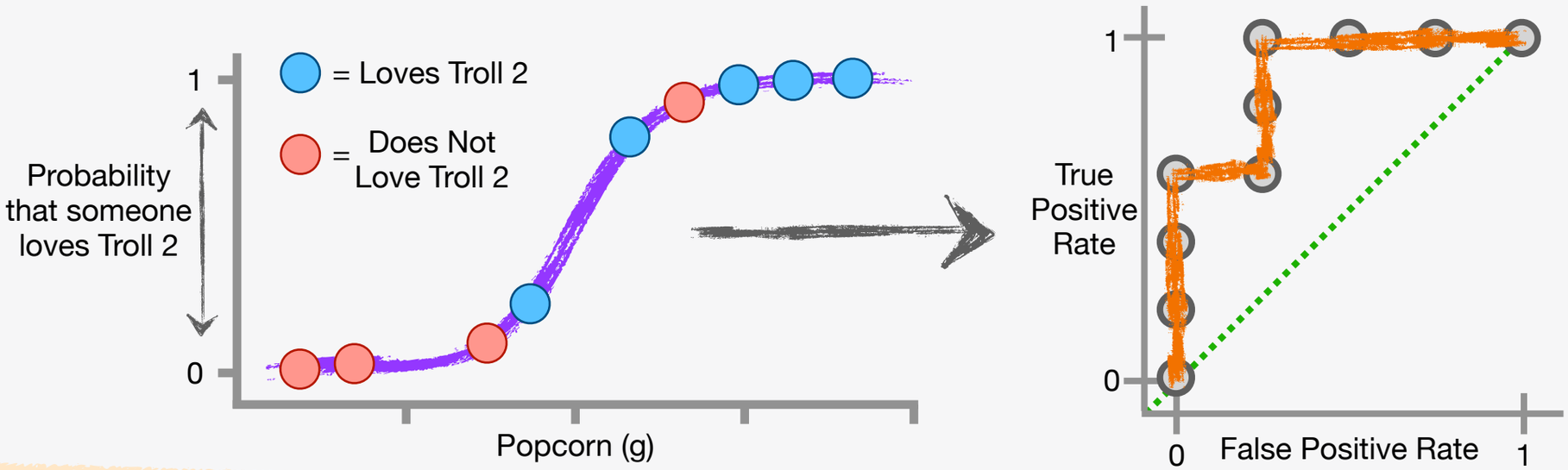
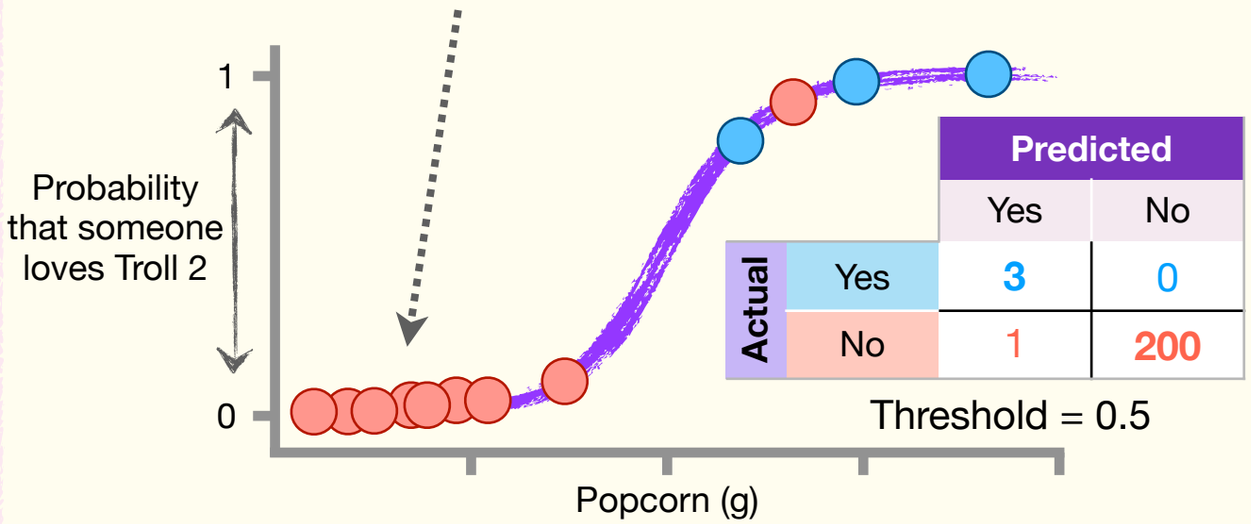
Precision Recall Graphs: Main Ideas Part 1

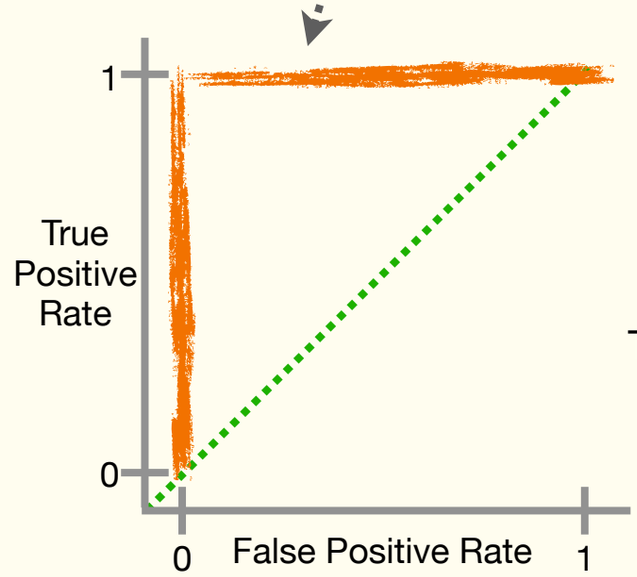
1 An **ROC** graph uses the **False Positive Rate** on the x-axis, and this is fine when the data are *balanced*, which in this case means there are similar numbers of people who **Love Troll 2** and who **Do Not Love Troll 2**.



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)…



…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**.



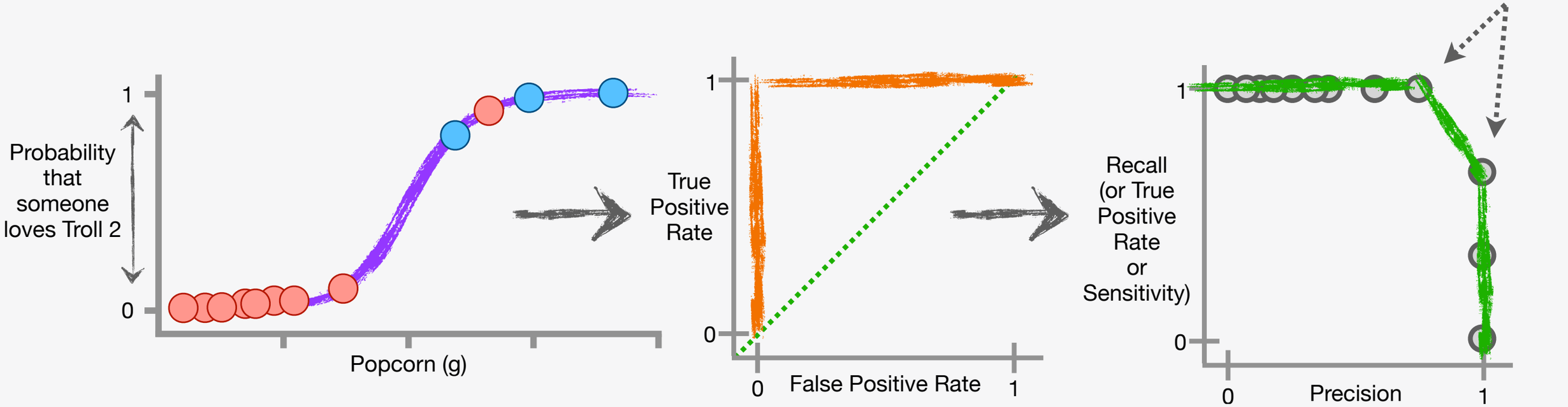
In other words, the **ROC** graph makes any model that simply predicts **No** **100%** of the time look really good.

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**.

Gentle Reminder:

True Positive

False Negative

Actual	Predicted: Yes	Predicted: No
Yes	**TP**	FN
No	FP	**TN**

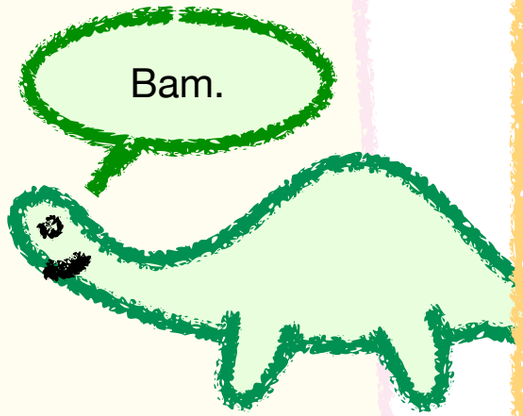
False Positive

True Negative

$$\textbf{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

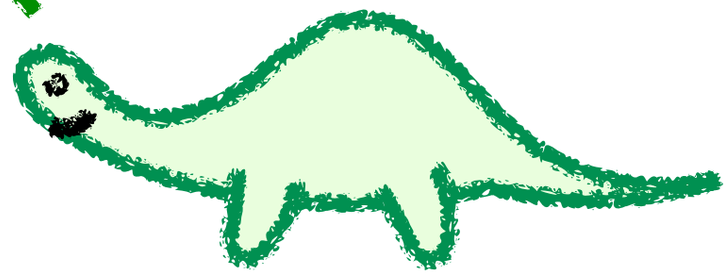
Actual	Predicted: Yes	Predicted: No
Yes	3	0
No	1	200

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!!!

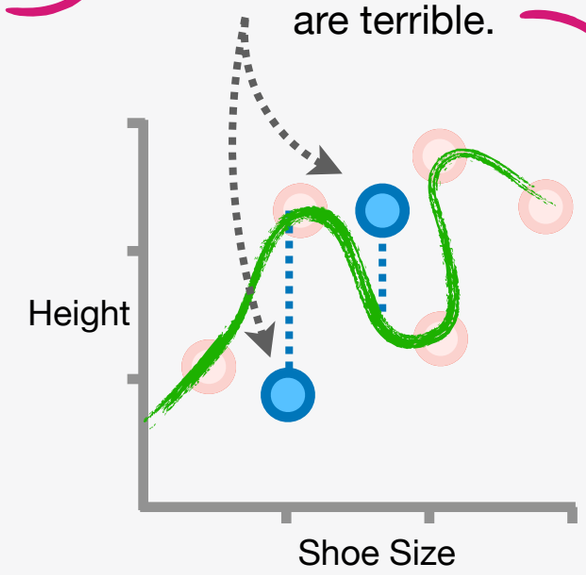
Regularization: Main Ideas

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.

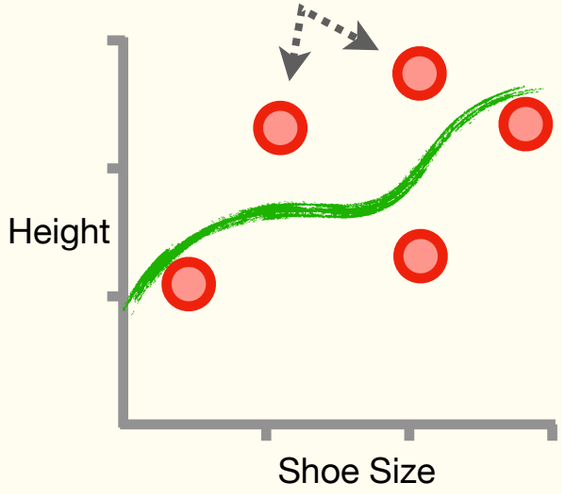


Using technical jargon, we would say that the **squiggle** has *low* **Bias** because it fits the **Training Data** well, but *high* **Variance** because it does a bad job with **New Data**.

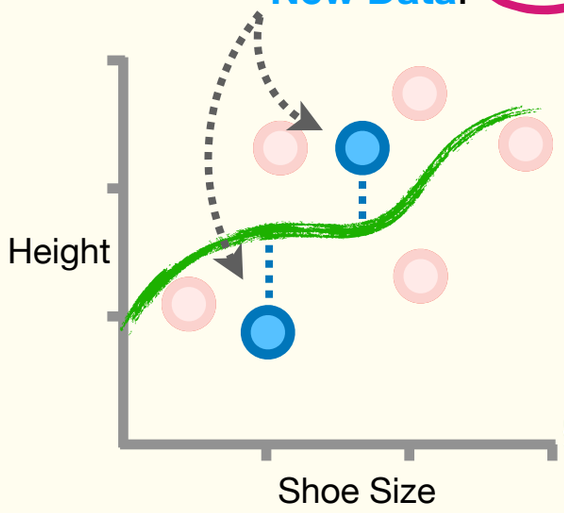
2

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**.



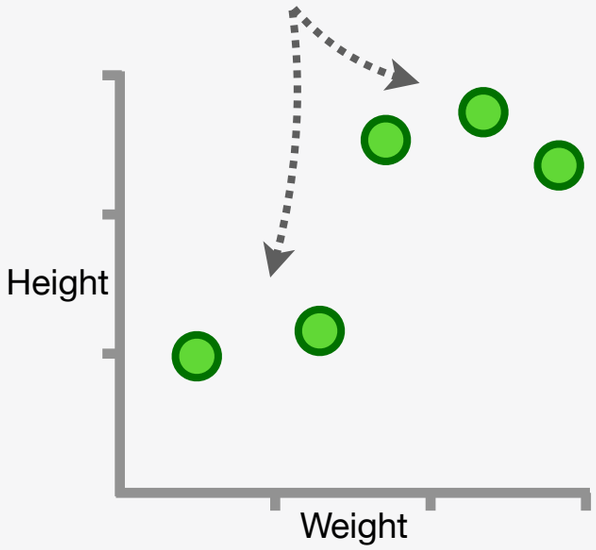
Using technical jargon, we would say that **Regularization** *increases* **Bias** a little bit, but in return, we get a big *decrease* in **Variance**.

BAM!!!

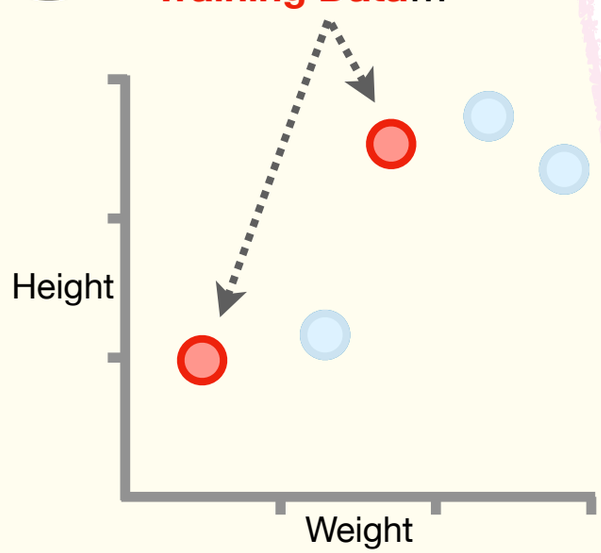
NOTE: In this chapter, we'll learn about the two main types of **Regularization**, **Ridge** and **Lasso**, in the context of **Linear Regression**, but they can be used with almost any machine learning algorithm to improve performance.

Ridge/Squared/L2 Regularization: Details Part 1

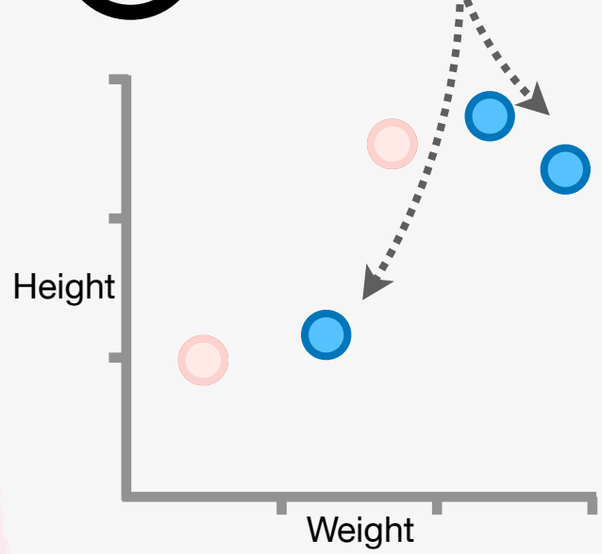
1 Let's imagine we measured the Height and Weight of **5** different people…



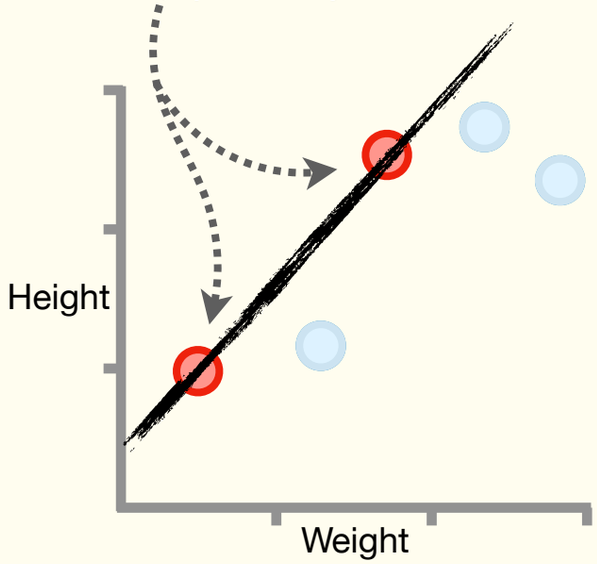
2 …and then we split those data into **Training Data**…



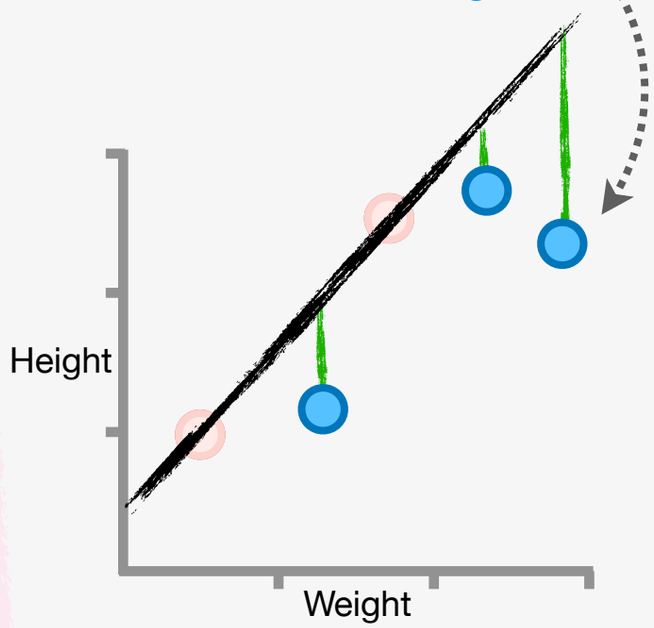
3 …and **Testing Data**.



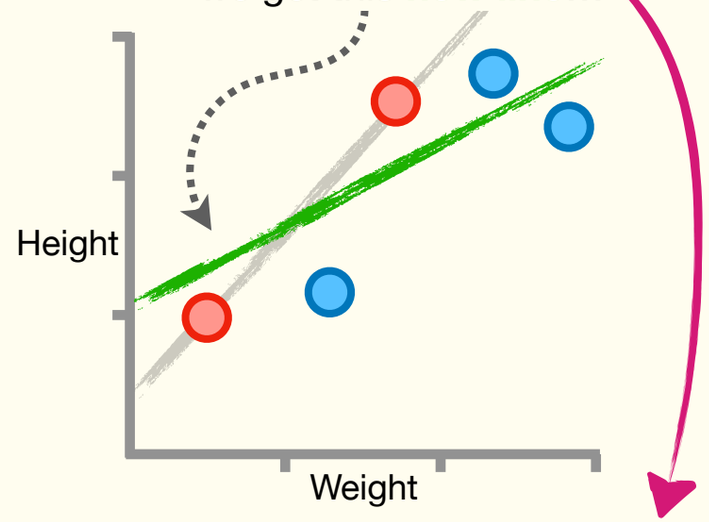
4 Then we fit a **line** to the **Training Data** that minimized the **Sum of the Squared Residuals** (**SSR**). Because we only have **2** points in the **Training Data**, the line fits it perfectly, and the **SSR = 0**…



5 …however, because the slope of the **line** is so steep, it does a bad job with the **Testing Data**.



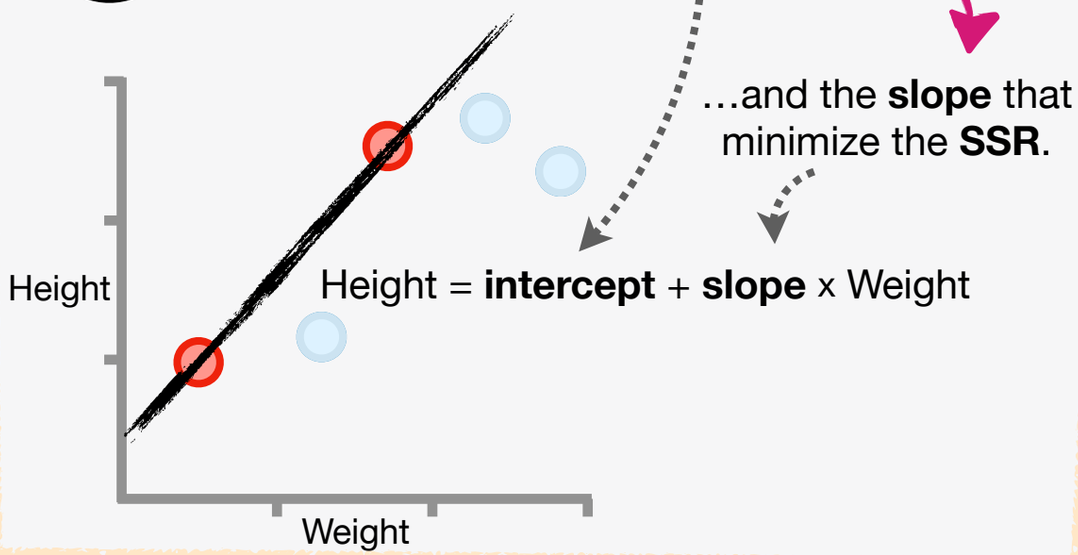
6 In contrast, after applying **Ridge Regularization**, also called **Squared** or **L2 Regularization**, we get this **new line**…



…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.

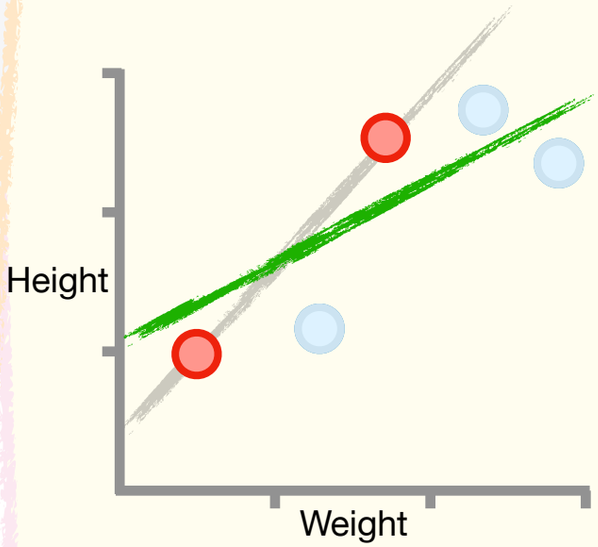
Ridge/Squared/L2 Regularization: Details Part 2

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**.

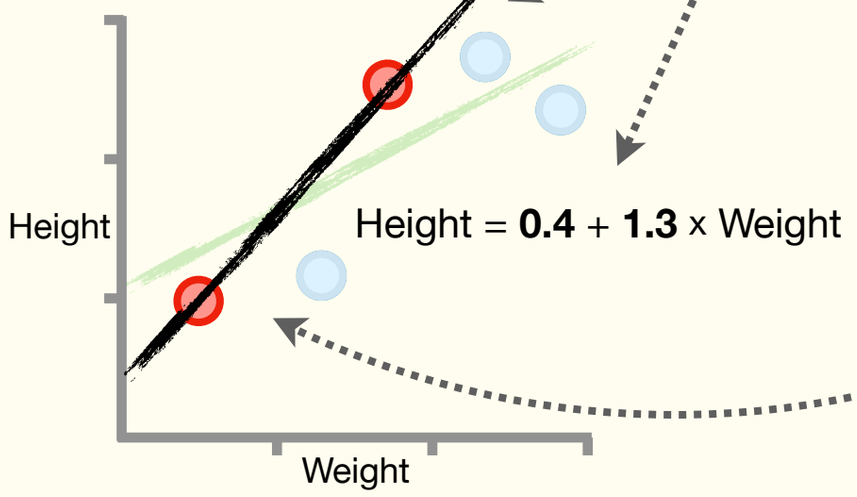
8 In contrast, when we use **Ridge Regularization** to optimize parameters, we simultaneously minimize the **SSR** and a penalty that's proportional to the square of the slope…



SSR + ***λ*** x 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…



10 Now, if we're using **Ridge Regularization**, we want to minimize this equation.

SSR + ***λ*** x slope^2 = 0 + 1 x 1.3^2 = 1.69

…and, we end up with **1.69** as the score for the **line** that fits the **Training Data** perfectly.

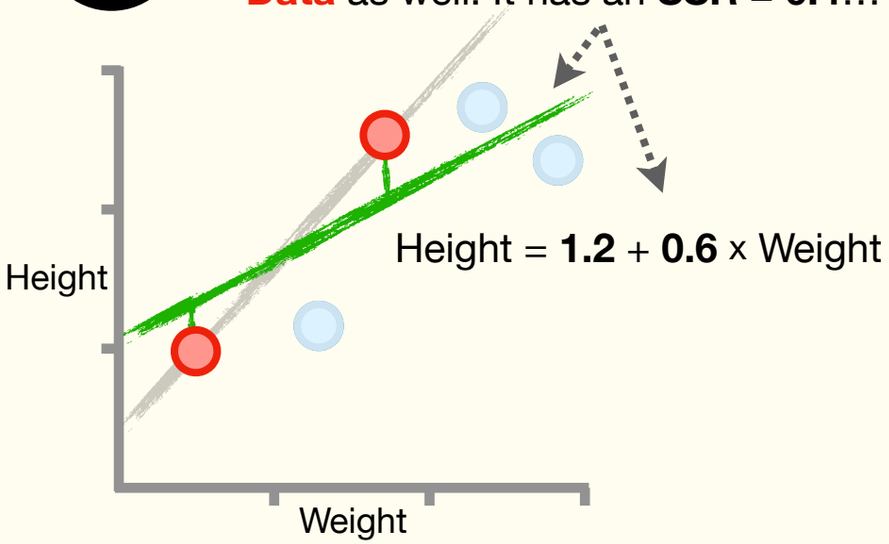
Because the **line** fits the **Training Data** perfectly, the **SSR = 0**…

…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…

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**…

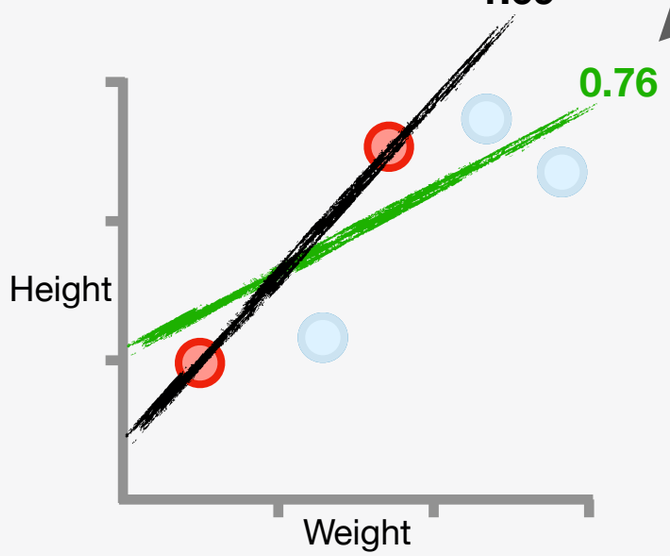


12 …so we plug in the **SSR**, **0.4**, the value for ***λ*** (***lambda***), which for now is **1**, and the slope, **0.6**, into the **Ridge Penalty**.

SSR + ***λ*** x slope^2 = 0.4 + 1 x 0.6^2 = 0.76

And when we do the math, we get **0.76**.

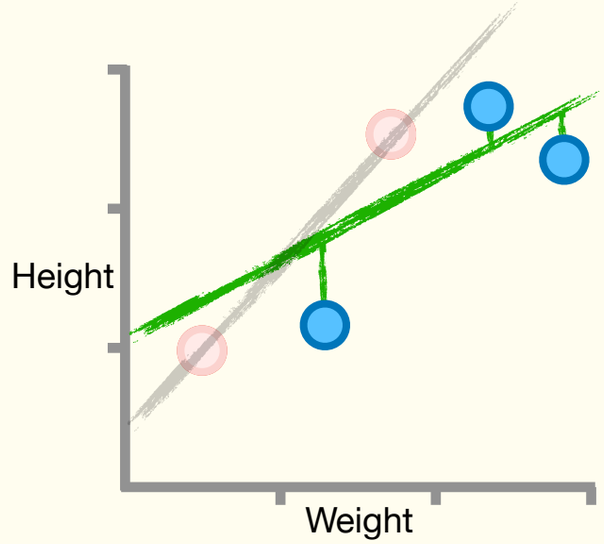
13 Thus, the **Ridge Score** for the **line** that fit the **Training Data** perfectly is **1.69**…



…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***).

SSR + ***λ*** x slope^2

Ridge/Squared/L2 Regularization: Details Part 4

15 When $\lambda = 0$…

SSR + λ x slope^2

= **SSR** + **0** x slope^2

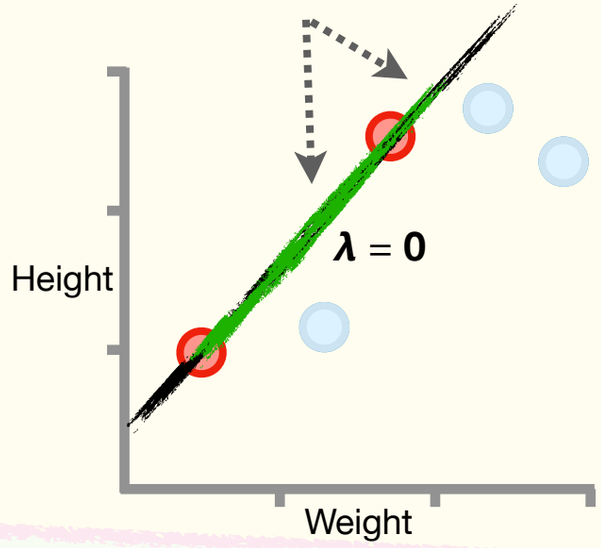
= **SSR** + **0**

= **SSR**

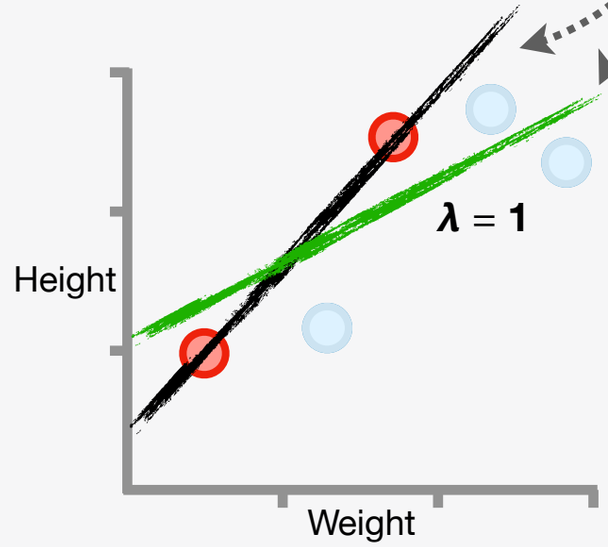
…then the whole **Ridge Penalty** is also **0**…

…and that means we'll only minimize the **SSR**, so it's as if we're *not using* **Ridge Regularization**…

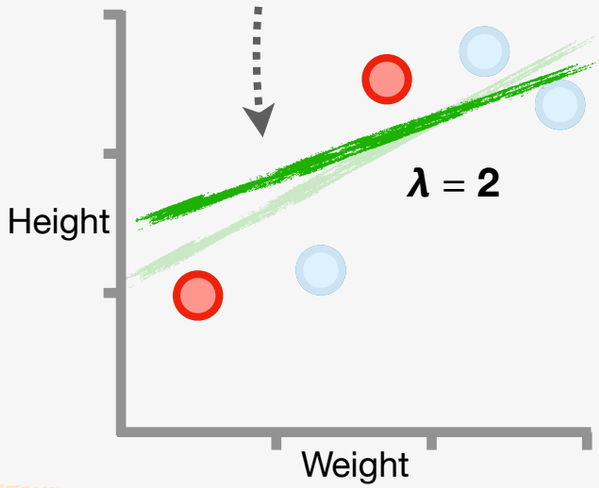
…and, as a result, the **new line**, derived from **Ridge Regularization**, is no different from a line that minimizes the **SSR**.



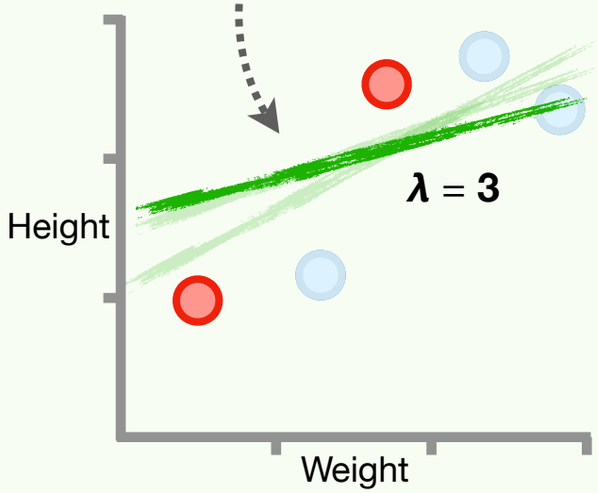
16 However, as we just saw, when $\lambda = 1$, we get a **new line** that has a smaller slope than when we only minimized the **SSR**.



17 When we increase λ to **2**, the slope gets even smaller…

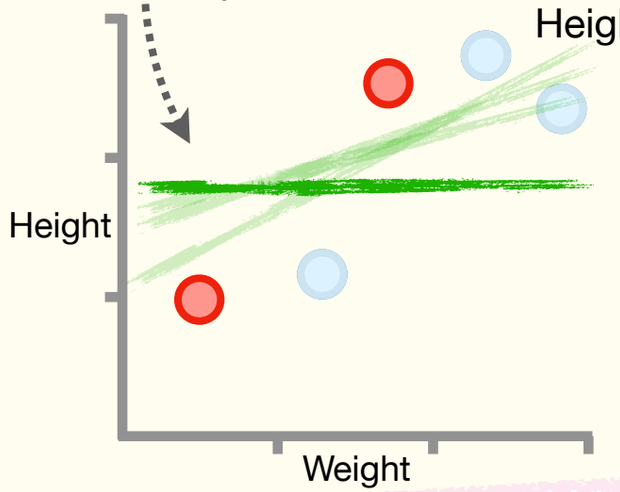


18 …and when we increase λ to **3**, the slope gets even smaller…



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.

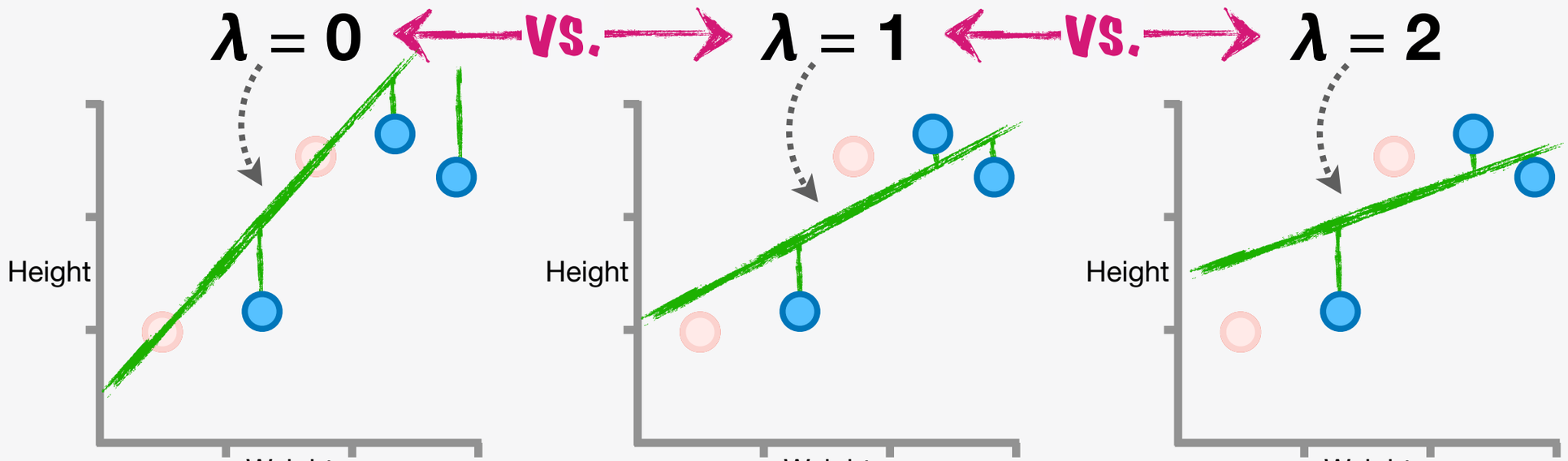
So, how do we pick a good value for λ?



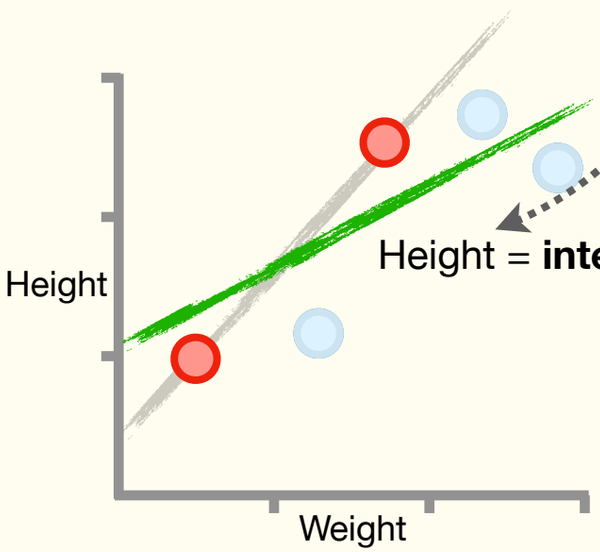
Ridge/Squared/L2 Regularization: Details Part 5

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…



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

…the **Ridge Penalty** contained only a single parameter, the slope.

SSR + λ x slope^2

However, if we had a more complicated model that used Weight, Shoe Size, and Age to predict Height…

Height = **intercept** + **slope$_w$** x Weight + **slope$_s$** x Shoe Size + **slope$_a$** x Age

…then the **Ridge Penalty** would include the sum of the squares of the **3** slopes associated with those variables.

SSR + λ x (slope$_w^2$ + slope$_s^2$ + slope$_a^2$)

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, **slope$_w$**, **slope$_s$**, and **slope$_a$**, but not equally.

Height = **intercept** + **slope$_w$** x Weight + **slope$_s$** x Shoe Size + **slope$_a$** x 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, **slope$_w$** and **slope$_s$**, will shrink a little bit…

…compared to the slope associated with the Airspeed of a Swallow, **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, **slope$_a$**, a lot will shrink the **Ridge Penalty** a lot…

…without increasing the **SSR**.

SSR + λ x (slope$_w^2$ + slope$_s^2$ + 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**…

$$\frac{d}{d\ intercept}\left(\text{SSR} + \lambda \times \text{slope}^2\right)$$

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

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

$$\frac{d}{d\ slope}\left(\text{SSR} + \lambda \times \text{slope}^2\right)$$

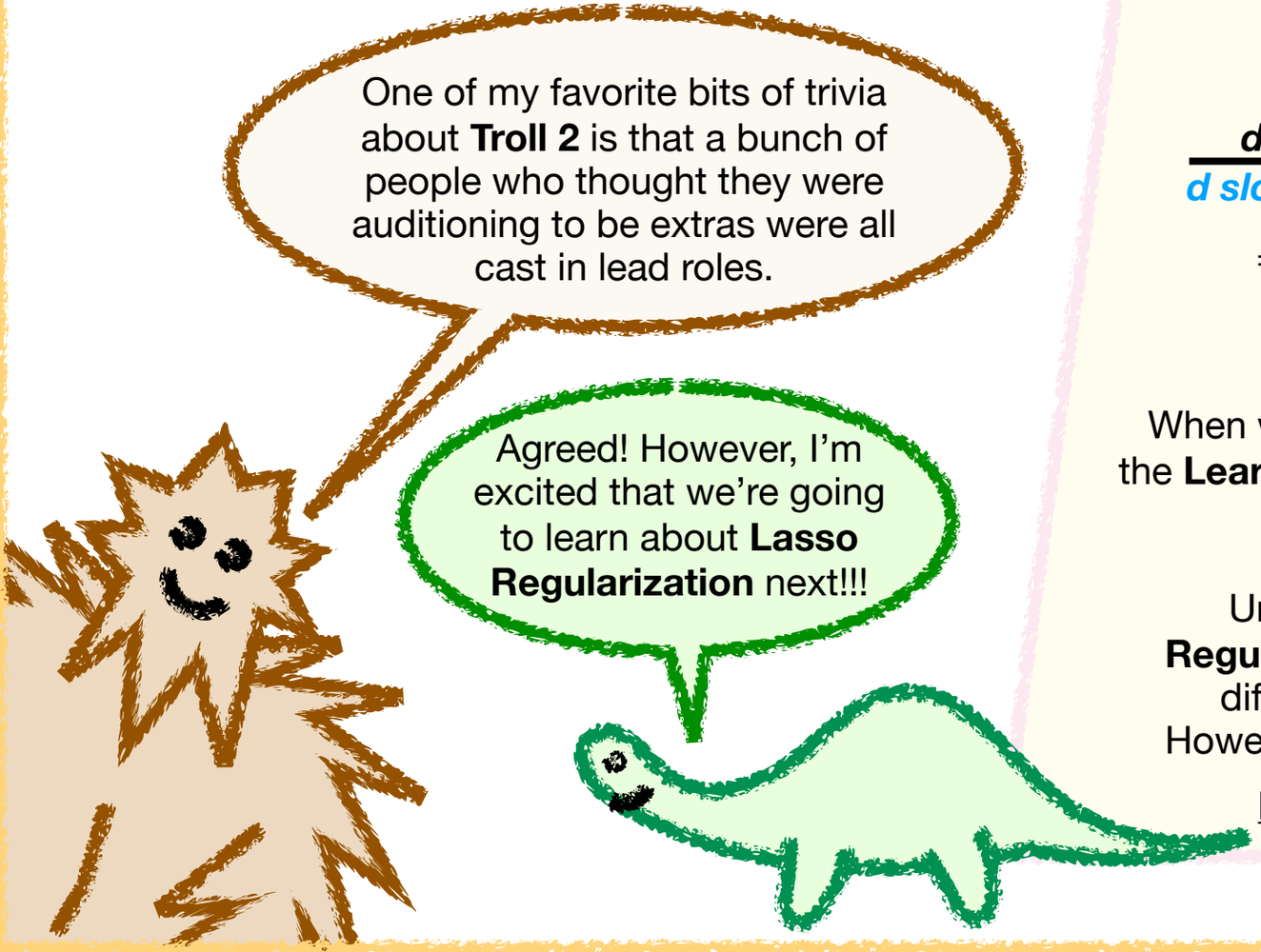
$$= -2 \times \text{Weight}\left(\text{Height} - (\text{intercept} + \text{slope} \times \text{Weight})\right)$$

$$+ 2 \times \lambda \times \text{slope}$$

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.

SSR + λ x slope^2

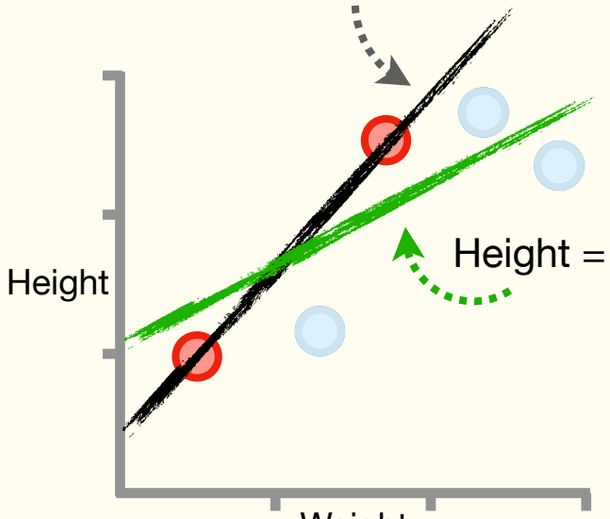
VS.

In contrast, in the **Lasso Penalty**, we take the parameter's absolute value.

SSR + λ x |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



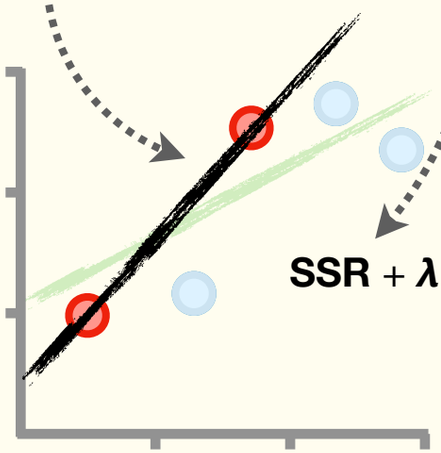
…and for the **green line**, which doesn't fit the **Training Data** as well.

Height = **1.2** + **0.6** x Weight

3 For the **black line**, which fits the **Training Data** perfectly, the **SSR** is **0**…

…for now we'll let λ **= 1**…

Height = **0.4** + **1.3** x Weight



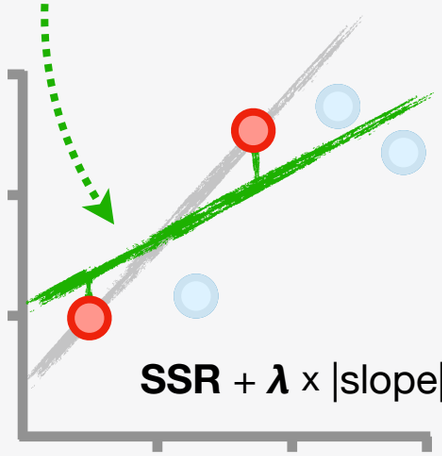
…and the absolute value of the slope is **1.3**…

SSR + λ x |slope| = 0 + 1 x 1.3 = 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 λ, 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.

SSR + λ x |slope| = 0.4 + 1 x 0.6 = 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!!!

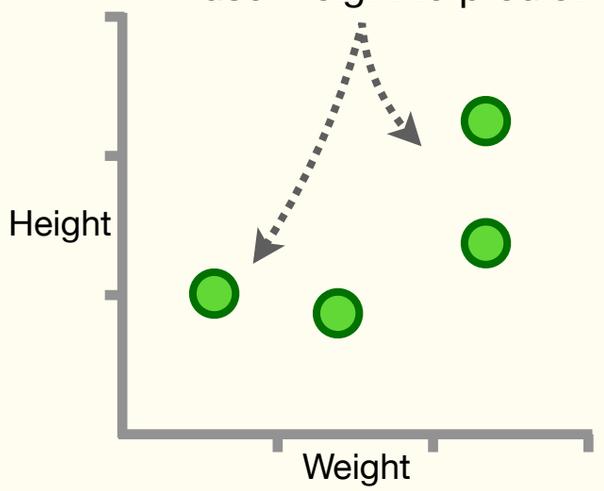
NOTE: Ridge and **Lasso Regularization** are frequently combined to get the best of both worlds.

DOUBLE BAM!!!

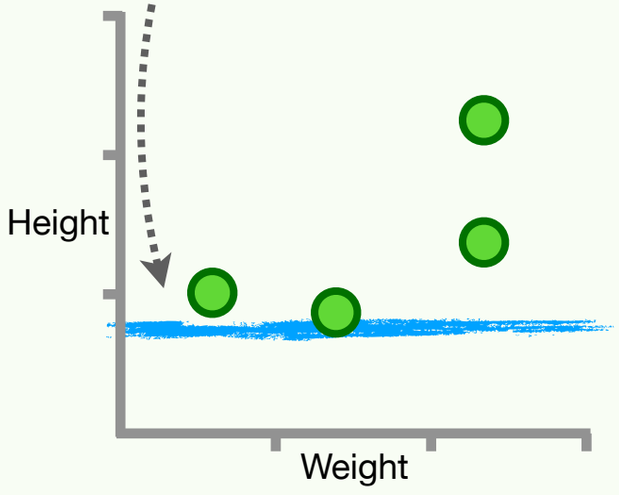
Ridge vs. Lasso Regularization: Details Part 1

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!!!

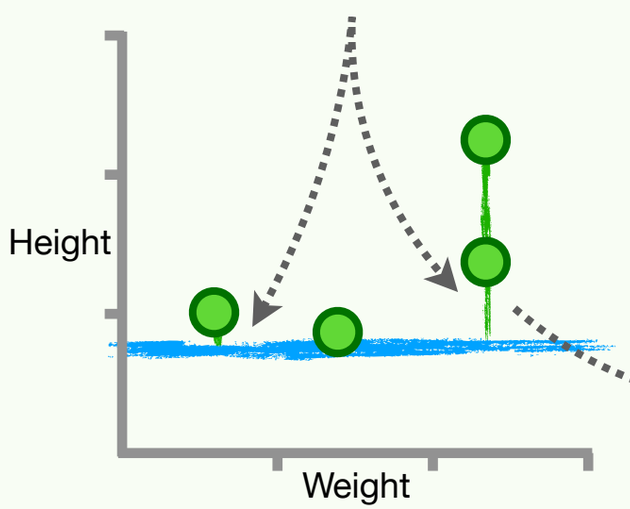
2 As always, we'll start with a super simple dataset where we want to use Weight to predict Height.



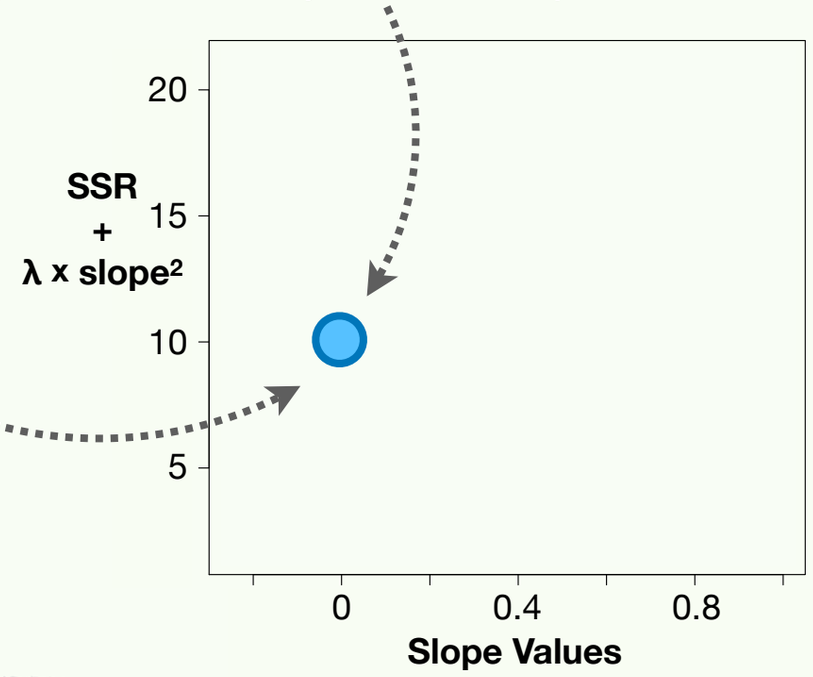
3 Now let's fit a **blue horizontal line** to the data, which is a terrible fit, but we'll improve it in a bit…



…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**…

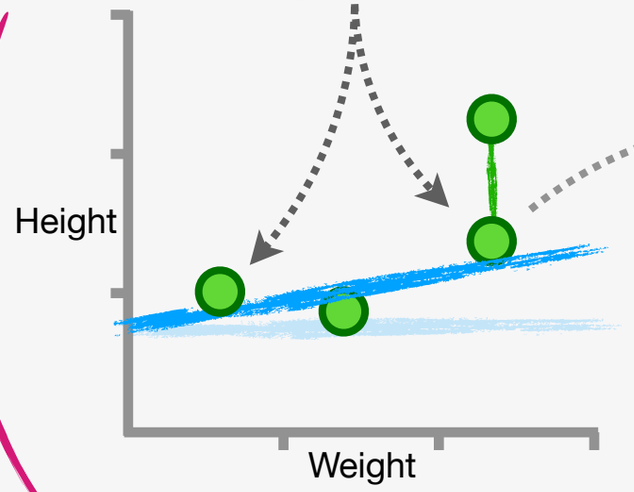


…and now let's plot the **Ridge Score** and the corresponding slope of the **blue horizontal line**, **0**, on a graph that has **SSR + λ x slope²** on the y-axis and slope on the x-axis.

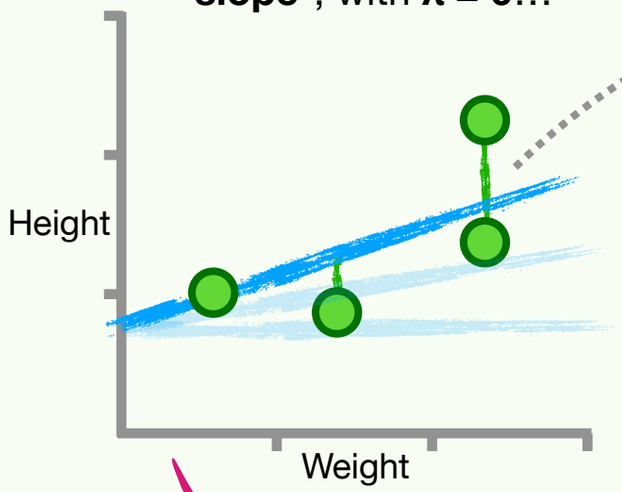


Ridge vs. Lasso Regularization: Details Part 2

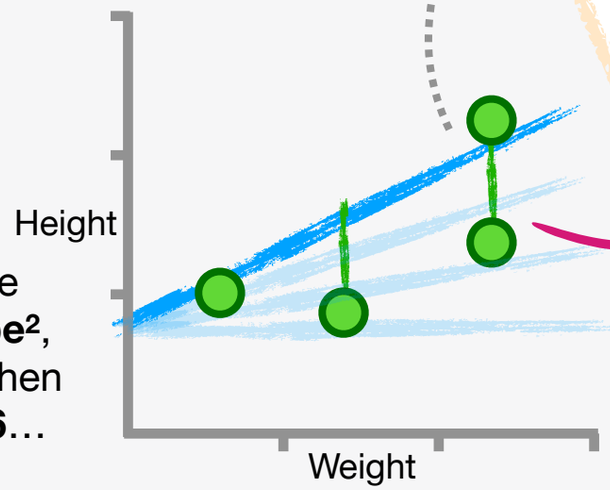
4 Now let's increase the slope to **0.2** and plot the **SSR + λ x slope²**, again with **λ = 0**…

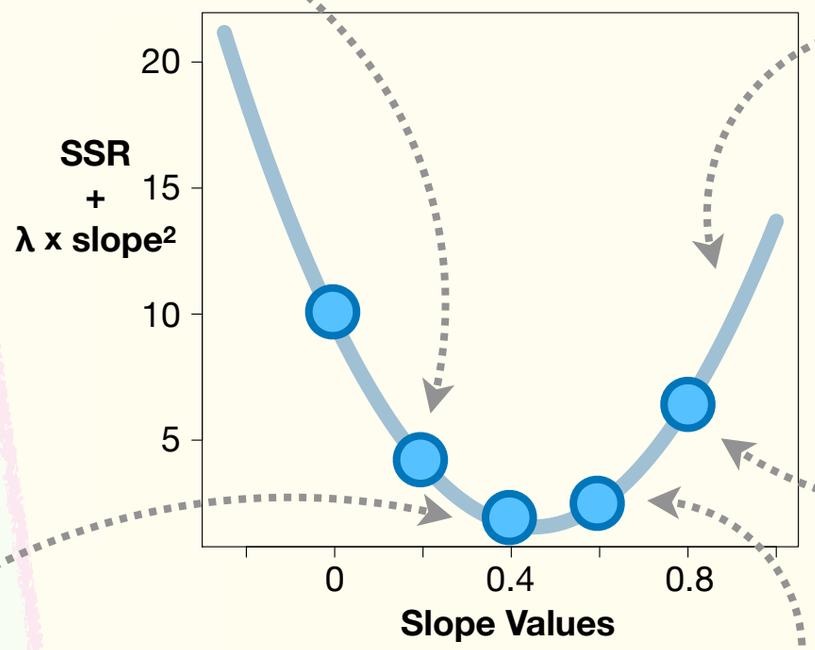


…and then increase the slope to **0.4** and plot the **SSR + λ x slope²**, with **λ = 0**…

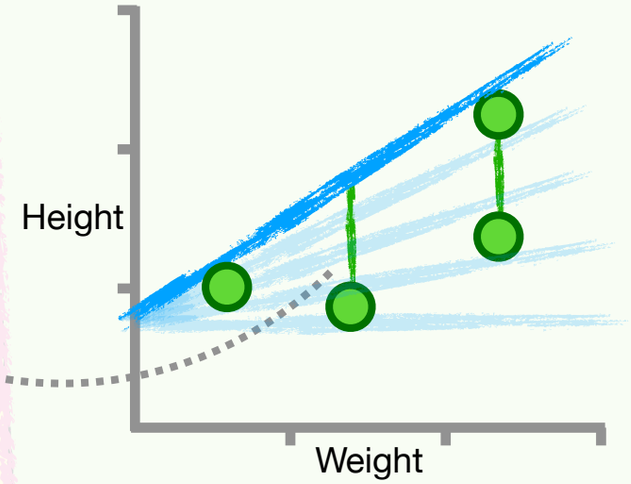


…and plot the **SSR + λ x slope²**, with **λ = 0** for when the slope is **0.6**…





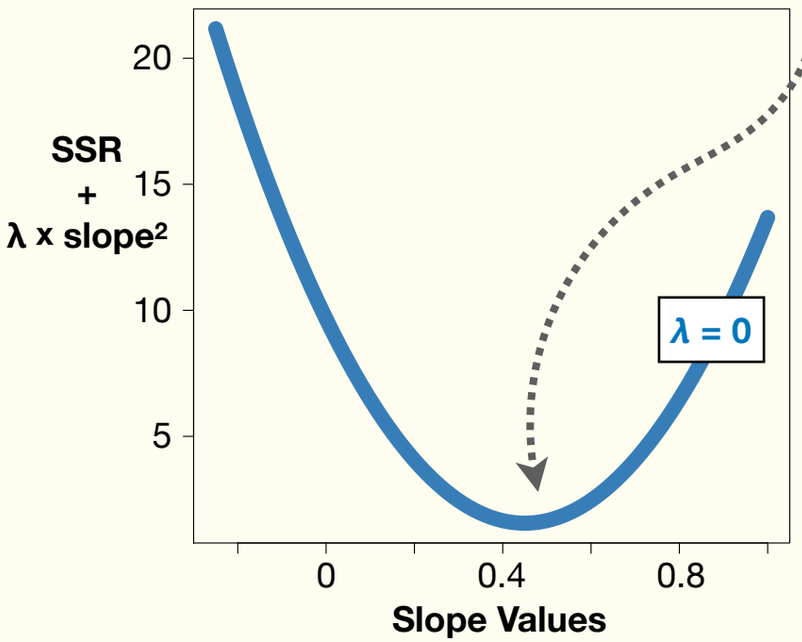
Ultimately, we can plot the **SSR + λ x slope²**, with **λ = 0**, as a function of the slope, and we get this **blue curve**.



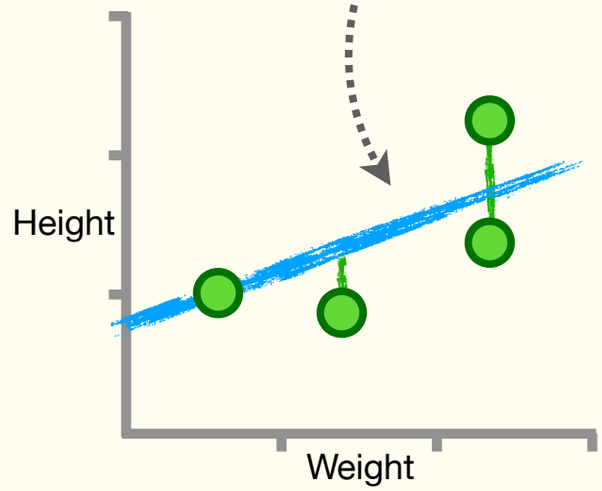
…and plot the **SSR + λ x slope²**, with **λ = 0** for when the slope is **0.8**.

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**…

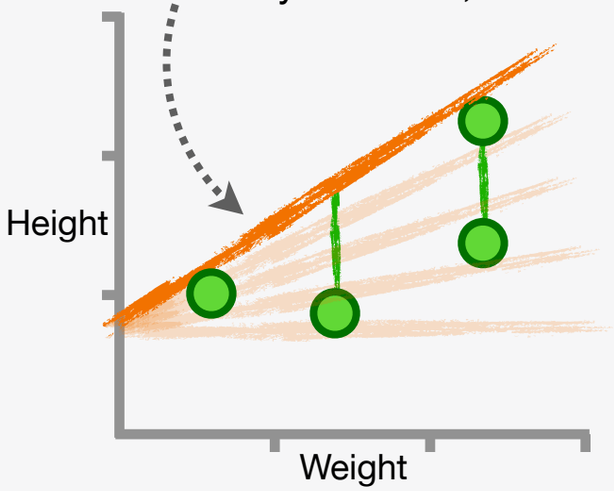


…which corresponds to this **blue line**.

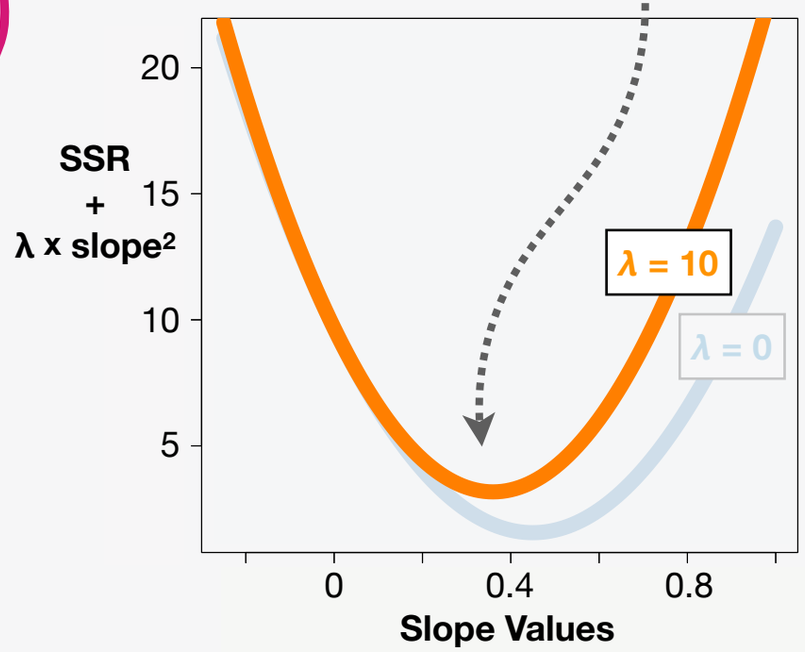


Ridge Regression makes me think of being at the top of a cold mountain. Brrr!! And **Lasso Regression** makes me think about cowboys. Giddy up!

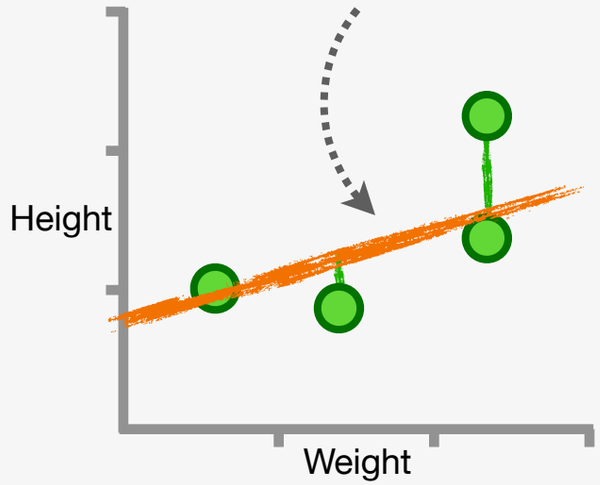
6 Now, just like before, let's calculate the **SSR + λ x slope²** for a bunch of different slopes, only this time, let **λ = 10**.



We get this **orange curve**, where we see that the slope that minimizes the **Ridge Score** is just under **0.4**…

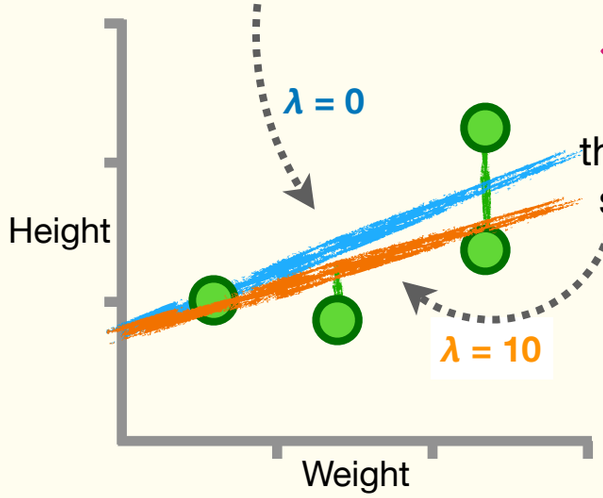


…which corresponds this **orange line**.



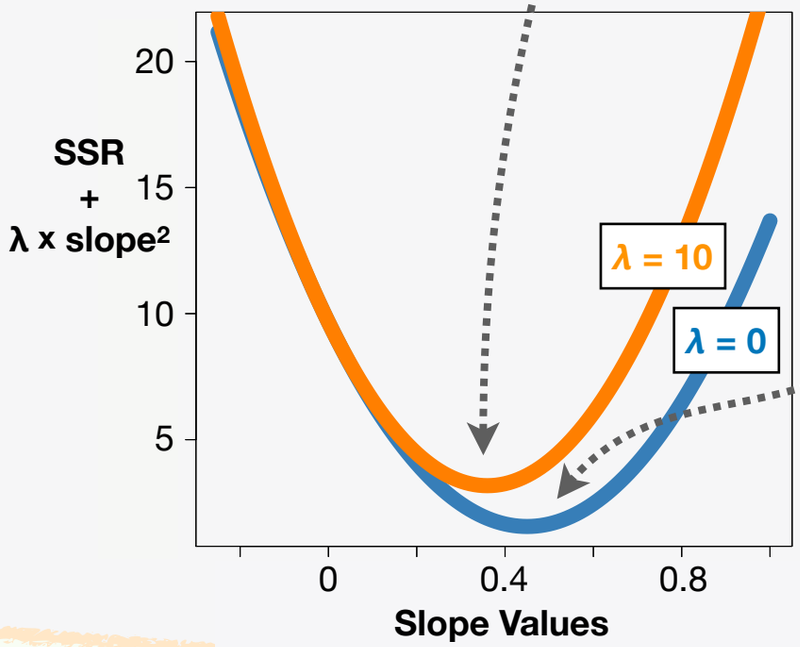
Ridge vs. Lasso Regularization: Details Part 4

7 When we compare the **blue line**, which has the optimal slope when $\lambda = 0$…



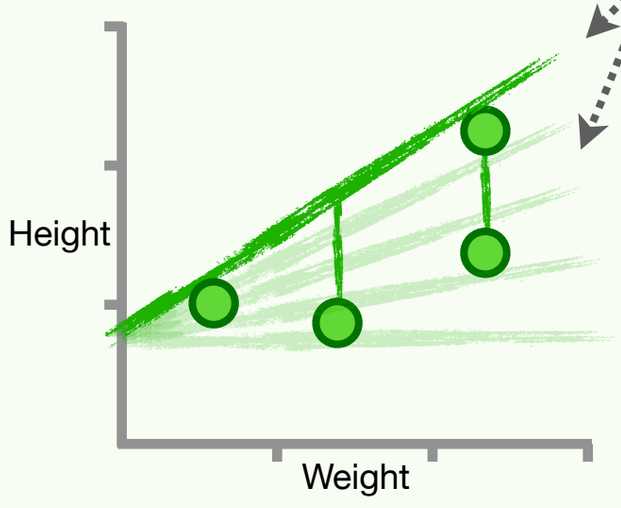
…to the **orange line**, which has the optimal slope when $\lambda = 10$, we see that $\lambda = 10$, which increases the **Ridge Penalty**, results in a smaller slope.

8 Likewise, we can see that when $\lambda = 10$, the lowest point on the **orange curve** corresponds to a slope value closer to **0**…

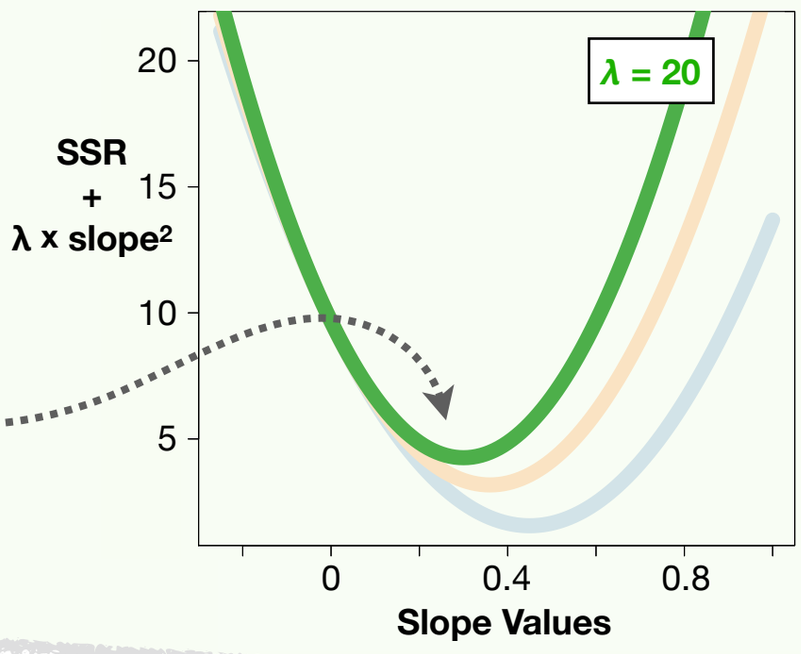


…than on the **blue curve**, when $\lambda = 0$.

9 Now we calculate the **SSR + λ x slope²** for a bunch of different slopes with $\lambda = 20$…

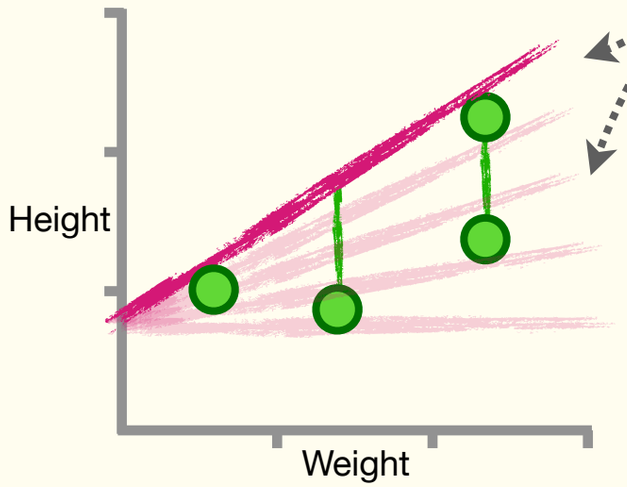


…and we get this **green curve**, and again, we see that the lowest point corresponds to a slope that's closer to **0**.

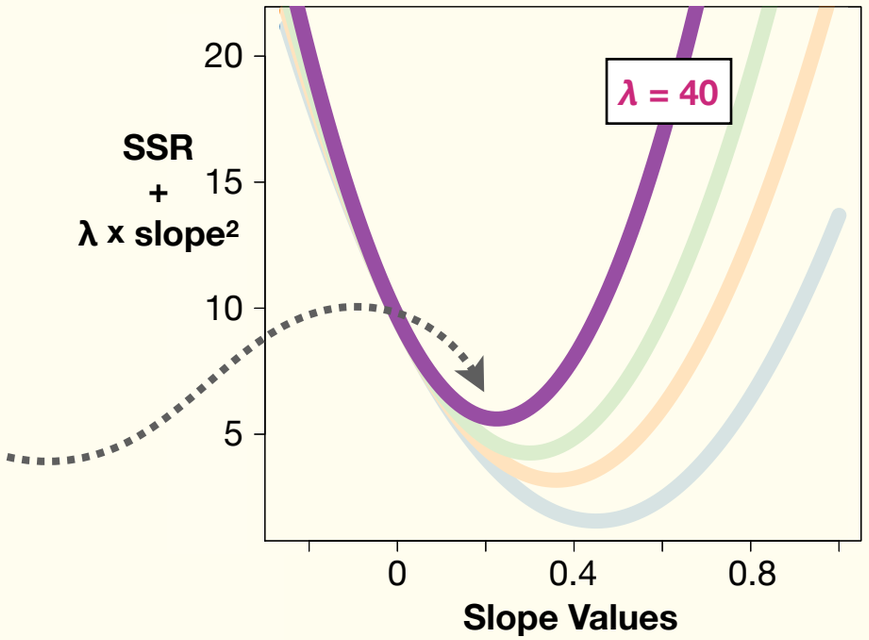


Ridge vs. Lasso Regularization: Details Part 5

10 Lastly, we calculate the **SSR + λ x slope²** 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:

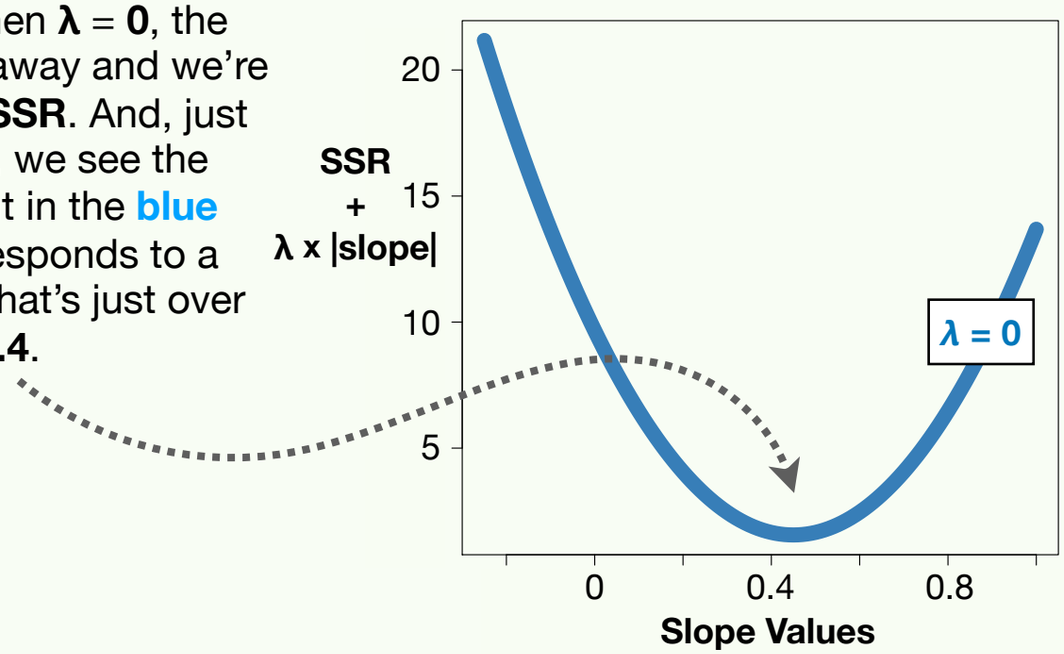
1) When we calculate **SSR + λ x slope²**, we get a nice curve for different values for the slope.

2) When we increase **λ**, the lowest point in the curve corresponds to a slope value closer to **0**, but not quite **0**.

BAM!!!

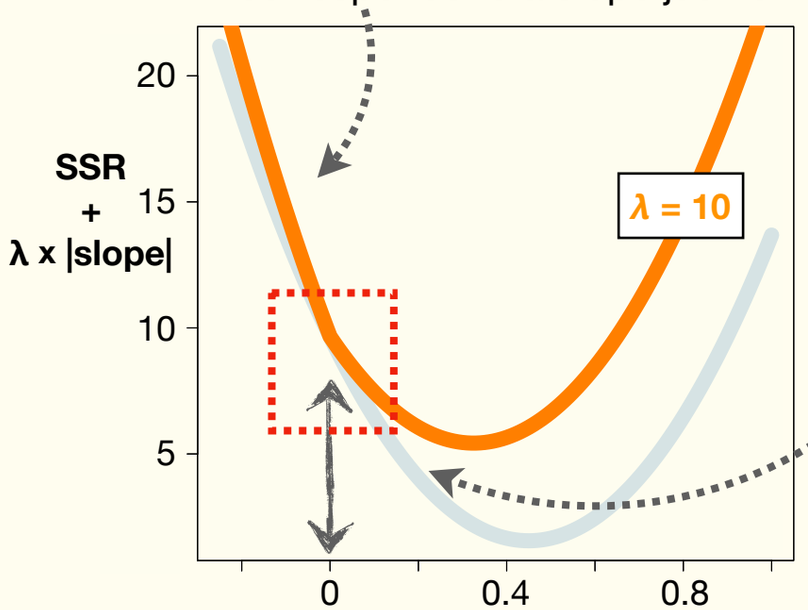
Now let's do the same thing using the **Lasso Penalty**, and calculate **SSR + λ x |slope|**!!!

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**.



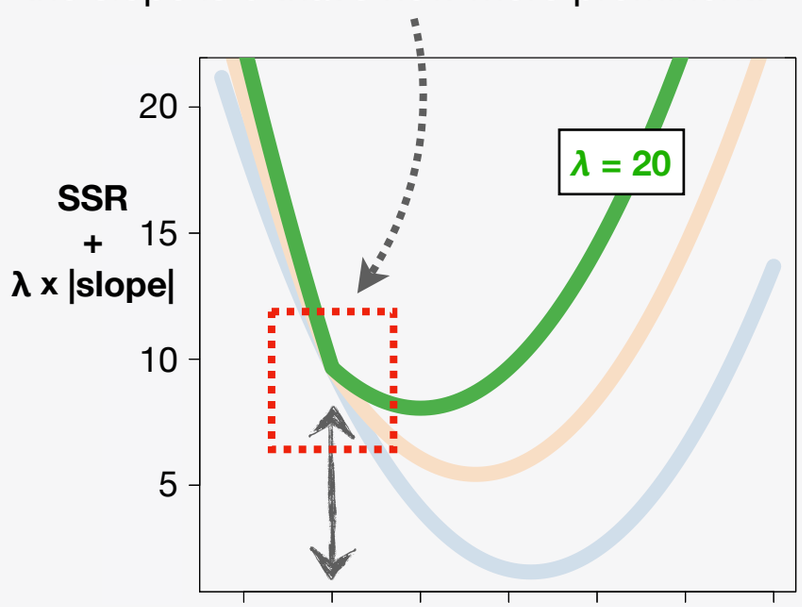
Ridge vs. Lasso Regularization: Details Part 6

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**.

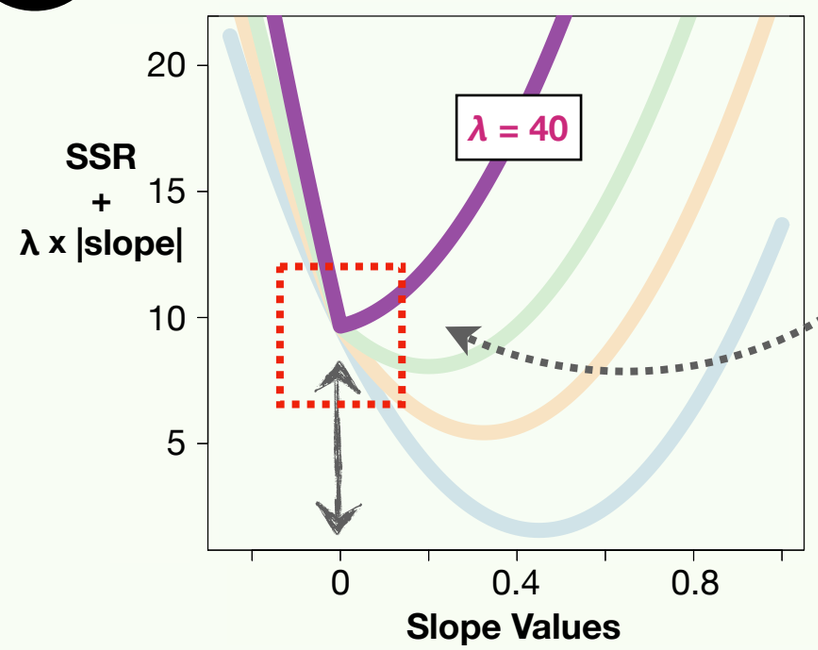


However, unlike before, this **orange shape** has a slight kink when the slope is **0**.

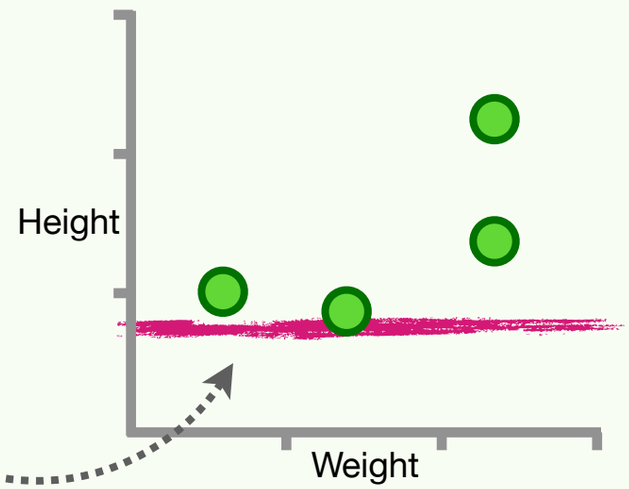
14 When we calculate the **SSR + λ x |slope|** and let **λ = 20** for a bunch of different values for the slope, we get this **green shape** with a kink where the slope is **0** that's now more prominent.



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…



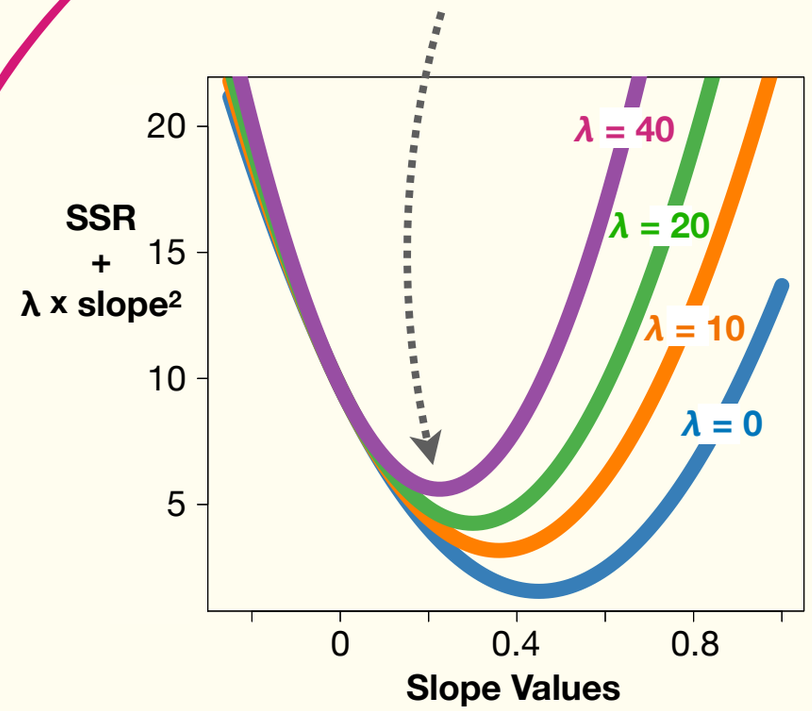
…and that means when **λ = 40**, the slope of the optimal line is **0**.



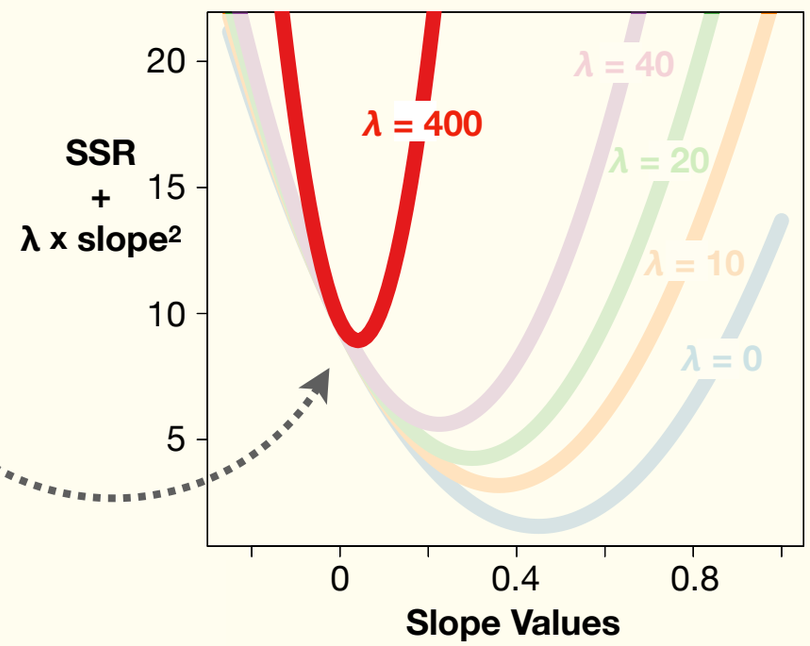
BAM!!!

Ridge vs. Lasso Regularization: Details Part 7

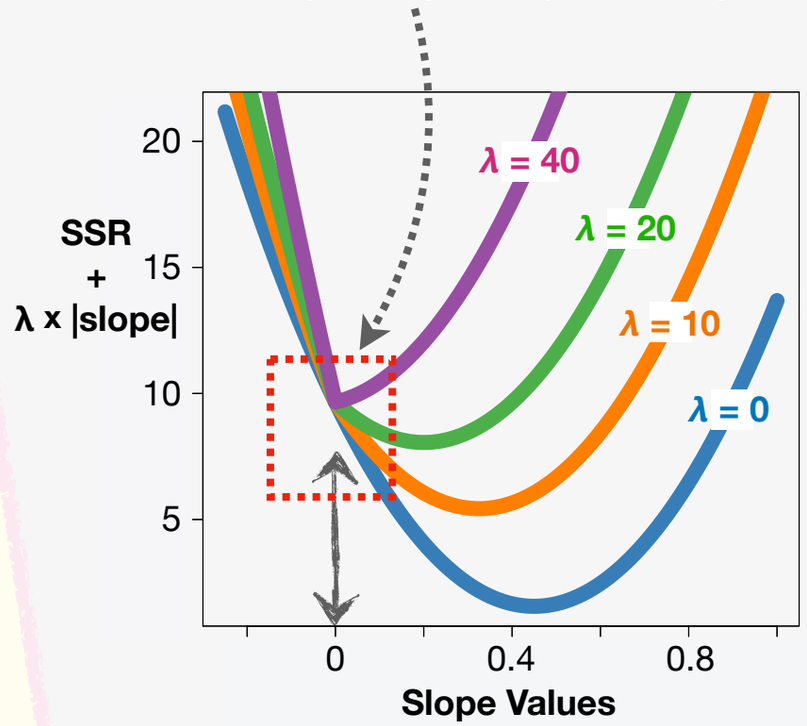
16 In summary, when we increase the **Ridge, Squared** the optimal value for the slope shifts toward **0**, but we retain a nice **parabola** shape, and the optimal slope is never **0** itself.



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!!!

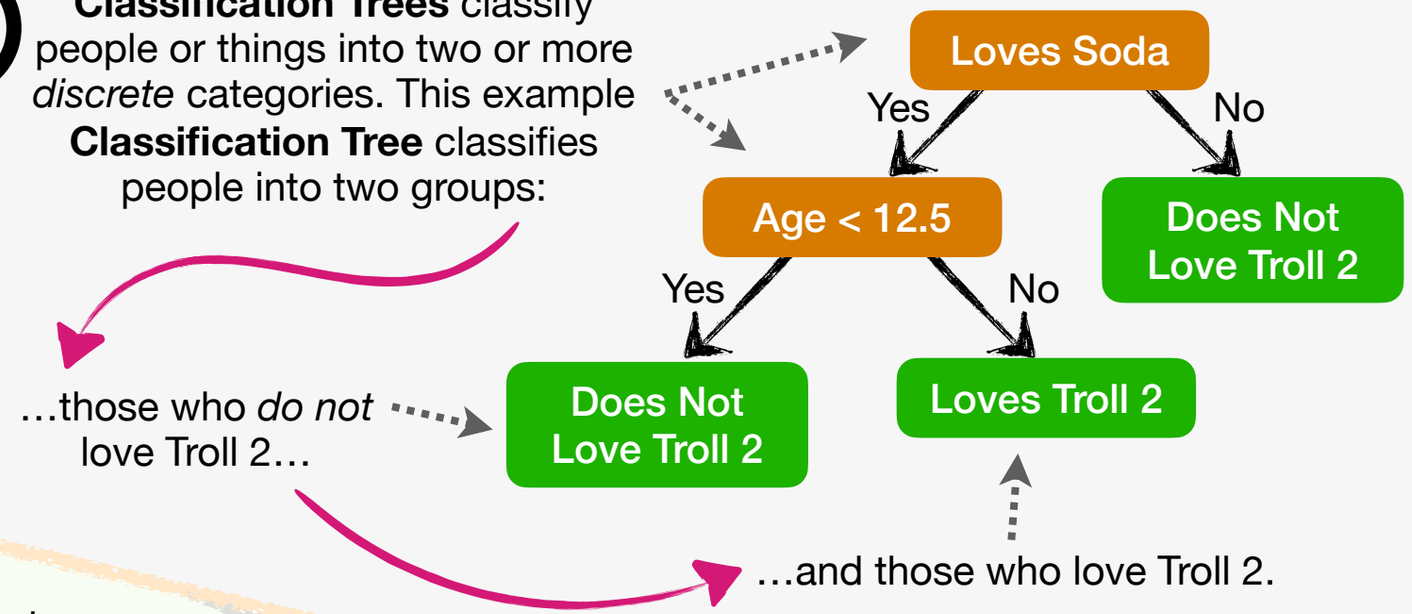
Chapter 10

Decision Trees!!!

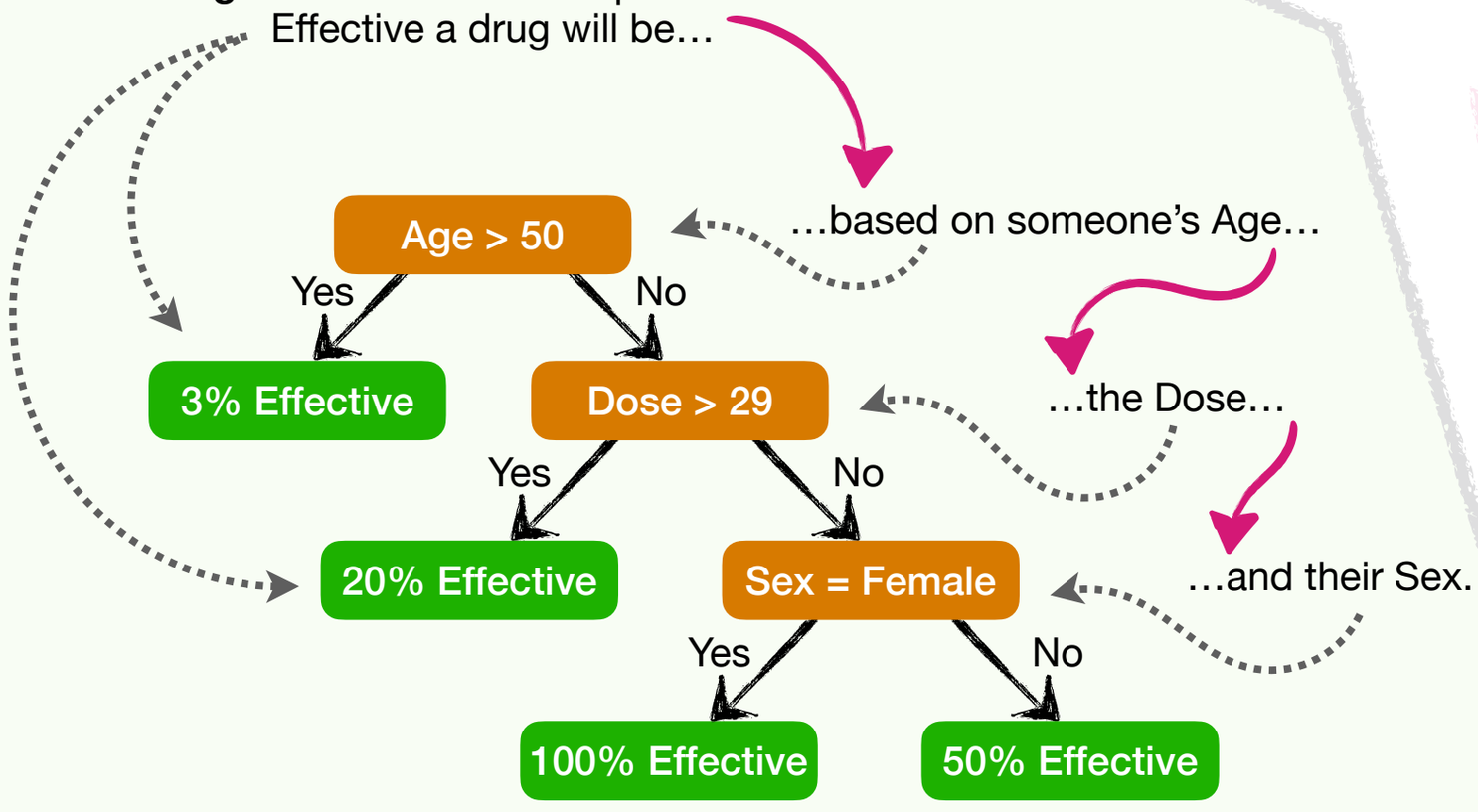
Classification and Regression Trees: Main Ideas

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:



3 In contrast, **Regression Trees** try to predict a *continuous* value. This example **Regression Tree** tries to predict how Effective a drug will be…



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…

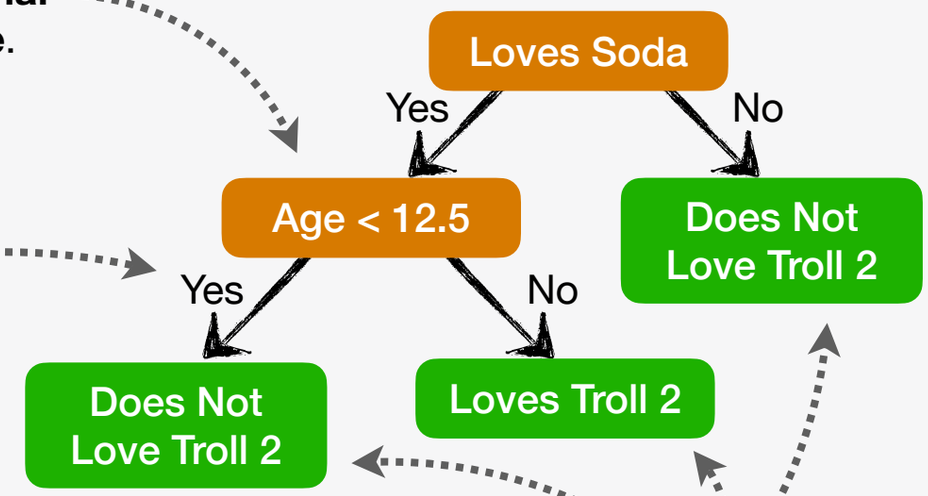
Terminology Alert!!! Decision Tree Lingo

1 The good news is that **Decision Trees** are pretty simple, and there's not too much terminology.

One thing that's weird about **Decision Trees** is that they're upside down! The roots are on top, and the leaves are on the bottom!

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

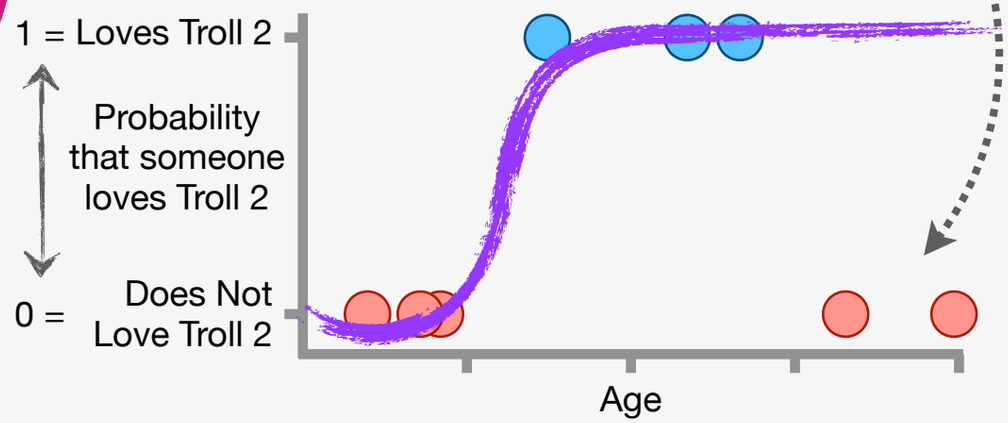
Classification Trees: Main Ideas

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

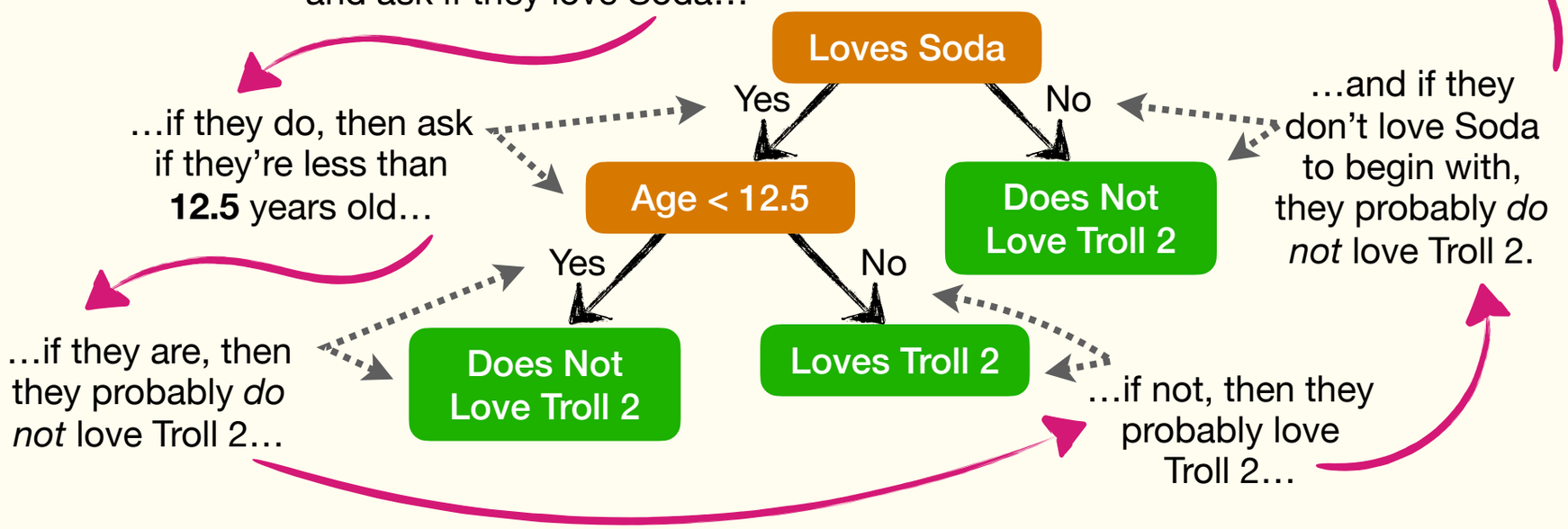
…that we want to use to predict if someone loves Troll 2, the **1990** blockbuster movie that was neither about trolls nor a sequel.

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…



BAM!!!

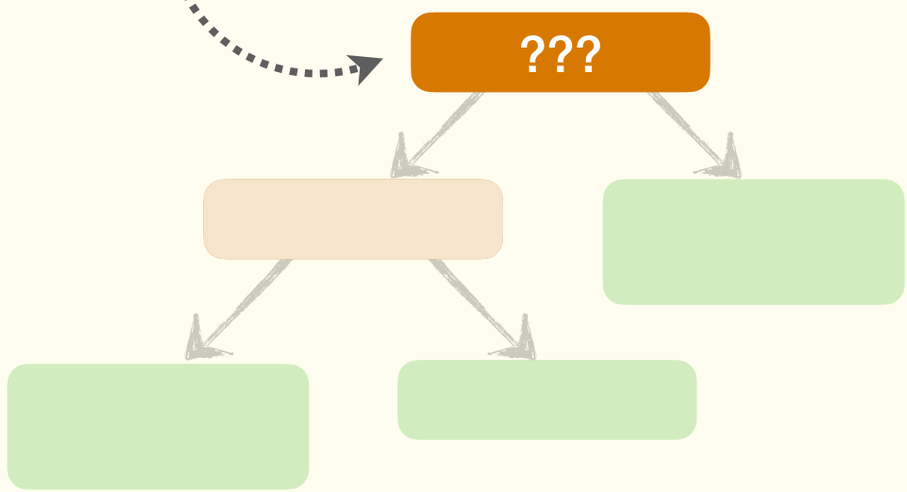
Building a Classification Tree: Step-by-Step

1 Given this **Training Dataset**, we want to build a **Classification Tree** that uses Loves Popcorn, Loves Soda, and Age…

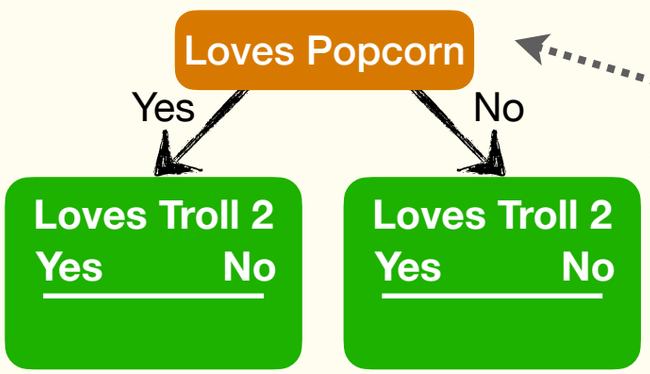
Loves Popcorn	Loves Soda	Age	Loves Troll 2
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

…to predict whether or not someone will love Troll 2.

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…

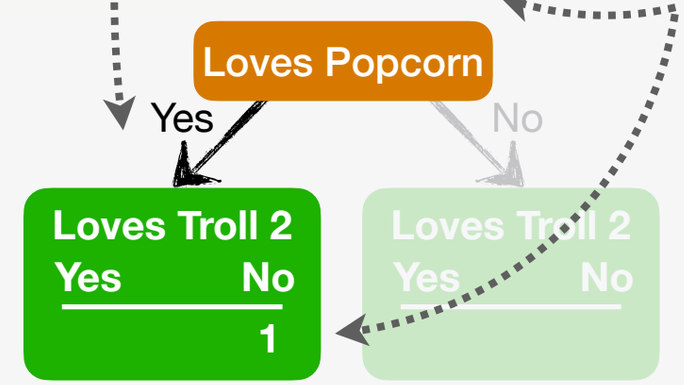


…by making a super simple tree that only asks if someone loves Popcorn and running the data down it.

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

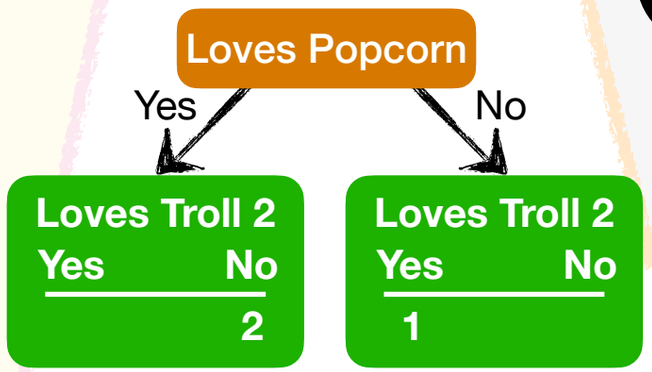
…and because they *do not* love Troll 2, we'll put a **1** under the word **No**.



Building a Classification Tree: Step-by-Step

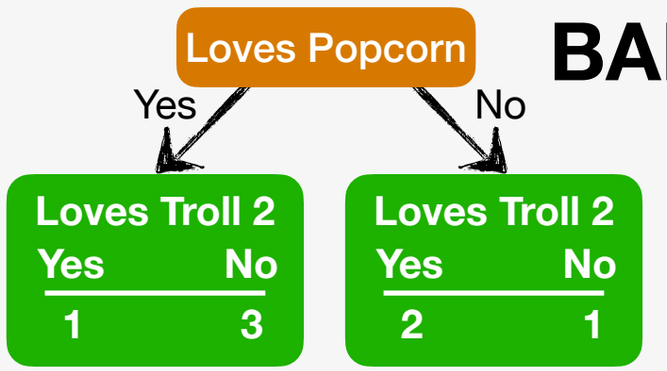
5 The second person also loves Popcorn, so they also go to the **Leaf** on the *left*, and because they do not love Troll 2, we increase **No** to **2**.

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**.

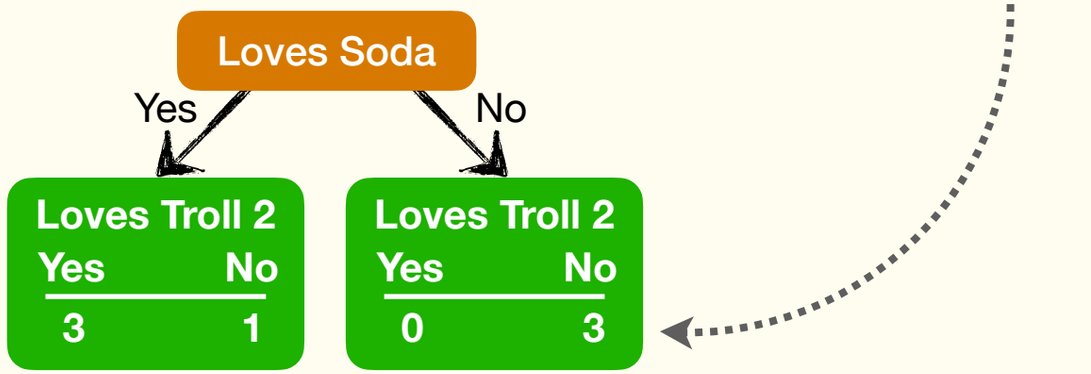
7 Likewise, we run the remaining rows down the tree, keeping track of whether or not each person loves Troll 2 or not.



BAM!!!

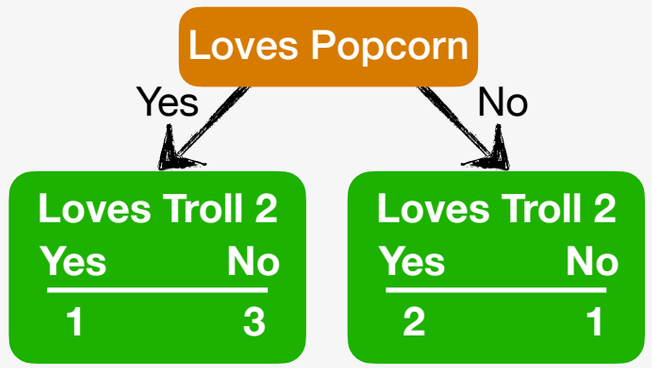
8 Then we do the same thing for Loves Soda.

Loves Popcorn	Loves Soda	Age	Loves Troll 2
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
...	...	...	...



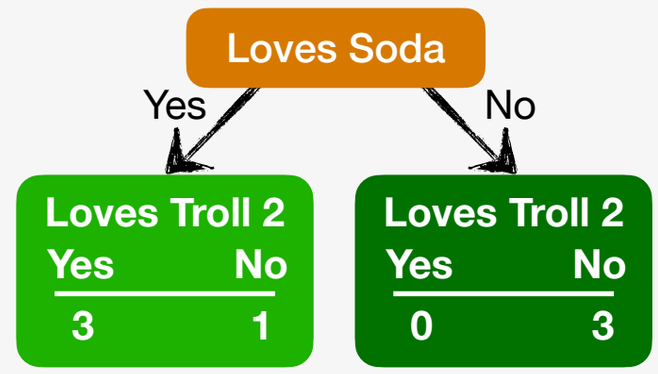
Building a Classification Tree: Step-by-Step

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**…

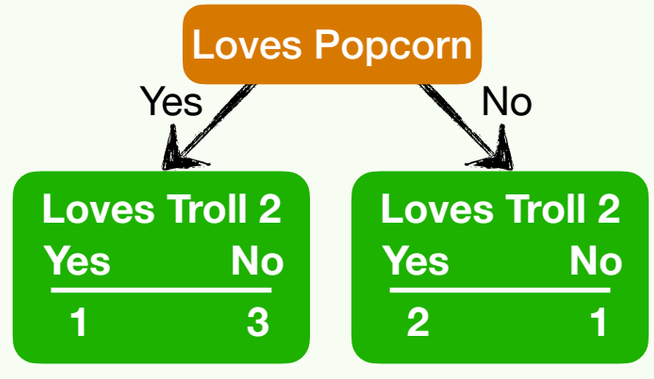
…it seems like Loves Soda does a better job classifying who loves and does not love Troll 2, but it would be nice if we could *quantify* the differences between Loves Popcorn and Loves Soda.

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**.

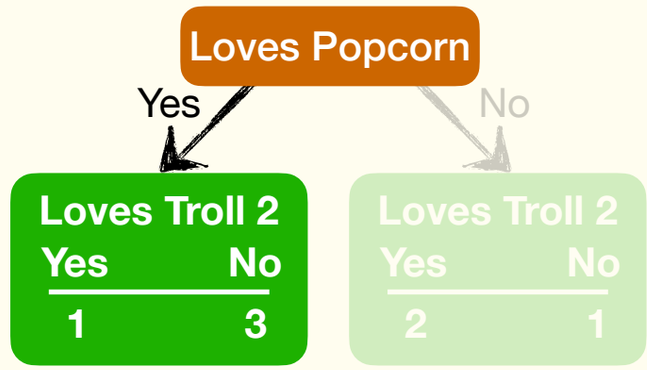
In theory, all of the methods give similar results, so we'll focus on **Gini Impurity** since it's very popular and I think it's the most straightforward.

We'll start by calculating the **Gini Impurity** to quantify the **Impurity** in the **Leaves** for loves Popcorn.



Building a Classification Tree: Step-by-Step

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**.



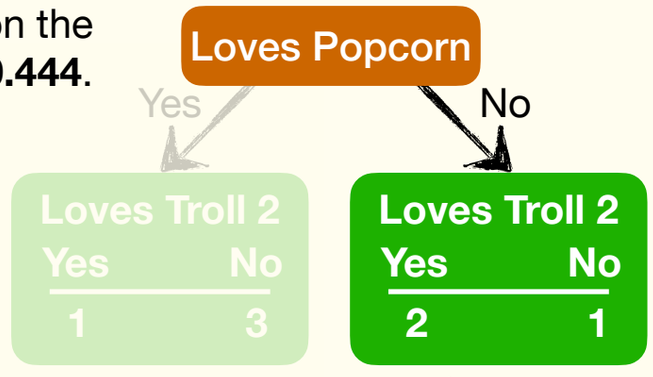
13 **Gini Impurity** = 1 - (the probability of "yes")^2 - (the probability of "no")^2 for a **Leaf**

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**.

$$= 1 - \left(\frac{\text{The number for } \mathbf{Yes}}{\text{The total for the } \mathbf{Leaf}}\right)^2 - \left(\frac{\text{The number for } \mathbf{No}}{\text{The total for the } \mathbf{Leaf}}\right)^2$$

$$= 1 - \left(\frac{1}{1+3}\right)^2 - \left(\frac{3}{1+3}\right)^2 = 0.375$$

15 For the **Leaf** on the *right*, we get **0.444**.



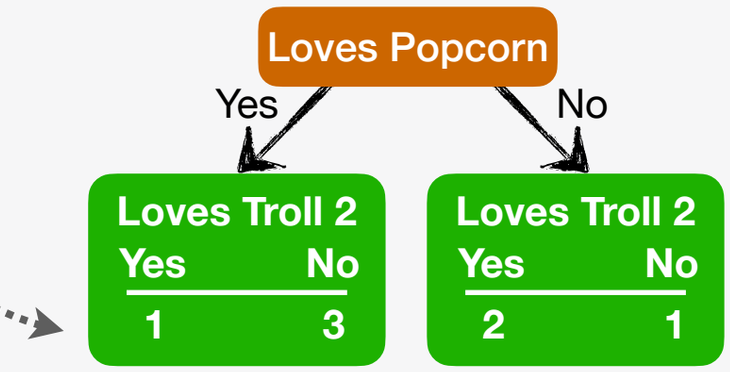
Gini Impurity = 1 - (the probability of "yes")^2 - (the probability of "no")^2 for a **Leaf**

$$= 1 - \left(\frac{2}{2+1}\right)^2 - \left(\frac{1}{2+1}\right)^2 = 0.444$$

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.



Gini Impurity = 0.375

Gini Impurity = 0.444

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**…

…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**…

$$\text{Total Gini Impurity} = \left(\frac{4}{4+3}\right)0.375 + \left(\frac{3}{4+3}\right)0.444 = 0.405$$

…and when we do the math, we get **0.405**.

BAM!!!

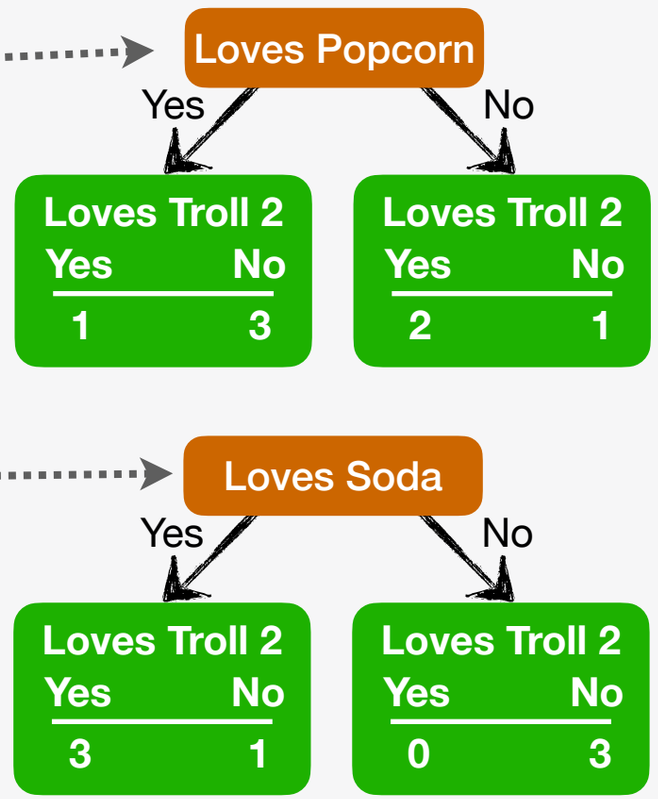
Building a Classification Tree: Step-by-Step

19 Now that we've calculated the **Gini Impurity** for Loves Popcorn, **0.405**…

…we can follow the same steps to calculate the **Gini Impurity** for Loves Soda, **0.214**.

Gini Impurity for **Loves Popcorn** = **0.405**

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.

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

However, because Age contains numeric data, and not just **Yes/No** values, calculating the **Gini Impurity** is a little more involved.

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

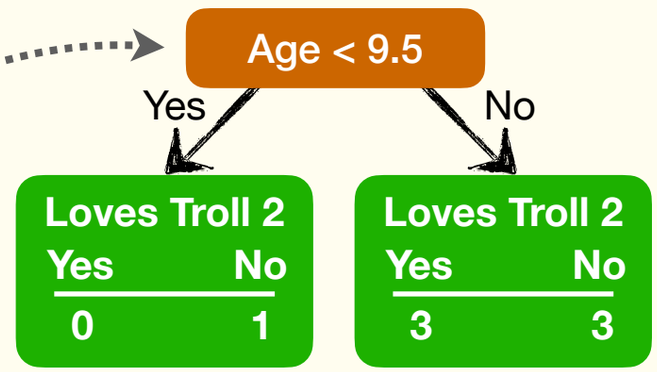
Building a Classification Tree: Step-by-Step

23 Lastly, we calculate the **Gini Impurity** values for each average Age.

Age	Loves Troll 2
7	No
12	No
18	Yes
35	Yes
38	Yes
50	No
83	No

9.5, 15, 26.5, 36.5, 44, 66.5

For example, the first average Age is **9.5**, so we use **9.5** as the threshold for splitting the rows into **2** leaves…



…and when we do the math, we get **0.429**.

$$\text{Total Gini Impurity} = \left(\frac{1}{1+6}\right)0.0 + \left(\frac{6}{1+6}\right)0.5$$

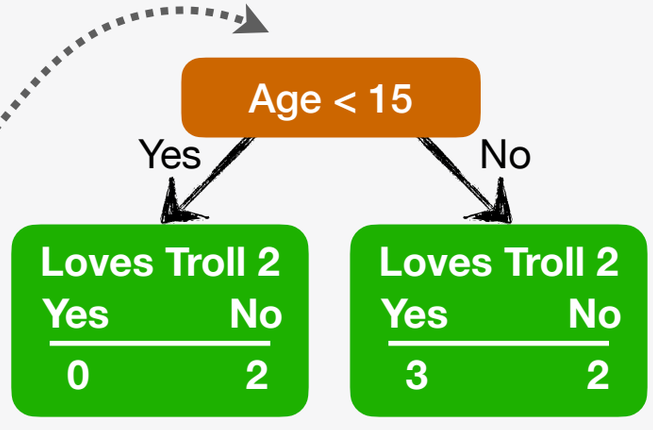
$$= 0.429$$

24 Ultimately, we end up with a **Gini Impurity** for each potential threshold for Age…

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	

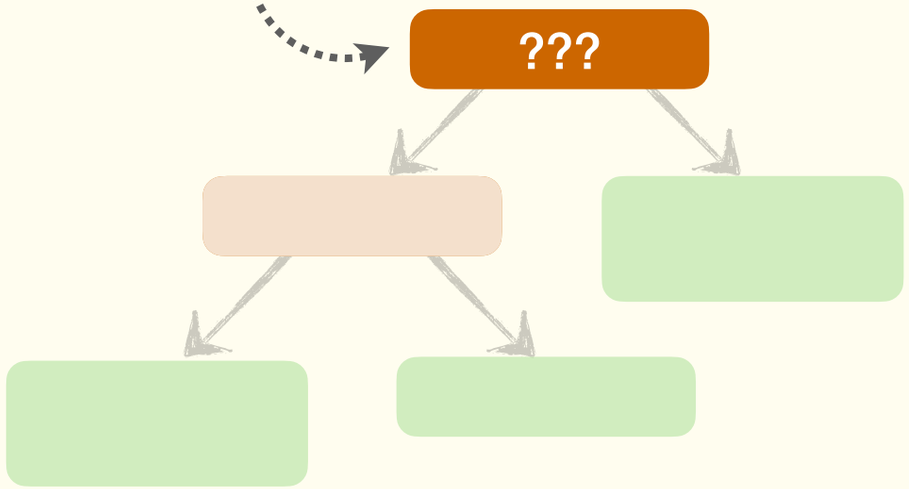
…and then we identify the thresholds with the lowest **Impurities**, and because the candidate thresholds **15** and **44** are tied for the lowest **Impurity**, **0.343**, we can pick either one for the **Root**. In this case, we'll pick **15**.

BAM!!!



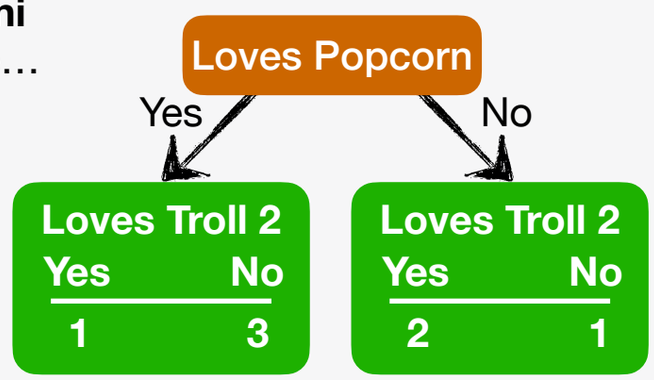
Building a Classification Tree: Step-by-Step

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…

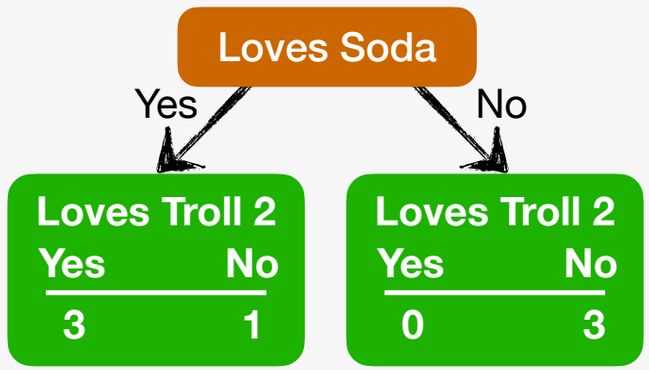


26 …so we calculated the **Gini Impurities** for each feature…

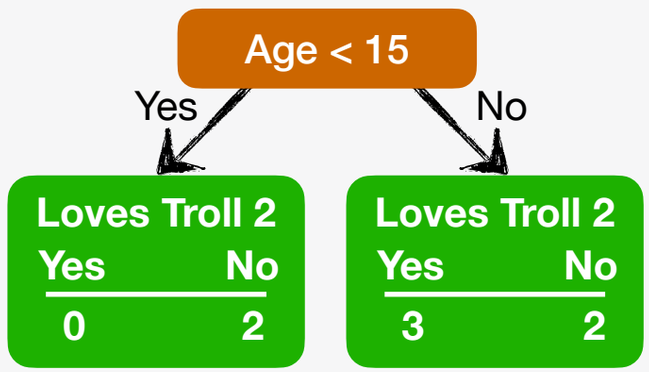
Gini Impurity for **Loves Popcorn** = **0.405**



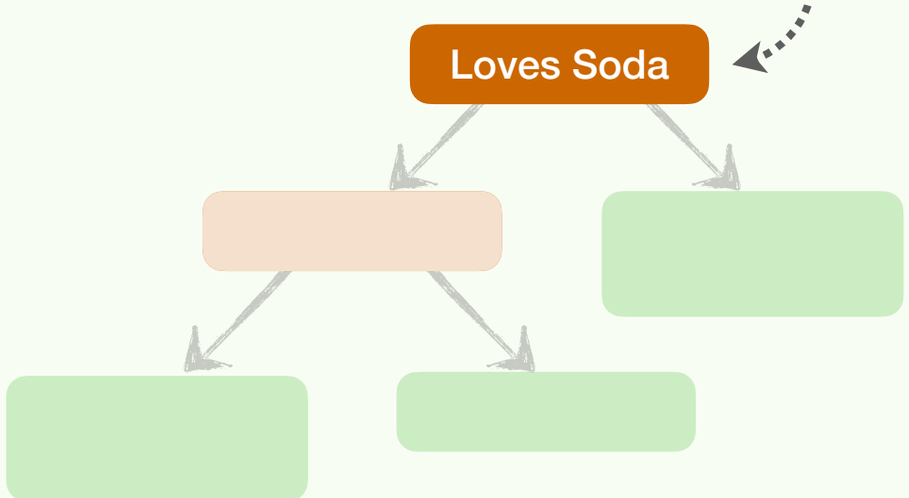
Gini Impurity for **Loves Soda** = **0.214**



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.

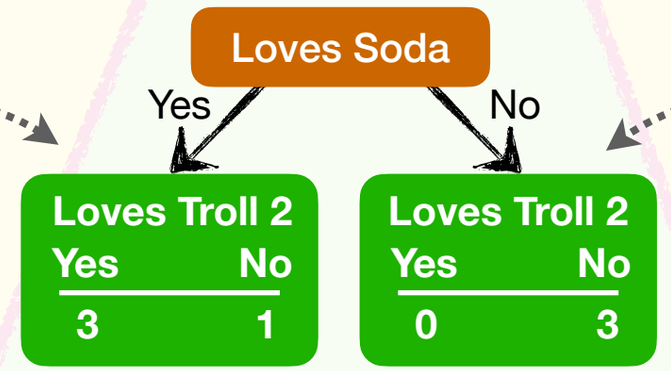


BAM!!!

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

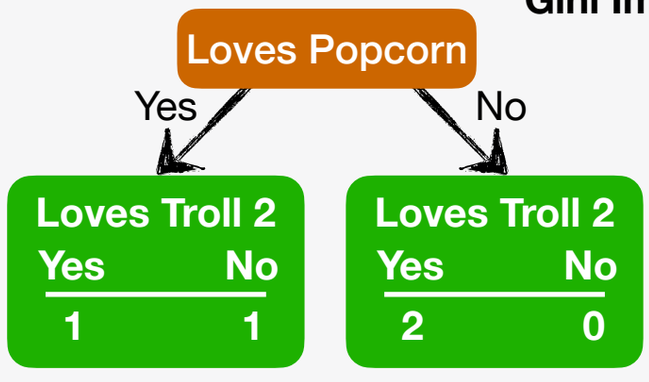


29 …and the **3** people who *do not* love Soda, all of whom *do not* love Troll 2, go to the **Node** on the *right*.

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

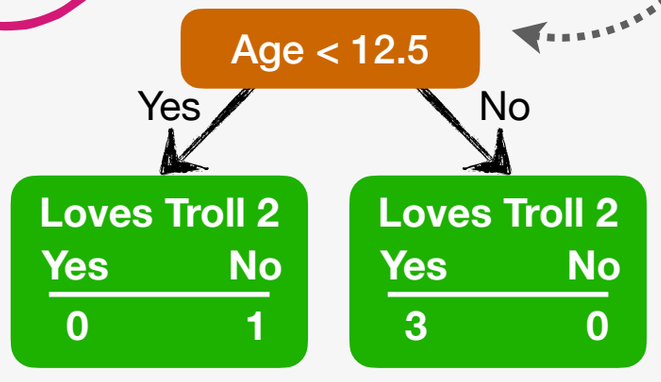
30 Now, because the **Node** on the *left* is **Impure**, we can split the **4** people in it based on Loves Popcorn or Age and calculate the **Gini Impurities**.

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

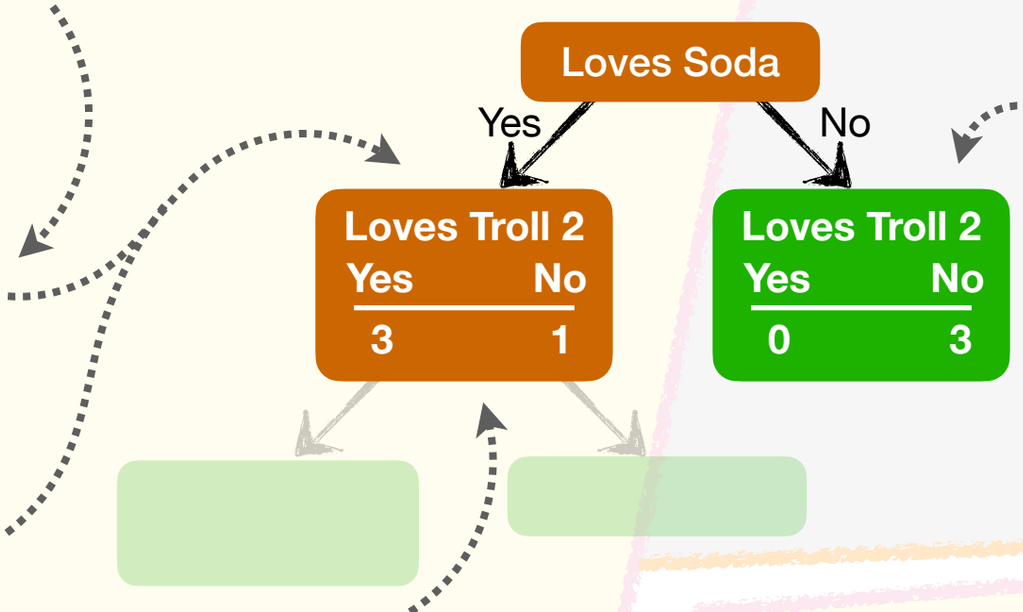
Loves Popcorn	Loves Soda	Age	Loves Troll 2
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No

12.5 · 26.5 · 36.5

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**…

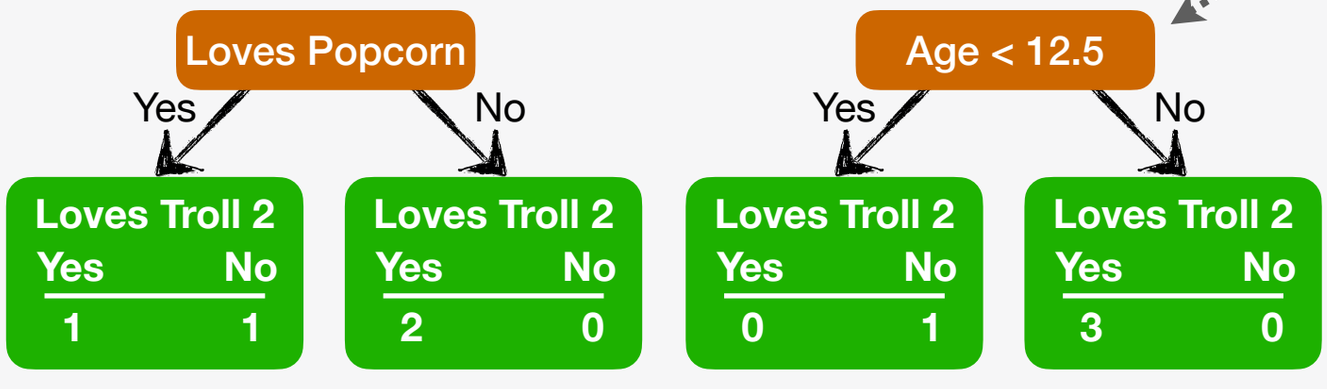
Loves Popcorn	Loves Soda	Age	Loves Troll 2
Yes	Yes	7	No
Yes	No	12	No
No	Yes	18	Yes
No	Yes	35	Yes
Yes	Yes	38	Yes
Yes	No	50	No
No	No	83	No



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.

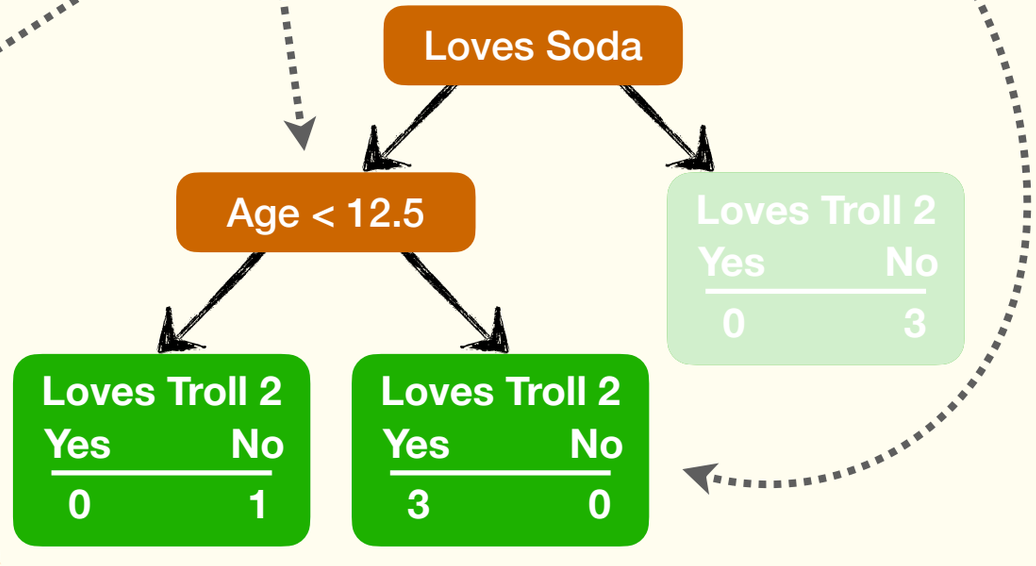
34 In contrast, because the **4** people who love Soda are a mixture of people who *do* and *do not* love Troll 2, we build simple trees with them based on Loves Popcorn and Age…



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**.



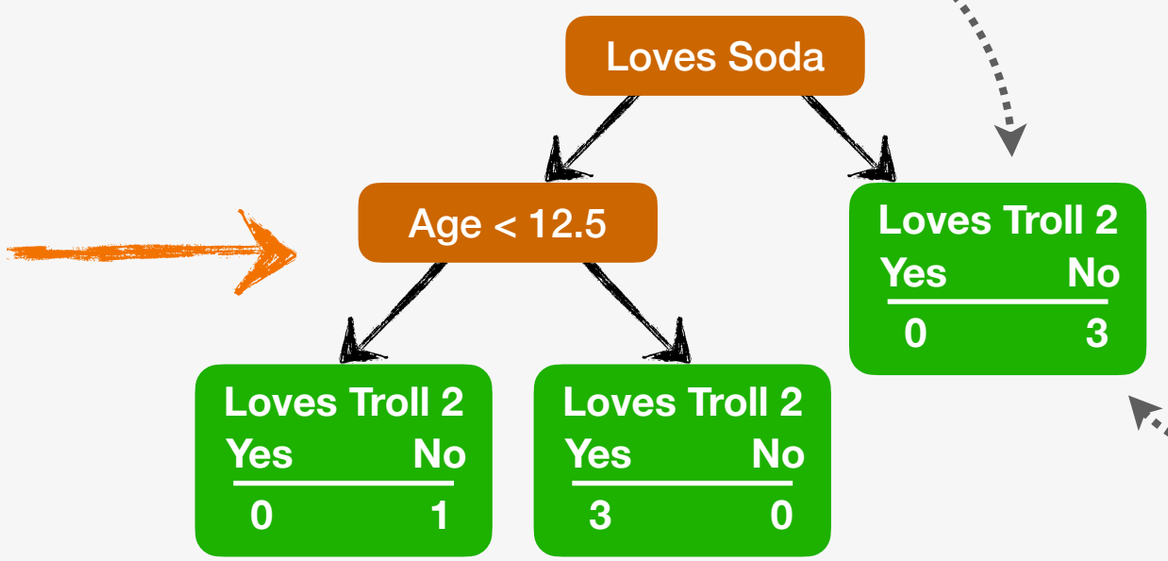
Building a Classification Tree: Step-by-Step

36 At this point, we've created a **Tree** from the **Training Data**.

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

Now the only thing remaining is to assign output values for each **Leaf**.

Generally speaking, the output of a **Leaf** is whatever category that has the most counts.

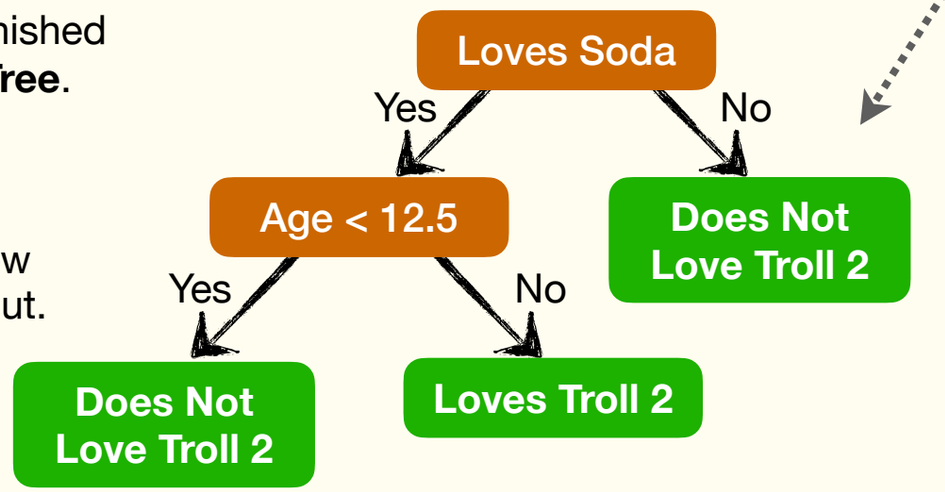


In other words, because the majority of the people in this **Leaf** *do not* love Troll 2, its output value is does not love Troll 2.

37 Hooray!!! After assigning output values to each **Leaf**, we've finally finished building a **Classification Tree**.

BAM?

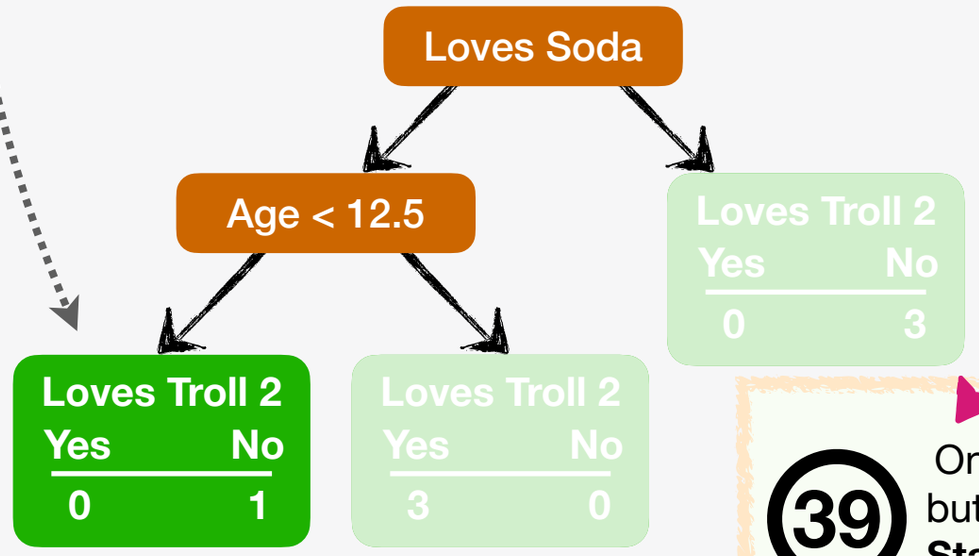
Not yet, there are still a few things we need to talk about.



Building a Classification Tree: Step-by-Step

38 When we built this tree, only one person in the **Training Data** made it to this **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



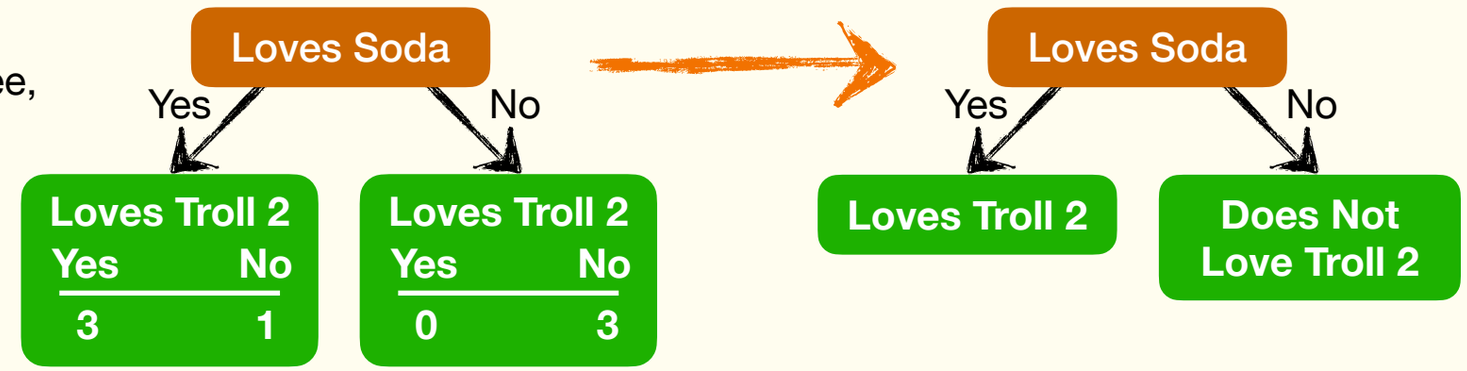
…and because so few people in the **Training Data** made it to that **Leaf**, it's hard to have confidence that the tree will do a great job making predictions with future data.

However, in practice, there are two main ways to deal with this type of problem.

39 One method is called **Pruning**, but we'll save that topic for **The StatQuest Illustrated Guide to Tree-Based Machine Learning!!!**

40 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!!!

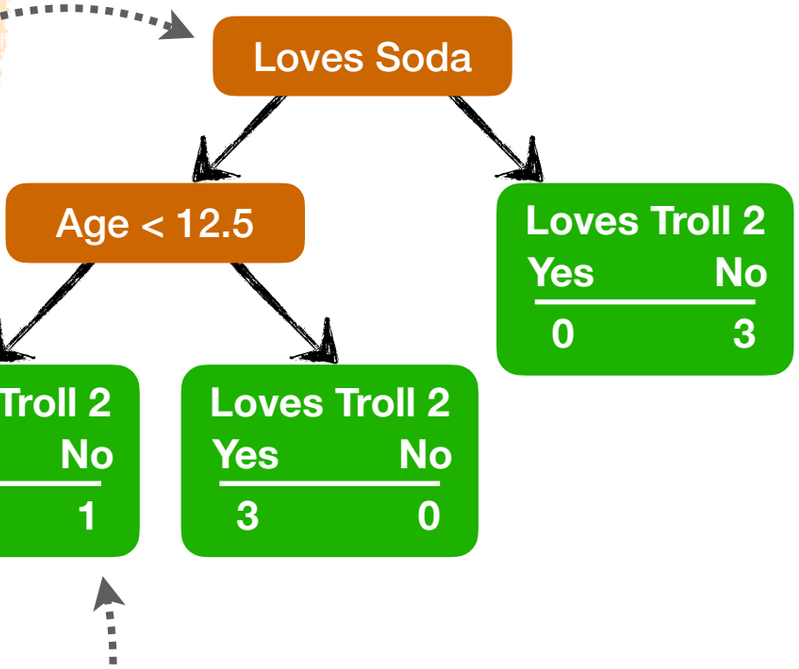
Now let's summarize how to build a **Classification Tree**.

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



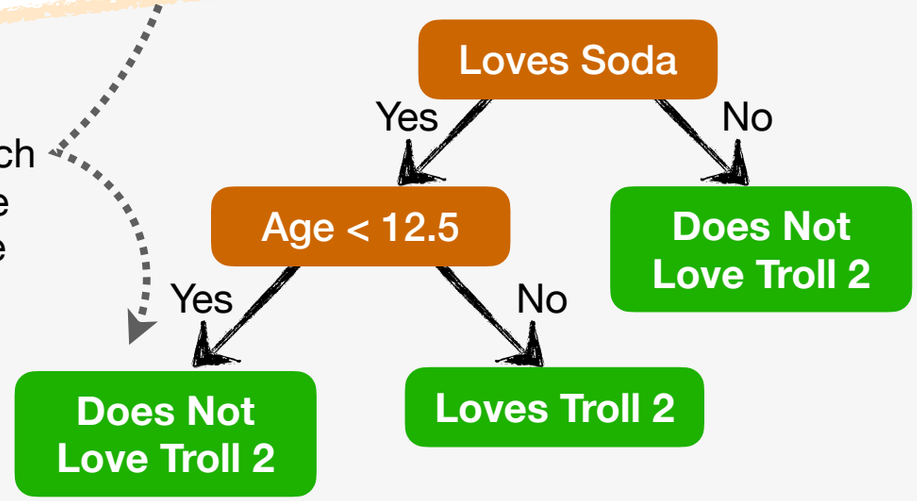
2 Then we used the **4** people who love Soda, which were a mixture of people who *do* and *do not* love Troll 2, to calculate **Gini Impurities** and selected **Age < 12.5** for the next **Node**.

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.

TRIPLE BAM!!



Now that we know all about **Classification Trees**, it's time for **Part Deux**, **Regression Trees**!!!