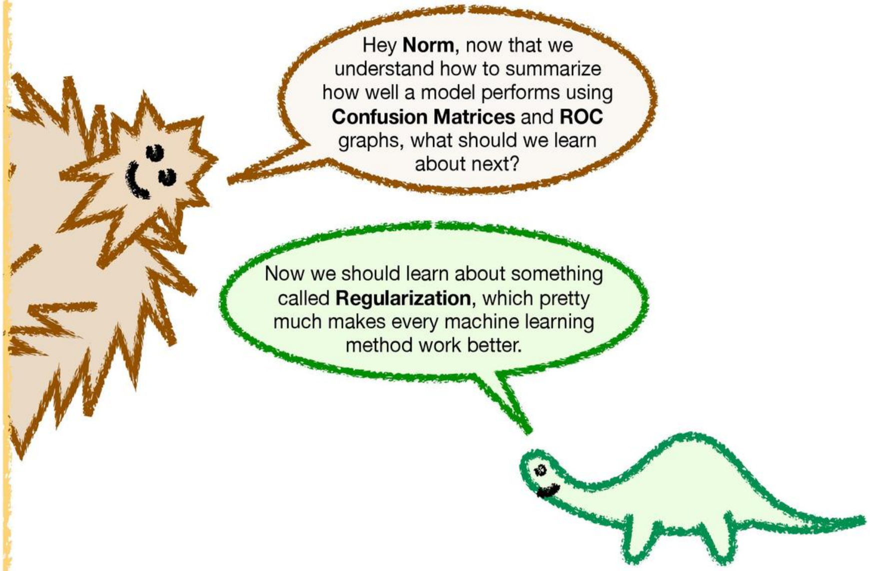
		Predicted	
		Yes	No
Actual	Yes	TP	FN
	No	FP	TN

True Positive
False Negative
False Positive
True Negative

		Predicted	
		Yes	No
Actual	Yes	3	0
	No	1	200

Threshold = 0.5




Hey **Norm**, now that we understand how to summarize how well a model performs using **Confusion Matrices** and **ROC** graphs, what should we learn about next?
Now we should learn about something called **Regularization**, which pretty much makes every machine learning method work better.

Chapter 09

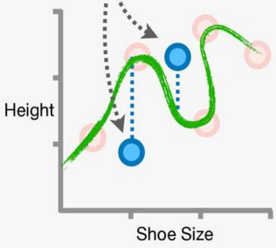
Preventing Overfitting with Regularization!!!

Regularization: Main Ideas

① **The Problem:** The fancier and more flexible a machine learning method is, the easier it is to overfit the **Training Data**.

In other words, this **squiggle** might fit the **Training Data** really well…

…but the predictions made with **New Data** are terrible.

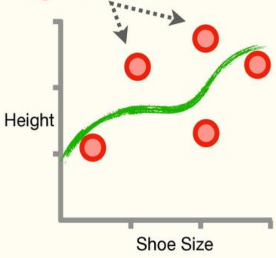


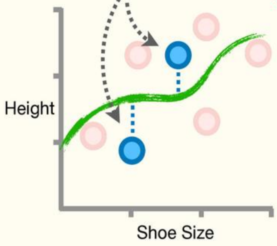
Using technical jargon, we would say that the **squiggle** has *low* **Bias** because it fits the **Training Data** well, but *high* **Variance** because it does a bad job with **New Data**.

② **A Solution:** One very common way to deal with overfitting the **Training Data** is to use a collection of techniques called **Regularization**. Essentially, **Regularization** reduces how sensitive the model is to the **Training Data**.

In this case, if we regularized the **squiggle**, then it would not fit the **Training Data** as well as it did before…

…but now it makes better predictions with **New Data**.

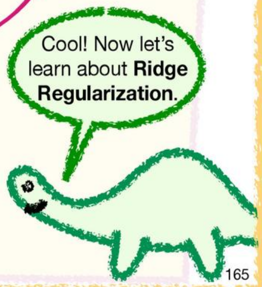




Using technical jargon, we would say that **Regularization** *increases* **Bias** a little bit, but in return, we get a big *decrease* in **Variance**.

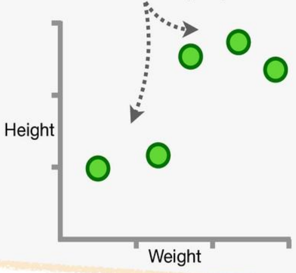
BAM!!!

NOTE: In this chapter, we'll learn about the two main types of **Regularization**, **Ridge** and **Lasso**, in the context of **Linear Regression**, but they can be used with almost any machine learning algorithm to improve performance.

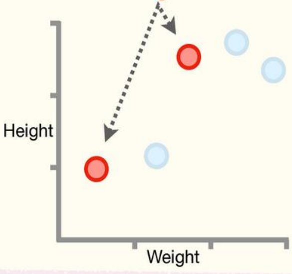


Ridge/Squared/L2 Regularization: Details Part 1

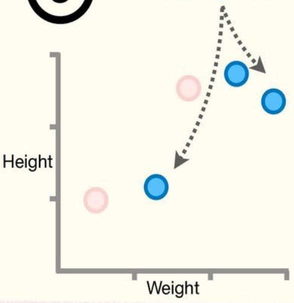
① Let's imagine we measured the Height and Weight of **5** different people...



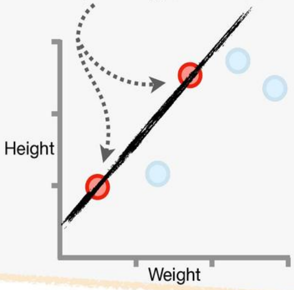
② ...and then we split those data into **Training Data**...



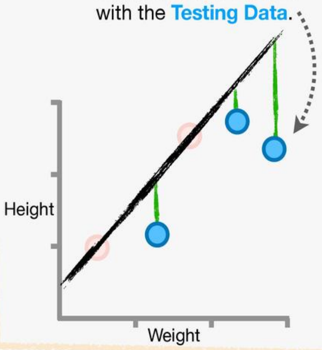
③ ...and **Testing Data**.



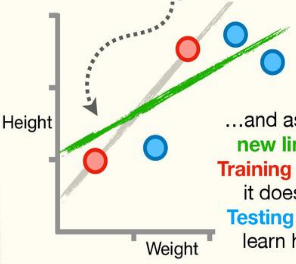
④ Then we fit a **line** to the **Training Data** that minimized the **Sum of the Squared Residuals** (**SSR**). Because we only have **2** points in the **Training Data**, the line fits it perfectly, and the **SSR** = **0**...



⑤ ...however, because the slope of the **line** is so steep, it does a bad job with the **Testing Data**.



⑥ In contrast, after applying **Ridge Regularization**, also called **Squared** or **L2 Regularization**, we get this **new line**...

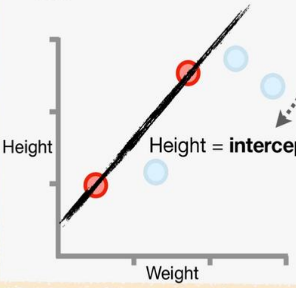


...and as you can see, the **new line** doesn't fit the **Training Data** perfectly, but it does better with the **Testing Data**. Read on to learn how it does this.

Ridge/Squared/L2 Regularization: Details Part 2

⑦ When we normally fit a line to **Training Data**, we want to find the **y-axis intercept**…

…and the **slope** that minimize the **SSR**.



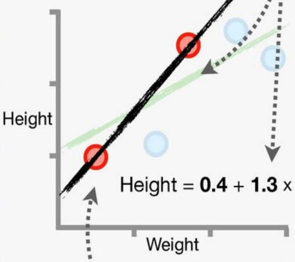
Height = **intercept** + **slope** x Weight

⑧ In contrast, when we use **Ridge Regularization** to optimize parameters, we simultaneously minimize the **SSR** and a penalty that's proportional to the square of the slope…

$\boxed{\textbf{SSR} + \boldsymbol{\lambda} \times \text{slope}^2}$

…where the Greek character **λ**, ***lambda***, is a positive number that determines how strong an effect **Ridge Regularization** has on the **new line**.

⑨ To get a better idea of how the **Ridge Penalty** works, let's plug in some numbers. We'll start with the **line** that fits the **Training Data** perfectly…



Height = **0.4** + **1.3** x Weight

⑩ Now, if we're using **Ridge Regularization**, we want to minimize this equation.

$$\textbf{SSR} + \boldsymbol{\lambda} \times \text{slope}^2 = 0 + 1 \times 1.3^2 \boxed{= 1.69}$$

Because the **line** fits the **Training Data** perfectly, the **SSR = 0**…

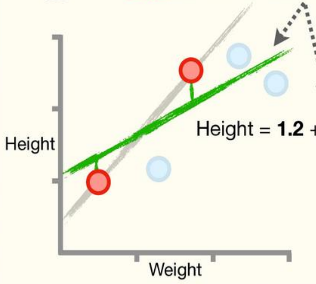
…and we'll talk more about **λ** (***lambda***) later, but for now, let's just set **λ = 1**…

…and since the slope is **1.3**, we just plug that in…

…and, we end up with **1.69** as the score for the **line** that fits the **Training Data** perfectly.

Ridge/Squared/L2 Regularization: Details Part 3

11 Now let's calculate the **Ridge Score** for the **new line** that doesn't fit the **Training Data** as well. It has an **SSR = 0.4**…



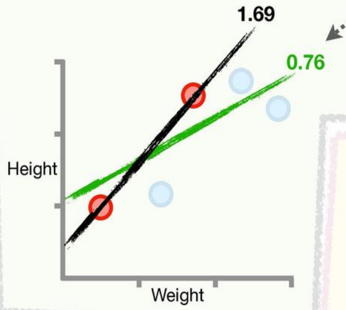
Height = **1.2** + **0.6** x Weight

12 …so we plug in the **SSR**, **0.4**, the value for ***λ*** (***lambda***), which for still is **1**, and the slope, **0.6**, into the **Ridge Penalty**.

And when we do the math, we get **0.76**.

$$\mathbf{SSR} + \boldsymbol{\lambda} \times \text{slope}^2 = 0.4 + 1 \times 0.6^2 \boxed{= 0.76}$$

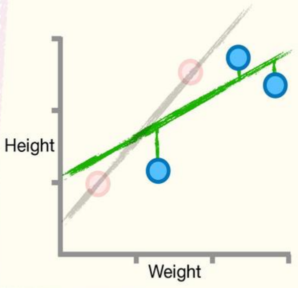
13 Thus, the **Ridge Score** for the **line** that fit the **Training Data** perfectly is **1.69**…



…and the **Ridge Score** for the **new line** that had a smaller slope and didn't fit the **Training Data** as well is **0.76**…

…and because we want to pick the line that minimizes the **Ridge Score**, we pick the **new line**.

14 Now, even though the **new line** doesn't fit the **Training Data** perfectly, it does a better job with the **Testing Data**. In other words, by *increasing* the **Bias** a little, we *decreased* the **Variance** a lot.



BAM!!!

However, you may be wondering how we found the **new line**, and that means it's time to talk a little bit more about ***λ*** (***lambda***).

$\mathbf{SSR} + \boldsymbol{\lambda} \times \text{slope}^2$

Ridge/Squared/L2 Regularization: Details Part 4

⑮ When λ = **0**…

$\mathbf{SSR} + \boldsymbol{\lambda} \times \text{slope}^2$

$= \mathbf{SSR} + \mathbf{0} \times \text{slope}^2$

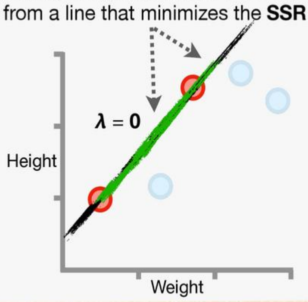
$= \mathbf{SSR} + \mathbf{0}$

$= \mathbf{SSR}$

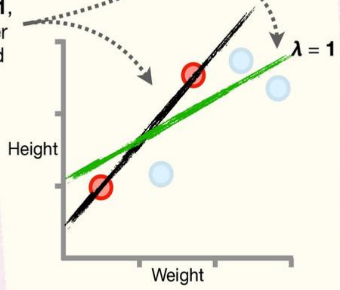
…then the whole **Ridge Penalty** is also **0**…

…and that means we'll only minimize the **SSR**, so it's as if we're *not using* **Ridge Regularization**…

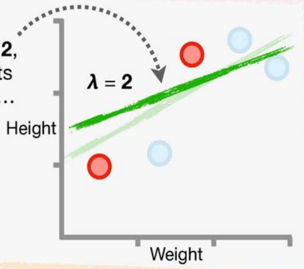
…and, as a result, the **new line**, derived from **Ridge Regularization**, is no different from a line that minimizes the **SSR**.



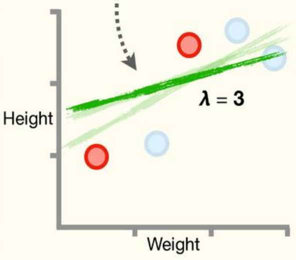
⑯ However, as we just saw, when λ = **1**, we get a **new line** that has a smaller slope than when we only minimized the **SSR**.



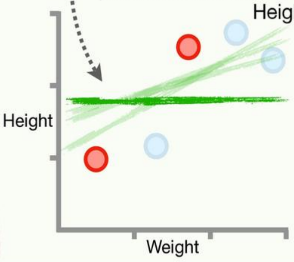
⑰ When we increase λ to **2**, the slope gets even smaller…



⑱ …and when we increase λ to **3**, the slope gets even smaller…



⑲ …and as we continue to increase λ, the slope gets closer and closer to **0** and the y-axis intercept becomes the average Height in the **Training Dataset** (**1.8**). In other words, Weight no longer plays a significant role in making predictions. Instead, we just use the mean Height.

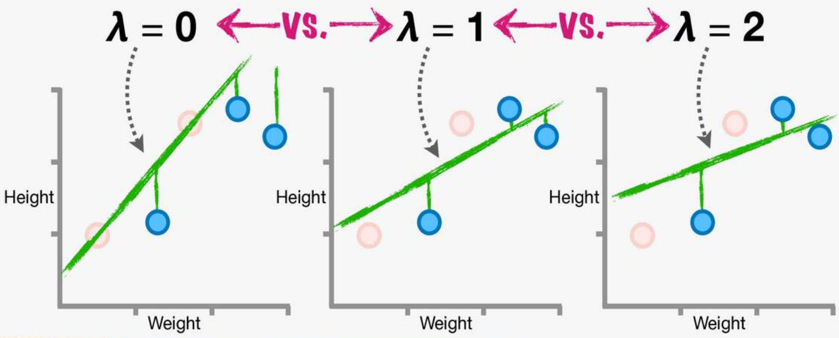


So, how do we pick a good value for λ?

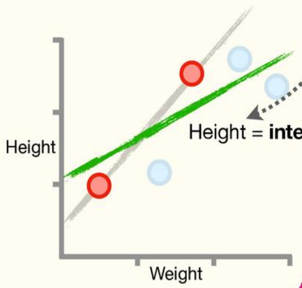
Ridge/Squared/L2 Regularization: Details Part 5

⑳ Unfortunately, there's no good way to know in advance what the best value for λ will be, so we just pick a bunch of potential values, including **0**, and see how well each one performs using **Cross Validation**.

BAM!!!



㉑ So far, our example has been super simple. Since we only used Weight to predict Height…



…the **Ridge Penalty** contained only a single parameter, the slope.

$$\text{Height} = \textbf{intercept} + \textbf{slope} \times \text{Weight}$$

$$\textbf{SSR} + \lambda \times \text{slope}^2$$

However, if we had a more complicated model that used Weight, Shoe Size, and Age to predict Height, and we would have slopes for each variable in the model…

$$\text{Height} = \textbf{intercept} + \textbf{slope}_{\textbf{w}} \times \text{Weight} + \textbf{slope}_{\textbf{s}} \times \text{Shoe Size} + \textbf{slope}_{\textbf{a}} \times \text{Age}$$

…and the **Ridge Penalty** would include the sum of the squares of the **3** slopes associated with those variables.

$$\textbf{SSR} + \lambda \times (\text{slope}_{w}^2 + \text{slope}_{s}^2 + \text{slope}_{a}^2)$$

NOTE: The **Ridge Penalty** never includes the **intercept** because the **intercept** doesn't directly affect how any of the variables (Weight, Shoe Size, or Age) predict Height.

Ridge/Squared/L2 Regularization: Details Part 6

22 When we apply **Ridge Regularization** to models with multiple parameters, like this one…

…it will shrink the parameters, $\textbf{slope}_w$, $\textbf{slope}_s$, and $\textbf{slope}_a$, but not equally.

$$\text{Height} = \textbf{intercept} + \textbf{slope}_w \times \text{Weight} + \textbf{slope}_s \times \text{Shoe Size} + \textbf{slope}_a \times \text{Airspeed of a Swallow}$$

23 For example, if Weight and Shoe Size are both useful for predicting Height, but Airspeed of a Swallow is not, then their slopes, $\textbf{slope}_w$ and $\textbf{slope}_s$, will shrink a little bit…

…compared to the slope associated with the Airspeed of a Swallow, $\textbf{slope}_a$, which will shrink a lot.

So, what causes this difference? When a variable, like Airspeed of a Swallow, is useless for making predictions, shrinking its parameter, $\textbf{slope}_a$, a lot will shrink the **Ridge Penalty** a lot…

…without increasing the **SSR**.

$$\textbf{SSR} + \boldsymbol{\lambda} \times (\text{slope}_w^2 + \text{slope}_s^2 + \text{slope}_a^2)$$

In contrast, if we shrink the slopes associated with Weight and Shoe Size, which are both useful for making predictions, then the **Ridge Penalty** would shrink, but the **SSR** would increase a lot.

BAM!!!

24 Now that we understand how the **Ridge Penalty** works, let's answer **2** of the most frequently asked questions about it. Then we'll learn about another type of **Regularization** called **Lasso**. Get pumped!!!

Ridge/Squared/L2 Regularization: FAQ

All of the examples showed how increasing λ, and thus decreasing the slope, made things better, but what if we need to increase the slope? Can Ridge Regularization ever make things worse?

As long as you try setting λ to **0**, when you're searching for the best value for λ, in theory, **Ridge Regularization** can never perform worse than simply finding the line that minimizes the **SSR**.

How do we find the optimal parameters using Ridge Regularization?

When we only have one slope to optimize, one way to find the line that minimizes the **SSR + the Ridge Penalty** is to use **Gradient Descent**. In this case, we want to find the optimal y-axis **intercept** and **slope**, so we take the derivative with respect to the **intercept**…

$$\frac{d}{d\ \textit{intercept}} (\ \mathbf{SSR} + \lambda \times \mathbf{slope}^2\ )$$

$$= -2 \times \mathbf{(}\ \text{Height} - (\mathbf{intercept} + \mathbf{slope} \times \text{Weight})\ \mathbf{)}$$

…and the derivative with respect to the **slope**.

$$\frac{d}{d\ \textit{slope}} (\ \mathbf{SSR} + \lambda \times \mathbf{slope}^2\ )$$

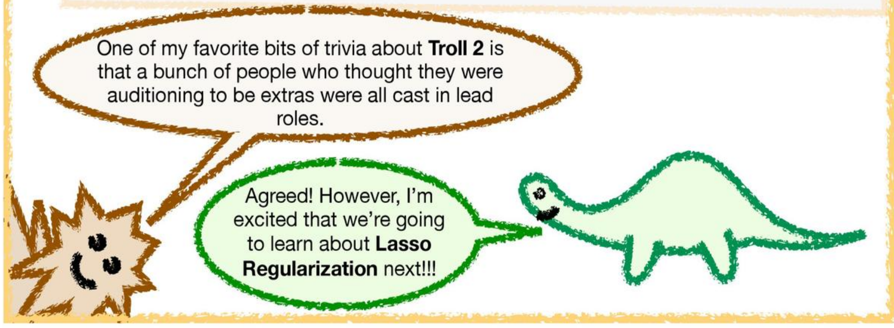
$$= -2 \times \text{Weight}\mathbf{(}\ \text{Height} - (\mathbf{intercept} + \mathbf{slope} \times \text{Weight})\ \mathbf{)}$$

$$+ 2 \times \lambda \times \mathbf{slope}$$

When we plug those derivatives into **Gradient Descent** and set the **Learning Rate** to **0.01**, we get the equations for the **new lines**.

Unfortunately, for more complicated models, or for **Lasso Regularization**, or for combinations of the two, we have to use a different approach that's outside of the scope of this book. However, interested readers can learn more by following this link:

https://web.stanford.edu/~hastie/TALKS/nips2005.pdf



Lasso/Absolute Value/L1 Regularization: Details Part 1

1 **Lasso Regularization**, also called **Absolute Value** or **L1 Regularization**, replaces the square that we use in the **Ridge Penalty** with the absolute value.

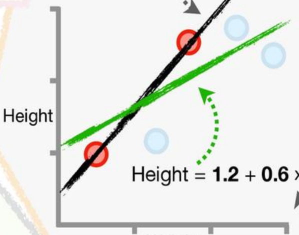
In the **Ridge Penalty**, we square the parameter.

In contrast, in the **Lasso Penalty**, we take the parameter's absolute value.

$\mathbf{SSR} + \boldsymbol{\lambda} \times \text{slope}^2$ **vs.** $\mathbf{SSR} + \boldsymbol{\lambda} \times |\text{slope}|$

2 For example, let's compare the **Lasso** scores for the **black line**, which fits the **Training Data** perfectly…

Height = **0.4** + **1.3** x Weight



Height = **1.2** + **0.6** x Weight

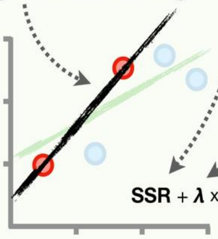
…and for the **green line**, which doesn't fit the **Training Data** as well.

3 For the **black line**, which fits the **Training Data** perfectly, the **SSR** is **0**…

…for now we'll let $\boldsymbol{\lambda}$ **= 1**…

…and the absolute value of the slope is **1.3**…

Height = **0.4** + **1.3** x Weight

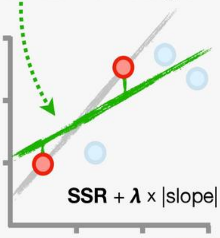


$\mathbf{SSR} + \boldsymbol{\lambda} \times |\text{slope}| = 0 + 1 \times 1.3 \boxed{= 1.3}$

…which give us **1.3** as the **Lasso** score for the **black line**.

4 In contrast, the **green line** has an **SSR = 0.4**…

Height = **1.2** + **0.6** x Weight



…so we plug in **0.4** for the **SSR**, **1** for $\boldsymbol{\lambda}$, and **0.6** for the slope and get **1.0** for the **Lasso** score.

And because **1 < 1.3**, we would pick the **green line**.

Bam.

$\mathbf{SSR} + \boldsymbol{\lambda} \times |\text{slope}| = 0.4 + 1 \times 0.6 \boxed{= 1.0}$

Lasso/Absolute Value/L1 Regularization: Details Part 2

⑤ The big difference between **Ridge** and **Lasso Regularization** is that **Ridge Regularization** can only shrink the parameters to be asymptotically close to **0**. In contrast, **Lasso Regularization** can shrink parameters all the way to **0**.

⑥ For example, if we applied **Ridge** and **Lasso Regularization**, separately, to this fancy model that predicted Height using Weight, Shoe Size, and the Airspeed of a Swallow…

Height = **intercept** + **slope**$_w$ x Weight + **slope**$_s$ x Shoe Size + **slope**$_a$ x Airspeed of a Swallow

…then regardless of how useless the variable Airspeed of a Swallow is for making predictions, **Ridge Regularization** will never get **slope**$_a$ = **0**.

In contrast, if Airspeed of a Swallow was totally useless, then **Lasso Regularization** can make **slope**$_a$ = **0**, resulting in a simpler model that no longer includes Airspeed of a Swallow.

Height = **intercept** + **slope**$_w$ x Weight + **slope**$_s$ x Shoe Size + ~~**slope**$_a$ x Airspeed of a Swallow~~

Height = **intercept** + **slope**$_w$ x Weight + **slope**$_s$ x Shoe Size

⑦ Thus, **Lasso Regularization** can exclude useless variables from the model and, in general, tends to perform well when we need to remove a lot of useless variables from a model.

In contrast, **Ridge Regularization** tends to perform better when most of the variables are useful.

BAM!!!

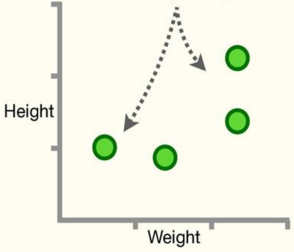
NOTE: Ridge and **Lasso Regularization** are frequently combined to get the best of both worlds.

DOUBLE BAM!!!

Ridge vs. Lasso Regularization: Details Part 1

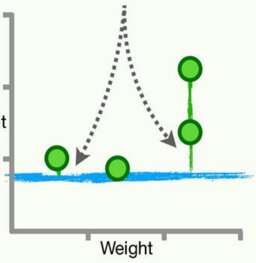
1 The critical thing to know about **Ridge** vs. **Lasso Regularization** is that **Ridge** works better when most of the variables are *useful* and **Lasso** works better when a lot of the variables are *useless*, and **Ridge** and **Lasso** can be combined to get the best of both worlds. That said, people frequently ask why only **Lasso** can set parameter values (e.g, the slope) to **0** and **Ridge** cannot. What follows is an illustration of this difference, so if you're interested, read on!!!

2 As always, we'll start with a super simple dataset where we want to use Weight to predict Height.

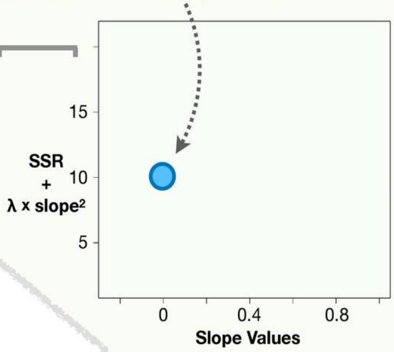


3 Now let's fit a **blue horizontal line** to the data, which is a terrible fit, but we'll improve it in a bit…

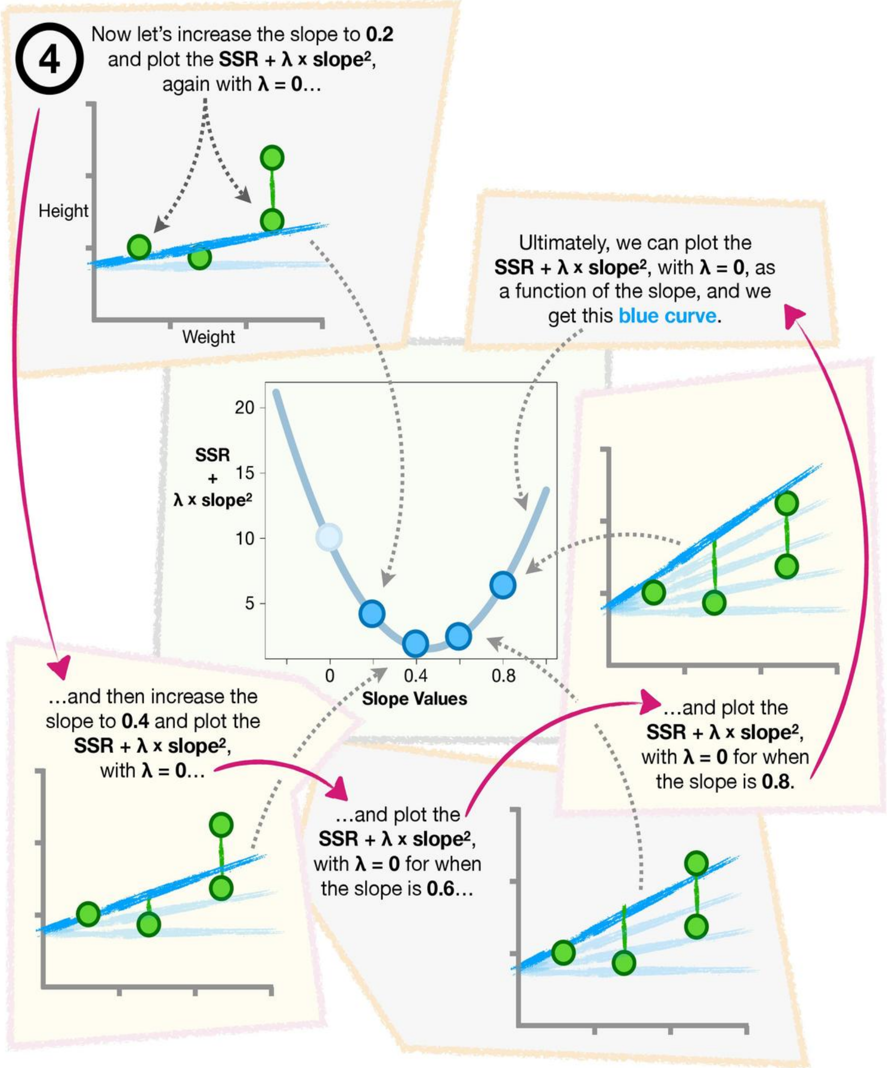
…and let's calculate the **Ridge Score**, **SSR + λ x slope²**, with **λ = 0**. In other words, since **λ = 0**, the **Ridge Score** will be the same as the **SSR**…



…and now let's plot the **Ridge Score** and the corresponding slope of the **blue horizontal line**, **0**, on a graph that has **SSR + λ x slope²** on the y-axis and slope on the x-axis.



Ridge vs. Lasso Regularization: Details Part 2



④ Now let's increase the slope to **0.2** and plot the **SSR + λ x slope²**, again with **λ = 0**…

…and then increase the slope to **0.4** and plot the **SSR + λ x slope²**, with **λ = 0**…

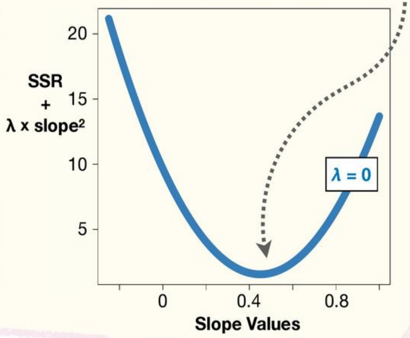
…and plot the **SSR + λ x slope²**, with **λ = 0** for when the slope is **0.6**…

…and plot the **SSR + λ x slope²**, with **λ = 0** for when the slope is **0.8**.

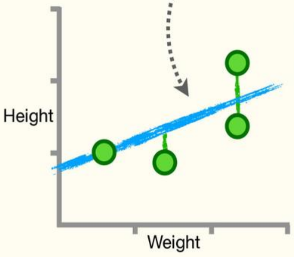
Ultimately, we can plot the **SSR + λ x slope²**, with **λ = 0**, as a function of the slope, and we get this **blue curve**.

Ridge vs. Lasso Regularization: Details Part 3

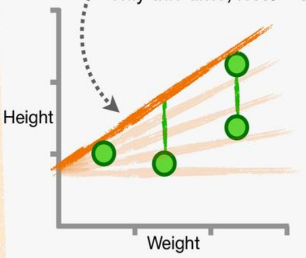
5 Now that we have this blue curve, we can see that the value for the slope that minimizes the **Ridge Score** is just over **0.4**…

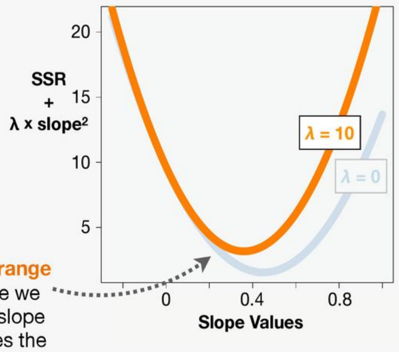


…which corresponds to this blue line.



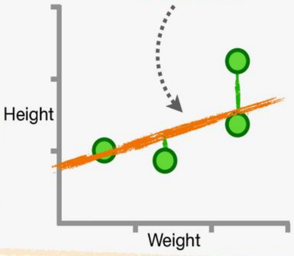
6 Now, just like before, let's calculate the **SSR + λ x slope²** for a bunch of different slopes, only this time, let **λ** = **10**.

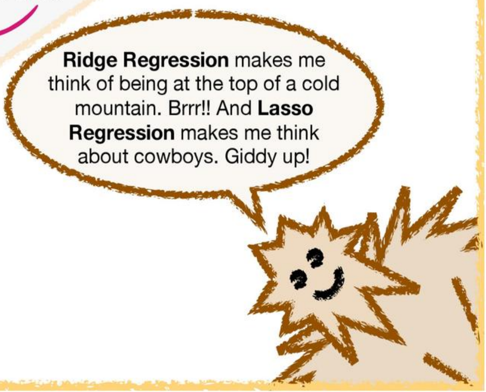




We get this orange curve, where we see that the slope that minimizes the **Ridge Score** is just under **0.4**…

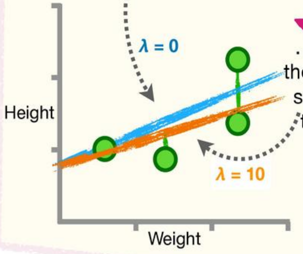
…which corresponds this orange line.





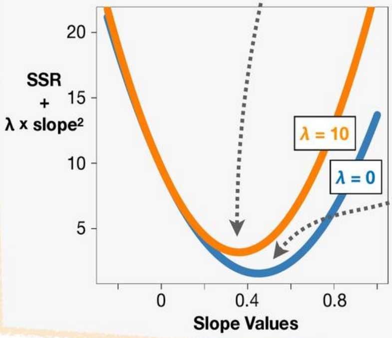
Ridge vs. Lasso Regularization: Details Part 4

7 When we compare the **blue line**, which has the optimal slope when **λ** = **0**…



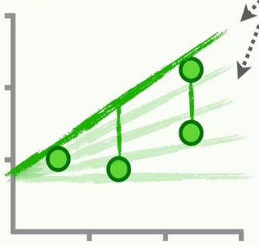
…to the **orange line**, which has the optimal slope when **λ** = **10**, we see that **λ** = **10**, which increases the **Ridge Penalty**, results in a smaller slope.

8 Likewise, we can see that when **λ** = **10**, the lowest point on the **orange curve** corresponds to a slope value closer to **0**…

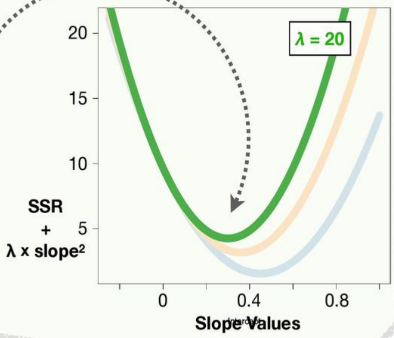


…than on the **blue curve**, when **λ** = **0**.

9 Now we calculate the **SSR + λ x slope^2** for a bunch of different slopes with **λ** = **20**…

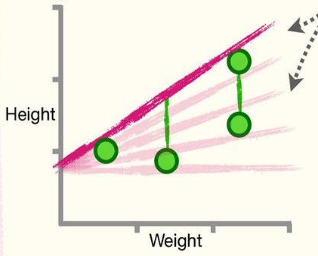


…and we get this **green curve**, and again, we see that the lowest point corresponds to a slope that's closer to **0**.

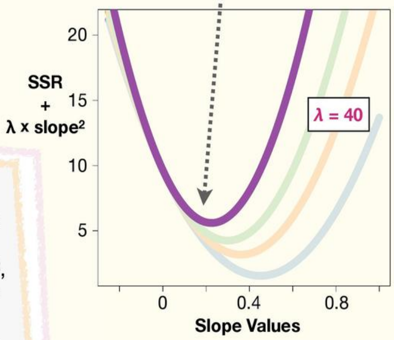


Ridge vs. Lasso Regularization: Details Part 5

10 Lastly, we calculate the **SSR + λ x slope^2** for a bunch of different slopes with **λ = 40**…



…and we get this **purple curve**, and again, we see that the lowest point corresponds to a slope that's closer to, but not quite, **0**.



11 To summarize what we've seen so far:

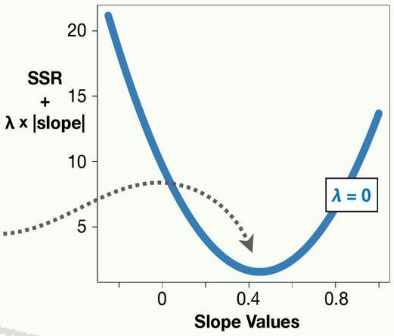
1) When we calculate **SSR + λ x slope^2**, we get a nice curve for different values for the slope.

2) When we increase **λ**, the lowest point in the curve corresponds to a slope value closer to **0**, but not quite **0**.

BAM!!!

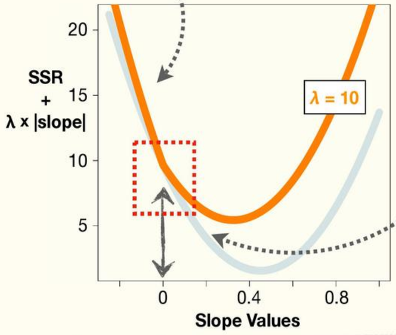
Now let's do the same thing using the **Lasso Penalty**, and calculate **SSR + λ x |slope|**!!!

12 When we calculate the **Lasso** score for a bunch of different values for the slope and let **λ = 0**, we get this **blue curve**, just like before, because when **λ = 0**, the penalty goes away and we're left with the **SSR**. And, just like before, we see the lowest point in the **blue curve** corresponds to a slope value that's just over **0.4**.



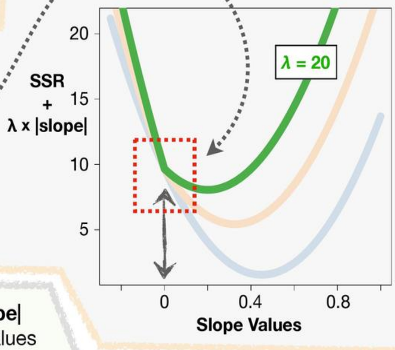
Ridge vs. Lasso Regularization: Details Part 6

⑬ Now when we calculate the **SSR + λ x |slope|** and let **λ** = **10** for a bunch of different values for the slope, we get this **orange shape**, and the lowest point corresponds to a slope just lower than **0.4**.

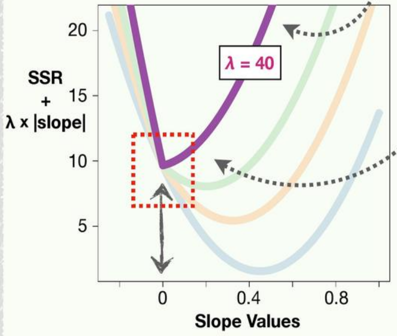


However, unlike before, this **orange shape** has a slight kink when the slope is **0**.

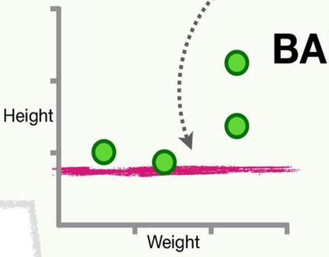
⑭ When we calculate the **SSR + λ x |slope|** and let **λ** = **20** for a bunch of different values for the slope, we get this **green shape** with a kink where the slope is **0** that's now more prominent.



⑮ When we calculate the **SSR + λ x |slope|** and let **λ** = **40** for a bunch of different values for the slope, we get this **purple shape** with a kink where the slope is **0**, and now that kink is also the lowest point in the shape…



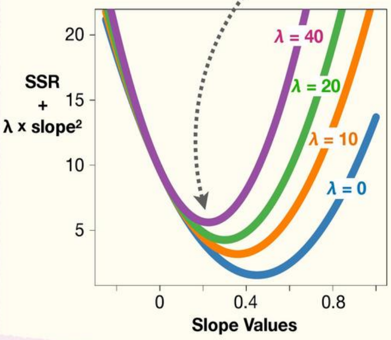
…and that means when **λ** = **40**, the slope of the optimal line is **0**.



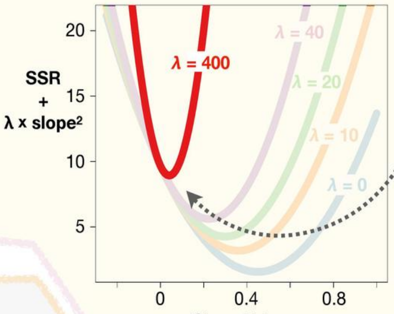
BAM!!!

Ridge vs. Lasso Regularization: Details Part 7

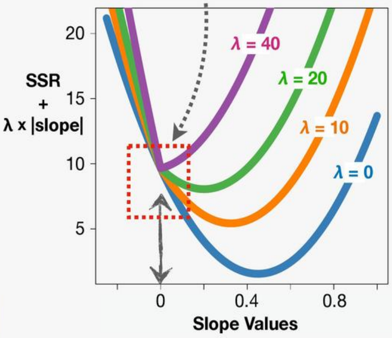
⑯ In summary, when we increase the **Ridge, Squared, or L2 Penalty** the optimal value for the slope shifts toward **0**, but we retain a nice **parabola** shape, and the optimal slope is never **0** itself.



NOTE: Even if we increase ***λ*** all the way to **400**, the **Ridge Penalty** gives us a smooth **red curve** and the lowest point corresponds to a slope value slightly greater than **0**.



⑰ In contrast, when we increase the **Lasso, Absolute Value**, or **L1 Penalty**, the optimal value for the slope shifts toward **0**, and since we have a kink at **0**, **0** ends up being the optimal slope.



TRIPLE BAM!!!

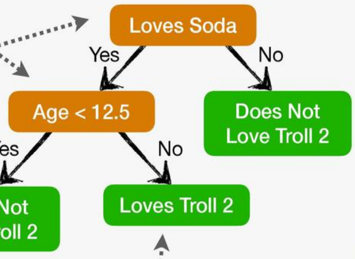
Chapter 10

Decision Trees!!!

Classification and Regression Trees: Main Ideas

1 There are two types of **Trees** in machine learning: trees for **Classification** and trees for **Regression**.

2 **Classification Trees** classify people or things into two or more *discrete* categories. This example **Classification Tree** classifies people into two groups:



…those who *do not* love Troll 2…

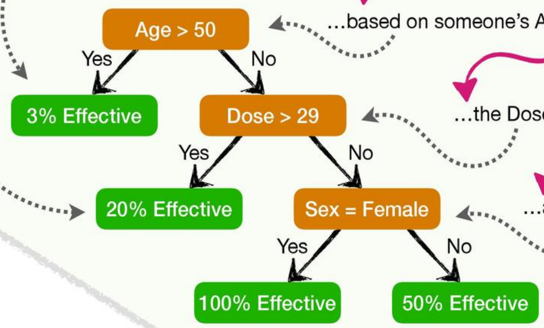
…and those who love Troll 2.

3 In contrast, **Regression Trees** try to predict a *continuous* value. This example **Regression Tree** tries to predict how Effective a drug will be…

…based on someone's Age…

…the Dose…

…and their Sex.



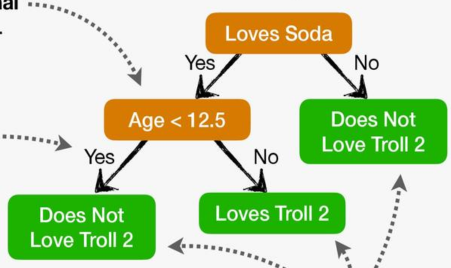
4 In this chapter, we'll cover the **Main Ideas** behind **Classification Trees** and **Regression Trees** and describe the most commonly used methods to build them. But first, it's time for the dreaded…

Terminology Alert!!! Decision Tree Lingo

① The good news is that **Decision Trees** are pretty simple, and there's not too much terminology.

The very top of the tree is called the **Root Node** or just the **Root**.

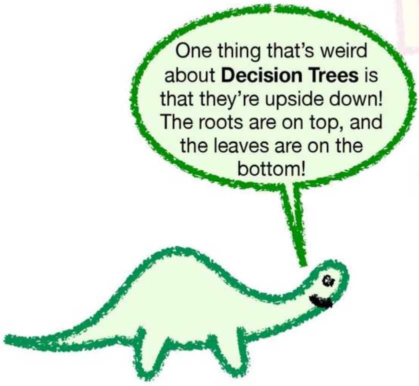
This is called an **Internal Node** or just a **Node**.



The arrows are called **Branches**. In this example, the **Branches** are labeled with **Yes** or **No**, but usually it's assumed that if a statement in a **Node** is ***True***, you go to the *Left*, and if it's ***False***, you go to the *Right*.

These are called **Leaf Nodes** or just **Leaves**.

② Now that we know the lingo, let's learn about **Classification Trees**!!!



Decision Trees
Part One:

Classification Trees

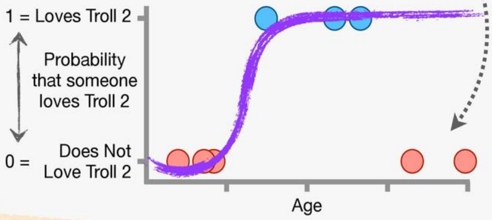
Classification Trees: Main Ideas

① **The Problem:** We have a mixture of **discrete** and **continuous** data…

Loves Popcorn	Loves Soda	Age	Loves Troll 2
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

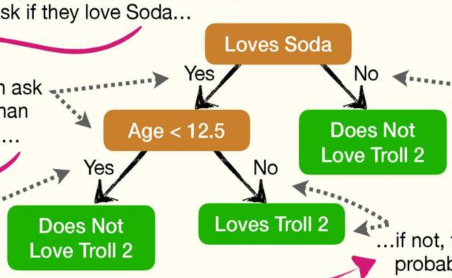
…that we want to use to predict if someone loves Troll 2, the **1990** blockbuster movie that was neither about trolls nor a sequel.

Unfortunately, we can't use **Logistic Regression** with these data because when we plot Age vs. Loves Troll 2, we see that fitting an **s-shaped squiggle** to the data would be a terrible idea: both young and old people *do not* love Troll 2, with the people who love Troll 2 in the middle. In this example, an **s-shaped squiggle** will incorrectly classify *all* of the older people.



② **A Solution:** A **Classification Tree**, which can handle all types of data, all types of relationships among the independent variables (the data we're using to make predictions, like Loves Soda and Age), and all kinds of relationships with the dependent variable (the thing we want to predict, which in this case is Loves Troll 2).

Classification Trees are relatively easy to interpret and use. If you meet a new person and want to decide if they love Troll 2 or not, you simply start at the top and ask if they love Soda…



…if they do, then ask if they're less than **12.5** years old…

…if they are, then they probably *do not* love Troll 2…

…if not, then they probably love Troll 2…

…and if they don't love Soda to begin with, they probably *do not* love Troll 2.

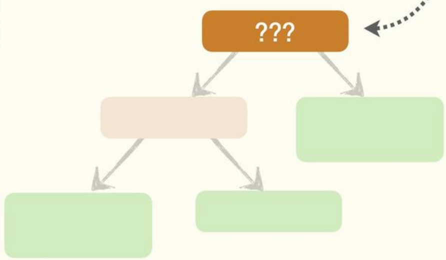
BAM!!!

Building a Classification Tree: Step-by-Step

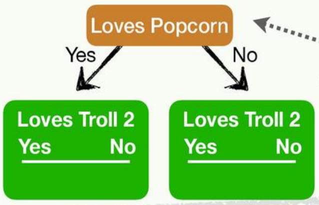
① Given this **Training Dataset**, we want to build a **Classification Tree** that uses Loves Popcorn, Loves Soda, and Age to predict whether or not someone will love Troll 2.

Loves Popcorn	Loves Soda	Age	Loves Troll 2
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

② The first thing we do is decide whether Loves Popcorn, Loves Soda, or Age should be the question we ask at the very top of the tree.



③ To make that decision, we'll start by looking at how well Loves Popcorn predicts whether or not someone loves Troll 2…

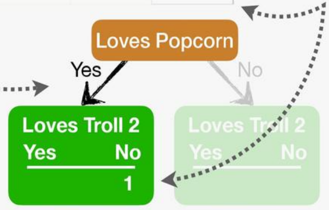


…by making a super simple tree that only asks if someone loves Popcorn and running the data down it.

④ For example, the first person in the **Training Data** loves Popcorn, so they go to the **Leaf** on the left…

Loves Popcorn	Loves Soda	Age	Loves Troll 2
Yes	Yes	7	No

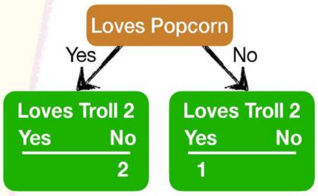
…and because they *do not* love Troll 2, we'll put a **1** under the word **No**.



Building a Classification Tree: Step-by-Step

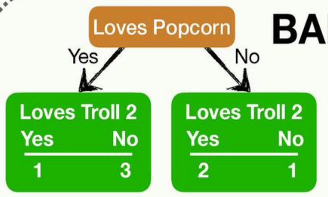
5 The second person also loves Popcorn, so they also go to the **Leaf** on the *left*, and because they do not love Troll 2, we increase **No** to **2**.

Loves Popcorn	Loves Soda	Age	Loves Troll 2
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No



6 The third person *does not* love Popcorn, so they go to the **Leaf** on the *right*, but they love Troll 2, so we put a **1** under the word **Yes**.

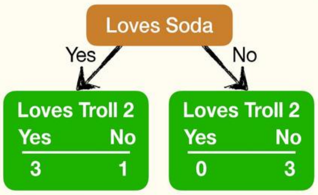
7 Likewise, we run the remaining rows down the tree, keeping track of whether or not each person loves Troll 2 or not.



BAM!!!

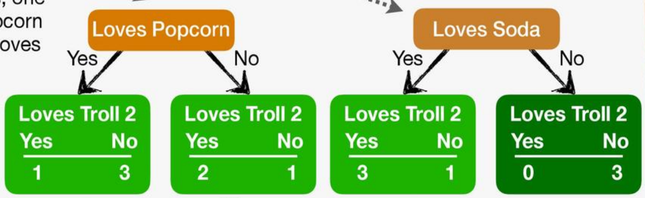
8 Then we do the same thing for Loves Soda.

Loves Popcorn	Loves Soda	Age	Loves Troll 2
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
...	...	...	...



Building a Classification Tree: Step-by-Step

9 Now, looking at the two little trees, one for Loves Popcorn and one for Loves Soda…



…we see that these three **Leaves** contain mixtures of people who love and do not love Troll 2.

In contrast, this **Leaf** only contains people who *do not* love Troll 2.

TERMINOLOGY ALERT!!!

Leaves that contain mixtures of classifications are called **Impure**.

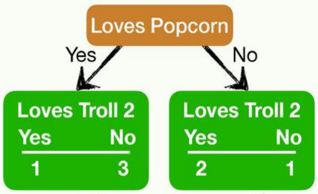
10 Because both **Leaves** in the Loves Popcorn tree are **Impure** and only one **Leaf** in the Loves Soda tree is **Impure**…

…it seems like Loves Soda does a better job classifying who loves and does not love Troll 2, but it would be nice if we could *quantify* the differences between Loves Popcorn and Loves Soda.

11 The good news is that there are several ways to quantify the **Impurity** of **Leaves** and **Trees**.

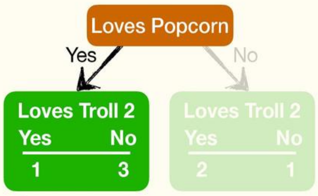
One of the most popular methods is called **Gini Impurity**, but there are also fancy-sounding methods like **Entropy** and **Information Gain**.

In theory, all of the methods give similar results, so we'll focus on **Gini Impurity** since it's very popular and I think it's the most straightforward.



We'll start by calculating the **Gini Impurity** to quantify the **Impurity** in the **Leaves** for loves Popcorn.

12 To calculate the **Gini Impurity** for Loves Popcorn, first we calculate the **Gini Impurity** for each individual **Leaf**. So, let's start by plugging the numbers from the *left* **Leaf** into the equation for **Gini Impurity**.



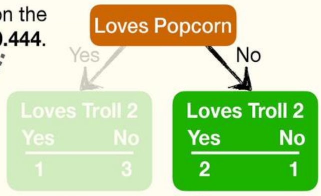
13 **Gini Impurity** for a **Leaf** $= 1 - (\text{the probability of "yes"})^2 - (\text{the probability of "no"})^2$

14 For the **Leaf** on the *left*, when we plug the numbers, **Yes = 1**, **No = 3**, and **Total = 1 + 3**, into the equation for **Gini Impurity**, we get **0.375**.

$$= 1 - \left(\frac{\text{The number for Yes}}{\text{The total for the Leaf}}\right)^2 - \left(\frac{\text{The number for No}}{\text{The total for the Leaf}}\right)^2$$

$$= 1 - \left(\frac{1}{1 + 3}\right)^2 - \left(\frac{3}{1 + 3}\right)^2 \boxed{= 0.375}$$

15 For the **Leaf** on the *right*, we get **0.444**.



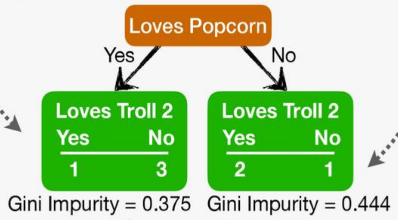
Gini Impurity for a **Leaf** $= 1 - (\text{the probability of "yes"})^2 - (\text{the probability of "no"})^2$

$$= 1 - \left(\frac{2}{2 + 1}\right)^2 - \left(\frac{1}{2 + 1}\right)^2 \boxed{= 0.444}$$

Building a Classification Tree: Step-by-Step

16 Now, because the **Leaf** on the *left* has **4** people in it…

…and the **Leaf** on the *right* only has **3**, the **Leaves** do not represent the same number of people.



So, to compensate for the differences in the number of people in each **Leaf**, the total **Gini Impurity** for Loves Popcorn is the **Weighted Average of the two Leaf Impurities.**

17 Total **Gini Impurity** = weighted average of **Gini Impurities** for the **Leaves**

18 The weight for the *left* **Leaf** is the total number of people in the **Leaf**, **4**…

$$\textbf{Total Gini Impurity} = \left(\frac{4}{4+3}\right)0.375 + \left(\frac{3}{4+3}\right)0.444 \boxed{= 0.405}$$

…divided by the total number of people in both **Leaves**, **7**…

…then we multiply that weight by its associated **Gini Impurity, 0.375.**

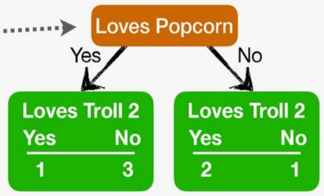
Now we add to that the weight for the *right* **Leaf**, the total number of people in the **Leaf**, **3**, divided by to the total in both **Leaves**, **7**…

…multiplied by the associated **Gini Impurity, 0.444**…

…and when we do the math, we get **0.405. BAM!!!**

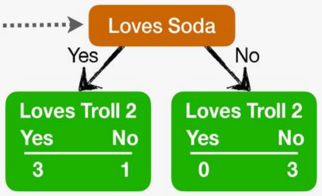
Building a Classification Tree: Step-by-Step

19 Now that we've calculated the **Gini Impurity** for Loves Popcorn, **0.405**…



Gini Impurity for **Loves Popcorn** = **0.405**

…we can follow the same steps to calculate the **Gini Impurity** for Loves Soda, **0.214**.



Gini Impurity for **Loves Soda** = **0.214**

20 The lower **Gini Impurity** for Loves Soda, **0.214**, confirms what we suspected earlier, that Loves Soda does a better job classifying people who love and do not love Troll 2. However, now that we've *quantified* the difference, we no longer have to rely on intuition. Bam!

21 Now we need to calculate the **Gini Impurity** for Age.

However, because Age contains numeric data, and not just **Yes/No** values, calculating the **Gini Impurity** is a little more involved.

Loves Popcorn	Loves Soda	Age	Loves Troll 2
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

Normally, the first thing we do is sort the rows by Age, from lowest to highest, but in this case, the data were already sorted, so we can skip this step.

22 The next thing we do is calculate the average Age for all adjacent rows.

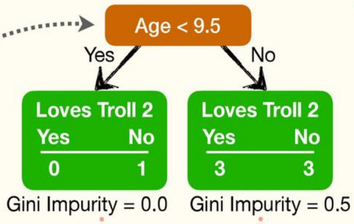
	Age	Loves Troll 2
	7	No
9.5		
	12	No
15		
	18	Yes
26.5		
	35	Yes
36.5		
	38	Yes
44		
	50	No
66.5		
	83	No

Building a Classification Tree: Step-by-Step

23 Lastly, we calculate the **Gini Impurity** values for each average Age.

Threshold	Age	Loves Troll 2
	7	No
9.5		
	12	No
15		
	18	Yes
26.5		
	35	Yes
36.5		
	38	Yes
44		
	50	No
66.5		
	83	No

For example, the first average Age is **9.5**, so we use **9.5** as the threshold for splitting the rows into **2** leaves…



$$\textbf{Total Gini Impurity} = \left(\frac{1}{1+6}\right)0.0 + \left(\frac{6}{1+6}\right)0.5$$

…and when we do the math, we get **0.429**.

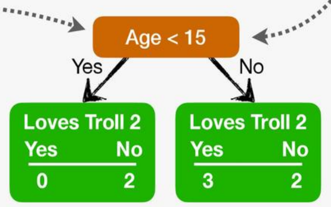
$$= 0.429$$

24 Ultimately, we end up with a **Gini Impurity** for each potential threshold for Age…

Threshold	Age	Loves Troll 2	
	7	No	
9.5			Gini Impurity = 0.429
	12	No	
15			Gini Impurity = 0.343
	18	Yes	
26.5			Gini Impurity = 0.476
	35	Yes	
36.5			Gini Impurity = 0.476
	38	Yes	
44			Gini Impurity = 0.343
	50	No	
66.5			Gini Impurity = 0.429
	83	No	

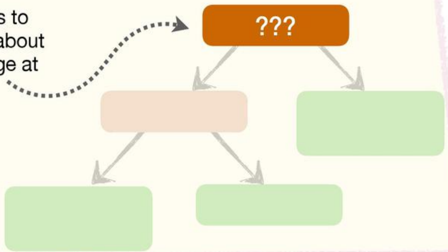
…and then we identify the thresholds with the lowest **Impurities**, and because the candidate thresholds **15** and **44** are tied for the lowest **Impurity**, **0.343**, we can pick either one for the **Root**. In this case, we'll pick **15**.

BAM!!!



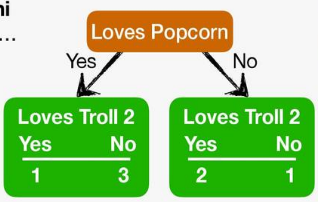
Building a Classification Tree: Step-by-Step

25 Now remember: our first goal was to determine whether we should ask about Loves Popcorn, Loves Soda, or Age at the very top of the tree…

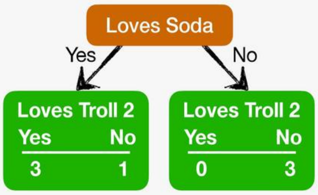


26 …so we calculated the **Gini Impurities** for each feature…

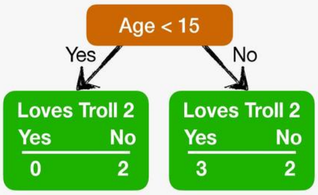
Gini Impurity for **Loves Popcorn** = **0.405**



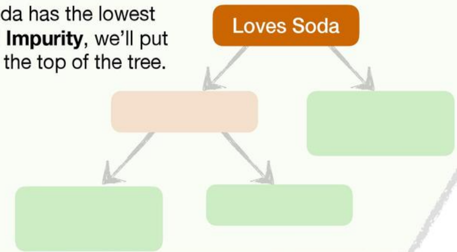
Gini Impurity for **Loves Soda** = **0.214**



Gini Impurity for **Age < 15** = **0.343**



27 …and because Loves Soda has the lowest **Gini Impurity**, we'll put it at the top of the tree.



BAM!!!

Building a Classification Tree: Step-by-Step

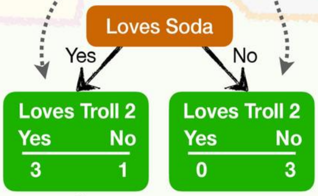
(28) With Loves Soda at the top of the tree, the **4** people who love Soda, including **3** who love Troll 2 and **1** who does not, go to the **Node** on the *left*…

Loves Popcorn	Loves Soda	Age	Loves Troll 2
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

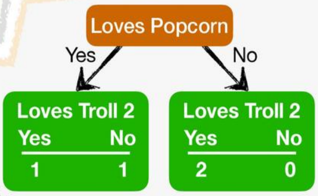
(29) …and the **3** people who *do not* love Soda, all of whom *do not* love Troll 2, go to the **Node** on the *right*.

Loves Popcorn	Loves Soda	Age	Loves Troll 2
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

(30) Now, because the **Node** on the *left* is **Impure**, we can split the **4** people in it based on Loves Popcorn or Age and calculate the **Gini Impurities**.

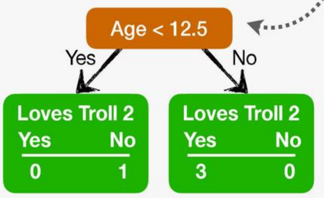


(31) When we split the **4** people who love Soda based on whether or not they love Popcorn, the **Gini Impurity** is **0.25**.

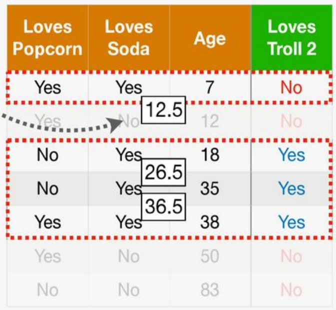


Gini Impurity = 0.25

However, when we split the **4** people based on Age **<** **12.5**, the **Gini Impurity** is **0**.



Gini Impurity = 0.0



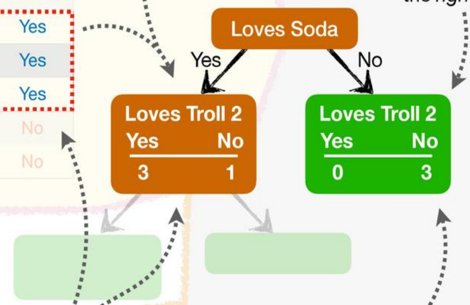
Loves Popcorn	Loves Soda	Age	Loves Troll 2
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

Building a Classification Tree: Step-by-Step

32 Now remember, earlier we put Loves Soda in the **Root** because splitting every person in the **Training Data** based on whether or not they love Soda gave us the *lowest* **Gini Impurity**. So, the **4** people who love Soda went to the *left* **Node**…

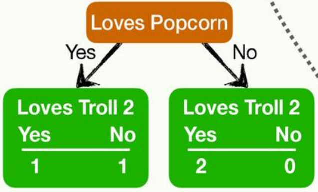
Loves Popcorn	Loves Soda	Age	Loves Troll 2
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

33 …and the **3** people who *do not* love Soda, all of whom *do not* love Troll 2, went to the **Node** on the *right*.

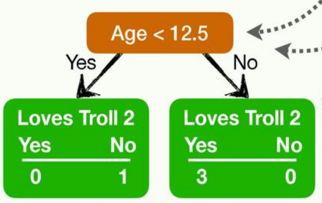


Now, because *everyone* in this **Node** *does not* love **Troll 2**, it becomes a **Leaf**, because there's no point in splitting the people up into smaller groups.

34 In contrast, because the **4** people who love Soda are a mixture of people who *do* and *do not* love Troll 2, we build simple trees with them based on Loves Popcorn and Age…



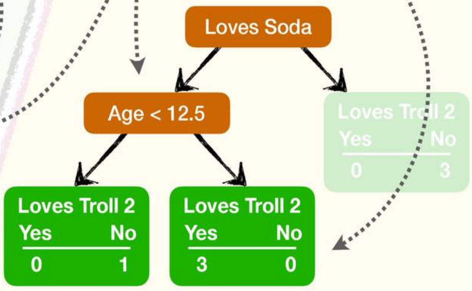
Gini Impurity = 0.25



Gini Impurity = 0.0

35 …and because Age **< 12.5** resulted in the lowest **Gini Impurity**, **0**, we add it to the tree.

And the new **Nodes** are **Leaves** because neither is **Impure**.

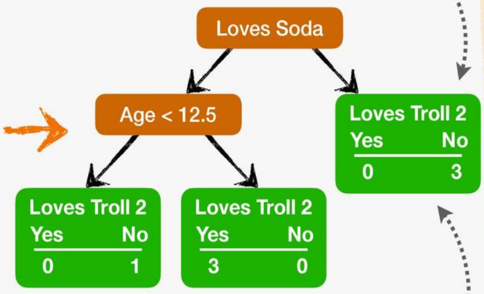


Building a Classification Tree: Step-by-Step

36 At this point, we've created a **Tree** from the **Training Data**.

Now the only thing remaining is to assign output values for each **Leaf**.

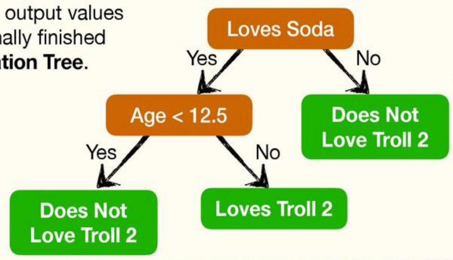
Loves Popcorn	Loves Soda	Age	Loves Troll 2
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

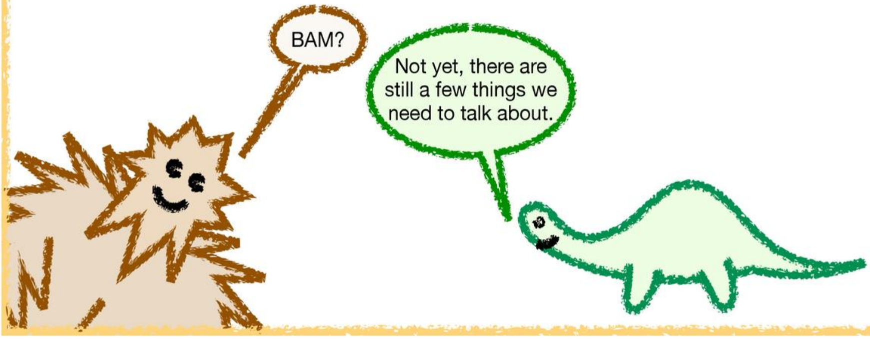

Loves Soda
Age < 12.5
Loves Troll 2 Yes No 0 3
Loves Troll 2 Yes No 0 1
Loves Troll 2 Yes No 3 0

37 Generally speaking, the output of a **Leaf** is whatever category that has the most counts.

In other words, because the majority of the people in this **Leaf** *do not* love Troll 2, its output value is does not love Troll 2.

38 Hooray!!! After assigning output values to each **Leaf**, we've finally finished building a **Classification Tree**.


Loves Soda
Yes
No
Age < 12.5
Does Not Love Troll 2
Yes
No
Does Not Love Troll 2
Loves Troll 2



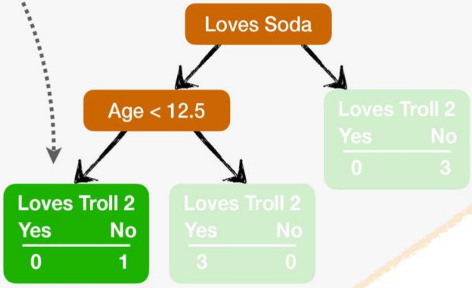
Building a Classification Tree: Step-by-Step

39 When we built this tree, only one person in the **Training Data** made it to this **Leaf**…

…and because so few people in the **Training Data** made it to that **Leaf**, it's hard to have confidence that the tree will do a great job making predictions with future data.

However, in practice, there are two main ways to deal with this type of problem.

Loves Popcorn	Loves Soda	Age	Loves Troll 2
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

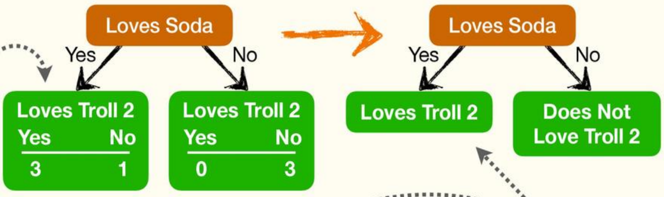


40 One method is called **Pruning**, but we'll save that topic for **The StatQuest Illustrated Guide to Tree-Based Machine Learning!!!**

41 Alternatively, we can put limits on how trees grow, for example, by requiring **3** or more people per **Leaf**.

If we did that with our **Training Data**, we would end up with this tree, and this **Leaf** would be **Impure**…

…but we would also have a better sense of the accuracy of our prediction because we know that only **75%** of the people in the **Leaf** love Troll 2.



NOTE: When we build a tree, we don't know in advance if it's better to require **3** people per **Leaf** or some other number, so we try a bunch, use **Cross Validation,** and pick the number that works best.

ALSO NOTE: Even though this **Leaf** is **Impure**, it still needs an output value, and because most of the people in this **Leaf** love Troll 2, that will be the output value.

BAM!!!

Now let's summarize how to build a **Classification Tree**.

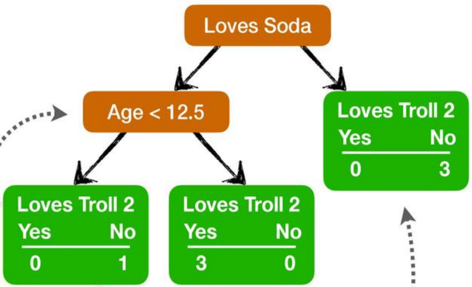
Building a Classification Tree: Summary

1 From the entire **Training Dataset**, we used **Gini Impurity** to select Loves Soda for the **Root** of the tree.

As a result, the **4** people who love Soda went to the *left* and the **3** people who do not love Soda went to the *right*.

BAM!

Loves Popcorn	Loves Soda	Age	Loves Troll 2
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

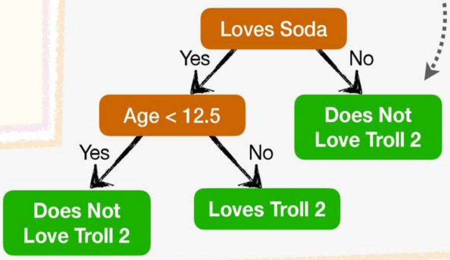


2 Then we used the **4** people who love Soda, which were a mixture of people who *do* and *do not* love Troll 2, to calculate **Gini Impurities** and selected **Age < 12.5** for the next **Node**.

Loves Popcorn	Loves Soda	Age	Loves Troll 2
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

Double BAM!!

3 Then we selected output values for each **Leaf** by picking the categories with the highest counts.

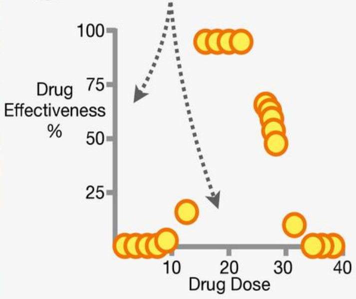


Decision Trees
Part Deux:

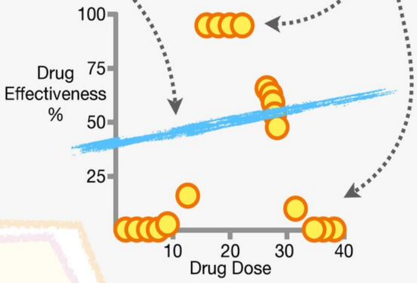
Regression Trees

Regression Trees: Main Ideas Part 1

① **The Problem:** We have this **Training Dataset** that consists of Drug Effectiveness for different Doses…

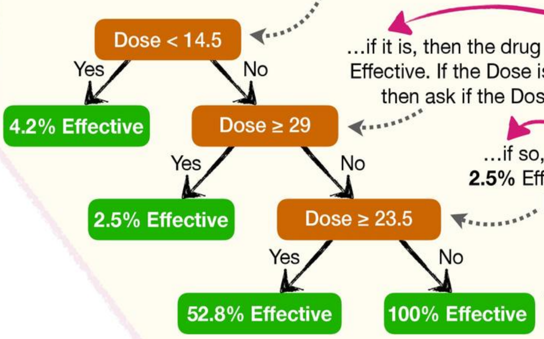


…and fitting a **straight line** to the data would result in terrible predictions because there are clusters of *ineffective* Doses that surround the *effective* Doses.



② **A Solution:** We can use a **Regression Tree**, which, just like a **Classification Tree**, can handle all types of data and all types of relationships among variables to make decisions, but now the output is a *continuous* value, which, in this case, is Drug Effectiveness.

Just like **Classification Trees**, **Regression Trees** are relatively easy to interpret and use. In this example, if you're given a new Dose and want to know how Effective it will be, you start at the top and ask if the Dose is **< 14.5**…



…if it is, then the drug will be **4.2%** Effective. If the Dose is not **< 14.5**, then ask if the Dose **≥ 29**…

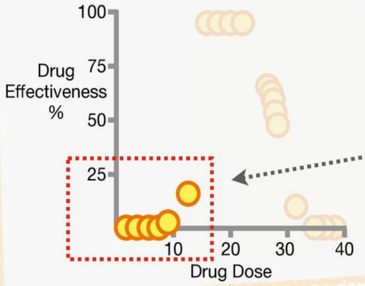
…if so, then the drug will be **2.5%** Effective. If not, ask if the Dose **≥ 23.5**…

…if so, then the drug will be **52.8%** Effective. If not, the drug will be **100%** Effective…

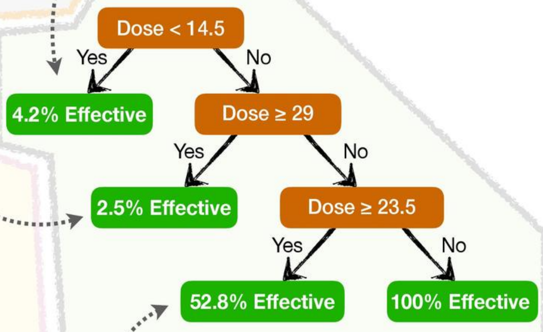
BAM!!!

Regression Trees: Main Ideas Part 2

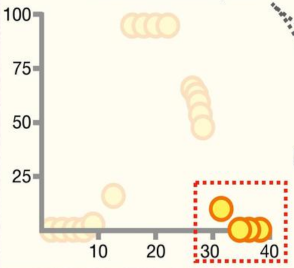
3 In this example, the **Regression Tree** makes good predictions because each **Leaf** corresponds to a different cluster of points in the graph.



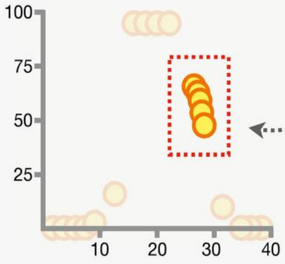
If we have a Dose **< 14.5**, then the output from the **Regression Tree** is the *average* Effectiveness from these **6** measurements, which is **4.2%**.



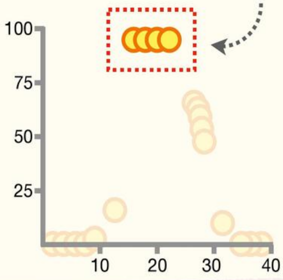
4 If the Dose is **≥ 29**, then the output is the *average* Effectiveness from these **4** measurements, **2.5%**.



5 If the Dose is between **23.5** and **29**, then the output is the *average* Effectiveness from these **5** measurements, **52.8%**.

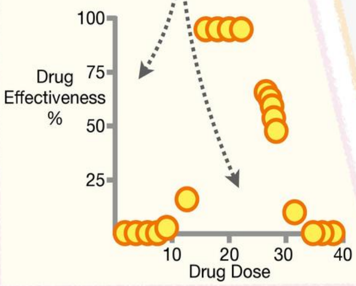


6 If the Dose is between **14.5** and **23.5**, then the output is the *average* Effectiveness from these **4** measurements, **100%**.

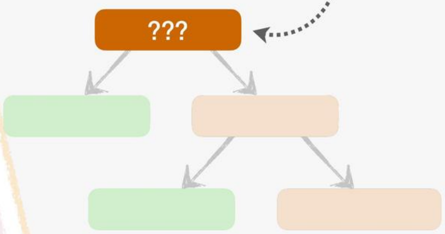


Building a Regression Tree: Step-by-Step

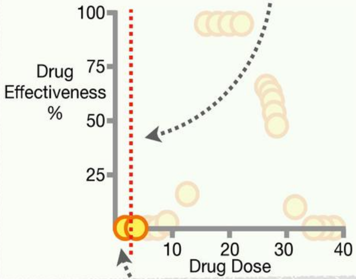
① Given these **Training Data**, we want to build a **Regression Tree** that uses Drug Dose to predict Drug Effectiveness.



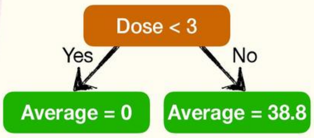
② Just like for **Classification Trees**, the first thing we do for a **Regression Tree** is decide what goes in the **Root**.



③ To make that decision, we calculate the average of the first **2** Doses, which is **3** and corresponds to this **dotted line**…

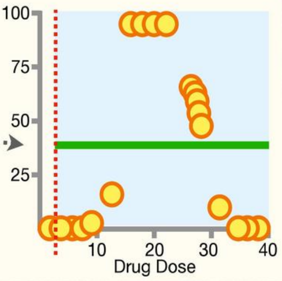


…and then we build a very simple tree that splits the measurements into **2** groups based on whether or not the Dose **<** **3**.



④ Because only one point has a Dose **<** **3**, and its average Effectiveness is **0**, we put **0** in the **Leaf** on the *left*.

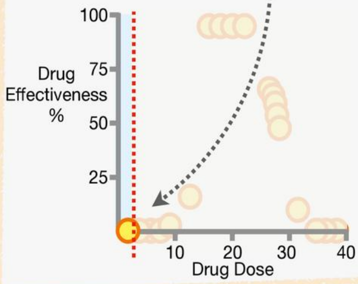
All other points have a **Dose ≥ 3**, and their average Effectiveness is **38.8**, so we put **38.3** in the **Leaf** on the *right*.

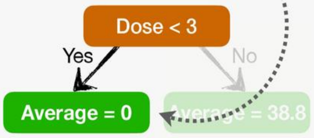


Building a Regression Tree: Step-by-Step

5 For the one point with Dose **< 3**, which has Effectiveness **= 0**…

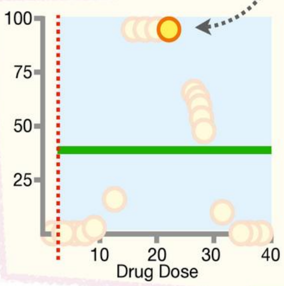
…the **Regression Tree** makes a pretty good prediction, **0**.

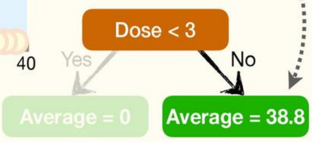




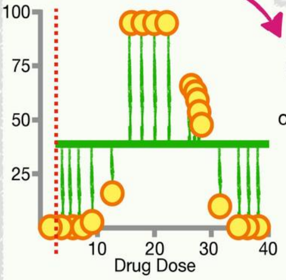
6 In contrast, for this specific point, which has Dose **> 3** and **100%** Effectiveness…

…the tree predicts that the Effectiveness will be **38.8**, which is a pretty bad prediction.





7 We can *visualize* how good or bad the **Regression Tree** is at making predictions by drawing the **Residuals**, the differences between the Observed and Predicted values.



We can also *quantify* how good or bad the predictions are by calculating the **Sum of the Squared Residuals** (**SSR**)…

$$(0 - 0)^2 + (0 - 38.8)^2 + (0 - 38.8)^2 + (0 - 38.8)^2$$

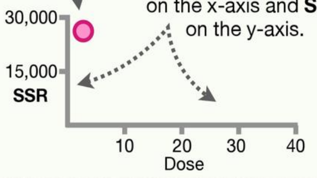
$$+ (5 - 38.8)^2 + (20 - 38.8)^2 + (100 - 38.8)^2$$

$$+ (100 - 38.8)^2 + \dots + (0 - 38.8)^2$$

$$\boxed{= 27{,}468.5}$$

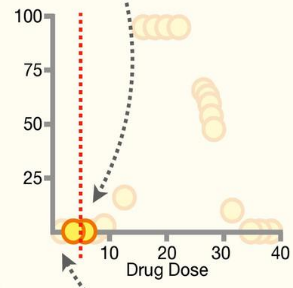
…and when the threshold for the tree is Dose **< 3**, then the **SSR = 27,468.5.**

Lastly, we can compare the **SSR** for different thresholds by plotting them on this graph, which has Dose on the x-axis and **SSR** on the y-axis.

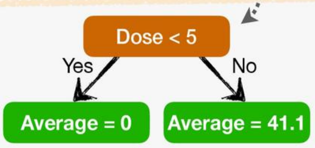


Building a Regression Tree: Step-by-Step

8 Now we shift the Dose threshold to be the average of the second and third measurements in the graph, **5**…

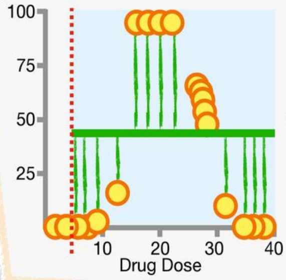


…and we build this super simple tree with Dose **< 5** at the **Root**.



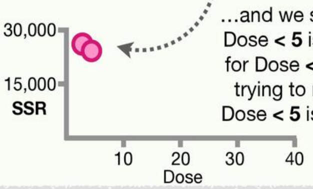
9 Because the average Effectiveness for the **2** points with Dose **< 5** is **0**, we put **0** in the **Leaf** on the *left*.

All of the other points have Dose **≥ 5**, and their average is **41.1**, so we put **41.1** in the **Leaf** on the *right*.



10 Now we calculate and plot the **SSR** for the new threshold, Dose **< 5**…

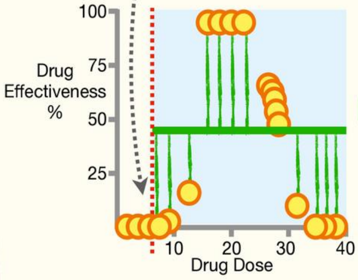
…and we see that the **SSR** for Dose **< 5** is less than the **SSR** for Dose **< 3**, and since we're trying to minimize the **SSR**, Dose **< 5** is a better threshold.

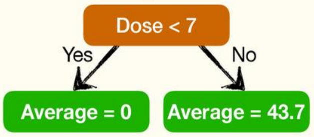


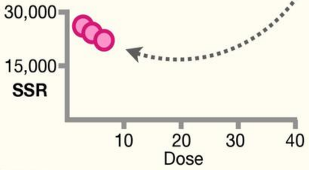
11 Then we shift the threshold to be the average of the third and fourth measurements, **7**…

…and that gives us this tree…

…and this point on the graph.

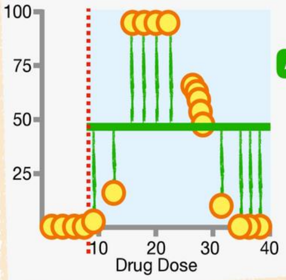


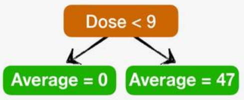


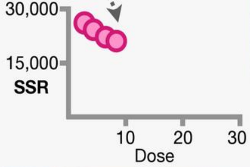


Building a Regression Tree: Step-by-Step

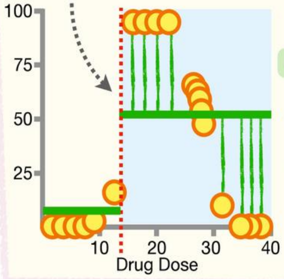
12 Then we just keep shifting the threshold to the average of every pair of consecutive Doses, create the tree, then calculate and plot the **SSR**.

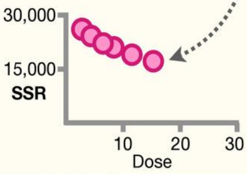




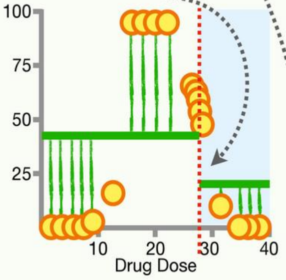


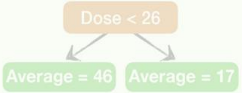
13 After shifting the Dose threshold over **2** more steps, the **SSR** graph looks like this.

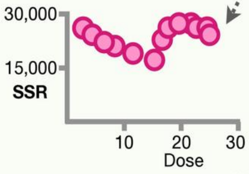




14 Then, after shifting the Dose threshold **7** more times, the **SSR** graph looks like this.

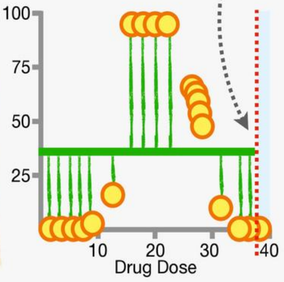


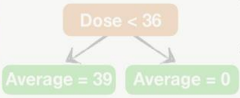


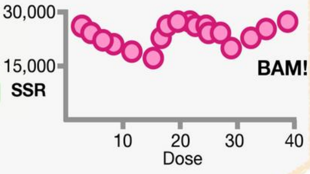


15 And finally, after shifting the Dose threshold all the way to the last pair of Doses…

…the **SSR** graph looks like this.

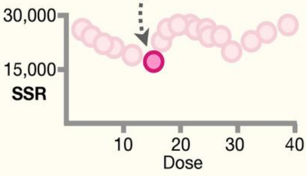




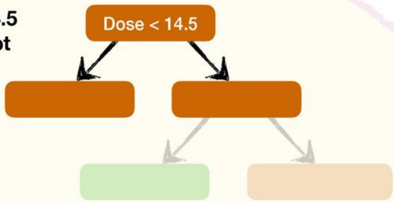


Building a Regression Tree: Step-by-Step

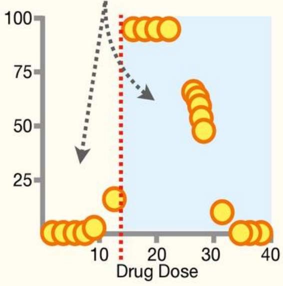
16 Looking at the **SSRs** for each Dose threshold, we see that Dose **< 14.5** had the smallest **SSR**…



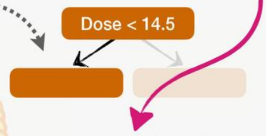
…so Dose **< 14.5** will be the **Root** of the tree…

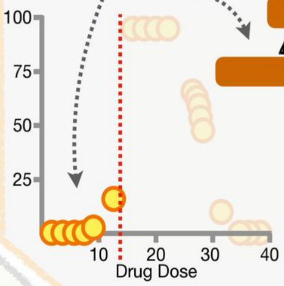


…which corresponds to splitting the measurements into two groups based on whether or not the Dose **< 14.5**.



17 Now, because the threshold in the **Root** of the tree is Dose **< 14.5**, these **6** measurements go into the **Node** on the *left*…

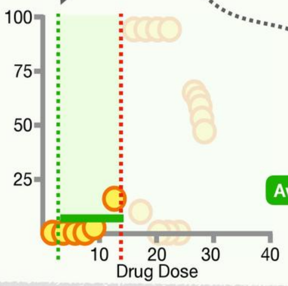


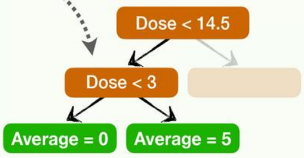


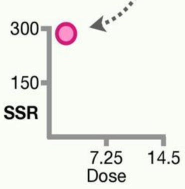
…and, *in theory*, we could subdivide these **6** measurements into smaller groups by repeating what we just did, only this time focusing only on these **6** measurements.

18 In other words, just like before, we can average the first two Doses and use that value, **3**, as a cutoff for splitting the **6** measurements with Dose **< 14.5** into **2** groups…

…then we calculate the **SSR** for just those **6** measurements and plot it on a graph.



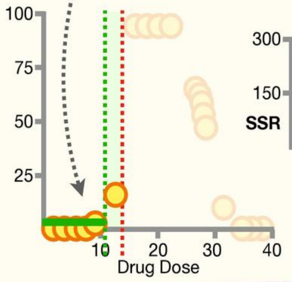


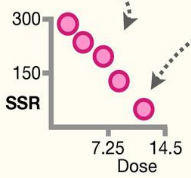


Building a Regression Tree: Step-by-Step

19 And after calculating the **SSR** for each threshold for the **6** measurements with Dose **< 14.5**, we end up with this graph…

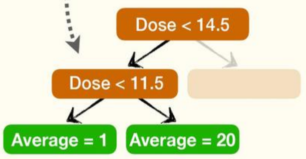
…and then we select the threshold that gives us the lowest **SSR**, Dose **< 11.5**, for the next **Node** in the tree.



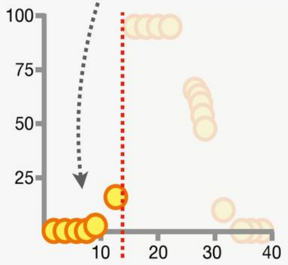


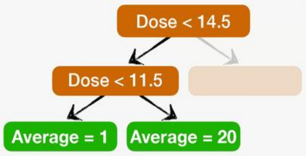
BAM?

No. No bam.



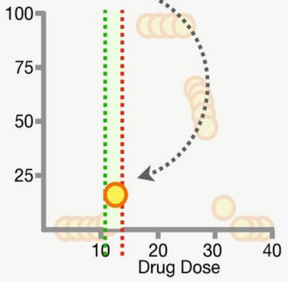
20 Earlier, we said *in theory* we could subdivide the **6** measurements with Dose **< 14.5** into smaller groups…





…but when we do, we end up with a single measurement in the **Leaf** on the *right* because there is the only one measurement with a Dose between **11.5** and **14.5**…

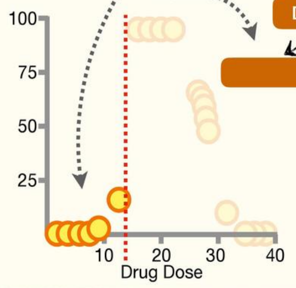
…and making a prediction based on a single measurement suggests that the tree is **Overfit** to the **Training Data** and may not perform well in the future.

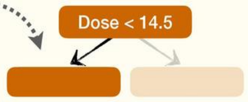


The simplest way to prevent this issue is to only split measurements when there are more than some minimum number, which is often **20**. However, since we have so little data in this specific example, we'll set the minimum to **7**.

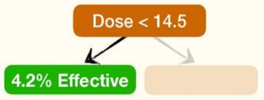
Building a Regression Tree: Step-by-Step

21 Now, because there are only **6** measurements with Dose **< 14.5**, there are only **6** measurements in the **Node** on the *left*…

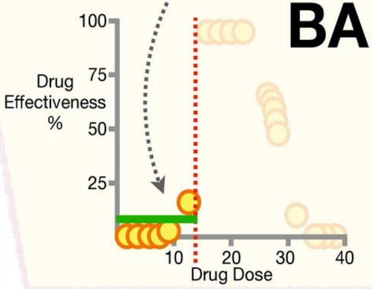




…and because we require a minimum of **7** measurements for further subdivision, the **Node** on the *left* will be a **Leaf**…

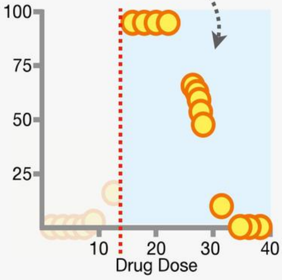


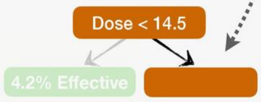
…and the output value for the **Leaf** is the average Effectiveness from the **6** measurements, **4.2%**.



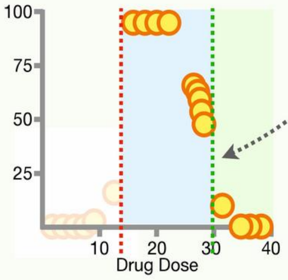
BAM!!!

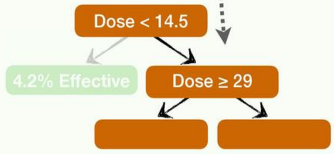
22 Now we need to figure out what to do with the **13** remaining measurements with Doses **≥ 14.5** that go to the **Node** on the *right*.





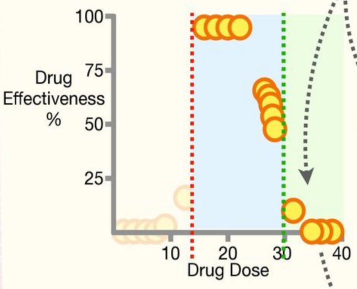
23 Since we have more than **7** measurements in the **Node** on the *right*, we can split them into two groups, and we do this by finding the threshold that gives us the lowest **SSR**.

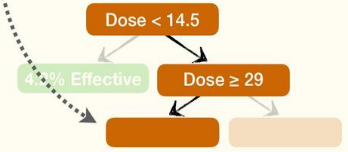




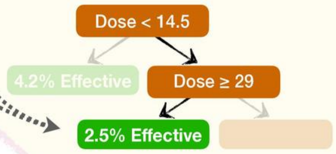
Building a Regression Tree: Step-by-Step

24 And because there are only **4** measurements with Dose ≥ **29**, there are only **4** measurements in the **Node** on the left…

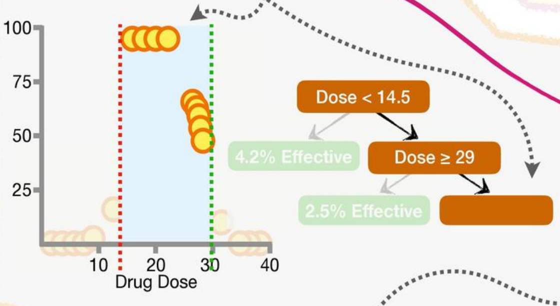




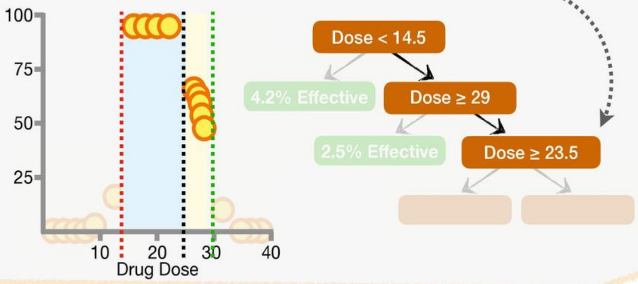
…and since the **Node** has fewer than **7** measurements, we'll make it a **Leaf**, and the output will be the average Effectiveness from those **4** measurements, **2.5%**.



25 Now, because we have more than **7** measurements with Doses between **14.5** and **29**, and thus, more than **7** measurements in the **Node** on the right…



…we can split the measurements into two groups by finding the Dose threshold that results in the lowest **SSR**.

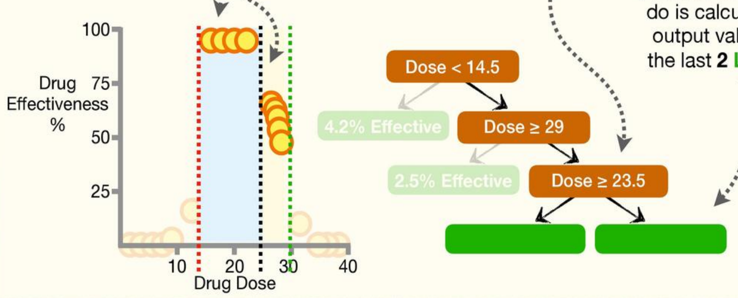


Building a Regression Tree: Step-by-Step

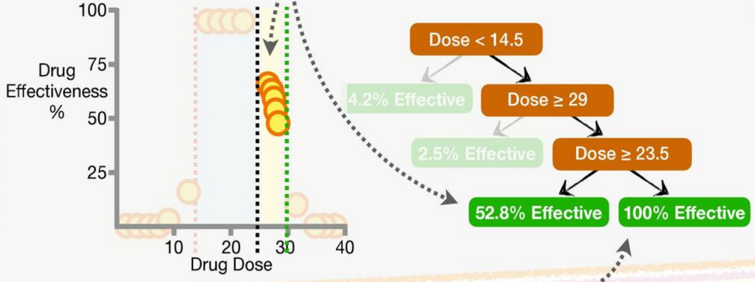
26 And since there are fewer than **7** measurements in each of these two groups…

…this will be the last split, because none of the **Leaves** has more than **7** measurements in them.

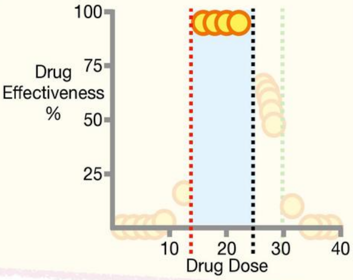
Now, all we have to do is calculate the output values for the last **2 Leaves**.



27 So, we use the average Drug Effectiveness for measurements with Doses between **23.5** and **29**, **52.8%**, as the output for **Leaf** on the *left*…



28 …and we use the average Drug Effectiveness for observations with Doses between **14.5** and **23.5**, **100%**, as the output for **Leaf** on the *right*.



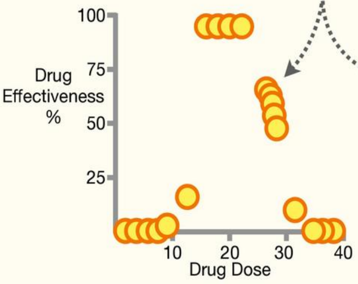
29 Now, at long last, we've finished building the **Regression Tree**.

DOUBLE BAM!!!


BUT WAIT!!! THERE'S MORE!!!

Building a Regression Tree With Multiple Features: Part 1

① So far, we've built a **Regression Tree** using a single predictor, Dose, to predict Drug Effectiveness.



Dose	Drug Effect
10	98
20	0
35	6
5	44
etc…	etc…

NOTE: Just like for **Classification Trees, Regression Trees** can use any type of variable to make a prediction. However, with **Regression Trees**, we always try to predict a continuous value.

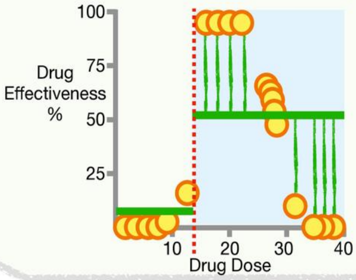
② Now let's talk about how to build a **Regression Tree** to predict Drug Effectiveness using Dose, Age, and Sex.

Dose	Age	Sex	Drug Effect
10	25	F	98
20	73	M	0
35	54	F	6
5	12	M	44
etc…	etc…	etc…	etc…

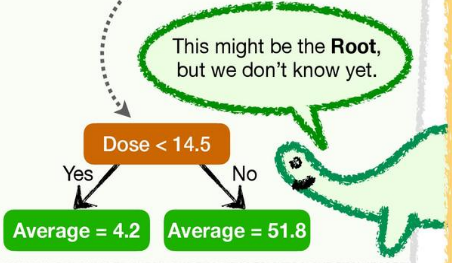
③ First, we completely ignore Age and Sex and only use Dose to predict Drug Effectiveness…

Dose	Age	Sex	Drug Effect
10	25	F	98
20	73	M	0
35	54	F	6
5	12	M	44
etc…	etc…	etc…	etc…

…and then we select the threshold that gives us the smallest **SSR**.



However, instead of that threshold instantly becoming the **Root**, it only becomes a *candidate* for the **Root**.

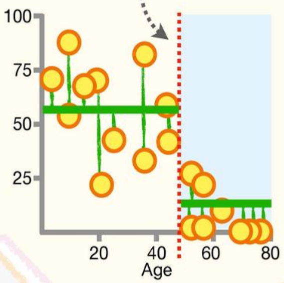


Building a Regression Tree With Multiple Features: Part 2

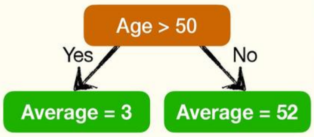
4 Then, we ignore Dose and Sex and only use Age to predict Effectiveness…

…and we select the threshold that gives us the smallest **SSR**…

Dose	Age	Sex	Drug Effect
10	25	F	98
20	73	M	0
35	54	F	6
5	12	M	44
etc…	etc…	etc…	etc…



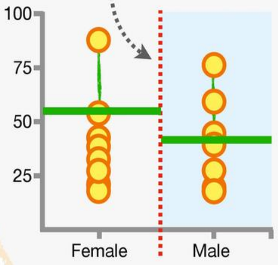
…and that becomes the second *candidate* for the **Root**.



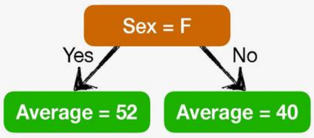
5 Lastly, we ignore Dose and Age and only use Sex to predict Effectiveness…

Dose	Age	Sex	Drug Effect
10	25	F	98
20	73	M	0
35	54	F	6
5	12	M	44
etc…	etc…	etc…	etc…

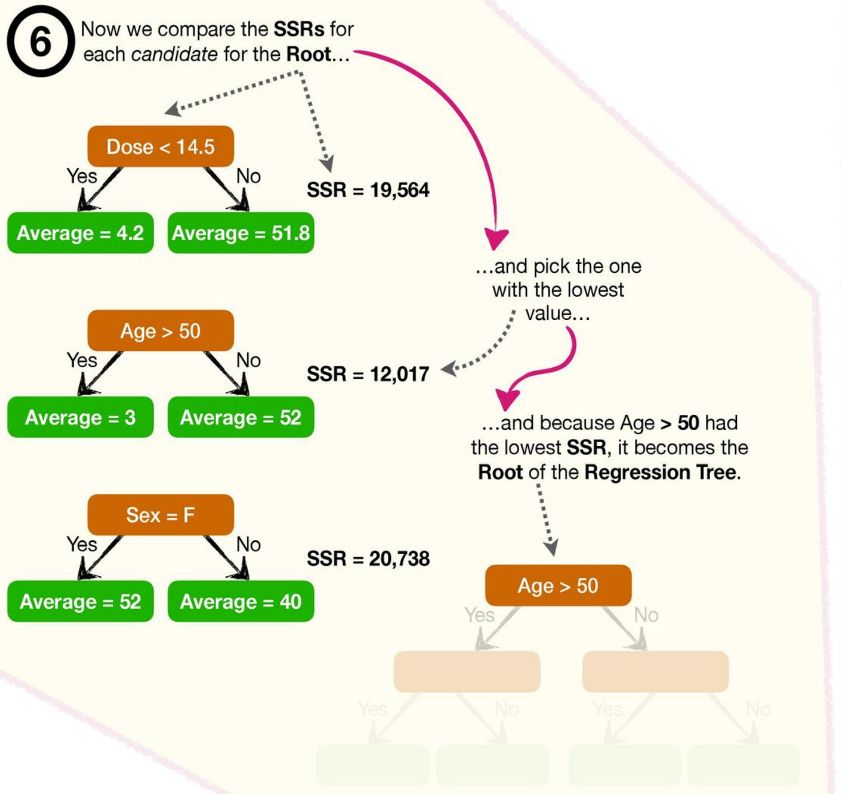
…and even though Sex only has one threshold for splitting the data, we still calculate the **SSR**, just like before…



…and that becomes the third *candidate* for the **Root**.



Building a Regression Tree With Multiple Features: Part 3



6 Now we compare the **SSRs** for each *candidate* for the **Root**…

…and pick the one with the lowest value…

…and because Age **> 50** had the lowest **SSR**, it becomes the **Root** of the **Regression Tree**.

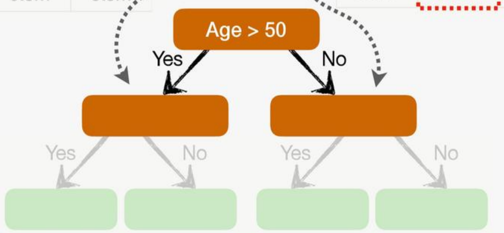
Building a Regression Tree With Multiple Features: Part 4

7 Now that Age **> 50** is the **Root**, the people in the **Training Data** who are older than **50** go to the **Node** on the *left*…

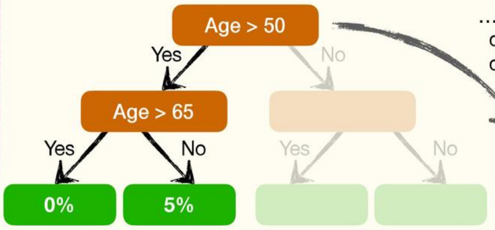
…and the people who are **≤ 50** go to the **Node** on the *right*.

Dose	Age	Sex	Drug Effect
10	25	F	98
20	73	M	0
35	54	F	6
5	12	M	44
etc…	etc…	etc…	etc…

Dose	Age	Sex	Drug Effect
10	25	F	98
20	73	M	0
35	54	F	6
5	12	M	44
etc…	etc…	etc…	etc…



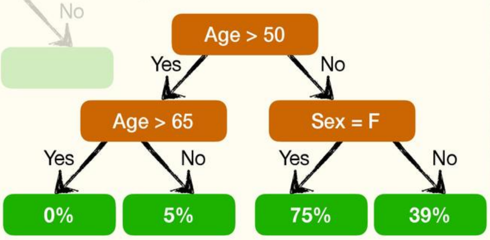
8 Then we grow the tree just like before, except now for each split we have **3** candidates, Dose, Age, and Sex, and we select whichever gives us the lowest **SSR**…



NOTE: The final prediction is Drug Effectiveness.

…until we can no longer subdivide the data any further. At that point, we're done building the **Regression Tree**.

TRIPLE BAM!!!



Chapter 11

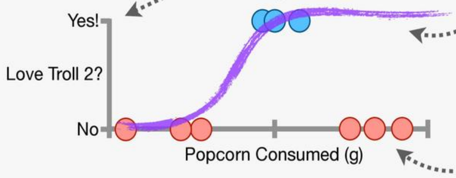
Support Vector Classifiers and Machines (SVMs)!!!

Support Vector Machines (SVMs): Main Ideas

1 **The Problem:** We measured how much Popcorn people ate, in grams, and we want to use that information to predict if someone will love Troll 2. The **red dots** represent people who do not love Troll 2. The **blue dots** represent people who do.

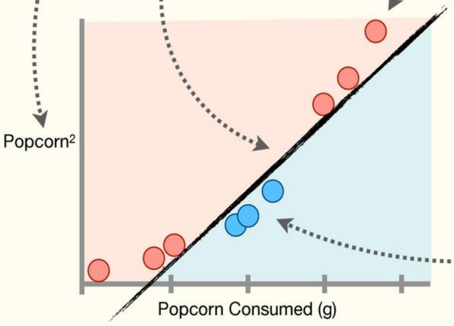
In theory, we could create a y-axis that represented whether or not someone loves Troll 2 and move the dots accordingly…

…and then use **Logistic Regression** to fit an **s-shaped squiggle** to the data…



…however, this **s-shaped squiggle** would make a terrible classifier because it would misclassify the people who ate the most Popcorn. Can we do better?

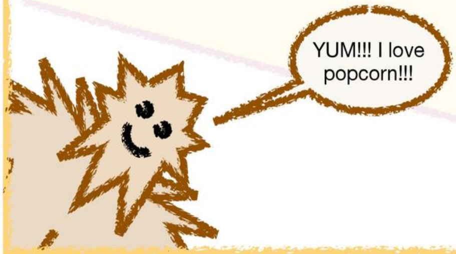
2 **A Solution:** A **Support Vector Machine** (**SVM**) can add a new axis to the data and move the points in a way that makes it relatively easy to draw a **straight line** that correctly classifies people.



In this example, the x-axis is the amount of Popcorn consumed by each person, and the new y-axis is the square of the Popcorn values.

Now, all of the points to the *left* of the **straight line** are people who do not love Troll 2…

…and all of the people on the *right* of the **straight line** loved Troll 2, so the **straight line** correctly separates the people who *love* and *do not love* Troll 2. **BAM!!!**



The first step in learning how a **Support Vector Machine** works is to learn about **Support Vector Classifiers**, so let's get started!!!

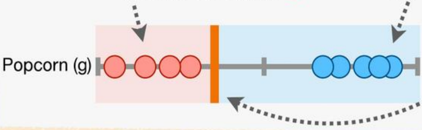
Support Vector Classifiers: Details Part 1

🔵 = Loves Troll 2

🔴 = Does Not Love Troll 2

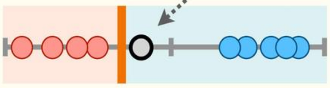
① If all the people who do not love Troll 2 didn't eat much Popcorn…

…and all of the people who love Troll 2 ate a lot of Popcorn…

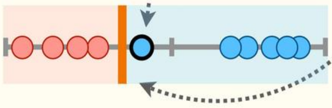


…then we could pick this threshold to classify people.

② However, as you've probably already guessed, this threshold is terrible because if someone new comes along and they ate this much Popcorn…



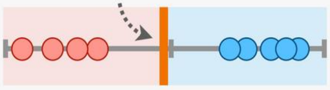
…then the threshold will classify them as someone who loves Troll 2, even though they're much closer to the people who do not love Troll 2.



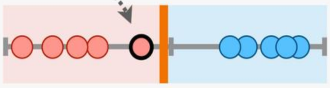
So, this threshold is really bad.

Can we do better? **YES!!!**

③ One way to improve the classifier is to use the midpoint between edges of each group of people.

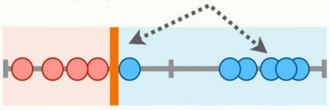


Now the person we'd previously classified as someone who loves Troll 2 is classified as someone who does not love Troll 2, which makes more sense.

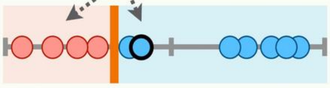


④ However, one problem with using the midpoint between the edges of each group as a classifier is that outliers, like this one, can ruin it.

Now, because of the outlier, which could be a mislabeled data point, the midpoint between the edges of each group is relatively far from most of the people who love Troll 2…



…and we're back to classifying that new person as someone who loves Troll 2, even though they're closer to a lot more people who do not love Troll 2.



Can we do better? **YES!!!**

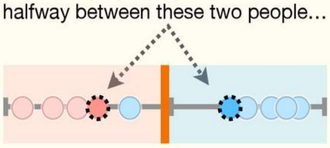
Support Vector Classifiers: Details Part 2

🔵 = Loves Troll 2

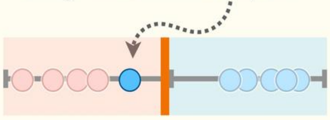
🔴 = Does Not Love Troll 2

5 One way to make a threshold for classification less sensitive to outliers is to allow misclassifications.

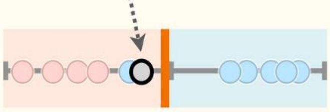
For example, if we put the threshold halfway between these two people…



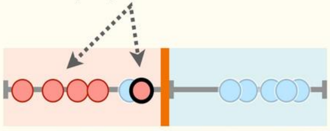
…then we'll misclassify this person as someone who does not love Troll 2, even though we're told that they do…



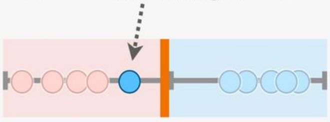
…but when a new person comes along…



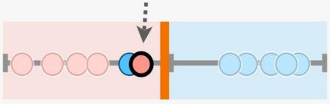
…they will be classified as not loving Troll 2, which makes sense since they're closer to most of the people who do not love Troll 2.



6 **NOTE:** Allowing the threshold to misclassify someone from the **Training Data**…

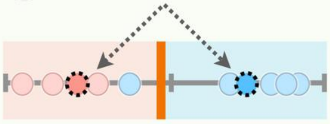


…in order to make better predictions…

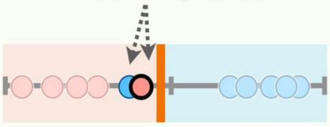


…is an example of the **Bias-Variance Tradeoff** mentioned in **Chapter 1**. By allowing for a misclassification, we avoid **Overfitting** the **Training Data** and increase the **Bias** a little, but we improve our predictions of new data, which suggests a reduction in **Variance**.

7 Alternatively, we could put the threshold halfway between these two people…



…however, this new threshold gives us the same result: one misclassification from the **Training Data** and the new person will be reasonably classified.



To be honest, the remaining combinations of pairs of points would give us similar results, so it's not super important which pair we pick.

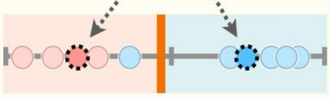
However, if we had a more complicated dataset, we could use **Cross Validation** to decide which pair of points should define the threshold and determine how many misclassifications of the **Training Data** to allow in order to get the best results.

Support Vector Classifiers: Details Part 3

🔵 = Loves Troll 2

🔴 = Does Not Love Troll 2

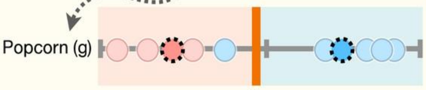
8 Now, imagine **Cross Validation** determined that putting a threshold halfway between these two points gave us the best results…



…then we would call this threshold a **Support Vector Classifier**.

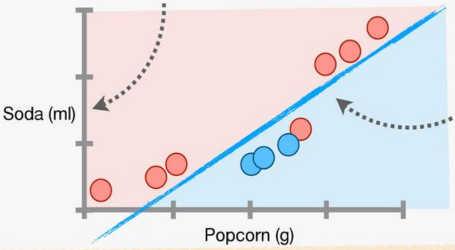
BAM.

NOTE: Because we only measured how much Popcorn people ate…



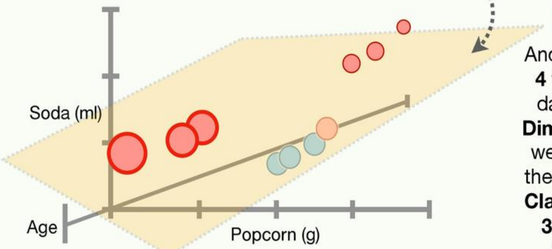
…the **Support Vector Classifier**, the threshold we use to decide if someone loves or does not love Troll 2, is just a **point** on a number line.

9 However, if, in addition to measuring how much Popcorn people ate, we also measured how much Soda they drank, then the data would be **2-Dimensional**…



…and a **Support Vector Classifier** would be a **straight line**. In other words, the **Support Vector Classifier** would be **1-Dimensional**.

10 If we measured Popcorn, Soda, and Age, then the data would be **3-Dimensional**, and the **Support Vector Classifier** would be a **2-Dimensional** **plane**.



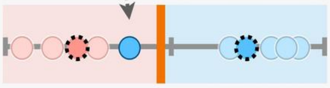
And if we measured **4** things, then the data would be **4-Dimensional**, which we can't draw, but the **Support Vector Classifier** would be **3-Dimensional**. Etc., etc., etc.

Support Vector Classifiers: Details Part 4

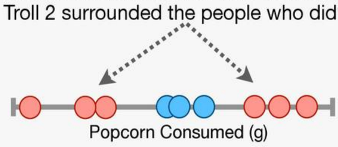
🔵 = Loves Troll 2

🔴 = Does Not Love Troll 2

⑪ **Support Vector Classifiers** seem pretty cool because they can handle outliers…

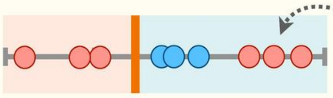


…but what if, in terms of Popcorn consumption, the people who do not love Troll 2 surrounded the people who did?

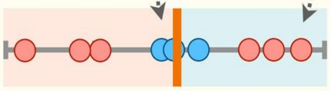


⑫ Now, no matter where we put the **Support Vector Classifier**, we'll make a lot of misclassifications.

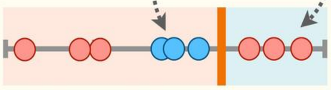
Ugh! These are misclassified!



Double Ugh!! These and these are misclassified!!!



Triple Ugh!!! These and these are misclassified!!! Can we do better???



⑬ **YES!!!** To see how we can do better, let's learn about **Support Vector Machines**!!!

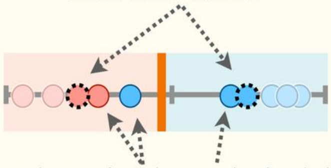
But first, we have to learn a little terminology…

Terminology Alert!!!

Oh no! It's the dreaded **Terminology Alert!!!**

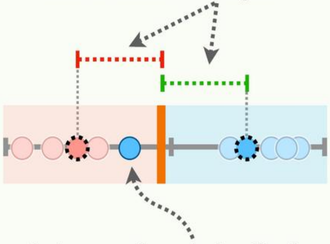
🔵 = Loves Troll 2
🔴 = Does Not Love Troll 2

① The name **Support Vector Classifier** comes from the fact that the points that define the threshold…



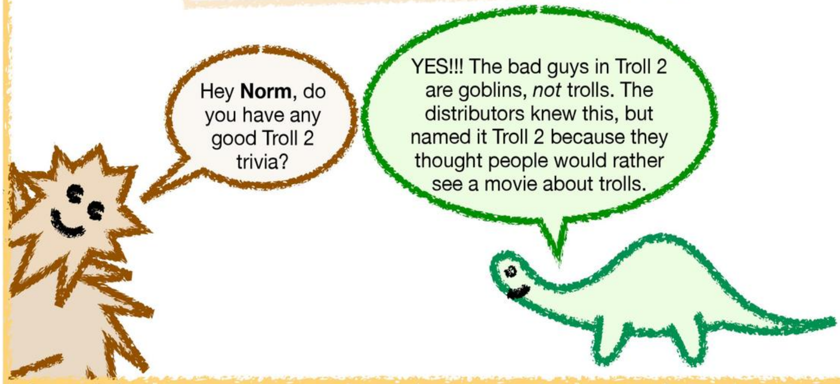
…and any points closer to the threshold are called **Support Vectors**.

② The distance between the points that define the threshold and the threshold itself is called the **Margin**.



…and when we allow misclassifications, the distance is called the **Soft Margin**.

Now that we understand the lingo, let's try to get a little intuition about how **Support Vector Machines** work before diving into the details.

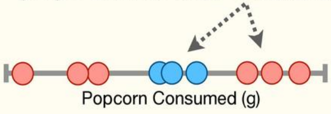


Support Vector Machines: Intuition Part 1

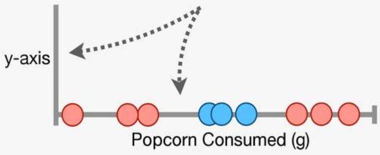
🔵 = Loves Troll 2

🔴 = Does Not Love Troll 2

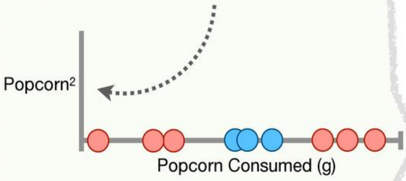
① To get a general intuition about how **Support Vector Machines** work, let's return to the dataset that has the people who love Troll 2 surrounded by people who do not.



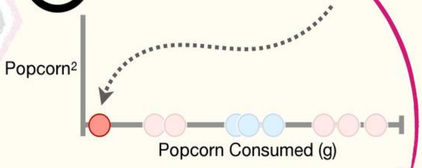
② Now, even though we only measured how much Popcorn each person ate, let's add a y-axis to the graph.



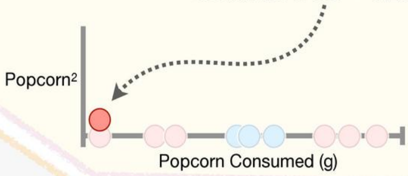
③ Specifically, we'll make the y-axis the square of the amount of Popcorn each person ate.



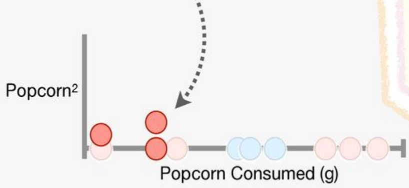
④ For example, because the first person only ate **0.5** grams of Popcorn, their x-axis coordinate is **0.5**…

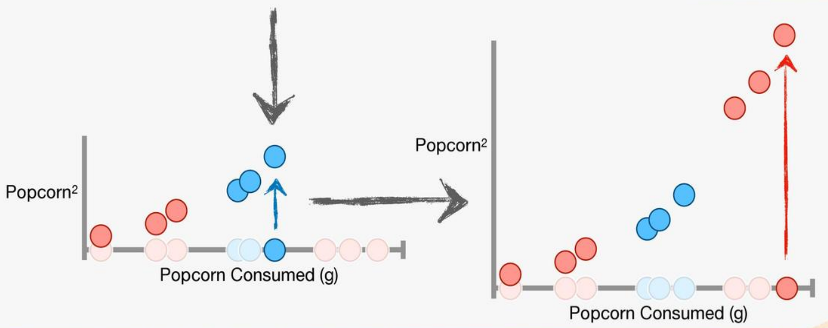


…and their y-axis coordinate is $\mathbf{0.5^2 = 0.25}$.



⑤ Now we just use $\mathbf{Popcorn^2}$ to get the y-axis coordinates for the remaining points.

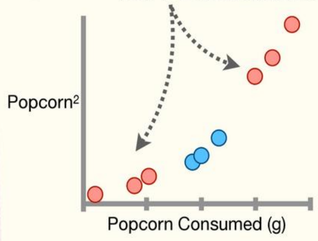




Support Vector Machines: Intuition Part 2

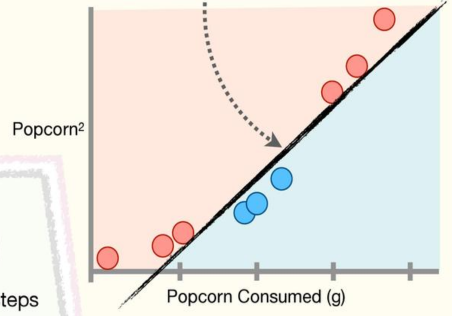
🔵 = Loves Troll 2
🔴 = Does Not Love Troll 2

6 Because each person has x- and y-axis coordinates, the data are now **2-Dimensional**…



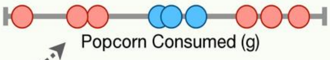
…and now that the data are **2-Dimensional**, we can draw a **Support Vector Classifier** that separates the people who love Troll 2 from the people who do not.

BAM!!!

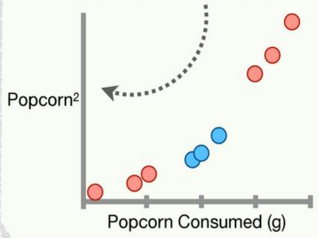


7 The **3** main steps for creating **Support Vector Machines** are…

a Start with low-dimensional data. In this case, we start with **1-Dimensional** data on a number line.

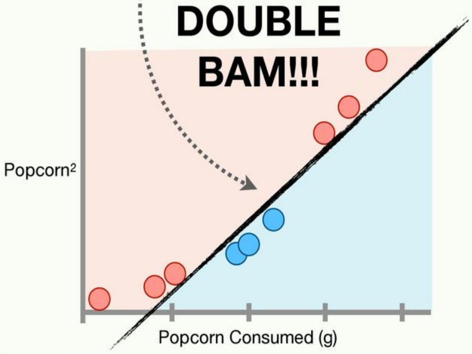


b Then use the existing data to create higher dimensions. In this example, we created **2-Dimensional** data by squaring the original Popcorn measurements.

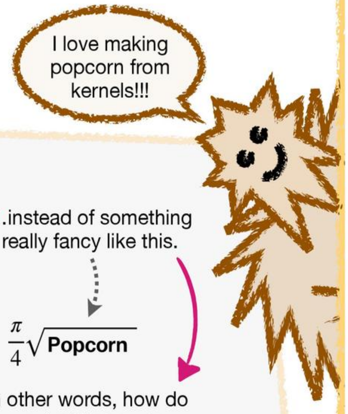


c Then find a **Support Vector Classifier** that separates the higher dimensional data into two groups.

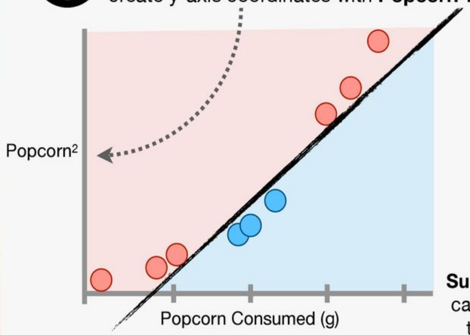
DOUBLE BAM!!!



Support Vector Machines (SVMs): Details Part 1



① Now, in the previous example, you may have been wondering why we decided to create y-axis coordinates with **Popcorn²**…



…instead of something really fancy like this.

$$\frac{\pi}{4}\sqrt{\textbf{Popcorn}}$$

In other words, how do we decide how to transform the data?

To make the mathematics efficient, **Support Vector Machines** use something called **Kernel Functions** that we can use to *systematically* find **Support Vector Classifiers** in higher dimensions.

Two of the most popular **Kernel Functions** are the **Polynomial Kernel** and the **Radial Kernel**, also known as the **Radial Basis Function**.

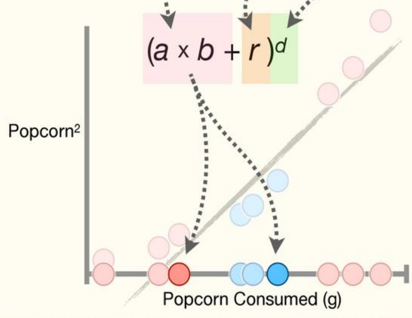
② The **Polynomial Kernel**, which is what we used in the last example, looks like this…

Polynomial Kernel: $(a \times b + r)^d$

…where ***a*** and ***b*** refer to two different observations in the data…

…***r*** determines the coefficient of the polynomial…

…and ***d*** determines the degree of the polynomial.



Support Vector Machines (SVMs): Details Part 2

Polynomial Kernel: $(a \times b + r)^d$

③ In the Popcorn example, we set ***r* = 1/2** and ***d* = 2**…

…and since we're squaring the term, we can expand it to be the product of two terms…

$$(a \times b + r)^d = (a \times b + \tfrac{1}{2})^2 = (a \times b + \tfrac{1}{2})(a \times b + \tfrac{1}{2})$$

…and we can multiply both terms together…

$$= a^2b^2 + \boxed{\tfrac{1}{2}ab + \tfrac{1}{2}ab} + \tfrac{1}{4}$$

…and then combine the middle terms…

$$= a^2b^2 + ab + \tfrac{1}{4}$$

…and, just because it will make things look better later, let's flip the order of the first two terms…

$$= ab + a^2b^2 + \tfrac{1}{4}$$

…and finally, this polynomial is equal to this **Dot Product**!

$$= (a, a^2, \tfrac{1}{2}) \cdot (b, b^2, \tfrac{1}{2})$$

NOTE: Dot Products sound fancier than they are. A **Dot Product** is just…

$$(\boxed{a}, a^2, \tfrac{1}{2}) \cdot (\boxed{b}, b^2, \tfrac{1}{2})$$

$$ab$$

…the first terms multiplied together…

$$(a, \boxed{a^2}, \tfrac{1}{2}) \cdot (b, \boxed{b^2}, \tfrac{1}{2})$$

$$ab + a^2b^2$$

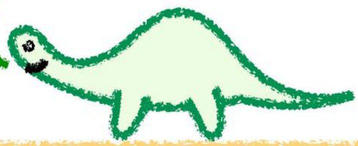
…plus the second terms multiplied together…

$$(a, a^2, \boxed{\tfrac{1}{2}}) \cdot (b, b^2, \boxed{\tfrac{1}{2}})$$

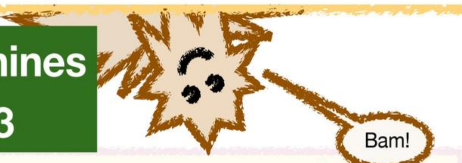
$$ab + a^2b^2 + \tfrac{1}{4}$$

…plus the third terms multiplied together.

Bam.



Support Vector Machines (SVMs): Details Part 3



4 To summarize the last page, we started with the **Polynomial Kernel**...

...set ***r* = 1/2** and ***d* = 2**...

...and, after a lot of math, ended up with this **Dot Product**...

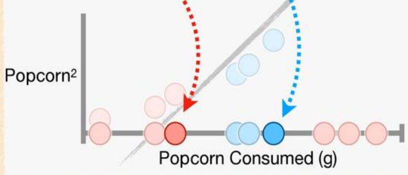
$$(a \times b + r)^d = (a \times b + \frac{1}{2})^2 = (a, a^2, \frac{1}{2}) \cdot (b, b^2, \frac{1}{2})$$

...and then **StatSquatch** learned that **Dot Products** sound fancier than they are...

...so now we need to learn why we're so excited about **Dot Products**!!!

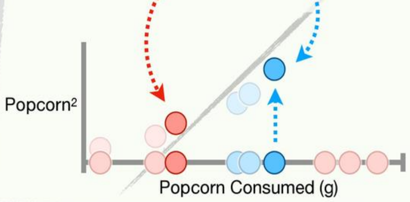
5 Since ***a*** and ***b*** refer to two different observations in the data, the first terms are their x-axis coordinates...

$$(\boxed{a}, a^2, \frac{1}{2}) \cdot (\boxed{b}, b^2, \frac{1}{2})$$



6 ...the second terms are their y-axis coordinates...

$$(a, \boxed{a^2}, \frac{1}{2}) \cdot (b, \boxed{b^2}, \frac{1}{2})$$



7 ... and the third terms are their z-axis coordinates.

$$(a, a^2, \boxed{\frac{1}{2}}) \cdot (b, b^2, \boxed{\frac{1}{2}})$$

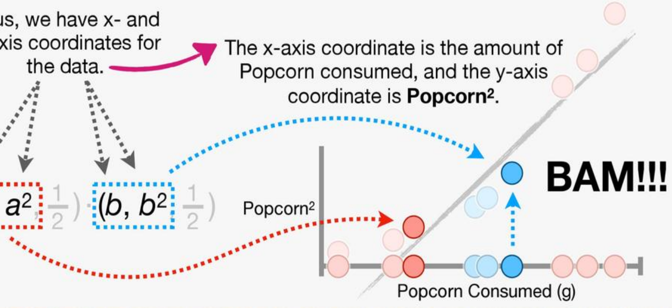
However, since they are **1/2** for both points (and every pair of points we select), we can ignore them.

$$(a, a^2, \cancel{\frac{1}{2}}) \cdot (b, b^2, \cancel{\frac{1}{2}})$$

8 Thus, we have x- and y-axis coordinates for the data.

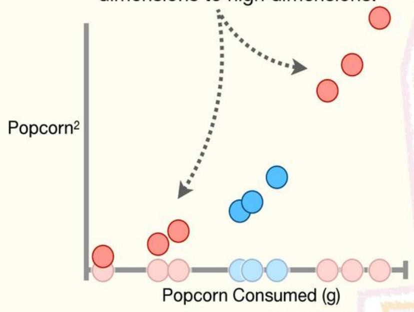
$$(\boxed{a, a^2}, \frac{1}{2}) \cdot (\boxed{b, b^2}, \frac{1}{2})$$

The x-axis coordinate is the amount of Popcorn consumed, and the y-axis coordinate is **Popcorn²**.

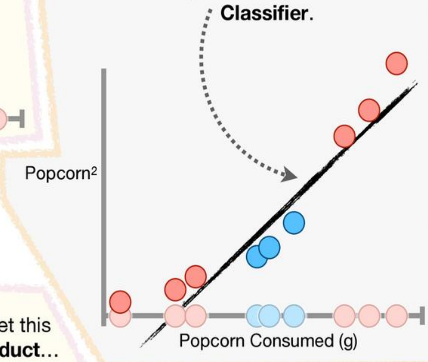


Support Vector Machines (SVMs): Details Part 4

9 The reason **Support Vector Machines** use **Kernels** is that they eliminate the need to actually transform the data from low dimensions to high dimensions.



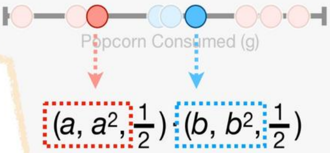
10 Instead, the **Kernels** use the **Dot Products** between every pair of points to compute their high-dimensional relationships and find the optimal **Support Vector Classifier**.

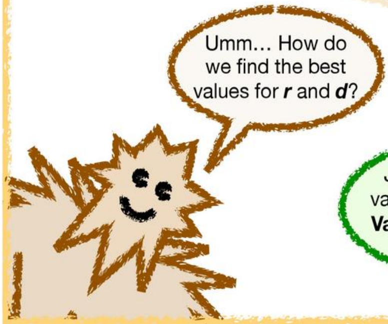


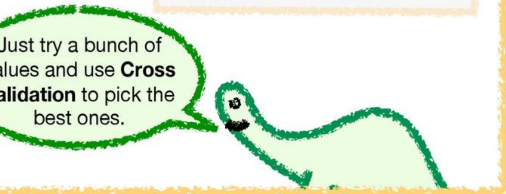
11 For example, when ***r* = 1/2** and ***d* = 2**... ...we get this **Dot Product**...

$$(a \times b + r)^d = (a \times b + \frac{1}{2})^2 = (a, a^2, \frac{1}{2}) \cdot (b, b^2, \frac{1}{2})$$

12 ...and since ***a*** and ***b*** refer to two different observations in the **Training Data**, we can plug their Popcorn values into the **Dot Product** and just do the math.

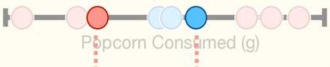






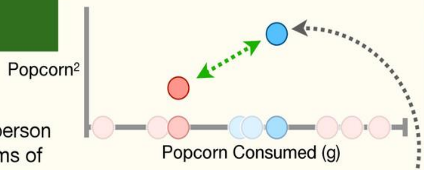
Support Vector Machines (SVMs): Details Part 5

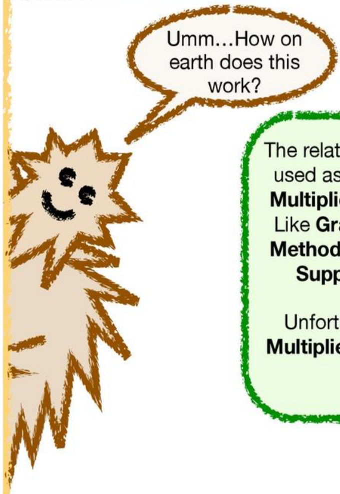
⑬ For example, if this person ate **5** grams of Popcorn… …and this person ate **10** grams of Popcorn… …then we plug the values, **5** and **10**, into the **Dot Product**… …do the math… …and we use this number, **2550.25**… …instead of the high-dimensional distance to find the optimal **Support Vector Classifier**.



$$\boxed{(a, a^2,} \frac{1}{2}) \; \boxed{(b, b^2,} \frac{1}{2})$$

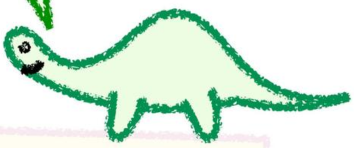
$$(5, 5^2, \frac{1}{2}) \cdot (10, 10^2, \frac{1}{2}) = (5 \times 10) + (5^2 \times 10^2) + (\frac{1}{2} \times \frac{1}{2}) = 2550.25$$





The relationships calculated by the **Dot Products** are used as input in a technique called the **Lagrangian Multiplier Method**, which is like **Gradient Descent**. Like **Gradient Descent**, the **Lagrangian Multiplier Method** is an iterative method that finds the optimal **Support Vector Classifier** one step at a time.

Unfortunately, the details of how the **Lagrangian Multiplier Method** works are outside of the scope of this book.
:(



⑭ Okay! Now that we understand most of the details of how **Support Vector Machines** and the **Polynomial Kernel** work, let's briefly get an intuition of how the **Radial Kernel** works.

BAM!!!

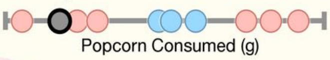
The Radial Kernel: Intuition

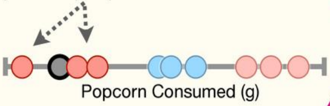
① Earlier in this chapter, I mentioned that two of the most popular **Kernel Functions** for **Support Vector Machines** are the **Polynomial Kernel** and the **Radial Kernel**, which is also called the **Radial Basis Function**.

Since we've already talked about the **Polynomial Kernel**, let's get an intuition about how the **Radial Kernel** works.

② The basic idea behind the **Radial Kernel** is that when we want to classify a new person…

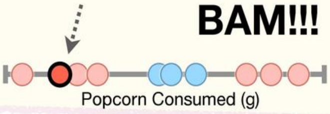
…we simply look to see how the closest points from the **Training Data** are classified. In this case, the closest points represent people who do not love Troll 2…





…and thus, we classify the new person as someone who does not love Troll 2.

BAM!!!



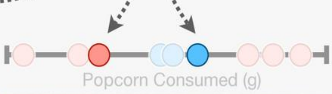
③ The equation for the **Radial Kernel** might look scary, but it's not that bad.

This Greek symbol ***γ***, ***gamma***, scales how much influence neighboring points have on classification…

…and we find a good value for ***γ*** (gamma) by trying a bunch of values and using **Cross Validation** to determine which is best…

$$e^{-\gamma(a-b)^2}$$

…and just like for the **Polynomial Kernel**, ***a*** and ***b*** refer to two different observations in the data.

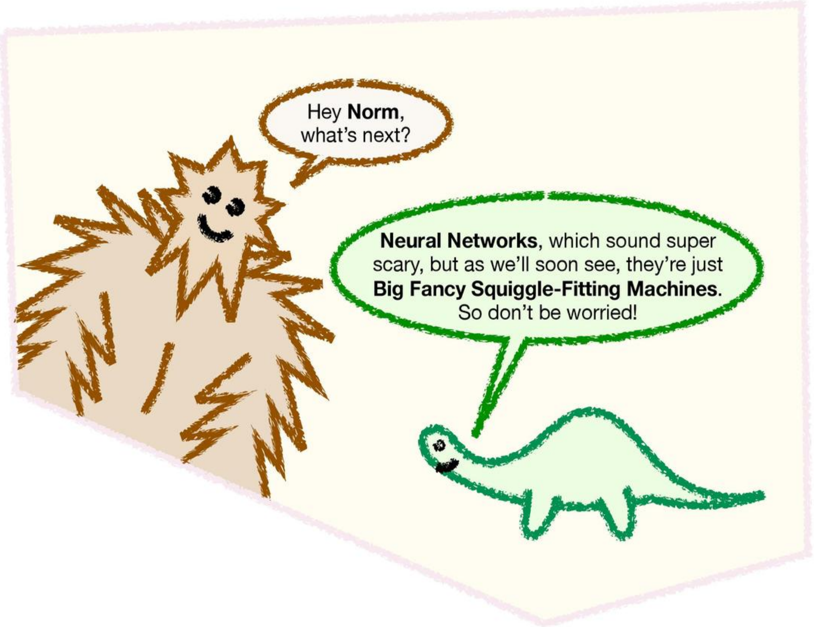


DOUBLE BAM!!!

④ Believe it or not, the **Radial Kernel** is like a **Polynomial Kernel**, but with ***r* = 0** and ***d* = infinity**…

$$(a \times b + r)^d = (a \times b + 0)^\infty$$

…and that means that the **Radial Kernel** finds a **Support Vector Classifier** in *infinite* dimensions, which sounds crazy, but the math works out. For details, scan, click, or tap this QR code to check out the **StatQuest**.


Hey **Norm**, what's next?
Neural Networks, which sound super scary, but as we'll soon see, they're just **Big Fancy Squiggle-Fitting Machines**. So don't be worried!

Chapter 12

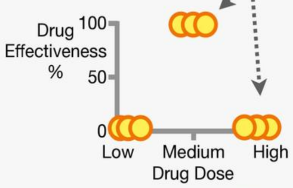
Neural Networks!!!

Neural Networks
Part One:

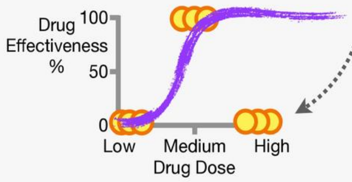
Understanding How Neural Networks Work

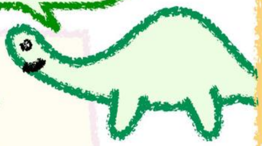
Neural Networks: Main Ideas

① **The Problem:** We have data that show the Effectiveness of a drug for different Doses....



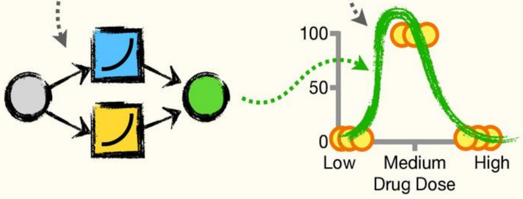
...and the **s-shaped squiggle** that **Logistic Regression** uses would not make a good fit. In this example, it would misclassify the high Doses.





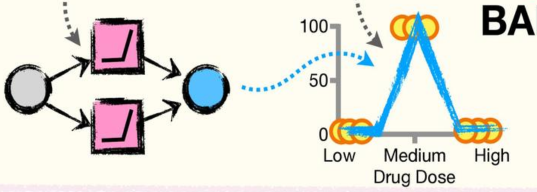
② **A Solution:** Although they sound super intimidating, all **Neural Networks** do is fit **fancy squiggles** or **bent shapes** to data. And like **Decision Trees** and **SVMs**, **Neural Networks** do fine with any relationship among the variables.

For example, we could get this **Neural Network**... ...to fit a **fancy squiggle** to the data...



...or we could get this **Neural Network**... ...to fit a **bent shape** to the data.

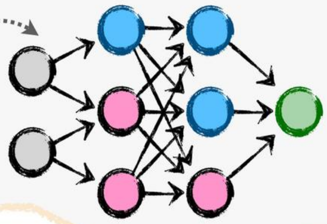
BAM!!!



Terminology Alert!!! Anatomy of a Neural Network

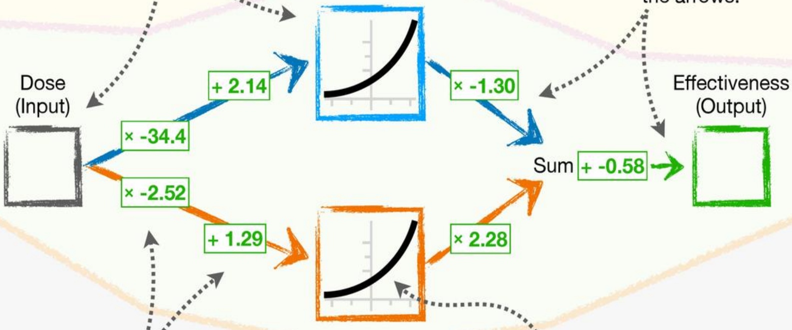
(Oh no! It's the dreaded **Terminology Alert!!!**)

① Although **Neural Networks** usually look like super complicated groups of neurons connected by synapses, which is where the name **Neural Network** comes from, they're all made from the same simple parts.



② **Neural Networks** consist of **Nodes**, the square boxes…

…and connections between **Nodes**, the arrows.



④ The numbers along the connections represent parameter values that were estimated when this **Neural Network** was fit to data using a process called **Backpropagation**. In this chapter, we'll see exactly what the parameters do and how they're estimated, step-by-step.

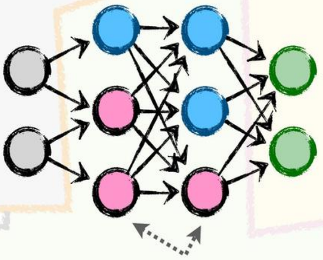
⑤ The bent or curved lines inside some of the **Nodes** are called **Activation Functions**, and they make **Neural Networks** flexible and able to fit just about any data.

Terminology Alert!!! Layers

(Oh no! *A Double* **Terminology Alert!!!**)

① **Neural Networks** are organized in **Layers**. Usually, a **Neural Network** has multiple **Input Nodes** that form an **Input Layer**…

② …and usually there are multiple **Output Nodes** that form an **Output Layer**…

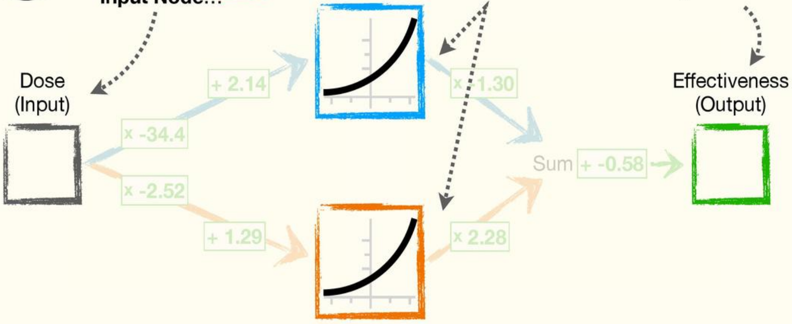


③ …and the **Layers** of **Nodes** between the **Input** and **Output Layers** are called **Hidden Layers**. Part of the *art* of **Neural Networks** is deciding how many **Hidden Layers** to use and how many **Nodes** should be in each one. Generally speaking, the more **Layers** and **Nodes**, the more complicated the shape that can be fit to the data.

④ In the example we'll use, we have a single **Input Node**…

…a single **Hidden Layer** with **2 Nodes** in it…

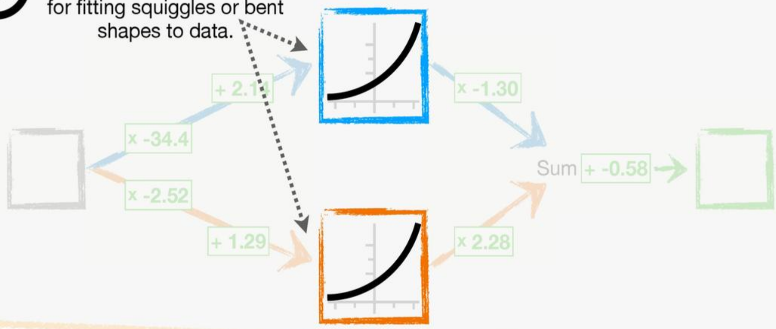
…and a single **Output Node**.



Terminology Alert!!! Activation Functions

(Oh no! *A TRIPLE* **TERMINOLOGY ALERT!!!**)

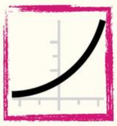
① The **Activation Functions** are the basic building blocks for fitting squiggles or bent shapes to data.



② There are lots of different **Activation Functions**. Here are three that are commonly used:

ReLU, which is short for **Rectified Linear Unit** and sounds like the name of a robot, is probably the most commonly used **Activation Function** with large **Neural Networks**. It's a **Bent Line**, and the bend is at ***x* = 0**.

SoftPlus, which sounds like a brand of toilet paper, is a modified form of the **ReLU Activation Function**. The big difference is that instead of the line being bent at **0**, we get a nice **Curve**.



Lastly, the **Sigmoid Activation Function** is an **s-shaped squiggle** that's frequently used when people teach **Neural Networks** but is rarely used in practice.

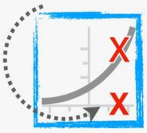
③ Although they might sound fancy, **Activation Functions** are just like the mathematical functions you learned when you were a teenager: you plug in an x-axis coordinate, do the math, and the output is a y-axis coordinate.

For example, the **SoftPlus** function is:

$$\textbf{SoftPlus}(x) = \textbf{log}(1 + e^x)$$

...where the **log()** function is the natural log, or **log base *e***, and ***e*** is **Euler's Number**, which is roughly **2.72**.

So, if we plug in an x-axis value, ***x* = 2.14**...

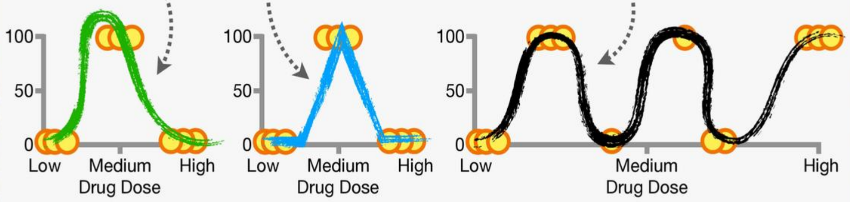


...then the **SoftPlus** will tell us the y-axis coordinate is **2.25**, because $\textbf{log}(1 + e^{2.14}) = \textbf{2.25}$.

Now let's talk about the main ideas of how **Activations Functions** work.

Activation Functions: Main Ideas

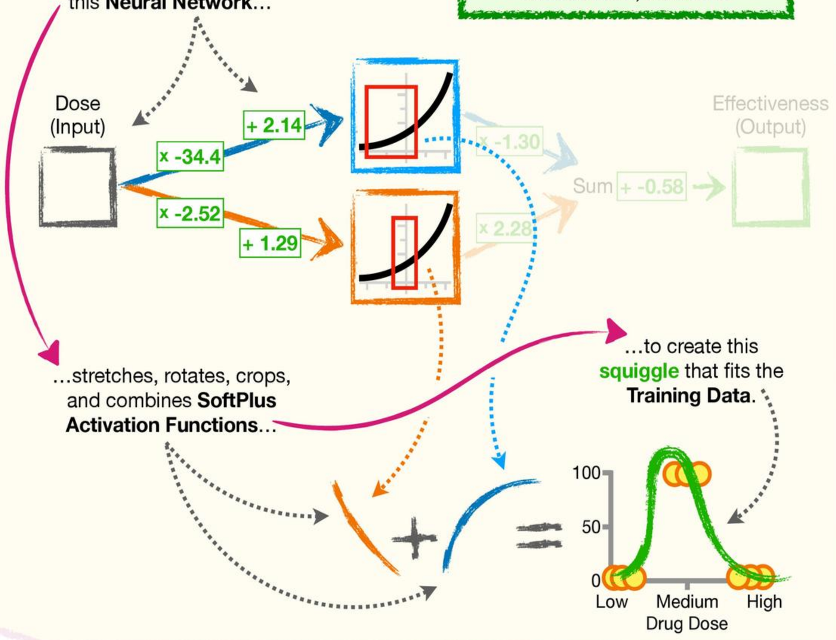
① **The Problem:** We need to create new, exciting shapes that can fit any dataset with a squiggle or bent lines, even when the dataset is super complicated.



② **The Solution: Neural Networks** stretch, rotate, crop, and combine **Activation Functions** to create new, exciting shapes that can fit anything!!!

NOTE: The purpose of this illustration is simply to show you where we're headed in the next section, so rather than think too hard about it, read on!!!

In the next section, **A Neural Network in Action: Step-by-Step**, we'll show you how this **Neural Network**…

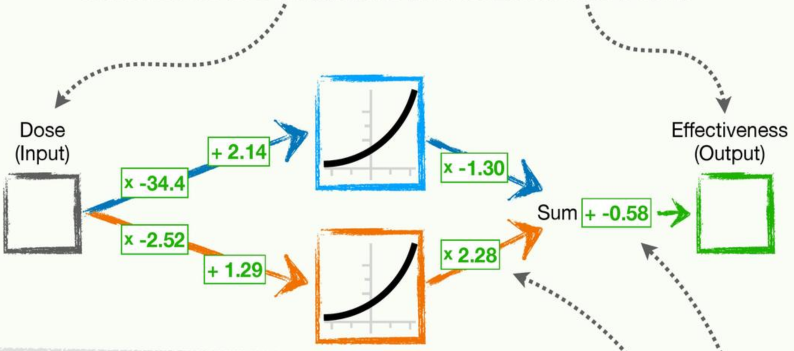


…stretches, rotates, crops, and combines **SoftPlus Activation Functions**…

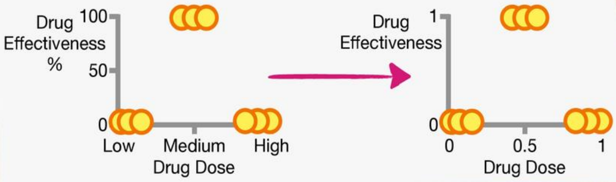
…to create this squiggle that fits the **Training Data**.

A Neural Network in Action: Step-by-Step

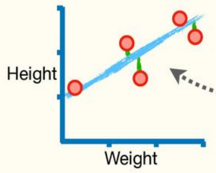
① A lot of people say that **Neural Networks** are black boxes and that it's difficult to understand what they're doing. Unfortunately, this is true for big, fancy **Neural Networks**. However, the good news is that it's *not* true for simple ones. So, let's walk through how this simple **Neural Network** works, one step at a time, by plugging in Dose values from low to high and seeing how it converts the Dose (input) into predicted Effectiveness (output).



② **NOTE:** To keep the math in this section simple, let's scale both the x- and y-axes so that they go from **0**, for *low* values, to **1**, for *high* values.



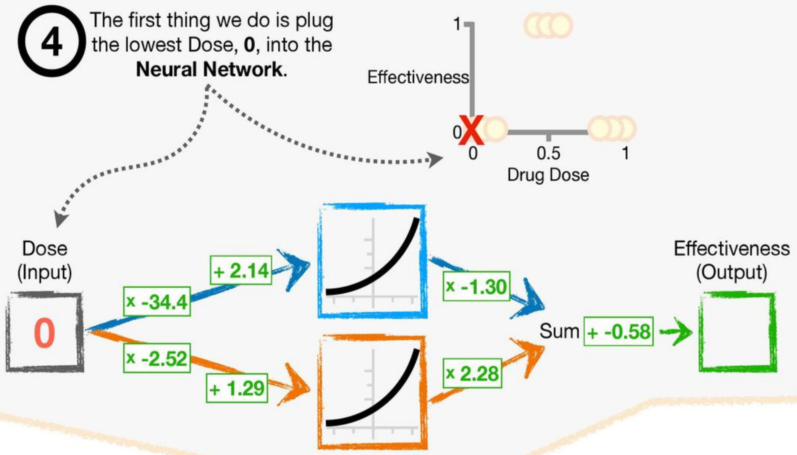
③ **ALSO NOTE:** These numbers are *parameters* that are estimated using a method called **Backpropagation**, and we'll talk about how that works, step-by-step, later. For now, just assume that these values have already been optimized, much like the **slope** and **intercept** are optimized when we fit a line to data.

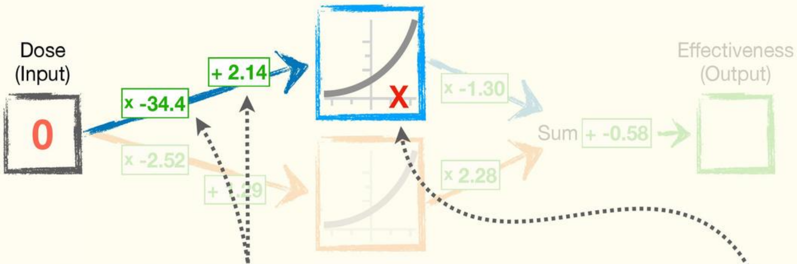


Height = **0.95** + **0.64** x Weight

A Neural Network in Action: Step-by-Step

4 The first thing we do is plug the lowest Dose, **0**, into the **Neural Network**.



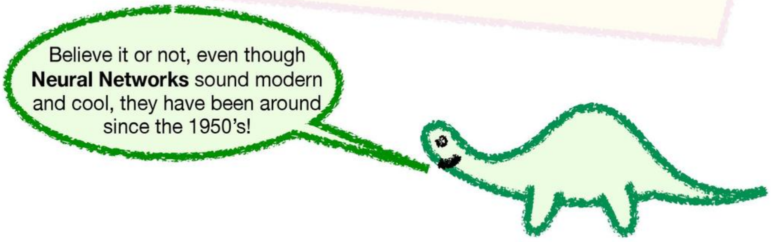


5 Now, the connection from the **Input Node** to the *top* **Node** in the **Hidden Layer** multiplies the Dose by **-34.4** and then adds **2.14**…

(**Dose** x -34.4) + 2.14 = x-axis coordinate

(**0** x -34.4) + 2.14 = x-axis coordinate = 2.14

…to get a new x-axis coordinate for the **Activation Function, 2.14**.



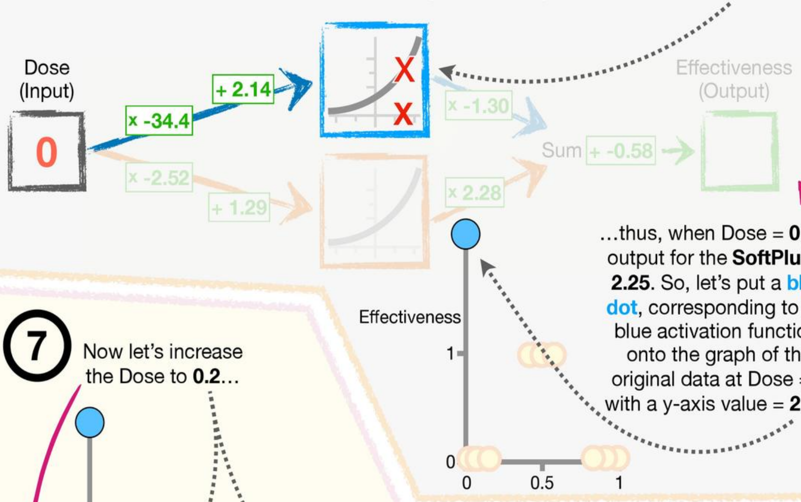
A Neural Network in Action: Step-by-Step

6 Then we plug the x-axis coordinate, **2.14** into the **SoftPlus Activation Function**…

…and do the math, just like we saw earlier…

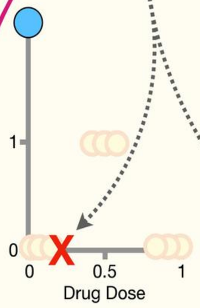
…and we get **2.25**. So, when the x-axis coordinate is **2.14**, the corresponding y-axis value is **2.25**…

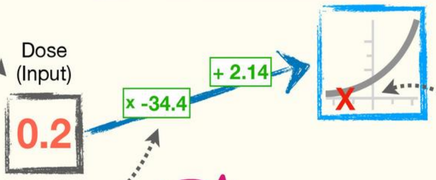
$$\textbf{SoftPlus}(x) = \textbf{SoftPlus}(2.14) = \textbf{log}(1 + e^{2.14}) = \text{y-axis coordinate} = \mathbf{2.25}$$



…thus, when Dose = **0**, the output for the **SoftPlus** is **2.25**. So, let's put a **blue dot**, corresponding to the blue activation function, onto the graph of the original data at Dose = **0** with a y-axis value = **2.25**.

7 Now let's increase the Dose to **0.2**…





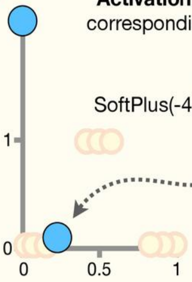
…and calculate the new x-axis coordinate by multiplying the Dose by **-34.4** and adding **2.14** to get **-4.74**…

(**Dose** x -34.4) + 2.14

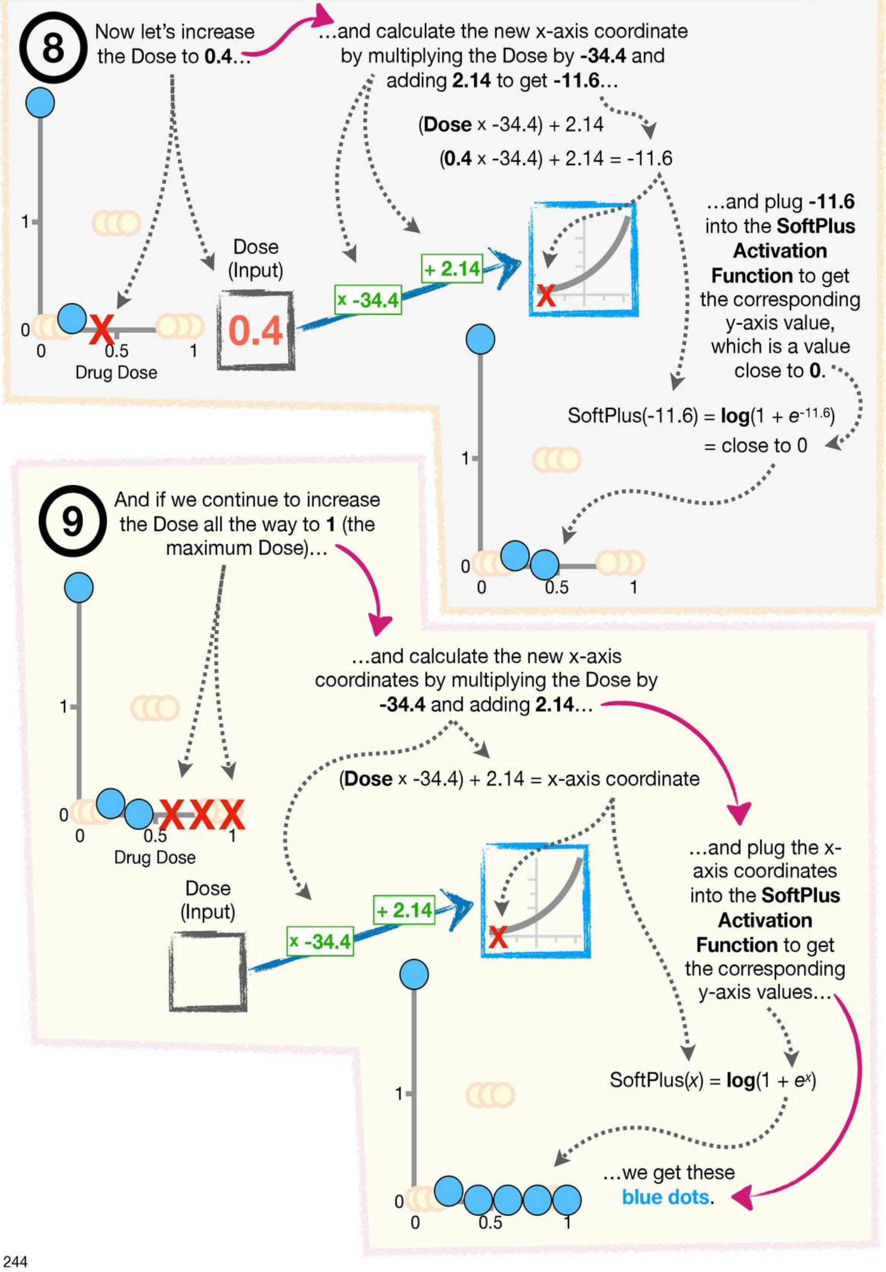
(**0.2** x -34.4) + 2.14 = -4.74

…and plug **-4.74** into the **SoftPlus Activation Function** to get the corresponding y-axis value, **0.01**.

$$\text{SoftPlus}(-4.74) = \textbf{log}(1 + e^{-4.74}) = 0.01$$



A Neural Network in Action: Step-by-Step



Now let's increase the Dose to **0.4**…

…and calculate the new x-axis coordinate by multiplying the Dose by **-34.4** and adding **2.14** to get **-11.6**…

$(\textbf{Dose} \times -34.4) + 2.14$

$(\textbf{0.4} \times -34.4) + 2.14 = -11.6$

…and plug **-11.6** into the **SoftPlus Activation Function** to get the corresponding y-axis value, which is a value close to **0**.

$\text{SoftPlus}(-11.6) = \textbf{log}(1 + e^{-11.6})$
$= \text{close to } 0$

And if we continue to increase the Dose all the way to **1** (the maximum Dose)…

…and calculate the new x-axis coordinates by multiplying the Dose by **-34.4** and adding **2.14**…

$(\textbf{Dose} \times -34.4) + 2.14 = \text{x-axis coordinate}$

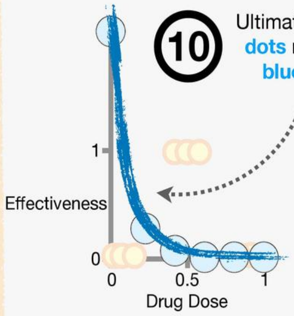
…and plug the x-axis coordinates into the **SoftPlus Activation Function** to get the corresponding y-axis values…

$\text{SoftPlus}(x) = \textbf{log}(1 + e^{x})$

…we get these **blue dots**.

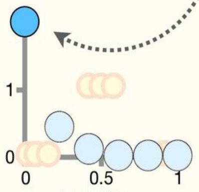
A Neural Network in Action: Step-by-Step

⑩ Ultimately, the **blue dots** result in this **blue curve**…

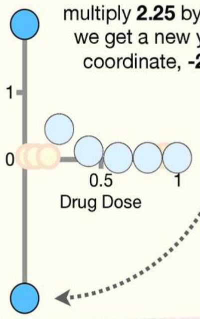


…and the next step in the **Neural Network** tells us to multiply the y-axis coordinates on the **blue curve** by **-1.30**.

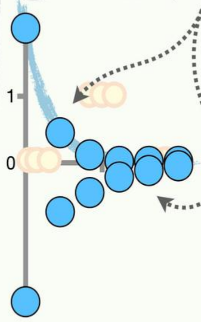
⑪ For example, when Dose = **0**, the current y-axis coordinate on the **blue curve** is **2.25**…



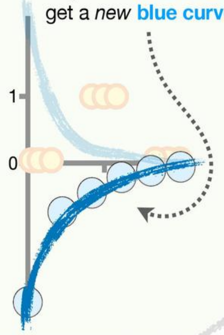
…and when we multiply **2.25** by **-1.30**, we get a new y-axis coordinate, **-2.93**.



⑫ Likewise, when we multiply all of the y-axis coordinates on the **blue curve** by **-1.30**…



…we end up flipping and stretching the *original* **blue curve** to get a *new* **blue curve**.



A Neural Network in Action: Step-by-Step

⑬ Okay, we just finished a major step, so this is a good time to review what we've done so far.

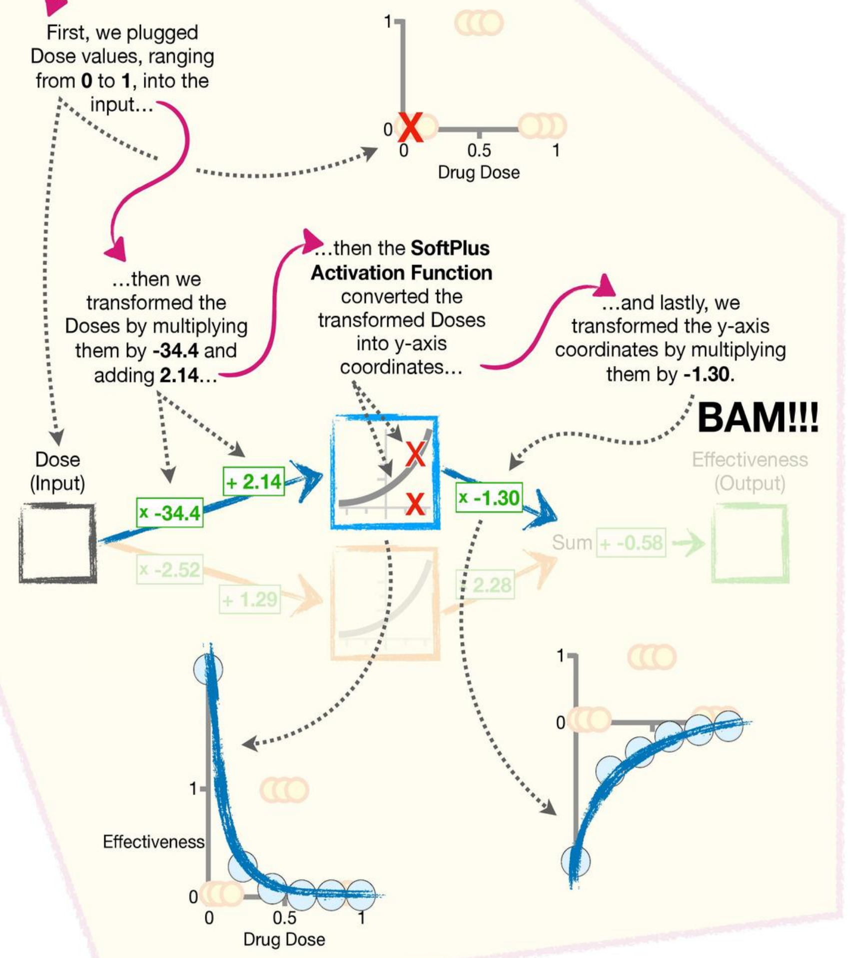
First, we plugged Dose values, ranging from **0** to **1**, into the input…

…then we transformed the Doses by multiplying them by **-34.4** and adding **2.14**…

…then the **SoftPlus Activation Function** converted the transformed Doses into y-axis coordinates…

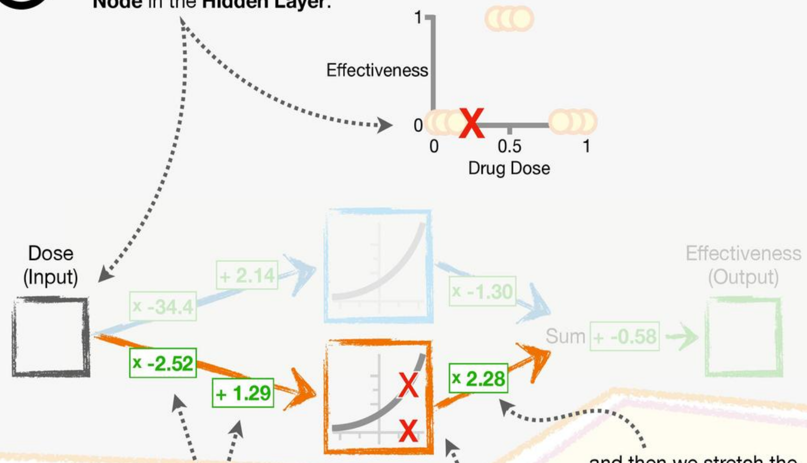
…and lastly, we transformed the y-axis coordinates by multiplying them by **-1.30**.

BAM!!!



A Neural Network in Action: Step-by-Step

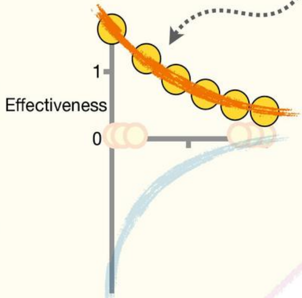
14 Now we run Dose values through the connection to the *bottom* **Node** in the **Hidden Layer**.

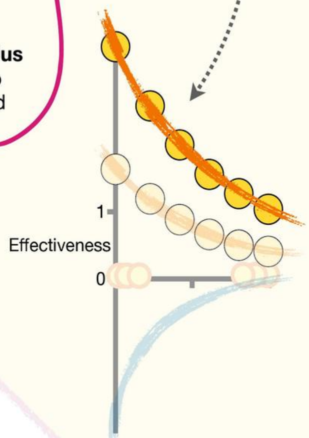


15 The good news is that the only difference between what we just did and what we're doing now is that now we multiply the Doses by **-2.52** and add **1.29**…

…before using the **SoftPlus Activation Function** to convert the transformed Doses into y-axis coordinates…

…and then we stretch the y-axis coordinates by multiplying them by **2.28** to get this **orange curve**.



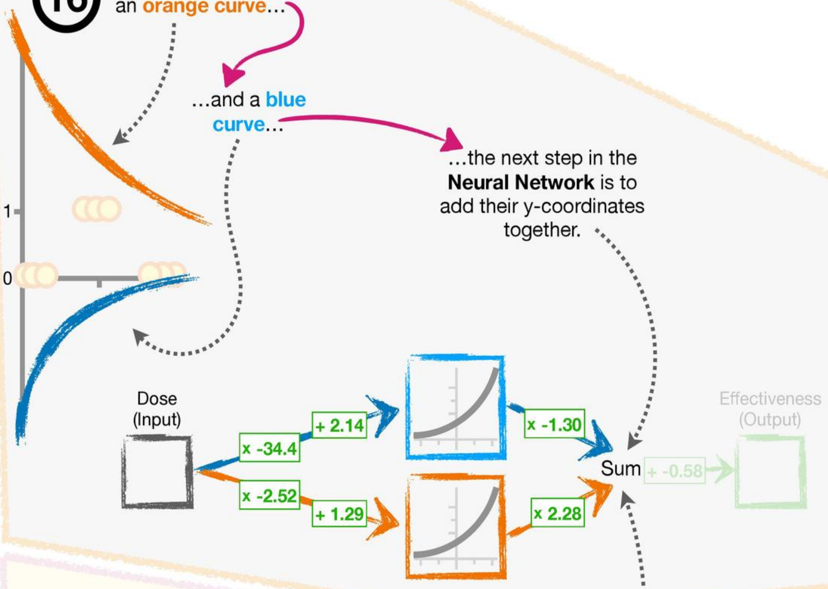


A Neural Network in Action: Step-by-Step

16 So, now that we have an **orange curve**…

…and a **blue curve**…

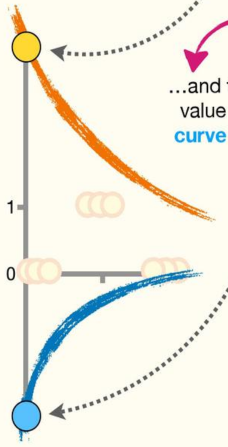
…the next step in the **Neural Network** is to add their y-coordinates together.

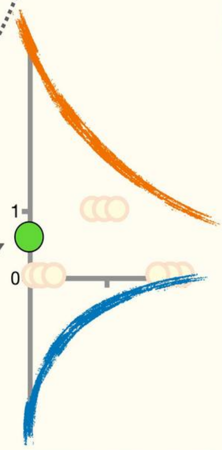


17 For example, when Dose = **0**, the y-axis value for the **orange curve** is **3.5**…

…and the y-axis value for **blue curve** is **-2.9**…

…thus, when Dose = **0**, the new y-axis value is the sum of the **orange** and **blue** y-axis values, **3.5** + **-2.9** = **0.6**. We'll keep track of that **0.6** value by putting a **green dot** here.



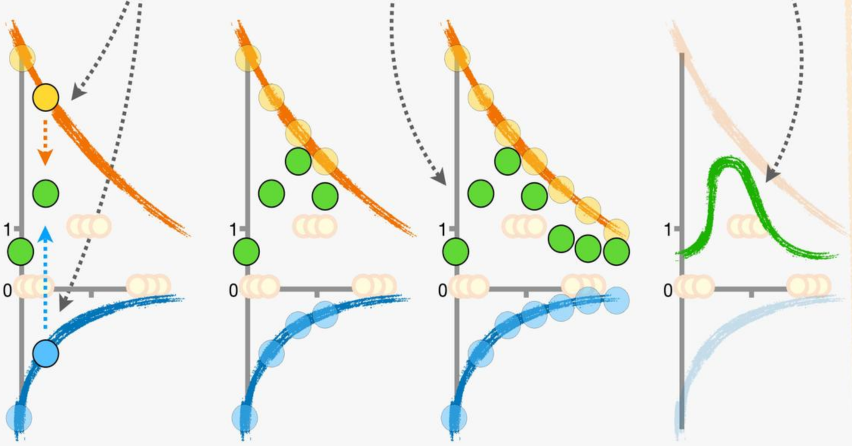


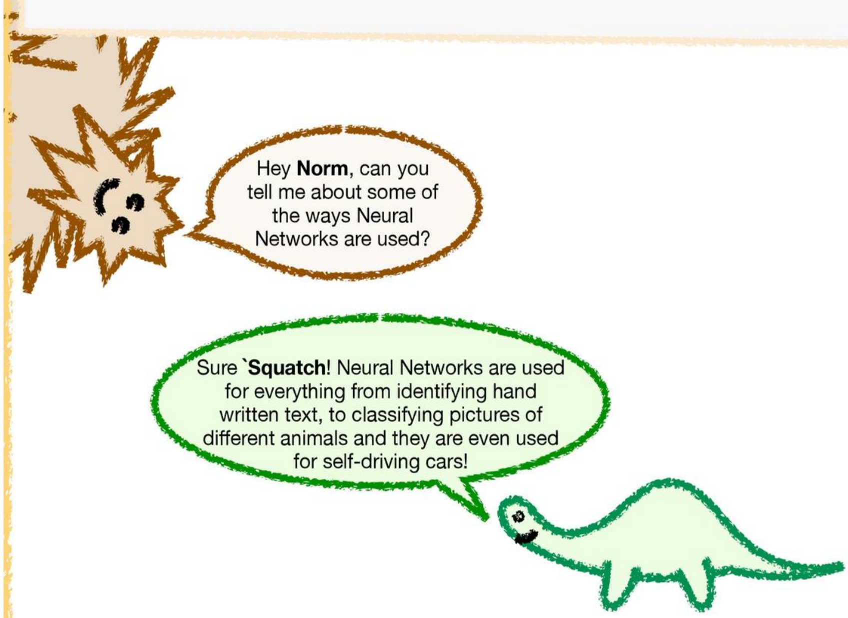
A Neural Network in Action: Step-by-Step

18 Then, for the remaining Dose values, we just add the y-axis coordinates of the **blue** and **orange** curves…

…plot the resulting **green dot** values…

…and, after connecting the dots, we ultimately end up with this **green squiggle**.

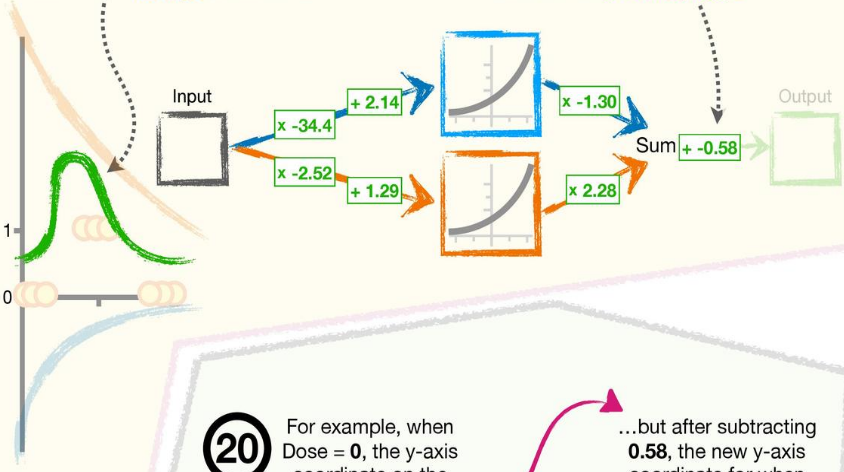




A Neural Network in Action: Step-by-Step

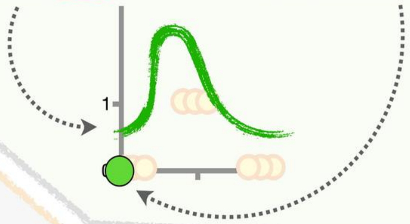
19 Now that we have this **green squiggle** from the **blue** and **orange** curves…

…we're ready for the final step, which is to add **-0.58** to each y-axis value on the **green squiggle**.

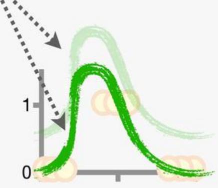


20 For example, when Dose = **0**, the y-axis coordinate on the **green squiggle** starts out at **0.6**…

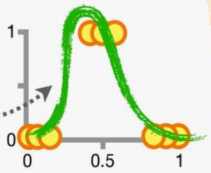
…but after subtracting **0.58**, the new y-axis coordinate for when Dose = **0** is **0.0** (rounded to the nearest **tenth**).



21 Likewise, subtracting **0.58** from all of the other y-axis coordinates on the **green squiggle** shifts it down…

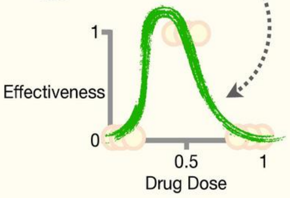


…and, ultimately, we end up with a **green squiggle** that fits the **Training Data**.

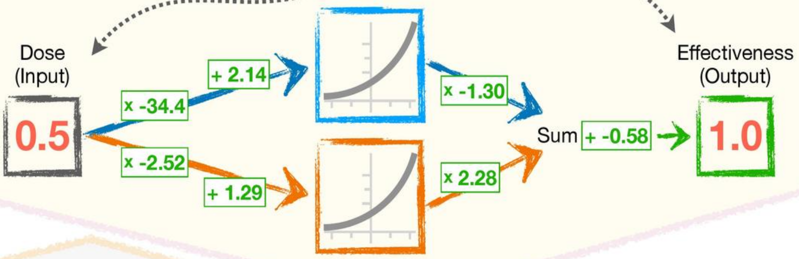


DOUBLE BAM!!!

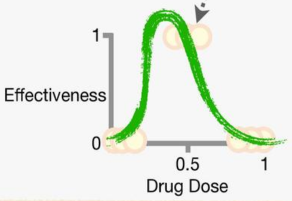
22 Hooray!!! At long last, we see the final **green squiggle**…



…that this **Neural Network** uses to predict Effectiveness from Doses.

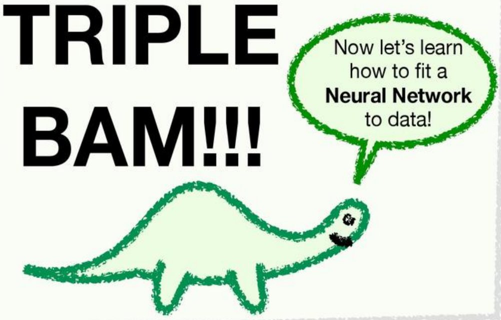


23 Now, if we're wondering if a medium Dose of **0.5** will be effective, we can look at the **green squiggle** and see that the output from the **Neural Network** will be **1**, and thus, Dose = **0.5** will be effective.



24 Alternatively, if we plug **Dose = 0.5** into the **Neural Network** and do the math, we get **1**, and thus, Dose = **0.5** will be effective.

TRIPLE BAM!!!

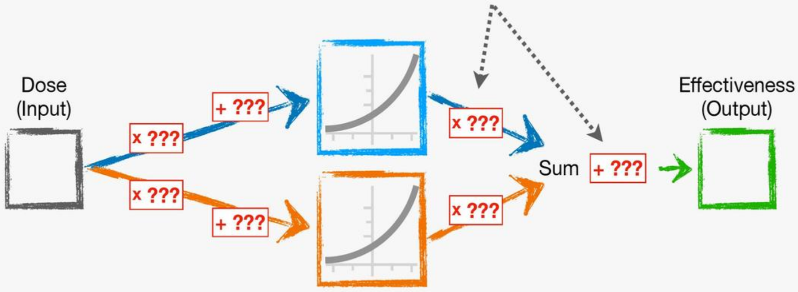


Neural Networks
Part Deux:

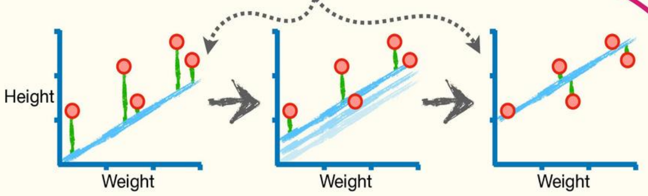
Fitting a Neural Network to Data with Backpropagation

Backpropagation: Main Ideas

① **The Problem:** Just like for **Linear Regression**, **Neural Networks** have parameters that we need to optimize to fit a squiggle or bent shape to data. How do we find the optimal values for these parameters?

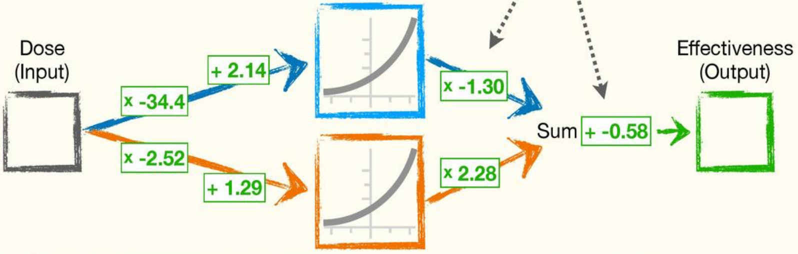


② **A Solution:** Just like for **Linear Regression**, we can use **Gradient Descent** (or **Stochastic Gradient Descent**) to find the optimal parameter values…



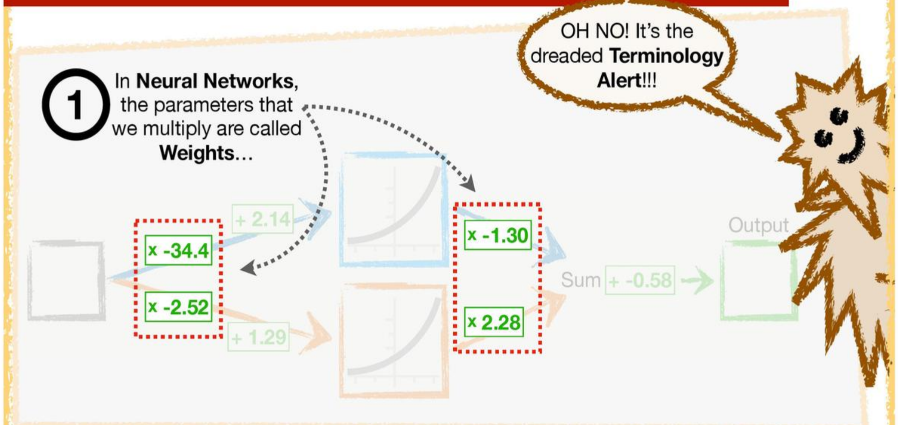
…however, we don't call it **Gradient Descent**. That would be too easy. Instead, because of how the derivatives are found for each parameter in a **Neural Network** (from the back to the front), we call it **Backpropagation**.

BAM!!!

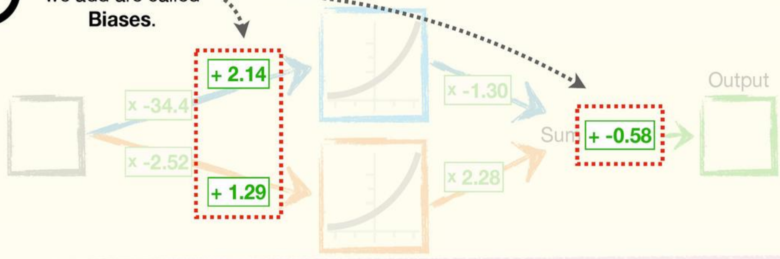


Terminology Alert!!! Weights and Biases

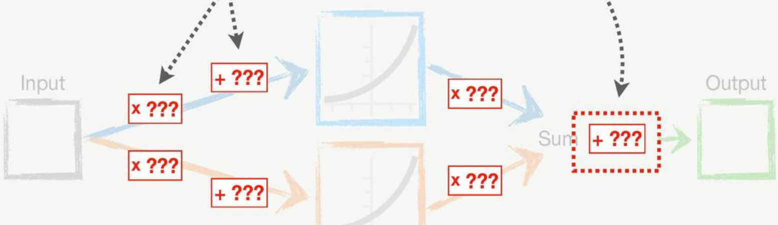
① In **Neural Networks**, the parameters that we multiply are called **Weights**…



② …and the parameters we add are called **Biases**.



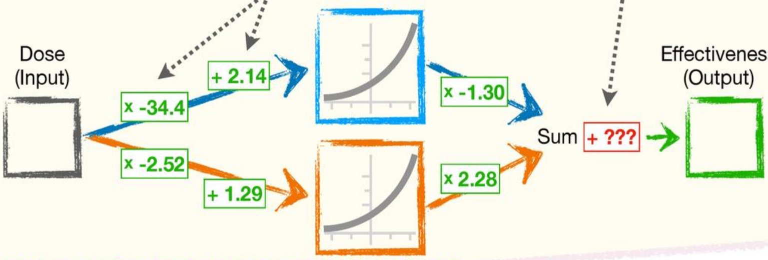
③ In the example that follows, we'll use **Backpropagation** to optimize the **Final Bias**. However, the same process and ideas apply to optimizing all of the parameters.



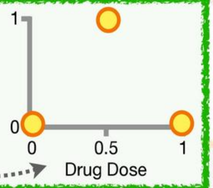
Backpropagation: Details Part 1

① In this example, let's assume that we already have the optimal values for all of the **Weights** and **Biases**…

…except for the **Final Bias**. So the goal is to use **Backpropagation** to optimize this **Final Bias**.

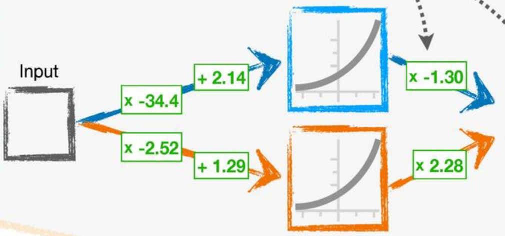


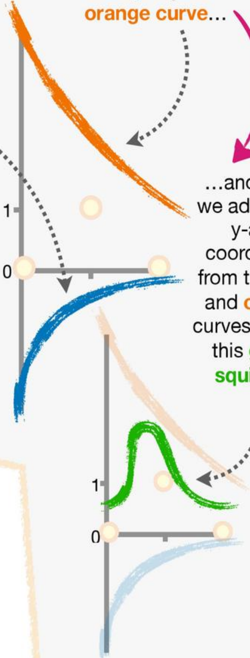
NOTE: To keep the math relatively simple, from now on, the **Training Data** will have only **3 Dose** values, **0**, **0.5**, and **1**.



② Now let's remember that when we plugged Doses into the input, the top of the **Hidden Layer** gave us this **blue curve**…

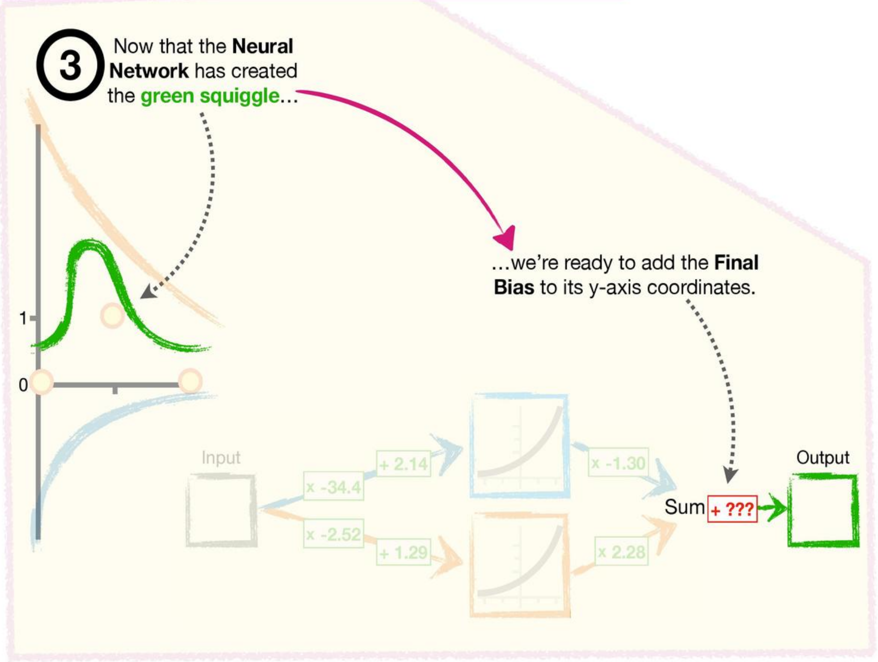
…and the bottom of the **Hidden Layer** gave us this **orange curve**…





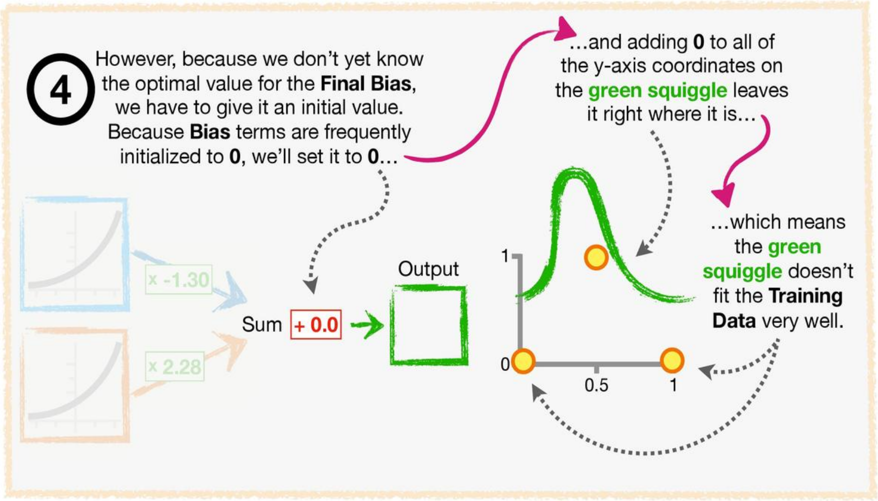
…and when we added the y-axis coordinates from the **blue** and **orange** curves, we got this **green squiggle**.

Backpropagation: Details Part 2



3 Now that the **Neural Network** has created the **green squiggle**…

…we're ready to add the **Final Bias** to its y-axis coordinates.



4 However, because we don't yet know the optimal value for the **Final Bias**, we have to give it an initial value. Because **Bias** terms are frequently initialized to **0**, we'll set it to **0**…

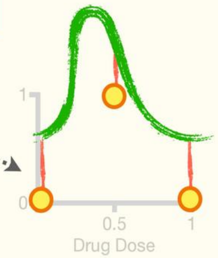
…and adding **0** to all of the y-axis coordinates on the **green squiggle** leaves it right where it is…

…which means the **green squiggle** doesn't fit the **Training Data** very well.

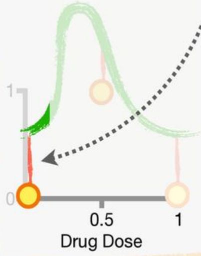
Backpropagation: Details Part 3

⑤ Now, just like we did for ***R*²**, **Linear Regression**, and **Regression Trees**, we can quantify how well the **green squiggle** fits all of the **Training Data** by calculating the **Sum of the Squared Residuals** (**SSR**).

$$\textbf{SSR} = \sum_{i=1}^{n} (\text{Observed}_i - \text{Predicted}_i)^2$$

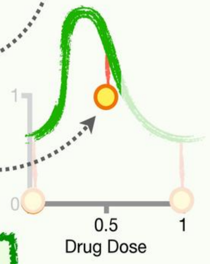


⑥ For example, for the first Dose, **0**, the **Observed** Effectiveness is **0** and the **green squiggle** created by the **Neural Network** predicts **0.57**, so we plug in **0** for the **Observed** value and **0.57** for the **Predicted** value into the equation for the **SSR**.

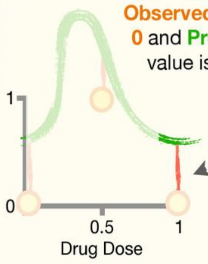


$$\begin{aligned} \textbf{SSR} &= (0 - 0.57)^2 \\ &+ (1 - 1.61)^2 \\ &+ (0 - 0.58)^2 \boxed{= 1.0} \end{aligned}$$

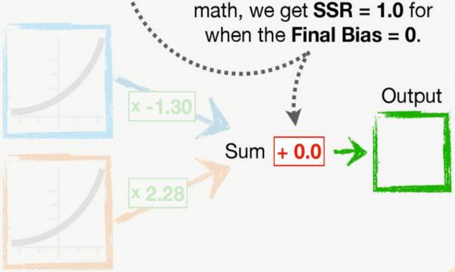
⑦ Then we add the **Residual** for when Dose = **0.5**. Now the **Observed** Effectiveness is **1**, but the **green squiggle** predicts **1.61**.



⑧ Then we add the **Residual** when Dose = **1**, where the **Observed** value is **0** and **Predicted** value is **0.58**.

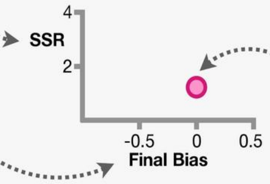


⑨ Lastly, when we do the math, we get **SSR** = **1.0** for when the **Final Bias** = **0**.

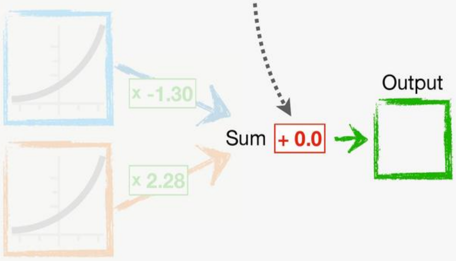


Backpropagation: Details Part 4

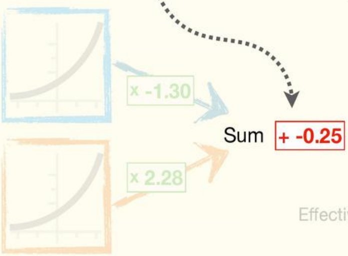
⑩ Now we can compare the **SSR** for different values for the **Final Bias** by plotting them on this graph that has values for the **Final Bias** on the x-axis and the corresponding **SSR** on the y-axis.



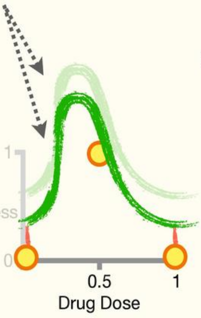
Thus, when the **Final Bias = 0**, we can draw a **pink dot** at **SSR = 1**.



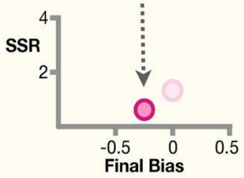
⑪ If we set the **Final Bias** to **-0.25**…



…then we shift the **green squiggle** down a little bit…



…and we can calculate the **SSR** and plot the value on our graph.

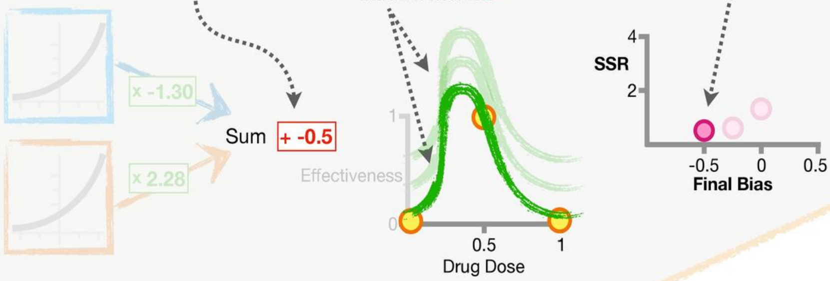


Backpropagation: Details Part 5

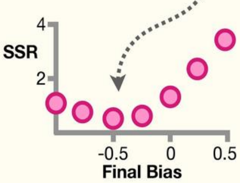
⑫ Setting the **Final Bias** to **-0.5**…

…shifts the **green squiggle** down a little bit more…

…and results in a slightly lower **SSR**.

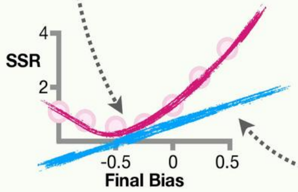


⑬ And if we try a bunch of different values for the **Final Bias**, we can see that the lowest **SSR** occurs when the **Final Bias** is close to **-0.5**…



⑭ …however, instead of just plugging in a bunch of numbers at random, we'll use **Gradient Descent** to quickly find the lowest point in the pink curve, which corresponds to the **Final Bias** value that gives us the minimum **SSR**…

…and to use **Gradient Descent**, we need the derivative of the **SSR** with respect to the **Final Bias**.

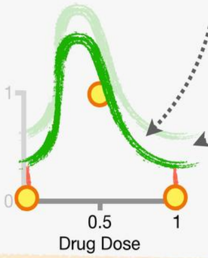


$$\frac{d\ SSR}{d\ Final\ Bias}$$

Backpropagation: Details Part 6

15 Remember, each **Predicted** value in the **SSR** comes from the **green squiggle**…

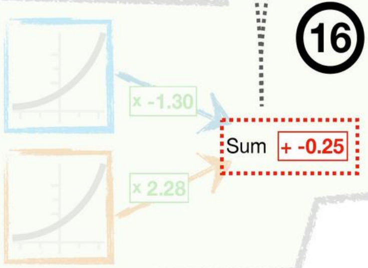
$$\textbf{SSR} = \sum_{i=1}^{n}(\text{Observed}_i - \boxed{\text{Predicted}_i})^2$$



$\boxed{\text{Predicted}}$ = **green squiggle**

= **blue curve** + **orange curve** + **Final Bias**

16 …and the **green squiggle** comes from the last part of the **Neural Network**, when we add the y-axis values from the **blue** and **orange** curves to the **Final Bias**.



17 Now, because the **Predicted** values link the **SSR**…

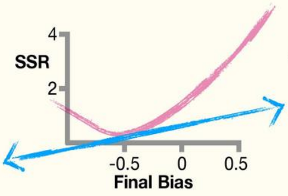
$$\textbf{SSR} = \sum_{i=1}^{n}(\text{Observed}_i - \boxed{\text{Predicted}_i})^2$$

…to the **Final Bias**…

$\boxed{\text{Predicted}}$ = **green squiggle** = **blue curve** + **orange curve** + **Final Bias**

…we can use **The Chain Rule** to solve for the derivative of the **SSR** with respect to the **Final Bias**.

$$\frac{d\ SSR}{d\ Final\ Bias}$$



Backpropagation: Details Part 7

⑱ **The Chain Rule** says that the derivative of the **SSR** with respect to the **Final Bias**…

$$\frac{d\ SSR}{d\ Final\ Bias} = \frac{d\ SSR}{d\ Predicted} \times \frac{d\ Predicted}{d\ Final\ Bias}$$

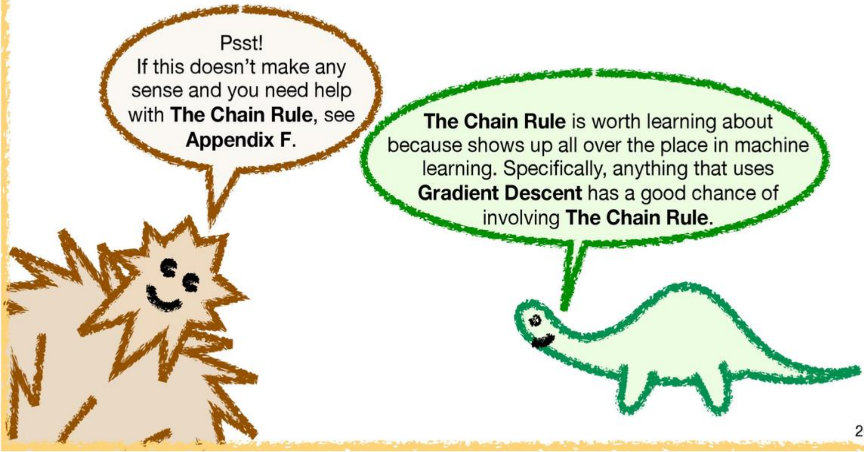
…is the derivative of the **SSR** with respect to the **Predicted** values…

$$\boxed{\mathbf{SSR}} = \sum_{i=1}^{n} (\text{Observed}_i - \boxed{\text{Predicted}_i})^2$$

…multiplied by the derivative of the **Predicted** values with respect to the **Final Bias**.

$$\boxed{\text{Predicted}} = \text{green squiggle} = \text{blue curve} + \text{orange curve} + \boxed{\textbf{Final Bias}}$$

BAM!!!



Backpropagation: Details Part 8

⑲ Now that we see that the derivative of the **SSR** with respect to the **Final Bias**…

…is the derivative of the **SSR** with respect to the **Predicted** values…

…multiplied by the derivative of the **Predicted** values with respect to the **Final Bias**…

$$\frac{d\ SSR}{d\ Final\ Bias} = \frac{d\ SSR}{d\ Predicted} \times \frac{d\ Predicted}{d\ Final\ Bias}$$

⑳ …we can solve for the *first* part, the derivative of the **SSR** with respect to the **Predicted** values…

…which, in turn, can be solved using **The Chain Rule**…

NOTE: For more details on how to solve for this derivative, see **Chapter 5**.

$$\frac{d\ SSR}{d\ Predicted} = \frac{d}{d\ Predicted} \sum_{i=1}^{n} (\text{Observed}_i - \text{Predicted}_i)^2$$

…by moving the square to the front…

…and multiplying everything by the derivative of the stuff inside the parentheses, which is **-1**…

$$= \sum_{i=1}^{n} 2 \times (\text{Observed}_i - \text{Predicted}_i) \times -1$$

…and, lastly, we simplify by multiplying **2** by **-1**.

$$\frac{d\ SSR}{d\ Predicted} = \sum_{i=1}^{n} -2 \times (\text{Observed}_i - \text{Predicted}_i)$$

BAM!!!

We solved for the *first* part of the derivative. Now let's solve for the *second* part.

Backpropagation: Details Part 9

㉑ The *second* part, the derivative of the **Predicted** values with respect to the **Final Bias**…

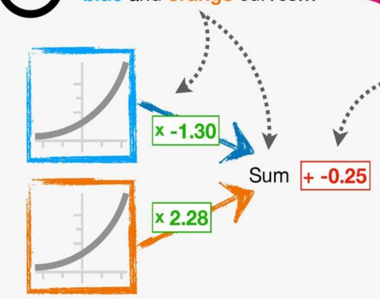
…is the derivative of the **green squiggle** with respect to the **Final Bias**…

$$\frac{d\ \textit{Predicted}}{d\ \textit{Final Bias}} = \frac{d}{d\ \textit{Final Bias}}\ \textbf{green squiggle}$$

…which, in turn, is the derivative of the sum of the **blue** and **orange** curves and the **Final Bias**.

$$= \frac{d}{d\ \textit{Final Bias}}(\textbf{blue curve} + \textbf{orange curve} + \textbf{Final Bias})$$

㉒ Now, remember that the **blue** and **orange** curves…



…were created before we got to the **Final Bias**…

…thus, the derivatives of the **blue** and **orange** curves with respect to the **Final Bias** are both **0** because they do not depend on the **Final Bias**…

$$\frac{d}{d\ \textit{Final Bias}}(\textbf{blue curve} + \textbf{orange curve} + \textbf{Final Bias}) = 0 + 0 + 1$$

…and the derivative of the **Final Bias** with respect to the **Final Bias** is **1**. So, when we do the math, the derivative of the Predicted values with respect to the **Final Bias** is **1**.

$$\boxed{\frac{d\ \textit{Predicted}}{d\ \textit{Final Bias}} = 1}$$

DOUBLE BAM!!!

We solved for the *second* part of the derivative.

Backpropagation: Details Part 10

㉓ Now, to get the derivative of the **SSR** with respect to the **Final Bias**, we simply plug in…

…the derivative of the **SSR** with respect to the **Predicted** values…

…and the derivative of the **Predicted** values with respect to the **Final Bias**.

$$\frac{d\ SSR}{d\ Predicted} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i)$$

$$\frac{d\ Predicted}{d\ Final\ Bias} = 1$$

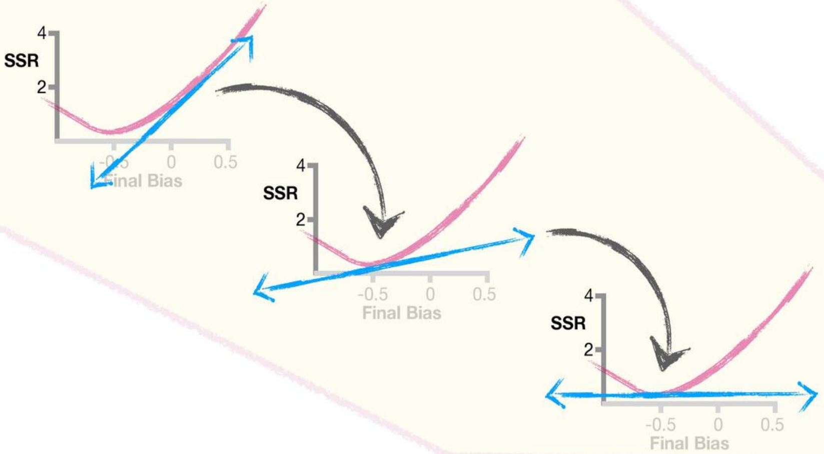
$$\frac{d\ SSR}{d\ Final\ Bias} = \frac{d\ SSR}{d\ Predicted} \times \frac{d\ Predicted}{d\ Final\ Bias}$$

$$\boxed{\frac{d\ SSR}{d\ Final\ Bias} = \sum_{i=1}^{n} -2 \times (Observed_i - Predicted_i) \times 1}$$

㉔ At long last, we have the derivative of the **SSR** with respect to the **Final Bias!!!**

TRIPLE BAM!!!

㉕ In the next section, we'll plug the derivative into **Gradient Descent** and solve for the optimal value for the **Final Bias**.

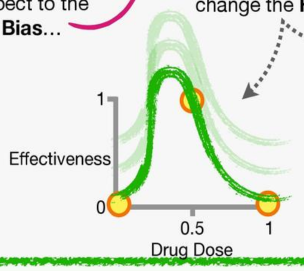


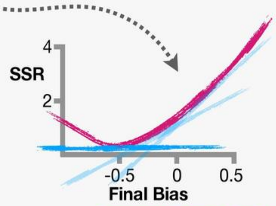
Backpropagation: Step-by-Step

① Now that we have the derivative of the **SSR** with respect to the **Final Bias**…

…which tells us how the **SSR** changes when we change the **Final Bias**…

…we can optimize the **Final Bias** with **Gradient Descent**.

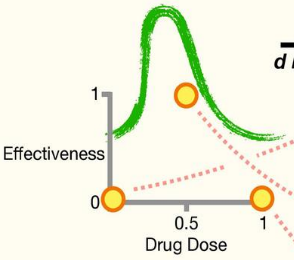




$$\frac{d\ SSR}{d\ Final\ Bias} = \sum_{i=1}^{n} -2 \times (\text{Observed}_i - \text{Predicted}_i) \times 1$$

NOTE: We're leaving the **"x 1"** term in the derivative to remind us that it comes from **The Chain Rule**. However, multiplying by **1** doesn't do anything, and you can omit it.

② First, we plug in the **Observed** values from the **Training Dataset** into the derivative of the **SSR** with respect to the **Final Bias**.



$$\frac{d\ SSR}{d\ Final\ Bias} = \sum_{i=1}^{n} -2 \times (\text{Observed}_i - \text{Predicted}_i)^2 \times 1$$

$$= -2 \times (\text{Observed}_1 - \text{Predicted}_1) \times 1$$

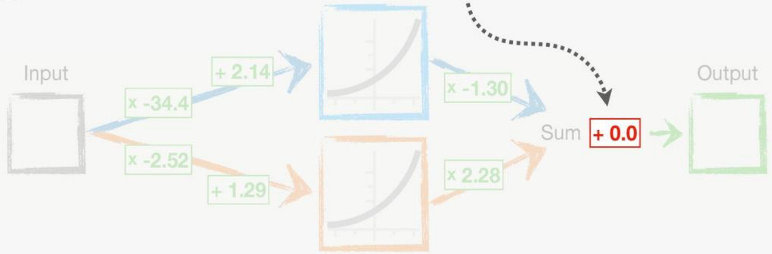
$$+ -2 \times (\text{Observed}_2 - \text{Predicted}_2) \times 1$$

$$+ -2 \times (\text{Observed}_3 - \text{Predicted}_3) \times 1$$

$$\frac{d\ SSR}{d\ Final\ Bias} = -2 \times (0 - \text{Predicted}_1) \times 1 + -2 \times (1 - \text{Predicted}_2) \times 1 + -2 \times (0 - \text{Predicted}_3) \times 1$$

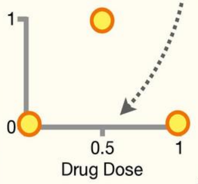
Backpropagation: Step-by-Step

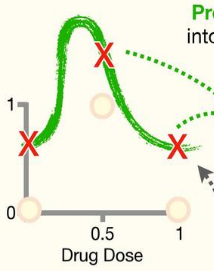
③ Then we initialize the **Final Bias** to a random value. In this case, we'll set the **Final Bias** to **0.0**.

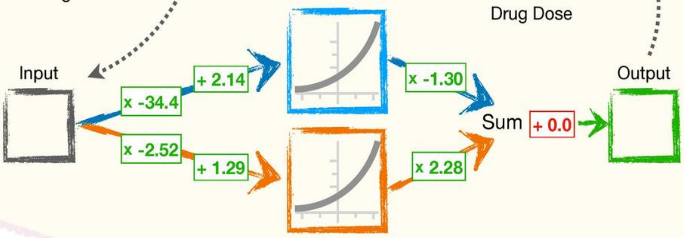


④ Then we run the **3 Doses** from the **Training Dataset**, **0**, **0.5** and **1**, through the **Neural Network** to get the **Predicted** values….

...and we plug the **Predicted** values into the derivative.







$$\frac{d\ SSR}{d\ Final\ Bias} = -2 \times (0 - \text{Predicted}) \times 1 + -2 \times (1 - \text{Predicted}) \times 1 + -2 \times (0 - \text{Predicted}) \times 1$$

$$= -2 \times (0 - 0.57) \times 1 + -2 \times (1 - 1.61) \times 1 + -2 \times (0 - 0.58) \times 1$$

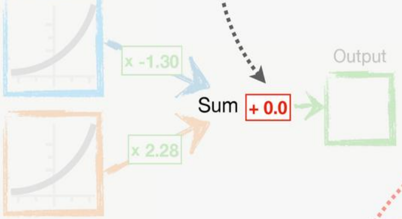
Backpropagation: Step-by-Step

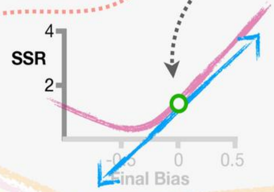
⑤ Now we evaluate the derivative at the current value for the **Final Bias**, which, at this point, is **0.0**.

When we do the math, we get **3.5**…

…thus, when the **Final Bias = 0**, the slope of this **tangent line** is **3.5**.

$$\frac{d\ SSR}{d\ Final\ Bias} = -2 \times (0 - 0.57) \times 1 + -2 \times (1 - 1.61) \times 1 + -2 \times (0 - 0.58) \times 1 \quad \boxed{= 3.5}$$





⑥ Then we calculate the **Step Size** with the standard equation for **Gradient Descent** and get **0.35**.

NOTE: In this example, we've set the **Learning Rate** to **0.1**.

Step Size = **Derivative** x **Learning Rate**

= **3.5** x **0.1**

= **0.35**

Gentle Reminder: The magnitude of the derivative is proportional to how big of a step we should take toward the minimum. The sign (+/-) tells us in what direction.

⑦ Lastly, we calculate a *new* value for the **Final Bias** from the *current* **Final Bias**…

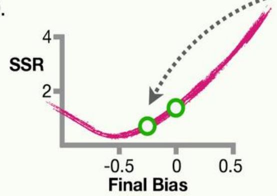
New Bias = **Current Bias** - **Step Size**

= **0.0** - **0.35**

= **-0.35**

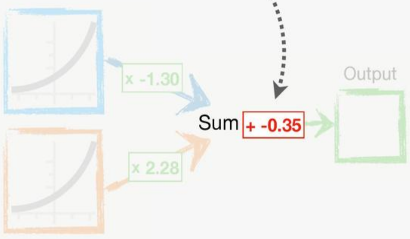
Remember, we initialized the **Final Bias** to **0.0**.

…and the *new* **Final Bias** is **-0.35**, which results in a lower **SSR**.

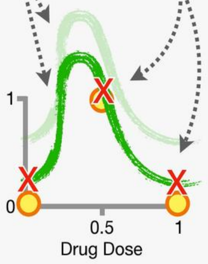


Backpropagation: Step-by-Step

8 With the new value for the **Final Bias**, **-0.35**…



…we shift the **green squiggle** down a bit, and the **Predicted** values are closer to the **Observed** values.



BAM!!!

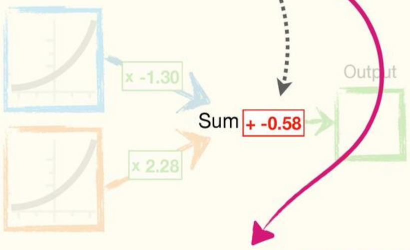
9 Now **Gradient Descent** iterates over the past three steps…

a Evaluate the derivative at the current value…

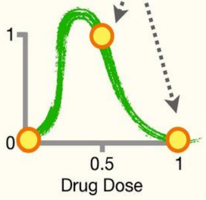
b Calculate the **Step Size**…

c Calculate the new value…

…and after **7** iterations, the **Final Bias** = **-0.58**…

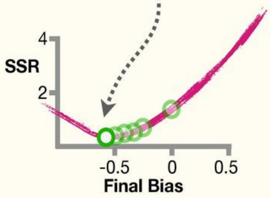


…and the **green squiggle** fits the **Training Data** really well…



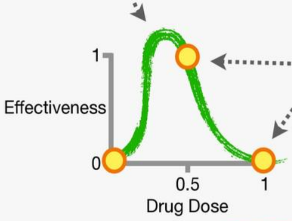
…and we've made it to the lowest **SSR**.

BAM!!!



Neural Networks: FAQ

Where the heck did this bump in the green squiggle come from?

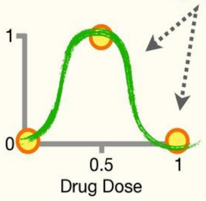


When we used **Backpropagation** to estimate the **Weights** and **Biases** in the **Neural Network**, we only calculated the **SSR** for the original **3** Doses, **0**, **0.5**, and **1**.

That means we only judge the **green squiggle** by how well it predicts Effectiveness at the original **3** Doses, **0**, **0.5**, and **1**, *and no other* Doses.

And that means the **green squiggle** can do whatever it wants in between the original **3** Doses, including making a strange bump that may or may not make good predictions. This is something I think about when people talk about using **Neural Networks** to drive cars. The **Neural Network** probably fits the **Training Data** really well, but there's no telling what it's doing between points, and that means it will be hard to predict what a self-driving car will do in new situations.

Wouldn't it be better if the green squiggle was a bell-shaped curve fit to the Training Data?



Maybe. Because we don't have any data between the **3** Doses in the **Training Data**, it's hard to say what the best fit would be.

When we have *Neural Networks*, which are cool and super flexible, why would we ever want to use *Logistic Regression*, which is a lot less flexible?

Neural Networks are cool, but deciding how many **Hidden Layers** to use and how many **Nodes** to put in each **Hidden Layer** and even picking the best **Activation Function** is a bit of an art form. In contrast, creating a model with **Logistic Regression** is a science, and there's no guesswork involved. This difference means that it can sometimes be easier to get **Logistic Regression** to make good predictions than a **Neural Network**, which might require a lot of tweaking before it performs well.

Furthermore, when we use a lot of variables to make predictions, it can be much easier to interpret a **Logistic Regression** model than a **Neural Network**. In other words, it's easy to know how **Logistic Regression** makes predictions. In contrast, it's much more difficult to understand how a **Neural Network** makes predictions.


Hooray!!!
Congratulations!!! TRIPLE BAM!!!
We made it to the end of an exciting book about machine learning!!!

Appendices!!!

Stuff you might have learned in school but forgot

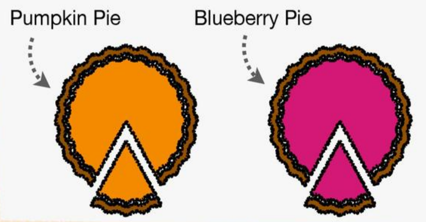
Appendix A:

Pie Probabilities

Appendix A: Pie Probabilities

1 We're in **StatLand**, where **70%** of the people prefer pumpkin pie and **30%** of the people prefer blueberry pie.

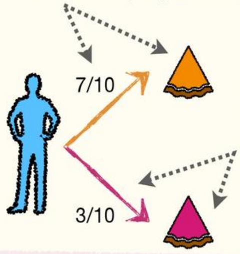
In other words, **7** out of **10** people in **StatLand**, or **7/10**, prefer pumpkin pie, and the remaining **3** out of **10** people, or **3/10**, prefer blueberry pie.



NOTE: In this case, randomly meeting one person who prefers pumpkin or blueberry pie does not affect the next person's pie preference. In probability lingo, we say that these two events, discovering the pie preferences from two randomly selected people, are **independent**. If, for some strange reason, the second person's preference was influenced by the first, the events would be **dependent**, and, unfortunately, the math ends up being different from what we show here.

2 Now, if we just randomly ask someone which type of pie they prefer, **7** out of **10** people, or **7/10ths** of the people, will say pumpkin…

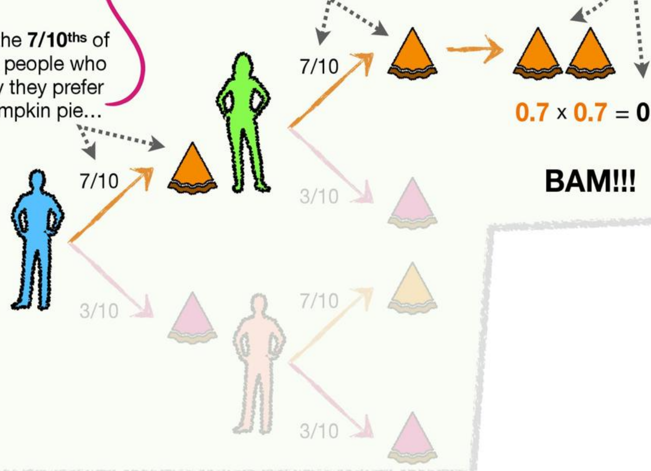
…and the remaining **3** out of **10** people, or **3/10ths**, will say blueberry.



3 Of the **7/10ths** of the people who say they prefer pumpkin pie…

…only **7/10ths** will be followed by people who also prefer pumpkin pie.

Thus, the probability that two people in a row will prefer pumpkin pie is **7/10ths** of the original **7/10ths** who preferred pumpkin pie, which is **7/10** × **7/10 = 49/100**, or **49%**.



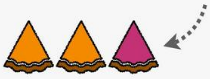
0.7 × 0.7 = **0.49**

BAM!!!

Appendix A: Pie Probabilities

4 Now let's talk about what happens when we ask a third person which pie they prefer.

Specifically, we'll talk about the probability that the first two people prefer pumpkin pie and the third person prefers blueberry.

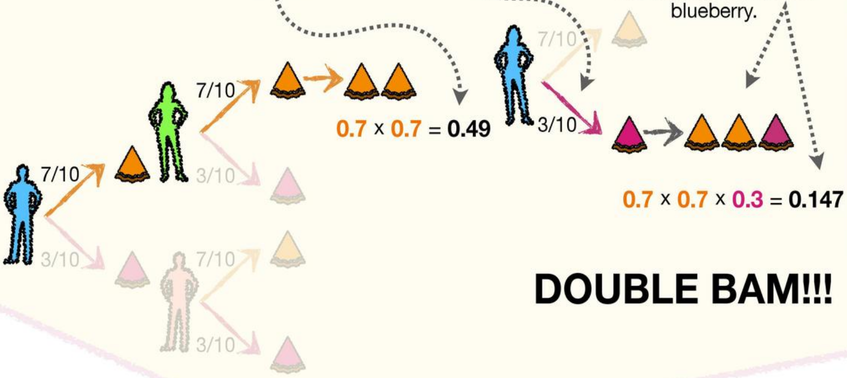


5 First, we just saw on the previous page that we will meet two people who both prefer pumpkin pie only **49%** of the time, or **49/100**.

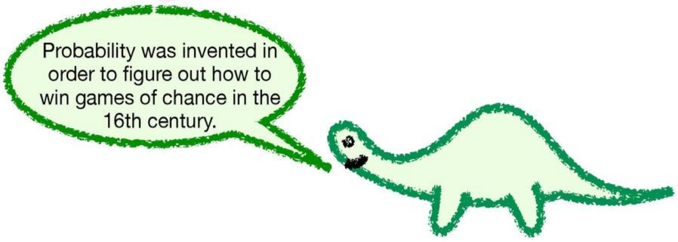
Now, of that **49/100**...

...only **3/10ths** will be followed by people who prefer blueberry pie.

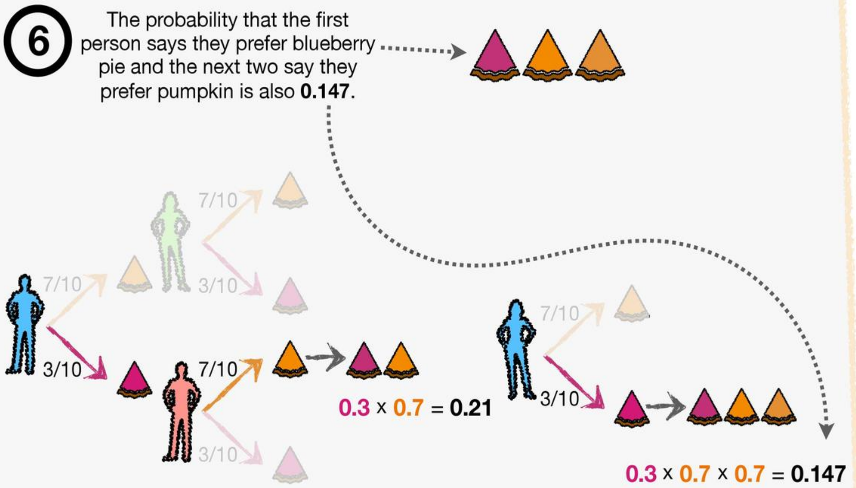
And to find **3/10ths** of the **49/100ths** who preferred pumpkin pie, we multiply: **3/10** x **49/100 = 147/1,000.** In other words, when we ask three random people their pie preferences, **14.7%** of the time the first two will prefer pumpkin pie and the third will prefer blueberry.



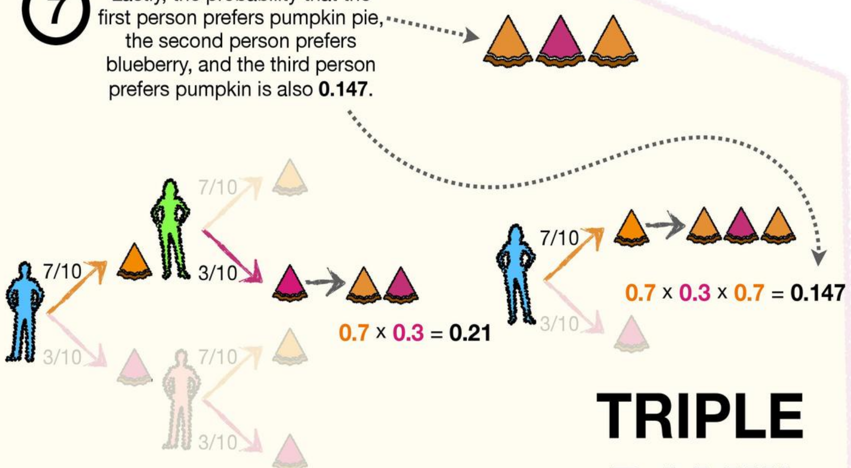
DOUBLE BAM!!!



⑥ The probability that the first person says they prefer blueberry pie and the next two say they prefer pumpkin is also **0.147**.



⑦ Lastly, the probability that the first person prefers pumpkin pie, the second person prefers blueberry, and the third person prefers pumpkin is also **0.147**.



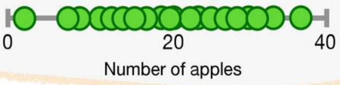
TRIPLE BAM!!!

Appendix B:

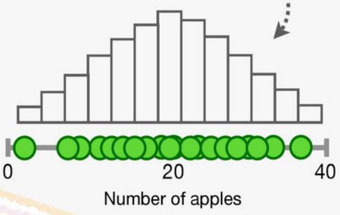
The Mean, Variance, and Standard Deviation

Appendix B: The Mean, Variance, and Standard Deviation

① Imagine we went to *all* **5,132 Spend-n-Save** food stores and counted the number of **green apples** that were for sale. We could plot the number of **green apples** at each store on this number line…

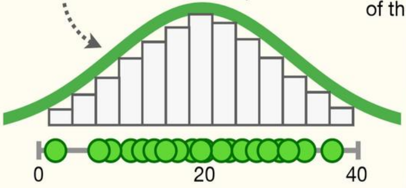


…but because there's a lot of overlap in the data, we can also draw a **Histogram** of the measurements.



② If we wanted to fit a **Normal Curve** to the data like this…

…then, first, we need to calculate the **Population Mean** to figure out where to put the center of the curve.

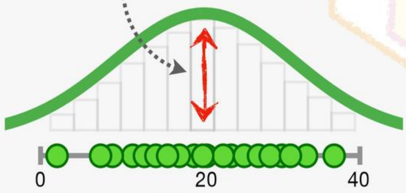


③ Because we counted the number of **green apples** in *all* **5,132 Spend-n-Save** stores, calculating the **Population Mean**, which is frequently denoted with the Greek character ***μ*** (***mu***), is relatively straightforward: we simply calculate the average of all of the measurements, which, in this case, is **20**.

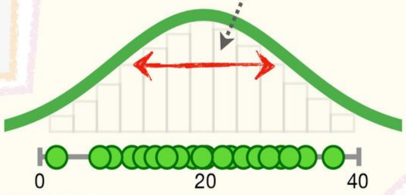
$$\textbf{Population Mean} = \mu = \frac{\text{Sum of Measurements}}{\text{Number of Measurements}}$$

$$= \frac{2 + 8 + ... + 37}{5132} = 20$$

④ Because the **Population Mean**, ***μ***, is **20**, we center the **Normal Curve** over **20**.

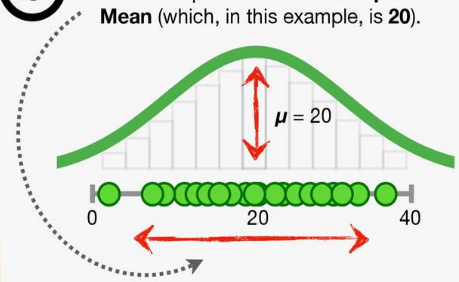


⑤ Now we need to determine the width of the curve by calculating the **Population Variance** (also called the **Population Variation**) and **Standard Deviation**.



Appendix B: The Mean, Variance, and Standard Deviation

6 In other words, we want to calculate how the data are spread around the **Population Mean** (which, in this example, is **20**).

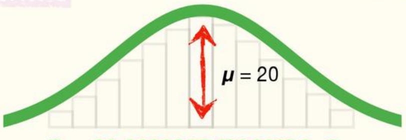


7 The formula for calculating the **Population Variance** is…

$$\textbf{Population Variance} = \frac{\sum (x - \mu)^2}{n}$$

…which is a pretty fancy-looking formula, so let's go through it one piece at a time.

8 The part in the parentheses, ***x* - *μ***, means we subtract the **Population Mean**, ***μ***, from each measurement, ***x***.



$$\textbf{Population Variance} = \frac{\sum \boxed{(x - \mu)}^2}{n}$$

For example, the first measurement is **2**, so we subtract ***μ***, which is **20**, from **2**…

$(2 - 20)$ $(8 - 20)$ $(28 - 20)$

$$\text{Population Variance} = \frac{\sum (x - \mu)^{\boxed{2}}}{n}$$

…then the square tells us to square each term…

$(2 - 20)^2$ $(8 - 20)^2$ $(28 - 20)^2$

$$\text{Population Variance} = \frac{\boxed{\sum} (x - \mu)^2}{n}$$

…and the Greek character **Σ** (**Sigma**) tells us to add up all of the terms…

$$\frac{(2 - 20)^2 + (8 - 20)^2 + \dots + (28 - 20)^2}{\text{Number of Measurements}}$$

…and lastly, we want the average of the squared differences, so we divide by the total number of measurements, ***n***, which, in this case, is *all* **Spend-n-Save** food stores, **5,132**.

$$\textbf{Population Variance} = \frac{\sum (x - \mu)^2}{\boxed{n}}$$

Appendix B: The Mean, Variance, and Standard Deviation

9 Now that we know how to calculate the **Population Variance**…

…when we do the math, we get **100**.

BAM?

Nope, not yet.

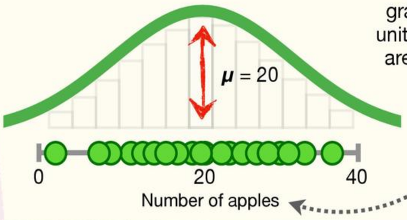
$$\text{Population Variance} = \frac{\sum (x - \mu)^2}{n} = \frac{(2 - 20)^2 + (8 - 20)^2 + \ldots + (28 - 20)^2}{5132} = 100$$

10 Because each term in the equation for **Population Variance** is squared…

…the units for the result, **100**, are **Number of Apples Squared**…

…and that means we can't plot the **Population Variance** on the graph, since the units on the x-axis are not squared.

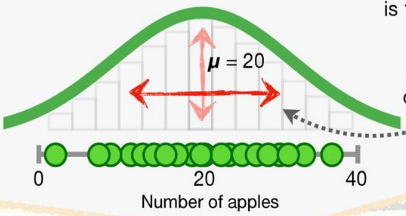
11 To solve this problem, we take the *square root* of the **Population Variance** to get the **Population Standard Deviation**…



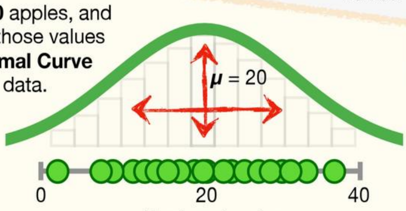
$$\text{Population Standard Deviation} = \sqrt{\frac{\sum (x - \mu)^2}{n}} = \sqrt{\text{Population Variance}} = \sqrt{100} = 10$$

…and because the **Population Variance** is **100**, the **Population Standard Deviation** is **10**…

…and we can plot that on the graph.



12 Now we have a graph that shows the **Population Mean**, **20**, plus and minus the **Population Standard Deviation**, **10** apples, and we can use those values to fit a **Normal Curve** to the data.

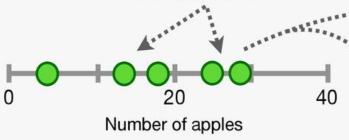


BAM!!!

13 **NOTE:** Before we move on, I want to emphasize that we almost never have the population data, so we almost never calculate the **Population Mean**, **Variance**, or **Standard Deviation**. Read on to learn what we do instead!

Appendix B: The Mean, Variance, and Standard Deviation

14 Instead of calculating **Population Parameters**, we *estimate* them from a relatively small number of measurements.

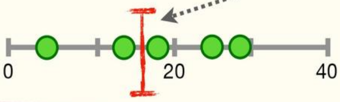


15 *Estimating* the **Population Mean** is super easy: we just calculate the average of the measurements we collected…

$$\textbf{Estimated Mean} = \frac{\text{Sum of Measurements}}{\text{Number of Measurements}}$$

$$= \frac{3 + 13 + 19 + 24 + 29}{5} = 17.6$$

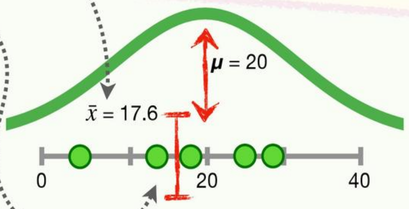
…and when we do the math, we get **17.6**.



16 **NOTE:** The **Estimated Mean**, which is often denoted with the symbol $\bar{x}$ (x-bar), is also called the **Sample Mean**…

…and due to the relatively small number of measurements used to calculate the **Estimated Mean**, it's different from the **Population Mean**.

A lot of **Statistics** is dedicated to quantifying and compensating for the differences between **Population Parameters**, like the **Mean** and **Variance**, and their *estimated* counterparts.



17 Now that we have an **Estimated Mean**, we can calculate an **Estimated Variance** and **Standard Deviation**. However, we have to compensate for the fact that we only have an **Estimated Mean**, which will almost certainly be different from the **Population Mean**.

18 Thus, when we calculate the **Estimated Variance** and **Standard Deviation** using the **Estimated Mean**…

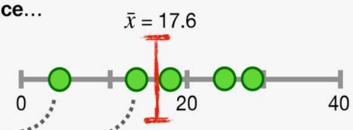
…we compensate for the difference between the **Population Mean** and the **Estimated Mean** by dividing by number of measurements minus **1**, ***n* - 1**, rather than ***n***.

$$\textbf{Estimated Variance} = \frac{\sum(x - \bar{x})^2}{n - 1}$$

$$\textbf{Estimated Standard Deviation} = \sqrt{\frac{\sum(x - \bar{x})^2}{n - 1}}$$

19 Now when we plug the data into the equation for the **Estimated Variance**...

$$\text{Estimated Variance} = \frac{\sum (x - \bar{x})^2}{n - 1}$$



$$= \frac{(3 - 17.6)^2 + (13 - 17.6)^2 + (19 - 17.6)^2 + (24 - 17.6)^2 + (29 - 17.6)^2}{5 - 1} = 101.8$$

...we get **101.8**, which is a pretty good estimate of the **Population Variance**, which, as we saw earlier, is **100**.

NOTE: If we had divided by ***n***, instead of ***n* - 1**, we would have gotten **81.4**, which is a significant *underestimate* of the true **Population Variance, 100**.

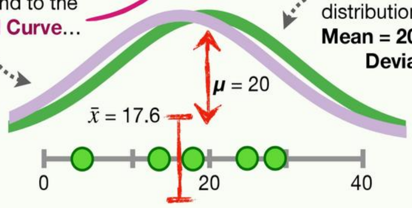
20 Lastly, the **Estimated Standard Deviation** is just the square root of the **Estimated Variance**...

...so, in this example, the **Estimated Standard Deviation** is **10.1**. Again, this is relatively close to the **Population** value we calculated earlier.

$$\text{Estimated Standard Deviation} = \sqrt{\frac{\sum (x - \bar{x})^2}{n - 1}} = \sqrt{\text{Estimated Variance}} = \sqrt{101.8} = 10.1$$

21 The **Estimated Mean, 17.6**, and **Standard Deviation, 10.1**, correspond to the **purple Normal Curve**...

...which isn't too far off from the true **Population** distribution in **green**, with **Mean = 20** and **Standard Deviation = 10**.



TRIPLE BAM!!!

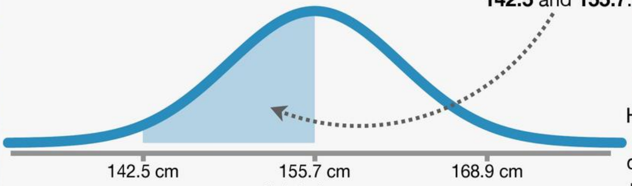
Appendix C:

Computer Commands for Calculating Probabilities with Continuous Probability Distributions

Appendix C: Computer Commands for Calculating Probabilities with Continuous Probability Distributions

① Given this **Normal Distribution**, with **Mean = 155.7** and **Standard Deviation = 6.6**…

…let's talk about ways we can get a computer to calculate the area under the curve between **142.5** and **155.7**.



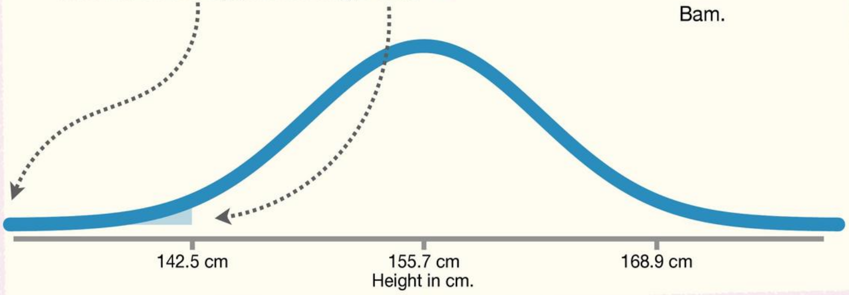
However, before we get into the specific commands that we can use in **Google Sheets**, **Microsoft Excel**, and ***R***, we need to talk about **Cumulative Distribution Functions**.

② A **Cumulative Distribution Function** (**CDF**) simply tells us the area under the curve up to a specific point.

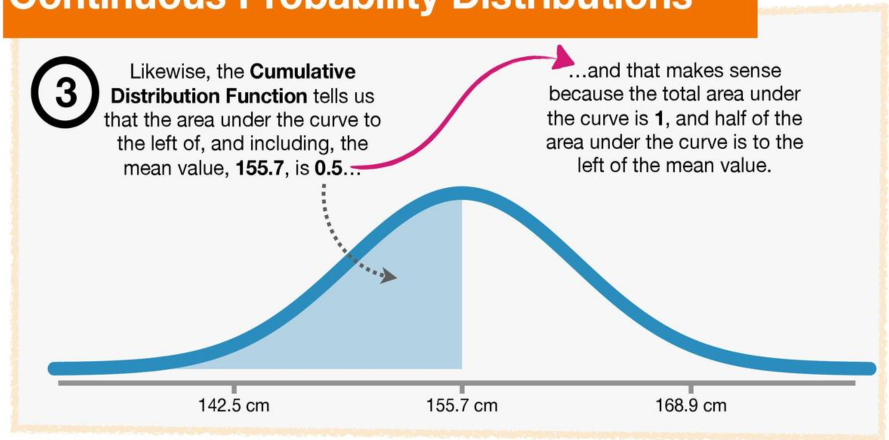
For example, given this **Normal Distribution** with **Mean = 155.7** and **Standard Deviation = 6.6**, a **Cumulative Distribution Function** for **142.5** would tell us the area under the curve for all x-axis values to the left of, and including, **142.5**.

In this case, the area is **0.02**.

Bam.

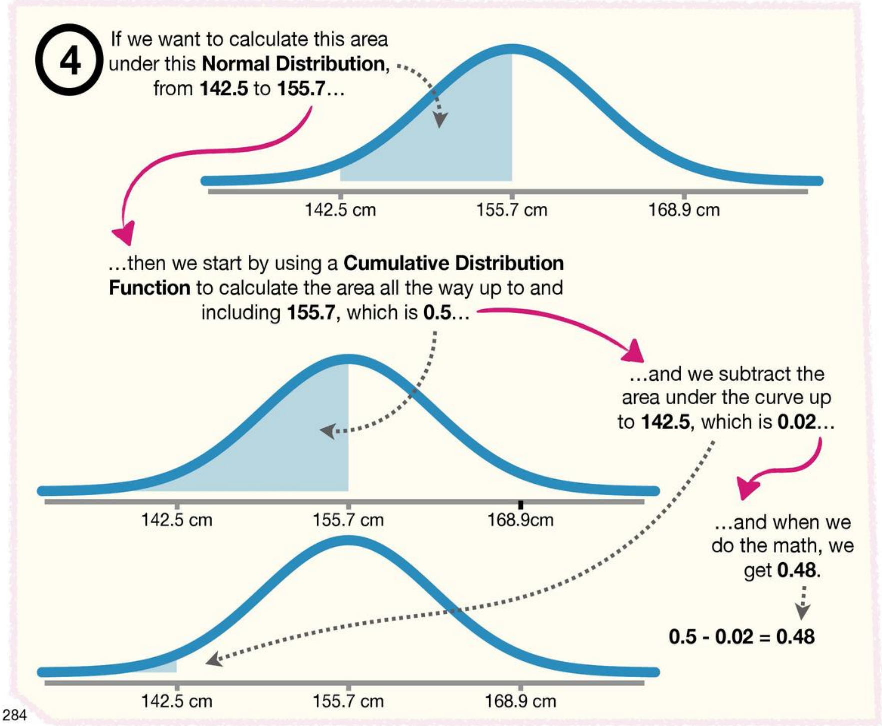


Appendix C: Computer Commands for Calculating Probabilities with Continuous Probability Distributions



Likewise, the **Cumulative Distribution Function** tells us that the area under the curve to the left of, and including, the mean value, **155.7**, is **0.5**…

…and that makes sense because the total area under the curve is **1**, and half of the area under the curve is to the left of the mean value.



If we want to calculate this area under this **Normal Distribution**, from **142.5** to **155.7**…

…then we start by using a **Cumulative Distribution Function** to calculate the area all the way up to and including **155.7**, which is **0.5**…

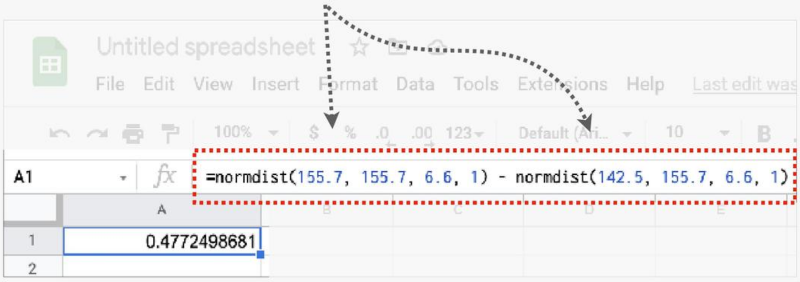
…and we subtract the area under the curve up to **142.5**, which is **0.02**…

…and when we do the math, we get **0.48**.

0.5 - 0.02 = 0.48

Appendix C: Computer Commands for Calculating Probabilities with Continuous Probability Distributions

5 Now, if we want to do those calculations with **Google Sheets** or **Microsoft Excel**, we use the **NORMDIST()** function.



```
=normdist(155.7, 155.7, 6.6, 1) - normdist(142.5, 155.7, 6.6, 1)
```

A1: 0.4772498681

6 The **NORMDIST()** function takes **4** arguments:

```
normdist( x-axis value, mean, standard deviation, use CDF)
```

…the x-axis value that we want to calculate the area under the curve to the left of, and including; in our example, this means we set this to either **155.7** or **142.5**…

…the **Mean** of the **Normal Distribution**, which in this example is **155.7**…

…the **Standard Deviation**, which in this example is **6.6**…

…and either **0** or **1**, depending on whether or not we want to use the **Cumulative Distribution Function** (**CDF**). In this example, we set it to **1** because we want to use the **CDF**.

Appendix C: Computer Commands for Calculating Probabilities with Continuous Probability Distributions

Gentle Reminder about the arguments for the **NORMDIST()** function:

```
normdist( x-axis value, mean, standard deviation, use CDF)
```

7 Now, if we want to use **NORMDIST()** to calculate the area under the curve between **142.5** and **155.7**…

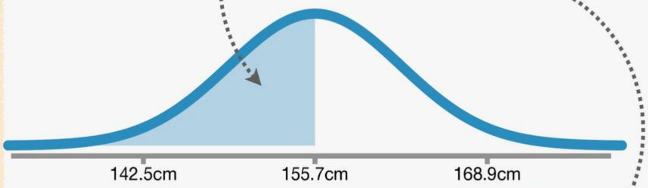
…we start by using **NORMDIST()** to calculate the area all the way up to and including **155.7**…

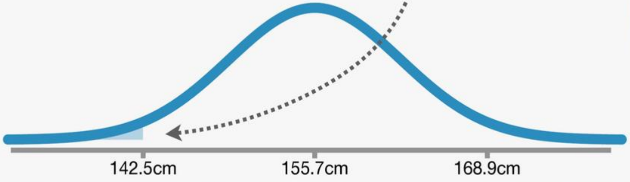
…and then use **NORMDIST()** to subtract the area under the curve up to **142.5** and the result is **0.48**.

```
normdist( 155.7, 155.7, 6.6, 1)

        - normdist( 142.5, 155.7, 6.6, 1) = 0.48
```

DOUBLE BAM!!





8 In the programming language called ***R***, we can get the same result using the **pnorm()** function, which is just like **NORMDIST()**, except we don't need to specify that we want to use a **CDF**.

```
pnorm(155.7, mean=155.7, sd=6.6)

        - pnorm(142.5, mean=155.7, sd=6.6) = 0.48
```

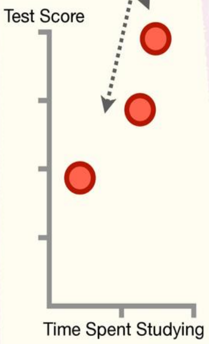
TRIPLE BAM!!!

Appendix D:

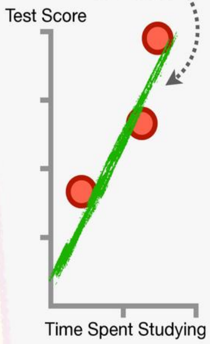
The Main Ideas of Derivatives

Appendix D: The Main Ideas of Derivatives

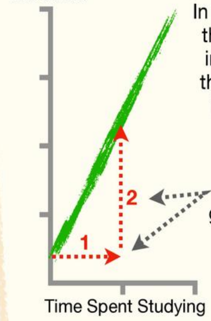
1 Imagine we collected Test Scores and Time Spent Studying from **3** people…



2 …and we fit a **straight line** to the data.



3 One way to understand the relationship between Test Scores and Time Spent Studying is to look at changes in Test Scores relative to changes in Time Spend Studying.



In this case, we see that for every unit increase in Time, there is a two-unit increase in Test Score.

In other words, we can say we go up **2** for every **1** we go over.

4 **NOTE:** The relationship "**2** up for every **1** over" holds even if we only go over **1/2** unit.

In this case, we see that for every half-unit increase in Time, there is a **2 x 0.5 = 1** unit increase in Test Score.

5 Because the "**2** up for every **1** over" relationship holds no matter how small a value for Time Spent Studying, we say that the **Derivative** of this **straight line**…

…is the change in Score (***d Score***) relative to the change in Time (***d Time***), which is **2**.

$$\frac{d\ Score}{d\ Time} = 2$$

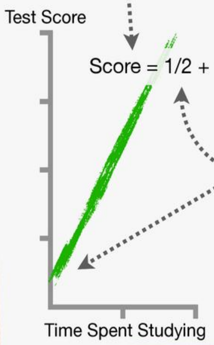
Now let's talk about how the **Derivative** is related to the **straight line**.

Appendix D: The Main Ideas of Derivatives

6

Here is the equation for the **straight line**.

The slope is **2**…

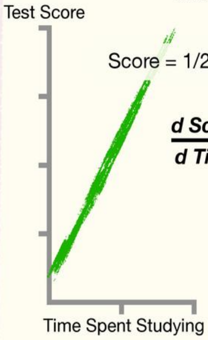


Score = 1/2 + (2 x Time)

…and the y-axis intercept is **1/2**.

7

Thus, we see that the slope, **2**, is equal to the **Derivative**, **2**, and both tell us to go "**2** up for every **1** over."



Score = 1/2 + $\boxed{2 \text{ x}}$ Time)

$$\frac{\textit{d Score}}{\textit{d Time}} = 2$$

8

Now let's stop talking about Studying and talk about Eating. This line has a slope of **3**, and, regardless of how small a value for Time Spent Eating, our Fullness goes up **3** times that amount.

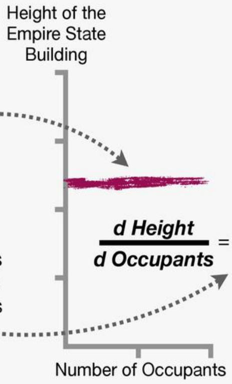
Thus, the **Derivative**, the change in Fullness relative to the change in Time, is **3**.

$$\frac{\textit{d Fullness}}{\textit{d Time}} = 3$$

9

When the slope is **0**, and the y-axis value never changes, regardless of the x-axis value…

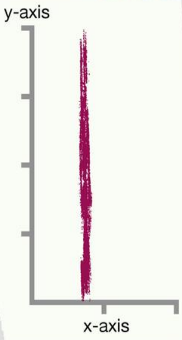
…then the **Derivative**, the change in the y-axis value relative to the change in the x-axis value, is **0**.



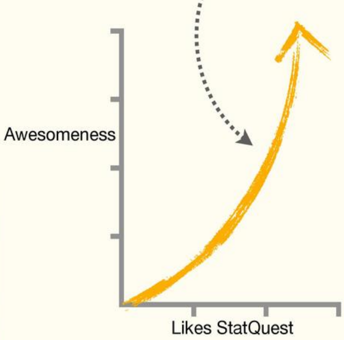
$$\frac{\textit{d Height}}{\textit{d Occupants}} = 0$$

10

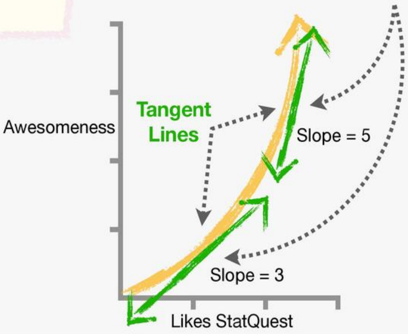
When the straight line is vertical, and the x-axis value never changes, then the **Derivative** is undefined. This is because it's impossible to measure the change in the y-axis value relative to the change in the x-axis value if the x-axis value never changes.



11 Lastly, when we have a curved line instead of a straight line…



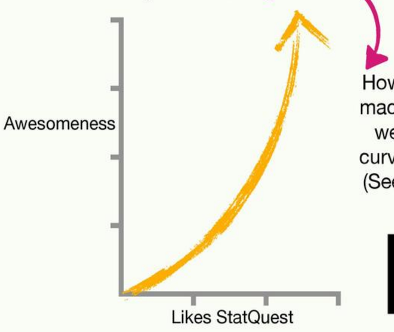
12 …the **Derivative** is the slope of any **straight line** that just barely touches the **curved line** at a single point.



Terminology Alert!!!

A straight line that touches the curve at a single point is called a **Tangent Line**.

13 Unfortunately, the **Derivatives** of curved lines are not as easy to determine as they are for **straight lines**.



However, the good news is that in machine learning, **99%** of the time we can find the **Derivative** of a curved line using **The Power Rule** (See **Appendix E**) and **The Chain Rule** (See **Appendix F**).

BAM!!!

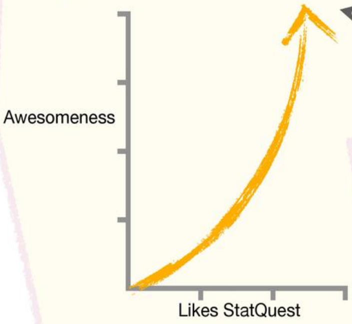
Appendix E:

The Power Rule

NOTE: This appendix assumes that you're already familiar with the concept of a derivative (**Appendix D**).

Appendix E: The Power Rule

① In this example, we have a **parabola** that represents the relationship between Awesomeness and Likes StatQuest.



This is the equation for the **parabola**:

$$\text{Awesomeness} = (\text{Likes StatQuest})^2$$

② We can calculate the derivative, the change in Awesomeness with respect to the change in Likes StatQuest…

$$\frac{d}{d \textit{ Likes StatQuest}} \text{Awesomeness}$$

…by first plugging in the equation for Awesomeness…

$$\frac{d}{d \textit{ Likes StatQuest}} (\text{Likes StatQuest})^2$$

…and then applying **The Power Rule**.

③ **The Power Rule** tells us to multiply Likes StatQuest by the power, which, in this case, is **2**…

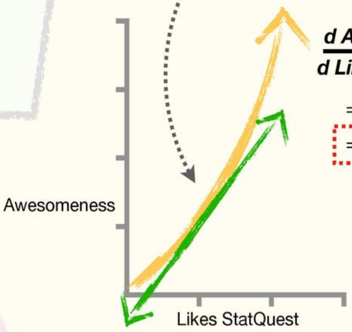
$$\frac{d}{d \textit{ Likes StatQuest}} (\text{Likes StatQuest})^2 = 2 \times \text{Likes StatQuest}^{2-1}$$

$$\boxed{= 2 \times \text{Likes StatQuest}}$$

…and raise Likes StatQuest by the original power, **2**, minus **1**…

…and since **2-1 = 1**, the derivative of Awesomeness with respect to Likes StatQuest is **2** times Likes StatQuest.

④ For example, when Likes StatQuest = **1**, the derivative, the slope of the **tangent line** is **2**.



$$\frac{\textit{d Awesomeness}}{\textit{d Likes StatQuest}} =$$

$$= 2 \times \text{Likes StatQuest}$$

$$\boxed{= 2 \times 1 = 2}$$

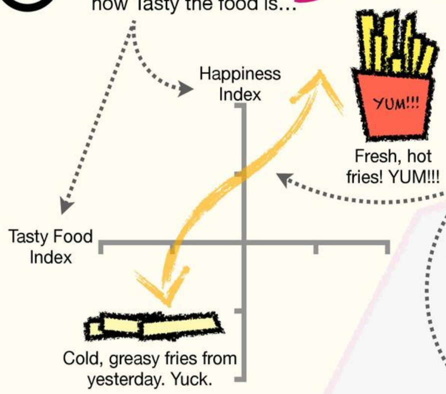
BAM!!!

Now let's look at a fancier example!!!

Appendix E: The Power Rule

5 Here we have a graph of how Happy people are relative to how Tasty the food is…

…and this is the equation for the **squiggle**:



$$\text{Happy} = 1 + \text{Tasty}^3$$

6 We can calculate the derivative, the change in Happiness with respect to the change in Tastiness…

$$\frac{d\,\text{Happy}}{d\,\text{Tasty}} = \frac{d}{d\,\text{Tasty}}\,\text{Happy}$$

…by plugging in the equation for Happy…

$$\frac{d}{d\,\text{Tasty}}(1 + \text{Tasty}^3)$$

…and taking the derivative of each term in the equation.

$$\frac{d}{d\,\text{Tasty}}(1 + \text{Tasty}^3) = \boxed{\frac{d}{d\,\text{Tasty}}1} + \boxed{\frac{d}{d\,\text{Tasty}}\text{Tasty}^3}$$

7 The constant value, **1**, doesn't change, regardless of the value for Tasty, so the derivative with respect to Tasty is **0**.

$$\frac{d}{d\,\text{Tasty}}1 = 0$$

The Power Rule tells us to multiply Tasty by the power, which, in this case is **3**…

…and raise Tasty by the original power, **3**, minus **1**.

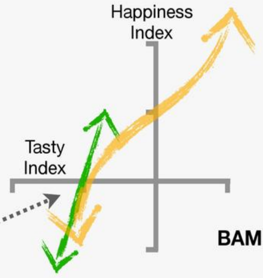
$$\frac{d}{d\,\text{Tasty}}\text{Tasty}^3 = 3 \times \text{Tasty}^{3-1}$$
$$= 3 \times \text{Tasty}^2$$

8 Lastly, we recombine both terms to get the final derivative.

$$\frac{d}{d\,\text{Tasty}}\text{Happy} = 0 + 3 \times \text{Tasty}^2$$
$$= 3 \times \text{Tasty}^2$$

9 Now, when Tasty = **-1**, the slope of the **tangent line** is **3**.

$$\frac{d}{d\,\text{Tasty}}3 \times \text{Tasty}^2$$
$$= 3 \times -1^2 = 3$$



BAM!!!

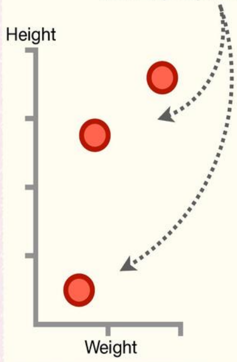
Appendix F:

The Chain Rule!!!

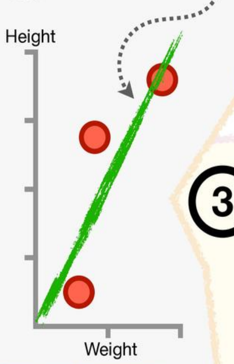
NOTE: This appendix assumes that you’re already familiar with the concept of a derivative (**Appendix D**) and **The Power Rule** (**Appendix E**).

Appendix F: The Chain Rule

① Here we have Weight and Height measurements from **3** people…

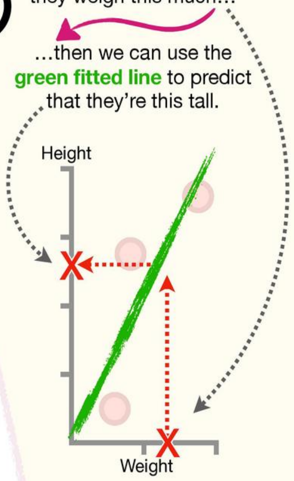


② …and we fit a line to the data.

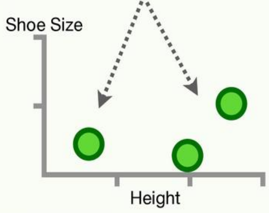


③ Now, if someone tells us that they weigh this much…

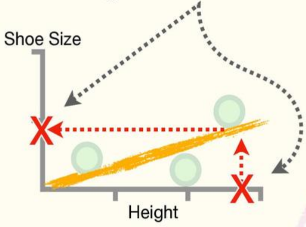
…then we can use the **green fitted line** to predict that they're this tall.

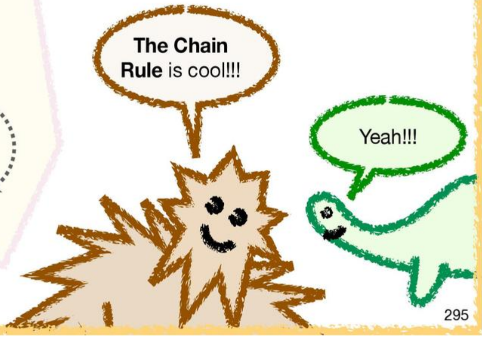


④ Here we have Height and Shoe Size measurements…

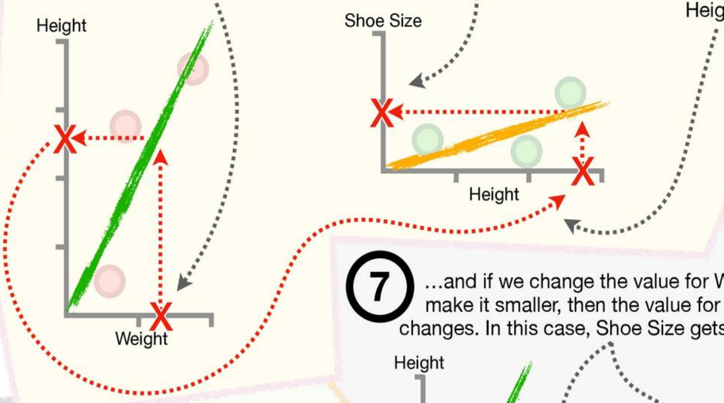


⑤ …and we can use an **orange fitted line** to predict Shoe Size from a Height measurement.





Appendix F: The Chain Rule

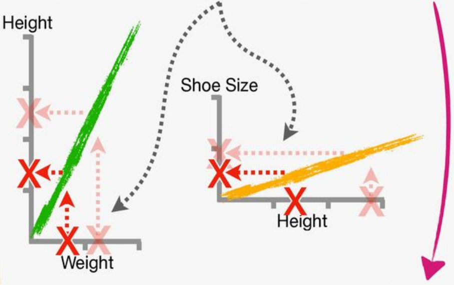


6 Now, if someone tells us that they weigh this much…

…then we can predict that this is their Shoe Size…

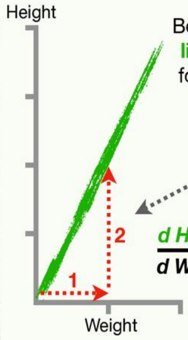
…because Weight and Shoe Size are connected by Height…

7 …and if we change the value for Weight, like make it smaller, then the value for Shoe Size changes. In this case, Shoe Size gets smaller, too.



Now, if we want to quantify how much Shoe Size changes when we change Weight, we need to calculate the derivative for Shoe Size with respect to Weight.

8 Because this **straight line** goes up **2** units for every **1** unit over, the slope and derivative are **2**…

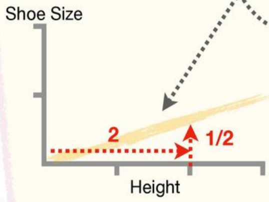


$$\frac{d\ Height}{d\ Weight} = 2$$

…and that means the equation for Height is:

$$Height = \frac{d\ Height}{d\ Weight} \times Weight = 2 \times Weight$$

9 Likewise, this straight line goes up **1/2** units for every **2** units over, so the slope and derivative are **1/4**…



$$\frac{d\ Size}{d\ Height} = \frac{1/2}{2} = \frac{1}{4}$$

…and that means the equation for Shoe Size is:

$$Shoe\ Size = \frac{d\ Size}{d\ Height} \times Height = \frac{1}{4} \times Height$$

Appendix F: The Chain Rule

⑩ Now, because Weight can predict Height…

$$\text{Height} = \frac{d\ \text{Height}}{d\ \text{Weight}} \times \text{Weight}$$

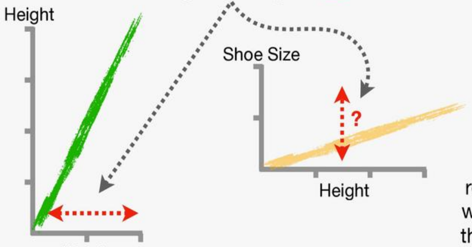
…and Height can predict Shoe Size…

$$\text{Shoe Size} = \frac{d\ \text{Size}}{d\ \text{Height}} \times \text{Height}$$

…we can plug the equation for Height into the equation for Shoe Size.

$$\text{Shoe Size} = \frac{d\ \text{Size}}{d\ \text{Height}} \times \frac{d\ \text{Height}}{d\ \text{Weight}} \times \text{Weight}$$

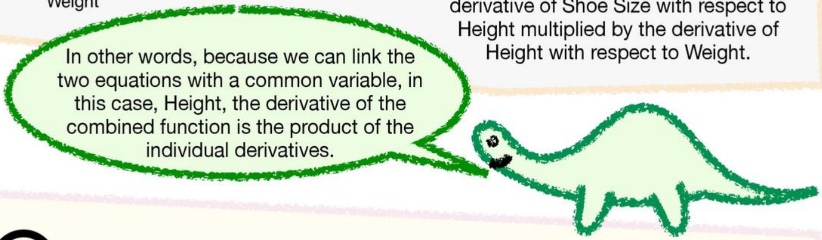
⑪ And if we want to determine how Shoe Size changes with respect to changes in Weight…



…we solve for the derivative of Shoe Size with respect to Weight…

$$\frac{d\ \text{Size}}{d\ \text{Weight}} = \frac{d\ \text{Size}}{d\ \text{Height}} \times \frac{d\ \text{Height}}{d\ \text{Weight}}$$

…and, using **The Power Rule** (or realizing that when we change Weight, we multiply it by both derivatives to get the new Shoe Size), we end up with the derivative of Shoe Size with respect to Height multiplied by the derivative of Height with respect to Weight.



⑫ Finally, we can plug in values for the derivatives…

Gentle Reminder:

$$\frac{d\ \text{Height}}{d\ \text{Weight}} = 2$$

$$\frac{d\ \text{Size}}{d\ \text{Height}} = \frac{1}{4}$$

$$\frac{d\ \text{Size}}{d\ \text{Weight}} = \frac{d\ \text{Size}}{d\ \text{Height}} \times \frac{d\ \text{Height}}{d\ \text{Weight}}$$

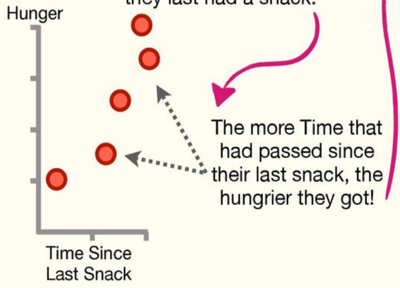
$$= \frac{1}{4} \times 2 = \frac{1}{2}$$

…and we see that when Weight increases by **1** unit, Shoe Size increases by **1/2**.

BAM!!!

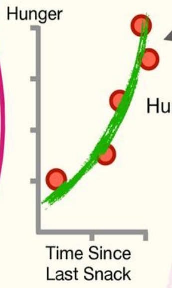
Appendix F: The Chain Rule, A More Complicated Example

① Now imagine we measured how Hungry a bunch of people were and how long it had been since they last had a snack.



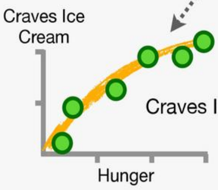
The more Time that had passed since their last snack, the hungrier they got!

So, we fit a quadratic line with an intercept of **0.5** to the measurements to reflect the increasing rate of Hunger.



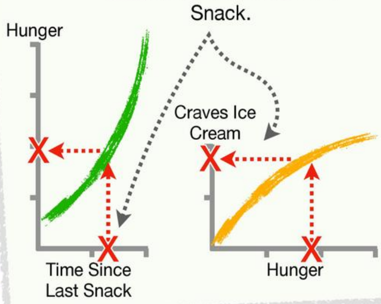
Hunger = $\text{Time}^2 + 0.5$

② Likewise, we fit a square root function to the data that shows how Hunger is related to craving ice cream.



Craves Ice Cream = $(\text{Hunger})^{1/2}$

③ Now we want to see how Craves Ice Cream changes relative to Time Since Last Snack.

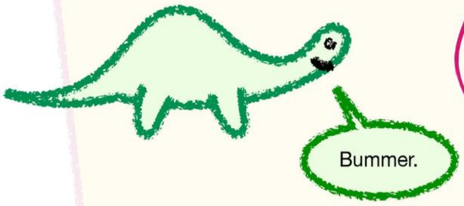


④ Unfortunately, when we plug the equation for Hunger into the equation for Craves Ice Cream…

Hunger = $\boxed{\text{Time}^2 + 0.5}$

Craves Ice Cream = $(\text{Hunger})^{1/2}$

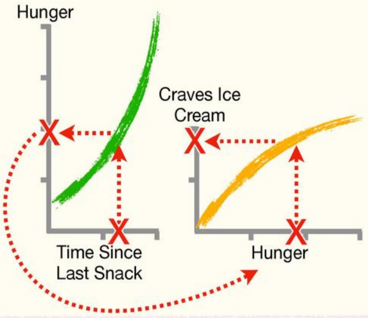
Craves Ice Cream = $(\text{Time}^2 + 0.5)^{1/2}$



…raising the sum by **1/2** makes it hard to find the derivative with **The Power Rule**.

Appendix F: The Chain Rule, A More Complicated Example

5 However, because Hunger links Time Since Last Snack to Craves Ice Cream, we can solve for the derivative using **The Chain Rule**!!!



6 **The Chain Rule** tells us that the derivative of Craves Ice Cream with respect to Time…

$$\frac{d\ \text{Craves}}{d\ \text{Time}} = \frac{d\ \text{Craves}}{d\ \text{Hunger}} \times \frac{d\ \text{Hunger}}{d\ \text{Time}}$$

…is the derivative of Craves Ice Cream with respect to Hunger…

…multiplied by the derivative of Hunger with respect to Time.

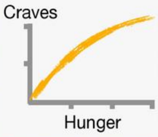
7 First, **The Power Rule** tells us that the derivative of Hunger with respect to Time is this equation:

$$\text{Hunger} = \text{Time}^2 + 0.5$$

$$\boxed{\frac{d\ \text{Hunger}}{d\ \text{Time}} = 2 \times \text{Time}}$$

8 Likewise, **The Power Rule** tells us that the derivative of Craves Ice Cream with respect to Hunger is this equation:

$$\text{Craves Ice Cream} = (\text{Hunger})^{1/2}$$



$$\boxed{\frac{d\ \text{Craves}}{d\ \text{Hunger}} = 1/2 \times \text{Hunger}^{-1/2} = \frac{1}{2 \times \text{Hunger}^{1/2}}}$$

9 Now we just plug the derivatives into **The Chain Rule**…

$$\frac{d\ \text{Craves}}{d\ \text{Time}} = \frac{d\ \text{Craves}}{d\ \text{Hunger}} \times \frac{d\ \text{Hunger}}{d\ \text{Time}}$$

$$= \frac{1}{2 \times \text{Hunger}^{1/2}} \times (2 \times \text{Time})$$

$$= \frac{2 \times \text{Time}}{2 \times \text{Hunger}^{1/2}}$$

$$\boxed{\frac{d\ \text{Craves}}{d\ \text{Time}} = \frac{\text{Time}}{\text{Hunger}^{1/2}}}$$

…and we see that when there's a change in Time Since Last Snack, the change in Craves Ice Cream is equal to Time divided by the square root of Hunger.

10

NOTE: In this example, it was obvious that Hunger was the link between Time Since Last Snack and Craves Ice Cream, which made it easy to apply **The Chain Rule**.

However, usually we get both equations jammed together like this…

$$\text{Craves Ice Cream} = (\text{Time}^2 + 0.5)^{1/2}$$

…and it's not so obvious how **The Chain Rule** applies. So, we'll talk about how to deal with this next!!!

BAM!!!

Appendix F: The Chain Rule, When The Link Is Not Obvious

① In the last part, we said that raising the sum by **1/2** makes it difficult to apply **The Power Rule** to this equation…

$$\text{Craves Ice Cream} = (\text{Time}^2 + 0.5)^{1/2}$$

…but there was an obvious way to link Time to Craves with Hunger, so we determined the derivative with **The Chain Rule**.

However, even when there's no obvious way to link equations, *we can create a link* so that we can still apply **The Chain Rule**.

② First, let's create a link between Time and Craves Ice Cream called **Inside**, which is equal to the stuff inside the parentheses…

$$\text{Inside} = \text{Time}^2 + 0.5$$

…and that means Craves Ice Cream can be rewritten as the square root of the stuff **Inside**.

$$\text{Craves Ice Cream} = (\text{Inside})^{1/2}$$

③ Now that we've created **Inside**, the link between Time and Craves, we can apply **The Chain Rule** to solve for the derivative.

The Chain Rule tells us that the derivative of Craves with respect to Time…

$$\frac{d\ \text{Craves}}{d\ \text{Time}} = \frac{d\ \text{Craves}}{d\ \text{Inside}} \times \frac{d\ \text{Inside}}{d\ \text{Time}}$$

…is the derivative of Craves with respect to **Inside**…

…multiplied by the derivative of **Inside** with respect to Time.

④ Now we use **The Power Rule** to solve for the two derivatives.

$$\frac{d\ \text{Craves}}{d\ \text{Inside}} = \frac{d}{d\ \text{Inside}}(\text{Inside})^{1/2} = 1/2 \times \text{Inside}^{-1/2}$$

$$= \frac{1}{2 \times \text{Inside}^{1/2}}$$

$$\frac{d\ \text{Inside}}{d\ \text{Time}} = \frac{d}{d\ \text{Time}}\text{Time}^2 + 0.5 = 2 \times \text{Time}$$

⑤ Lastly, we plug them into **The Chain Rule**…

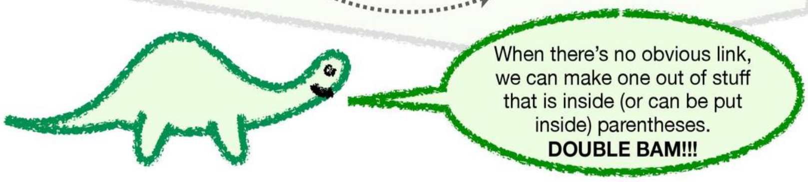
$$\frac{d\ \text{Craves}}{d\ \text{Time}} = \frac{d\ \text{Craves}}{d\ \text{Inside}} \times \frac{d\ \text{Inside}}{d\ \text{Time}}$$

$$\frac{d\ \text{Craves}}{d\ \text{Time}} = \frac{1}{2 \times \text{Inside}^{1/2}} \times (2 \times \text{Time})$$

$$= \frac{2 \times \text{Time}}{2 \times \text{Hunger}^{1/2}}$$

$$\boxed{\frac{d\ \text{Craves}}{d\ \text{Time}} = \frac{\text{Time}}{\text{Hunger}^{1/2}}}$$

…and just like when the link, Hunger, was obvious, when we created a link, **Inside**, we got the exact same result. **BAM!!!**



Acknowledgments

Acknowledgments

The idea for this book came from comments on my YouTube channel. I'll admit, when I first saw that people wanted a **StatQuest** book, I didn't think it would be possible because I didn't know how to explain things in writing the way I explained things with pictures. But once I created the **StatQuest** study guides, I realized that instead of *writing* a book, I could *draw* a book. And knowing that I could draw a book, I started to work on this one.

This book would not have been possible without the help of many, many people.

First, I'd like to thank all of the **Triple BAM** supporters on Patreon and YouTube: U-A Castle, J. Le, A. Izaki, Gabriel Robet, A. Doss, J. Gaynes, Adila, A. Takeh, J. Butt, M. Scola, Q95, Aluminum, S. Pancham, A. Cabrera, and N. Thomson.

I'd also like to thank my copy editor, **Wendy Spitzer**. She worked magic on this book by correcting a million errors, giving me invaluable feedback on readability, and making sure each concept was clearly explained. Also, I'd like to thank **Adila**, **Gabriel Robet**, **Mahmud Hasan**, **PhD**, **Ruizhe Ma**, **Samuel Judge** and **Sean Moran** who helped me with everything from the layout to making sure the math was done properly.

Lastly, I'd like to thank Will Falcon and the whole team at Lightning AI.

Index

Index