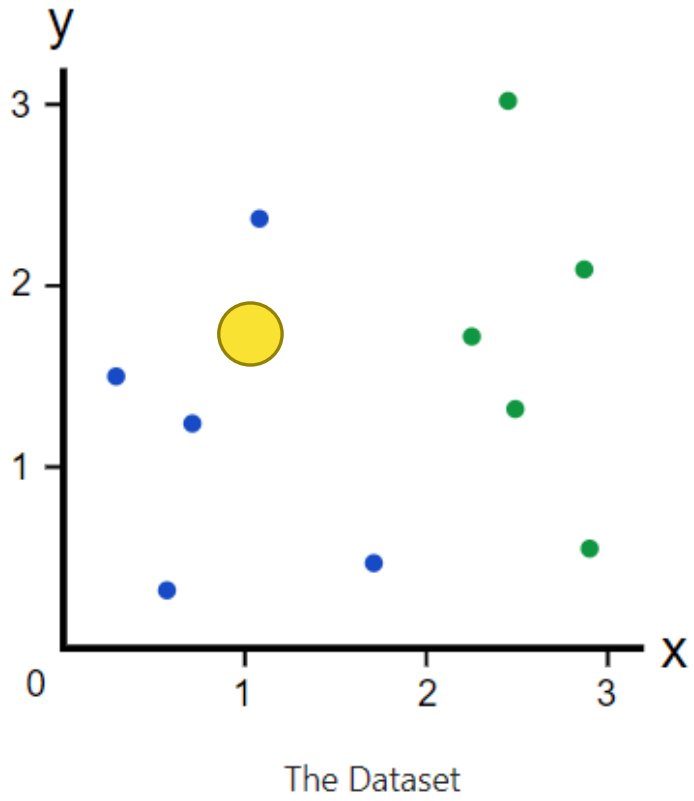
Decision Tree

DECISION TREE!

IN A NUTSHELL…

If I told you that there was a new point with an x coordinate of 1, what color do you think it'd be?



The Dataset

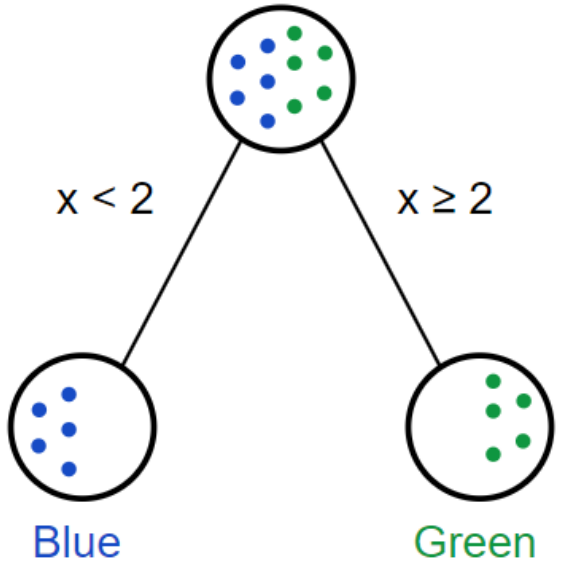
Credit: https://victorzhou.com/blog/intro-to-random-forests/

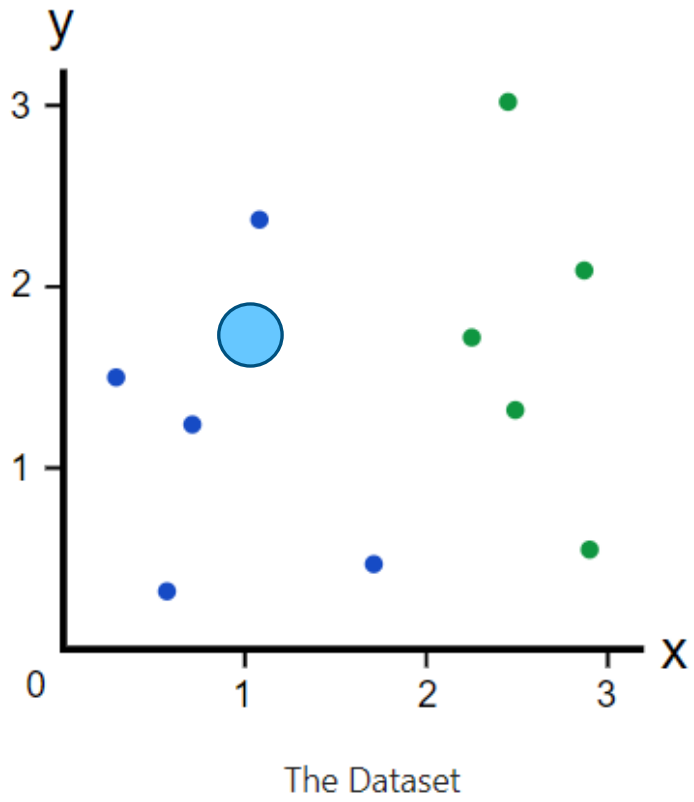
DECISION TREE!

IN A NUTSHELL…

Blue, right?

You just evaluated a decision tree in your head:





The Dataset

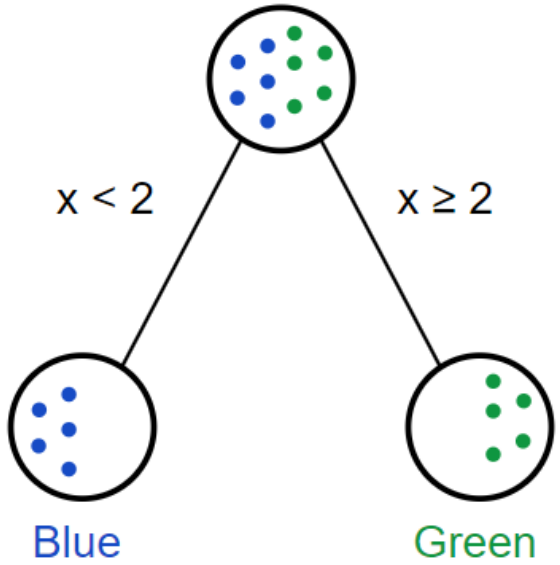
DECISION TREE!

IN A NUTSHELL…

That’s a simple decision tree with one **decision node** that **tests** x < 2.

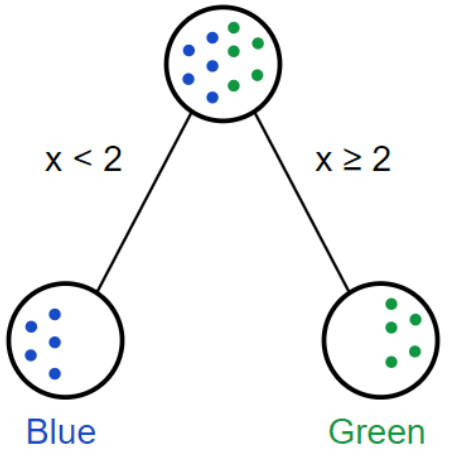
If the test passes (x < 2), we take the left **branch** and pick Blue.

If the test fails (x ≥2), we take the right **branch** and pick Green.



DECISION TREE!

IN A NUTSHELL…

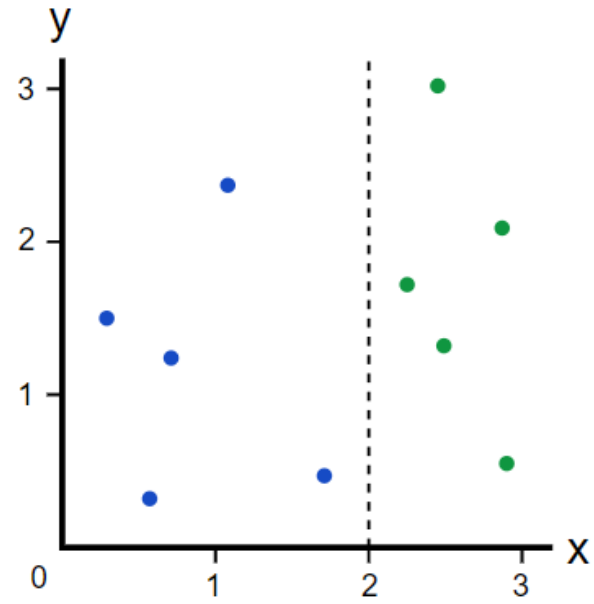


Decision Trees are often used to answer that kind of question:

Given a **labelled** dataset, how should we **classify** new samples?

Labelled: Our dataset is labelled because each point has a class (color): blue or green.

Classify: To classify a new datapoint is to assign a class (color) to it.



The Dataset, split at x=2

DECISION TREE!

IN A NUTSHELL…

Here’s a dataset that has 3 classes now instead of 2:

Our old decision tree **doesn’t work so well anymore**.

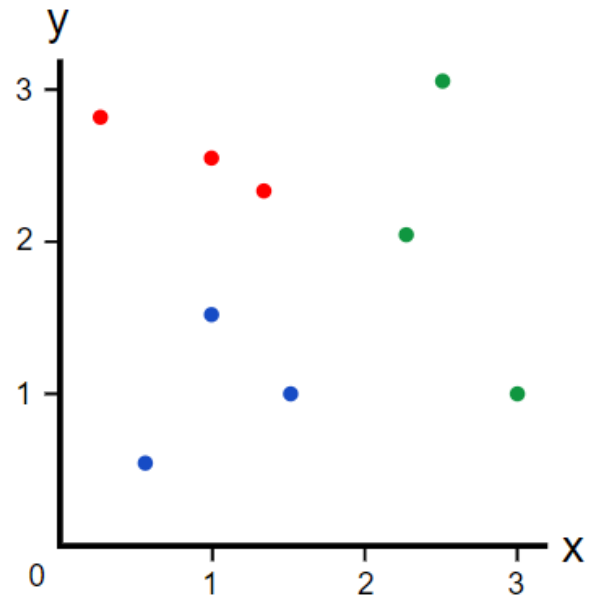
Given a new point (x, y):

If x ≥2:

We can still confidently classify it as green.

If x <2:

We can’t immediately classify it as blue - it could be red, too.

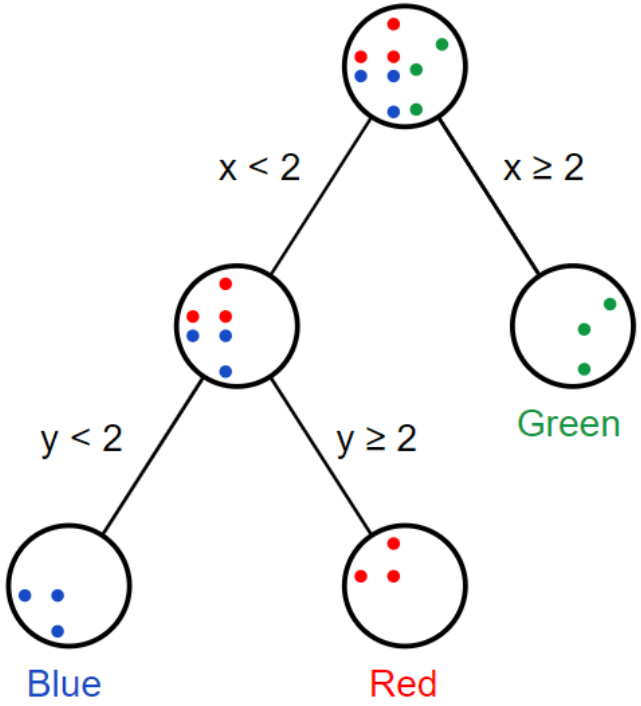


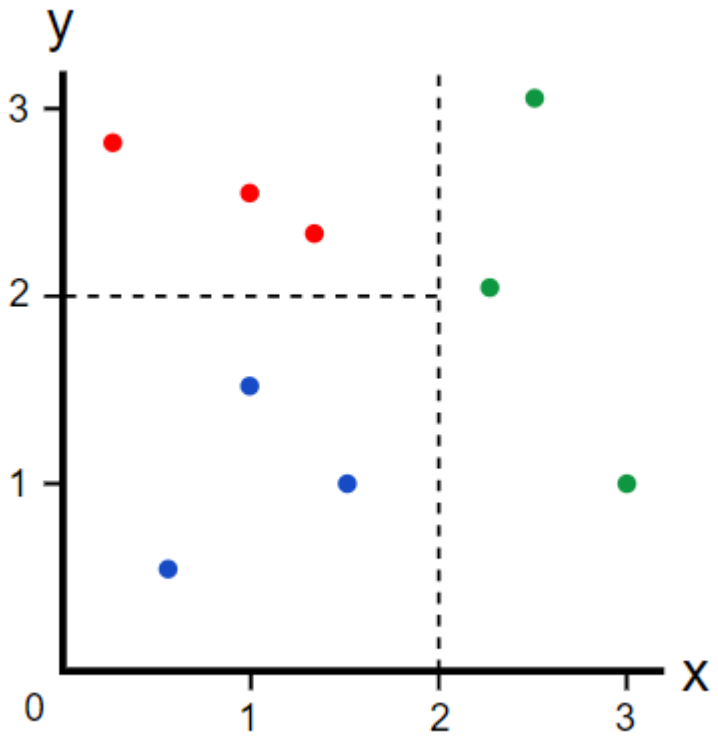
The Dataset v2

DECISION TREE!

IN A NUTSHELL…

We need to add another decision node to our decision tree:





DECISION TREE!

IN A NUTSHELL…

How to train a decision tree??

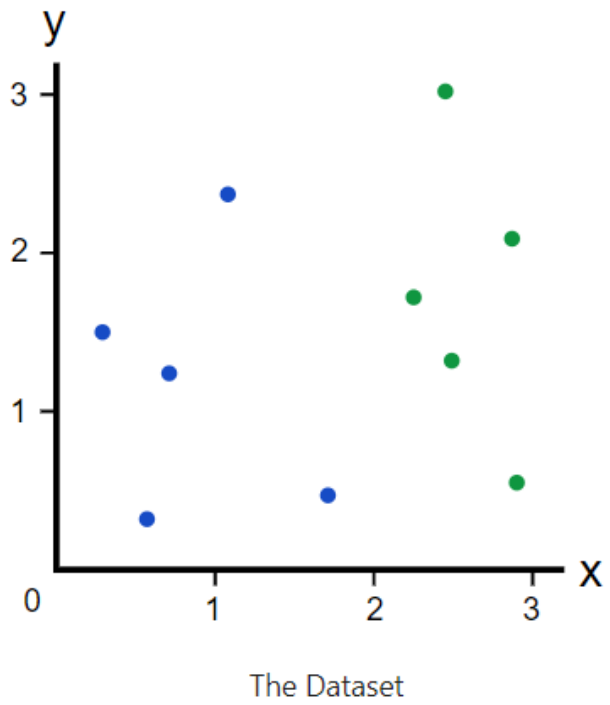
Training a decision tree consists of iteratively splitting the current data into two branches.

Credit: https://victorzhou.com/blog/gini-impurity/

DECISION TREE!

IN A NUTSHELL…

Suppose we have the following data:



The Dataset

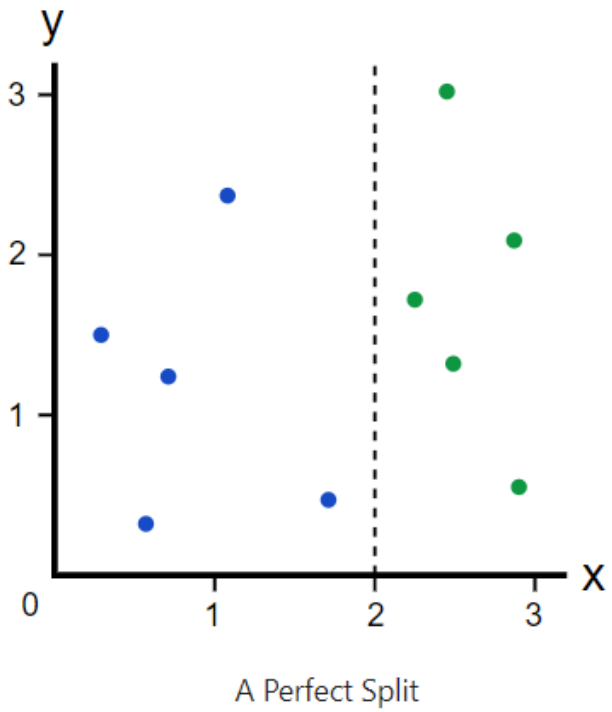
Right now, we have 1 branch with 5 blues and 5 greens.

DECISION TREE!

IN A NUTSHELL…

Suppose we have the following data:

Let’s make a split at x = 2:



A Perfect Split

This is a **perfect** split! It breaks our dataset perfectly into two branches:

Left branch, with 5 blues.

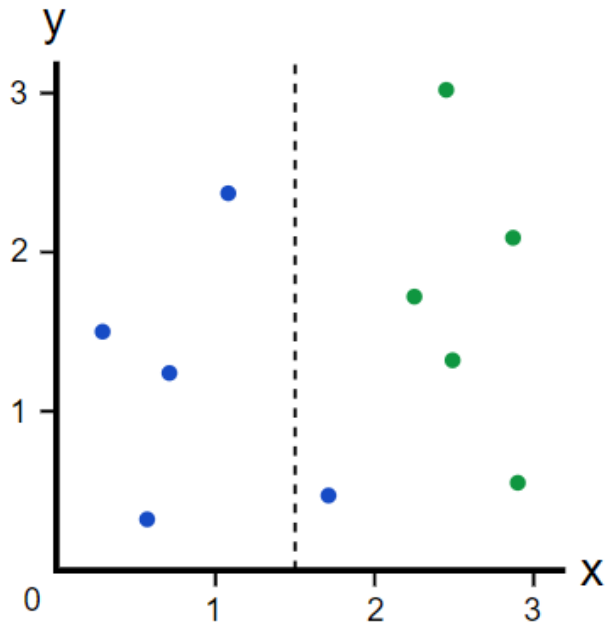
Right branch, with 5 greens.

DECISION TREE!

IN A NUTSHELL…

Suppose we have the following data:

What if we'd made a split at x = 1.5x=1.5 instead?



An Imperfect Split

This **imperfect** split breaks our dataset into these branches:

Left branch, with 4 blues.

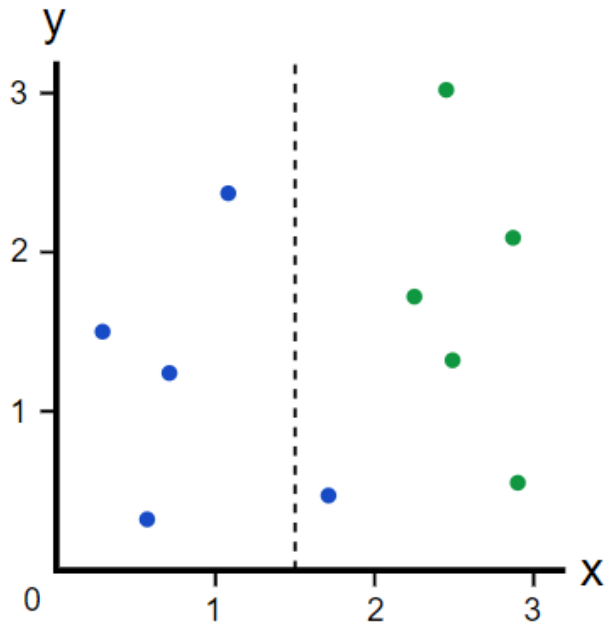
Right branch, with 1 blue and 5 greens.

DECISION TREE!

IN A NUTSHELL…

Suppose we have the following data:

What if we'd made a split at x = 1.5x=1.5 instead?



An Imperfect Split

It's obvious that this split is worse, but **how can we quantify that?**

DECISION TREE!

IN A NUTSHELL…

Being able to measure the quality of a split becomes even more important if we add a third class, reds. Imagine the following split:

Branch 1, with 3 blues, 1 green, and 1 red.
Branch 2, with 3 greens and 1 red.

Compare that against this split:

Branch 1, with 3 blues, 1 green, and 2 reds.
Branch 2, with 3 greens.

Which split is better? It's no longer immediately obvious.

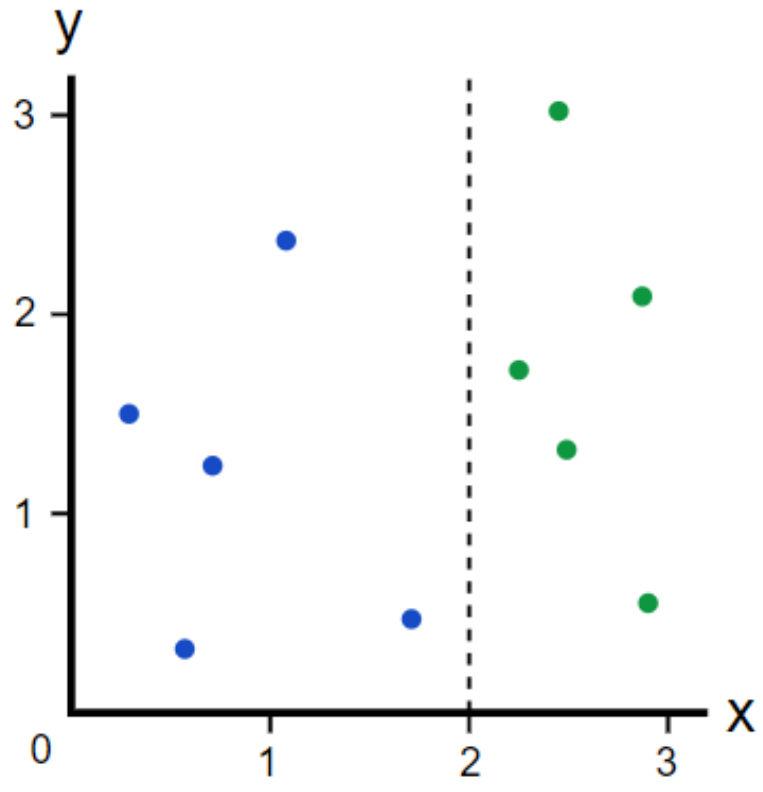
DECISION TREE!

GINI IMPURITY

Suppose we:

Randomly pick a datapoint in our dataset, then **Randomly classify it according to the class distribution in the dataset**.

For our dataset, we'd classify it as blue 5/10 of the time and as green 5/10 of the time, since we have 5 datapoints of each color.
What's the probability we classify the datapoint incorrectly? The answer to that question is the Gini Impurity.



A Perfect Split

DECISION TREE!

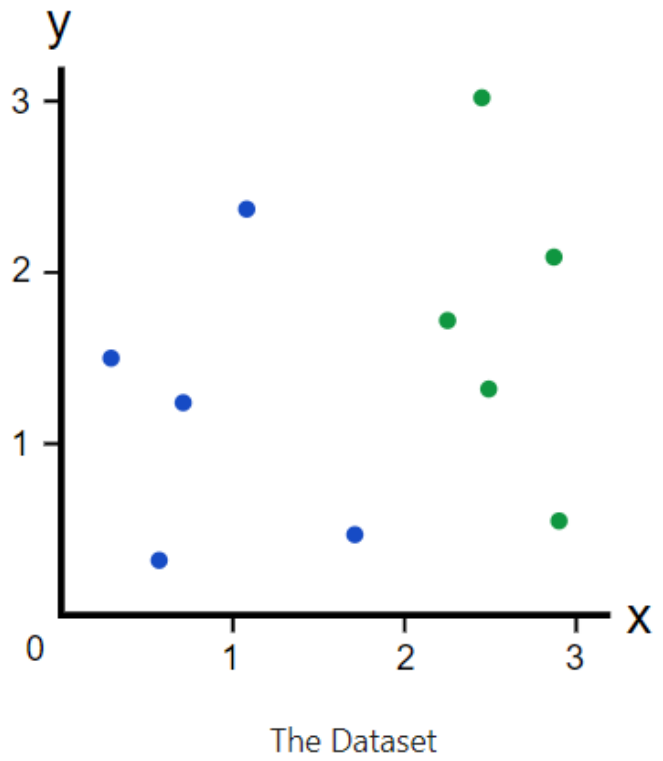
EXAMPLE 1: THE WHOLE DATASET

Let’s calculate the Gini Impurity of our entire dataset.

If we randomly pick a datapoint, it’s **either blue (50%) or green (50%).**

Now, we randomly classify our datapoint according to the class distribution. Since we have 5 of each color, we classify it as blue 50% of the time and as green 50% of the time.

What’s the probability we classify our datapoint **incorrectly**?



The Dataset

DECISION TREE!

EXAMPLE 1: THE WHOLE DATASET

We only classify it incorrectly in 2 of the events above. Thus, our total probability is 25% + 25% = 50%, so the Gini Impurity is 0.5.

Event	Probability
Pick Blue, Classify Blue ✓	25%
Pick Blue, Classify Green ✗	25%
Pick Green, Classify Blue ✗	25%
Pick Green, Classify Green ✓	25%

GINI IMPURITY

THE MATH (JUST FYI)

If we have C total classes and $p(i)$ is the probability of picking a datapoint with class i, then the Gini Impurity is calculated as

$$G = \sum_{i=1}^{C} p(i) * (1 - p(i))$$

For the example above, we have $C = 2$ and $p(1) = p(2) = 0.5$, so

$$\begin{aligned} G &= p(1) * (1 - p(1)) + p(2) * (1 - p(2)) \\ &= 0.5 * (1 - 0.5) + 0.5 * (1 - 0.5) \\ &= \boxed{0.5} \end{aligned}$$

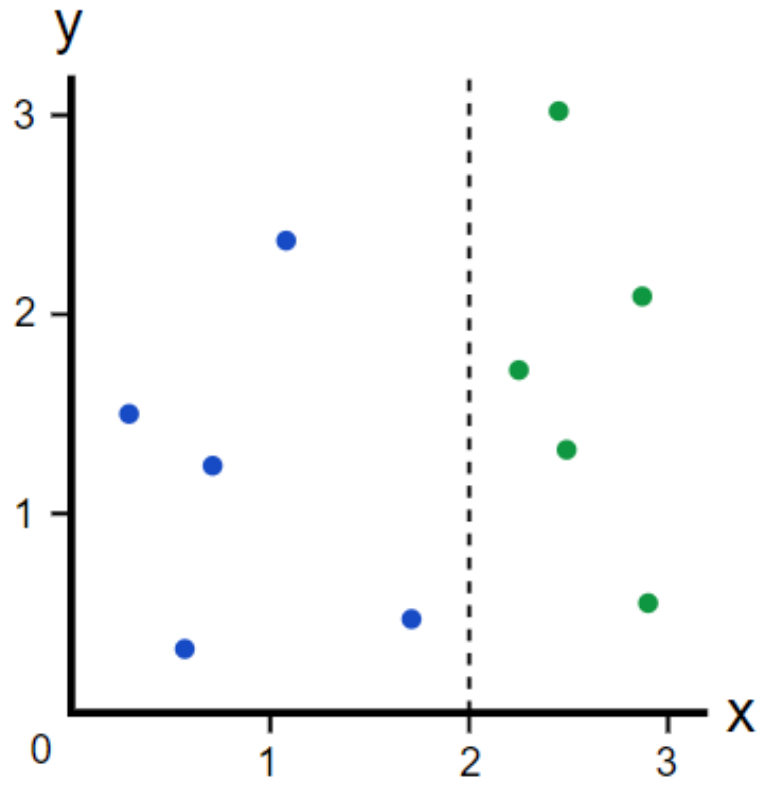
which matches what we calculated!

DECISION TREE!

EXAMPLE 2: A PERFECT SPLIT

Let’s go back to the perfect split we had.

What are the Gini Impurities of the two branches after the split?



A Perfect Split

DECISION TREE!

EXAMPLE 2: A PERFECT SPLIT

Left Branch has only blues, so its Gini Impurity is

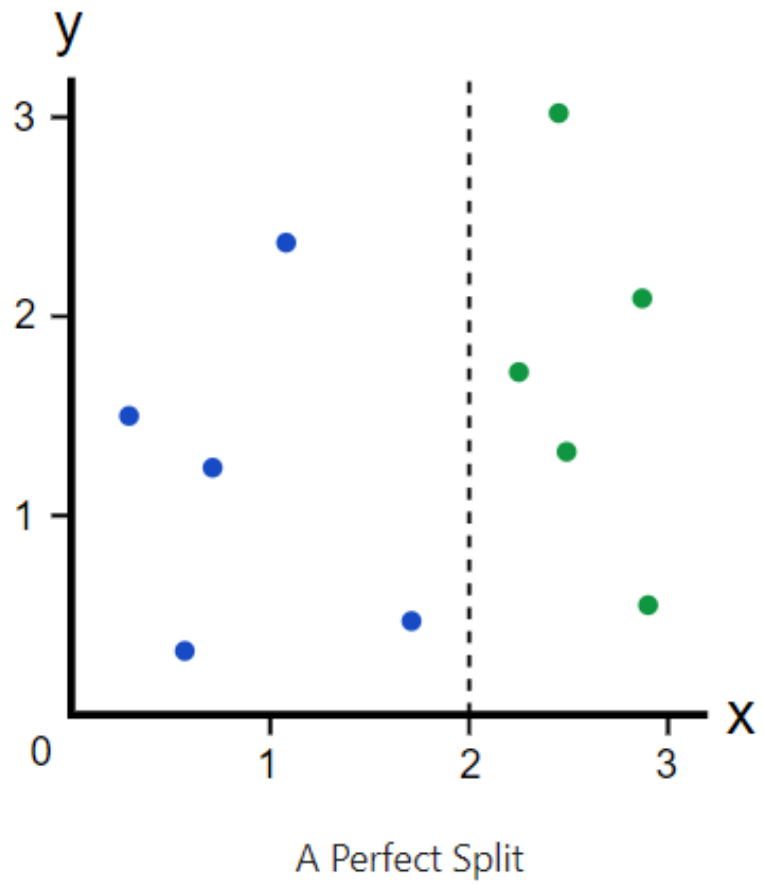
$$G_{left} = 1 * (1 - 1) + 0 * (1 - 0) = \boxed{0}$$

Right Branch has only greens, so its Gini Impurity is

$$G_{right} = 0 * (1 - 0) + 1 * (1 - 1) = \boxed{0}$$

Both branches have 0 impurity!

The perfect split turned a dataset with 0.5 impurity into 2 branches with 0 impurity.



A Perfect Split

DECISION TREE!

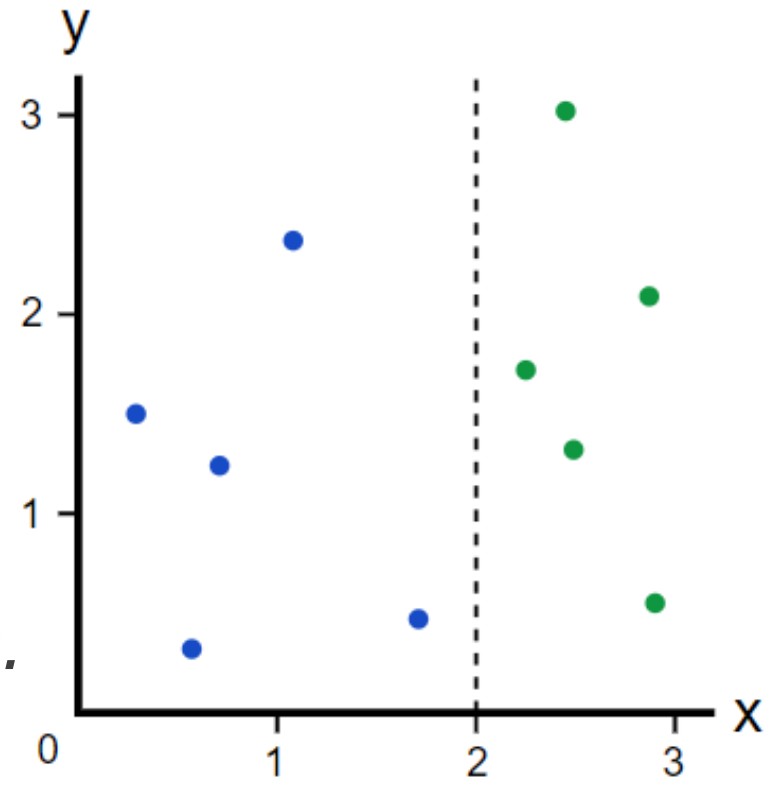
EXAMPLE 2: A PERFECT SPLIT

A Gini Impurity of 0 is the lowest and best possible impurity.

It can only be achieved when everything is the same class (e.g. only blues or only greens).

Which means…

Leaf nodes all have a Gini Impurity of 0.



A Perfect Split

DECISION TREE!

EXAMPLE 3: AN IMPERFECT SPLIT

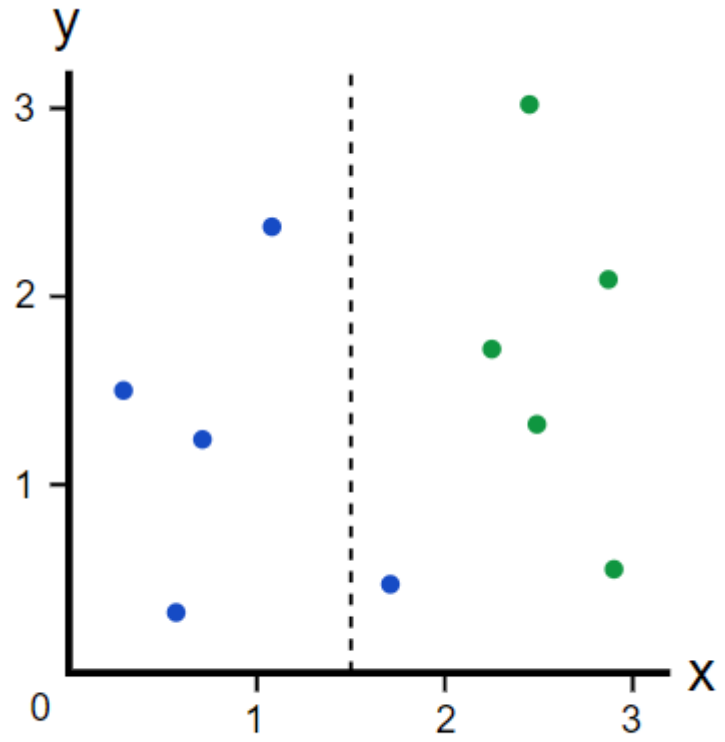
Finally, let's return to our imperfect split.

Left Branch has only blues, so we know that:

$$G_{left} = \boxed{0}.$$

Right Branch has 1 blue and 5 greens, so:

$$\begin{aligned} G_{right} &= \frac{1}{6} * (1 - \frac{1}{6}) + \frac{5}{6} * (1 - \frac{5}{6}) \\ &= \frac{5}{18} \\ &= \boxed{0.278} \end{aligned}$$



An Imperfect Split

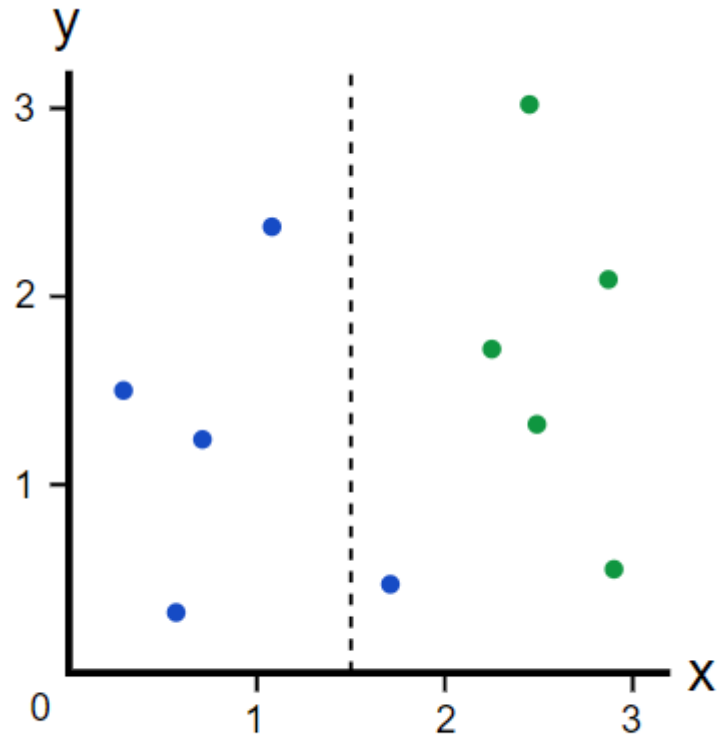
DECISION TREE!

PICKING THE BEST SPLIT

It’s finally time to answer the question we posed earlier:

how can we quantitatively evaluate the quality of a split?

Let’s take a look at the imperfect split again…



An Imperfect Split

DECISION TREE!

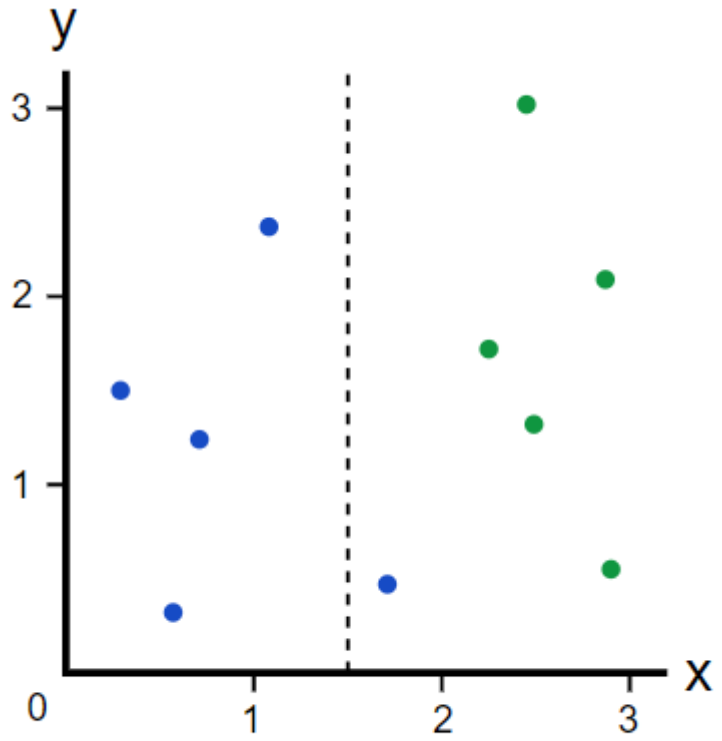
PICKING THE BEST SPLIT

We’ve already calculated the Gini Impurities for:

Before the split (the entire dataset): 0.5

Left Branch: 0

Right Branch: 0.278



An Imperfect Split

DECISION TREE!

PICKING THE BEST SPLIT

We'll determine the quality of the split by **weighting the impurity of each branch by how many elements it has**.

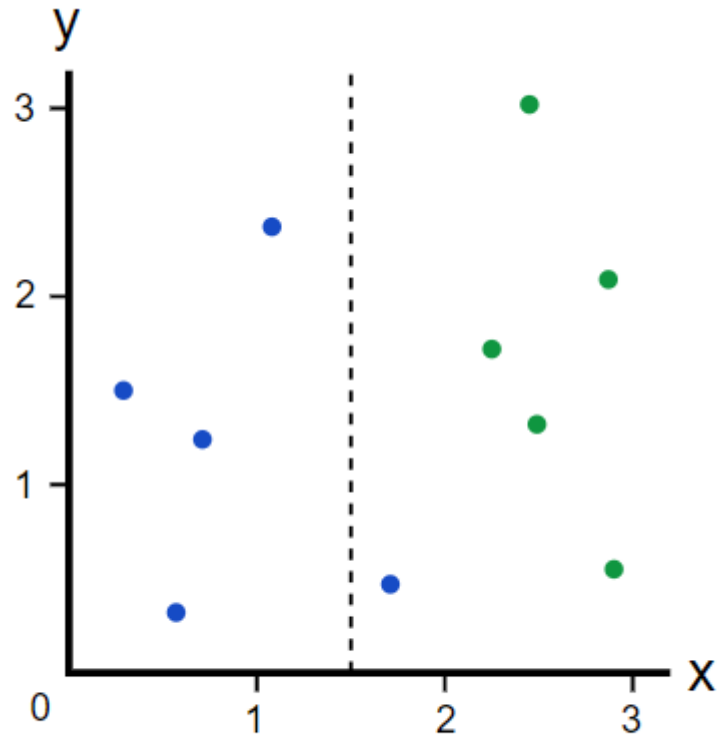
Since Left Branch has 4 elements and Right Branch has 6, we get:

$$(0.4 * 0) + (0.6 * 0.278) = 0.167$$

Thus, the amount of impurity we've "removed" with this split is:

$$0.5 - 0.167 = \boxed{0.333}$$

This is the Gini Gain.



An Imperfect Split

DECISION TREE!

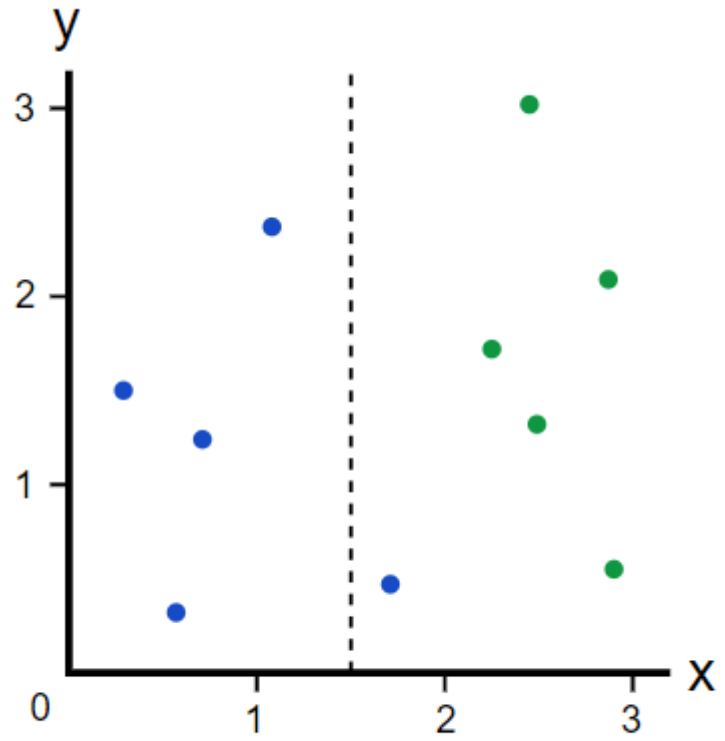
PICKING THE BEST SPLIT

This is what's used to pick the best split in a decision tree!

Higher Gini Gain = Better Split.

For example, it's easy to verify that the Gini Gain of the perfect split on our dataset is 0.5 > 0.333.

Gini Impurity is the probability of *incorrectly* classifying a randomly chosen element in the dataset if it were randomly labeled *according to the class distribution* in the dataset.



An Imperfect Split

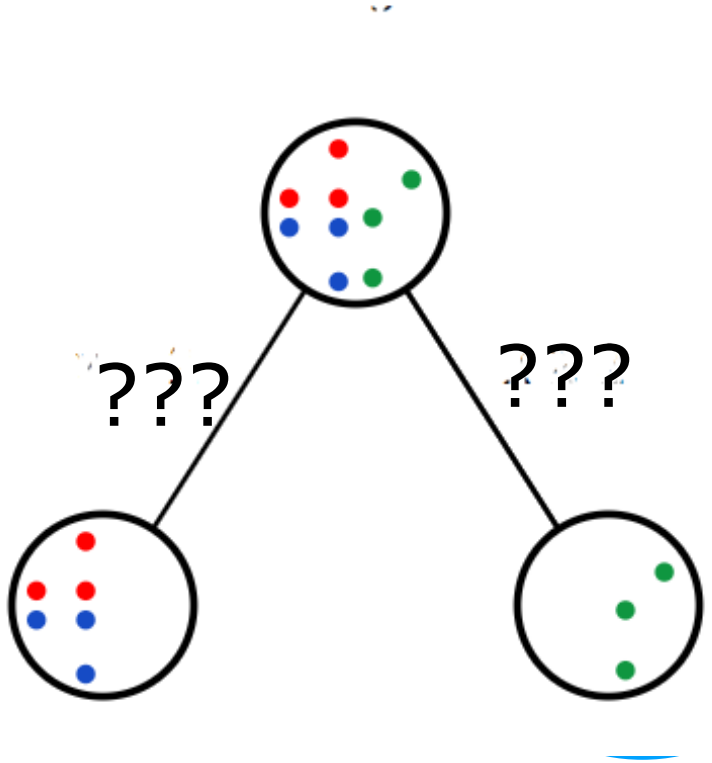
DECISION TREE!

PICKING THE ROOOOOOOOT NODE

Back to the problem of determining our root decision node. Now that we have a way to evaluate splits, all we have to do to is find the best split possible!

For the sake of simplicity, we're just going to **try every possible split** and use the best one (the one with the highest Gini Gain).

This is not the fastest way to find the best split, but it is the easiest to understand.



DECISION TREE!

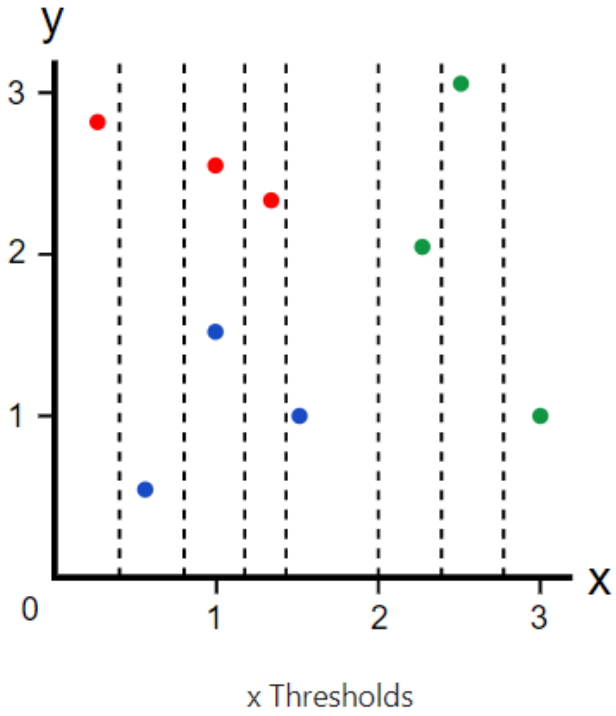
PICKING THE ROOOOOOOOT NODE

Trying every split means trying

Every feature (*x* or *y*).

All “unique” thresholds.

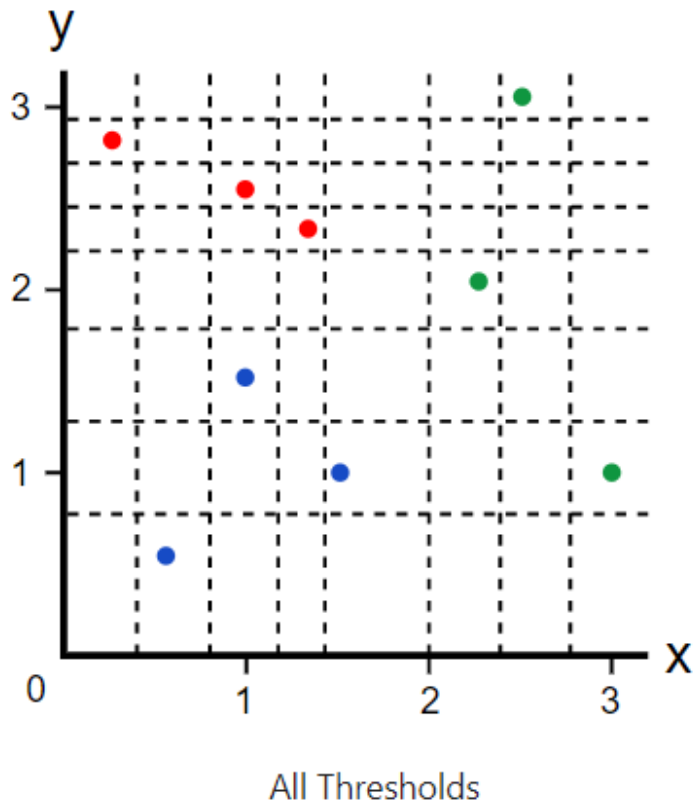
We only need to try thresholds that produce different splits.



DECISION TREE!

PICKING THE ROOOOOOOOT NODE

Split	Left Branch	Right Branch	Gini Gain
$x = 0.4$	●	● ● ● ● ● ●	0.083
$x = 0.8$	● ●	● ● ● ● ● ●	0.048
$x = 1.1$	● ● ● ●	● ● ● ●	0.133
$x = 1.3$	● ● ● ●	● ● ●	0.233
$x = 2$	● ● ● ● ●	● ●	0.333
$x = 2.4$	● ● ● ● ● ●	● ●	0.191
$x = 2.8$	● ● ● ● ● ● ●	●	0.083
$y = 0.8$	●	● ● ● ● ● ●	0.083
$y = 1.2$	● ● ●	● ● ● ● ●	0.111
$y = 1.8$	● ● ● ●	● ● ● ●	0.233
$y = 2.1$	● ● ● ●	● ● ●	0.233
$y = 2.4$	● ● ● ● ●	● ●	0.111
$y = 2.7$	● ● ● ● ● ●	● ●	0.048
$y = 2.9$	● ● ● ● ● ● ●	●	0.083

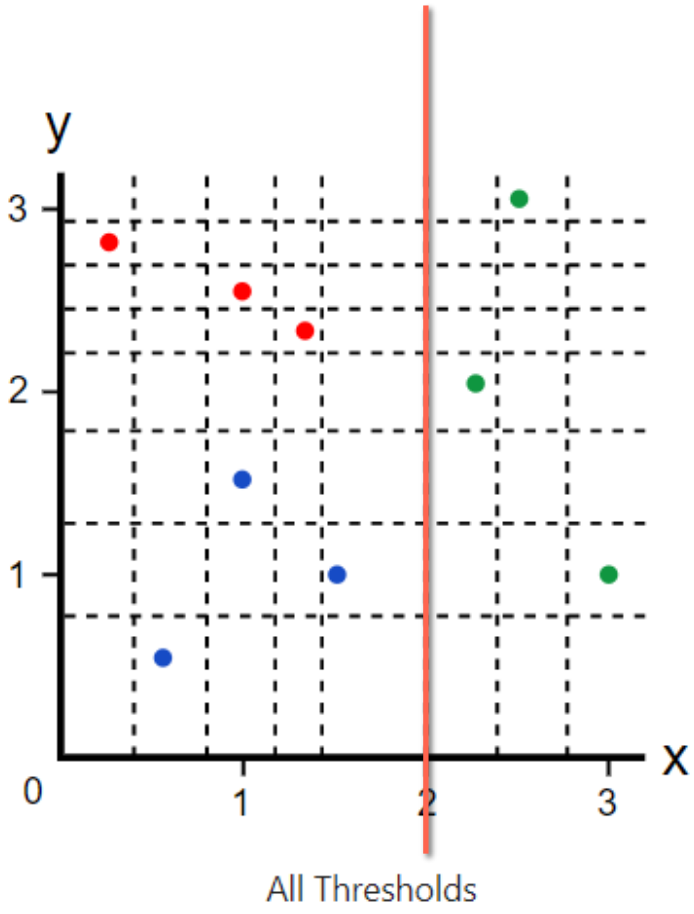


All Thresholds

DECISION TREE!

PICKING THE ROOOOOOOOT NODE

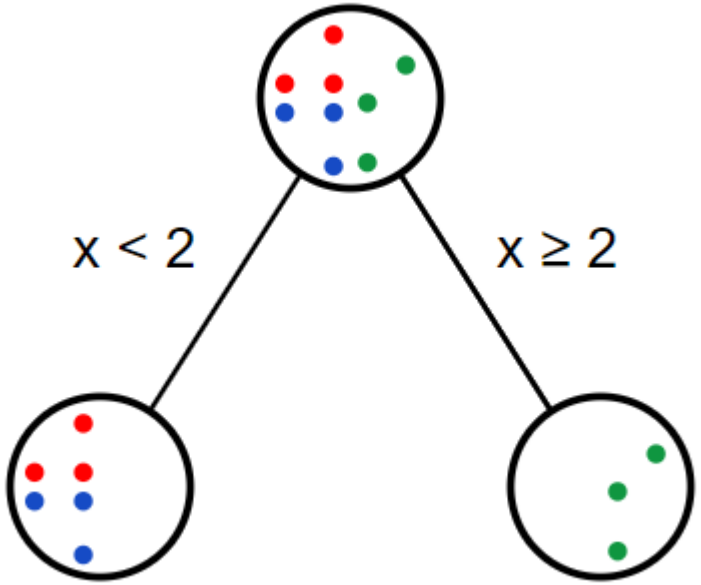
Split	Left Branch	Right Branch	Gini Gain
$x = 0.4$	●	● ● ● ● ● ●	0.083
$x = 0.8$	● ●	● ● ● ● ● ●	0.048
$x = 1.1$	● ● ● ●	● ● ● ●	0.133
$x = 1.3$	● ● ● ●	● ● ●	0.233
$x = 2$	● ● ● ● ●	● ●	0.333
$x = 2.4$	● ● ● ● ● ●	● ●	0.191
$x = 2.8$	● ● ● ● ● ● ●	●	0.083
$y = 0.8$	●	● ● ● ● ● ●	0.083
$y = 1.2$	● ● ●	● ● ● ● ●	0.111
$y = 1.8$	● ● ● ●	● ● ● ●	0.233
$y = 2.1$	● ● ● ●	● ● ●	0.233
$y = 2.4$	● ● ● ● ●	● ●	0.111
$y = 2.7$	● ● ● ● ● ●	● ●	0.048
$y = 2.9$	● ● ● ● ● ● ●	●	0.083



DECISION TREE!

PICKING THE ROOOOOOOOT NODE

This is how our tree looks like right now…



DECISION TREE!

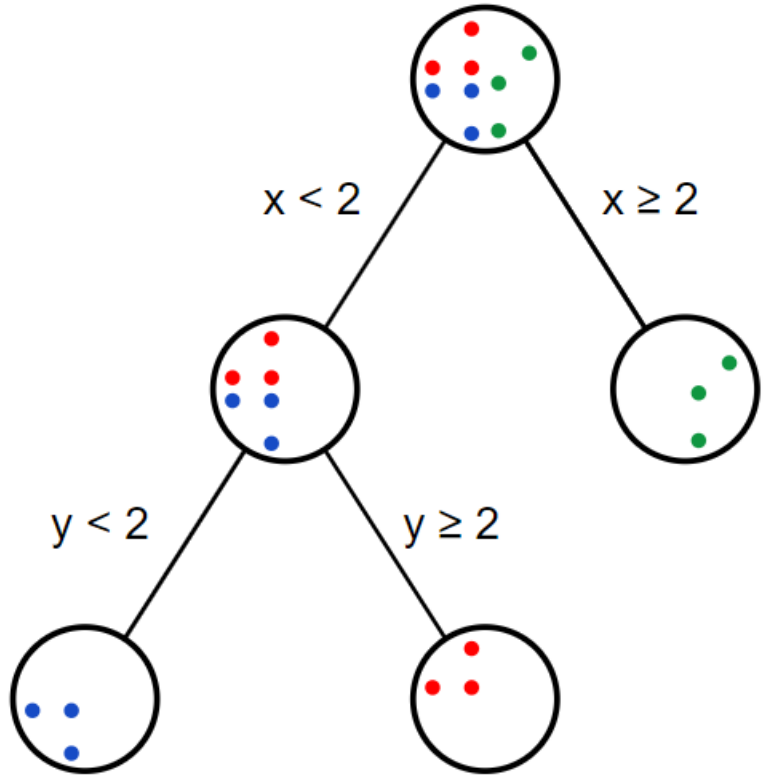
BUILDING THE TREE

Time to make our second decision node.

Let's (arbitrarily) go to the left branch. **We're now only using the datapoints that would take the left branch** (i.e. the datapoints satisfying x < 2), specifically the 3 blues and 3 reds.

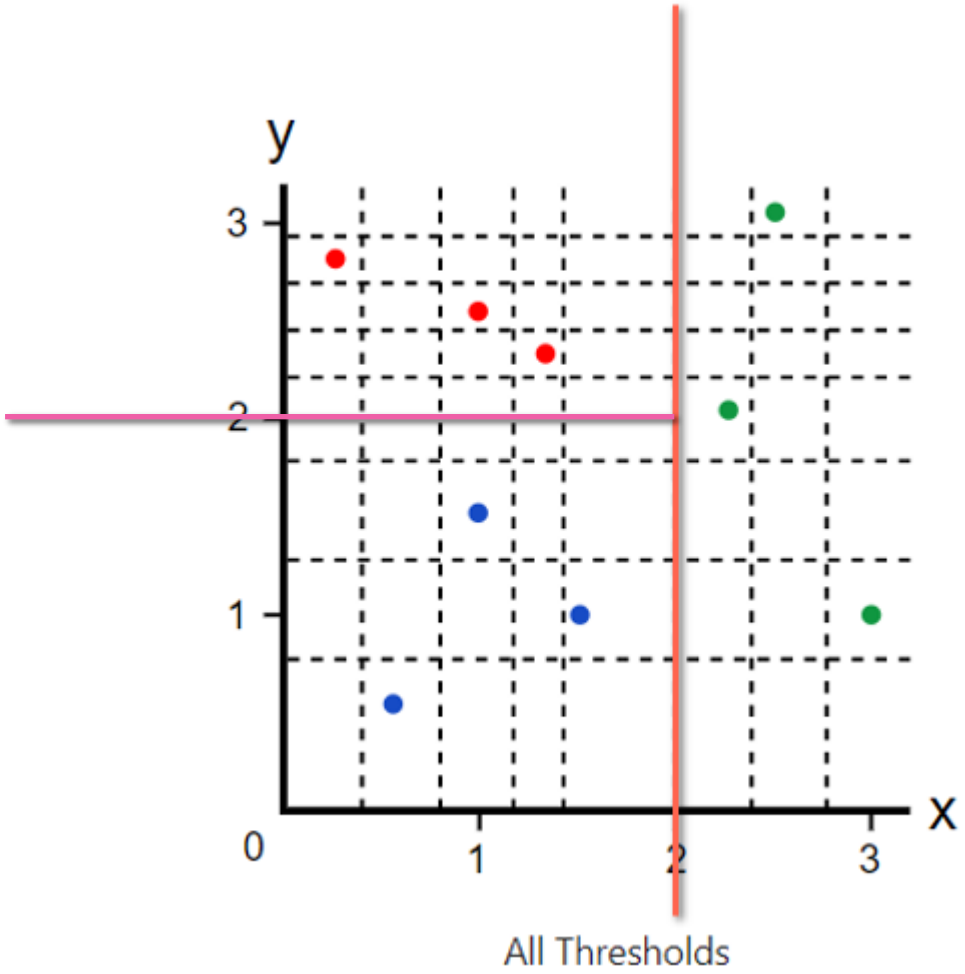
To build our second decision node, **we just do the same thing!** We try every possible split for the 6 datapoints we have and realize that y = 2 is the best split.

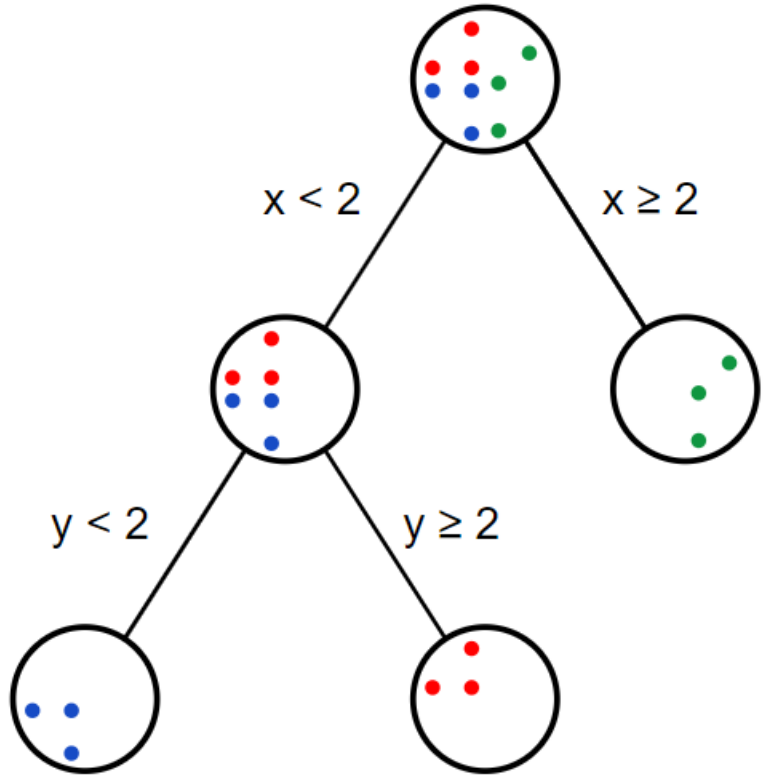
We make that into a decision node and now have this:



DECISION TREE!

BUILDING THE TREE





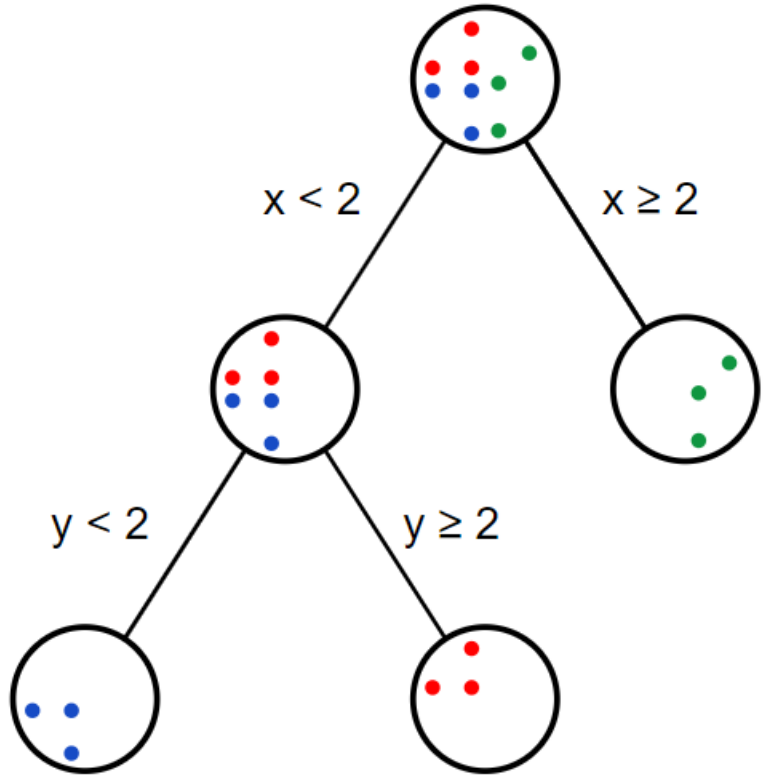
DECISION TREE!

WHEN TO STOP?

Now that we reached the bottom (leaf) of the tree:

Again, we try all the possible splits, but they all are equally good.

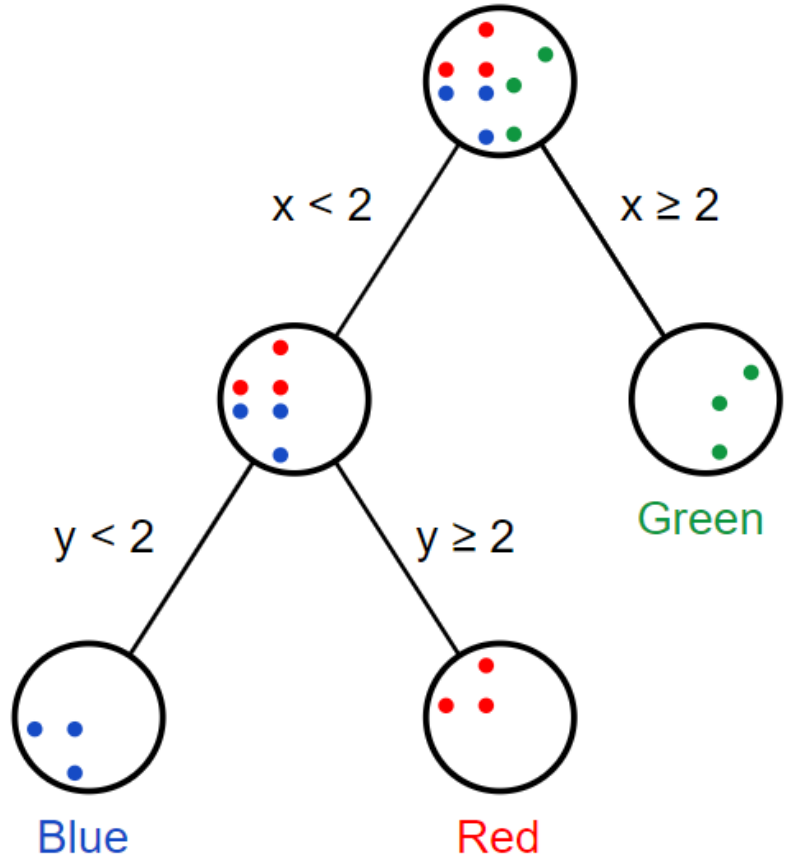
Have a Gini Gain of 0 (the Gini Impurity was already 0 and can’t go any lower).



DECISION TREE!

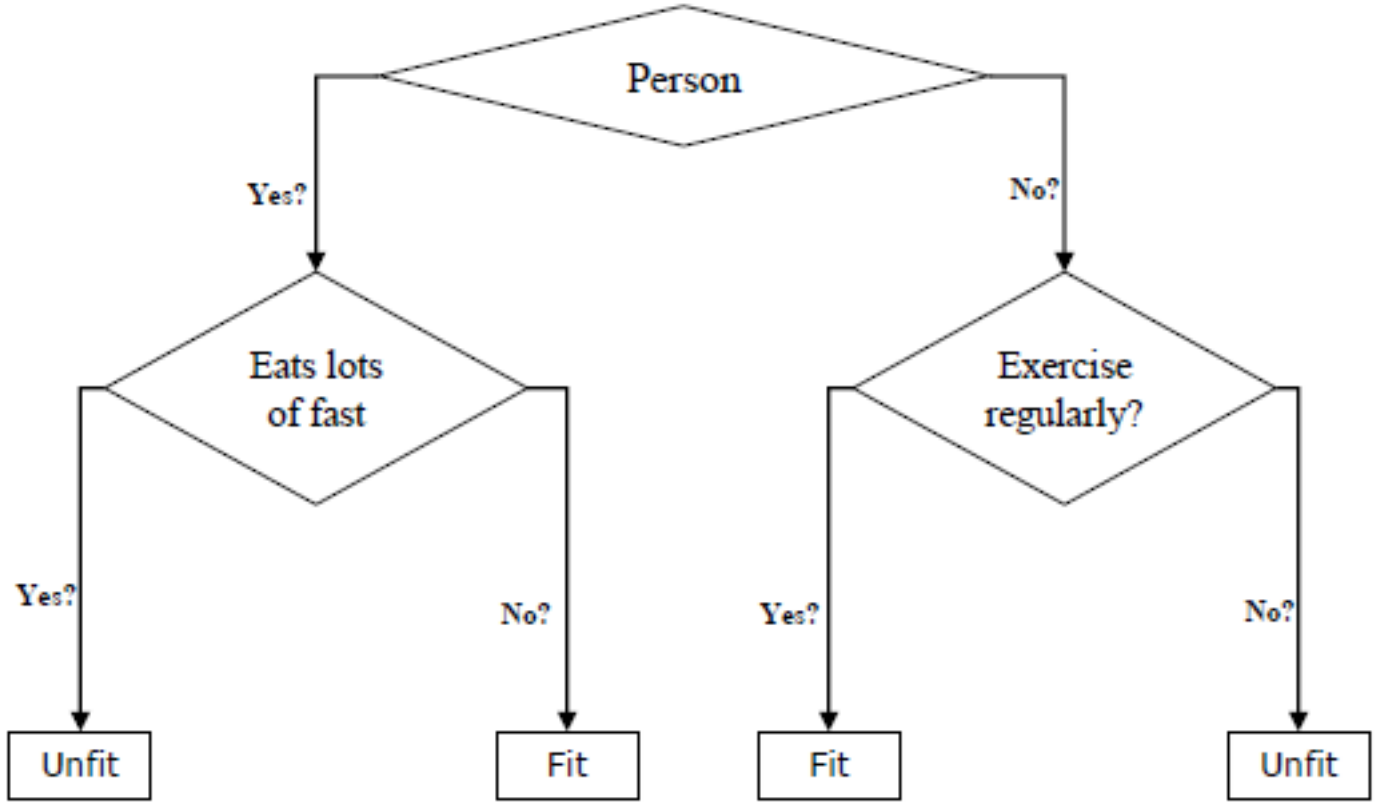
WHEN TO STOP?

Once all possible branches in our decision tree end in leaf nodes, we're done.



SO,
WHAT IS IT?

Decision tree algorithm creates a model that predicts the value of a target variable based on several input variables.



SOUNDS MAGICAL. ANY DOWNSIDES?

Of course! ~~Stop dreaming there is no all perfect algorithm out there~~

The main downsides of Decision Trees are their tendency to over-fit.

- If your tree goes too big (deep), you will overfit pretty fast!

They are also unable to grasp relationships between features.

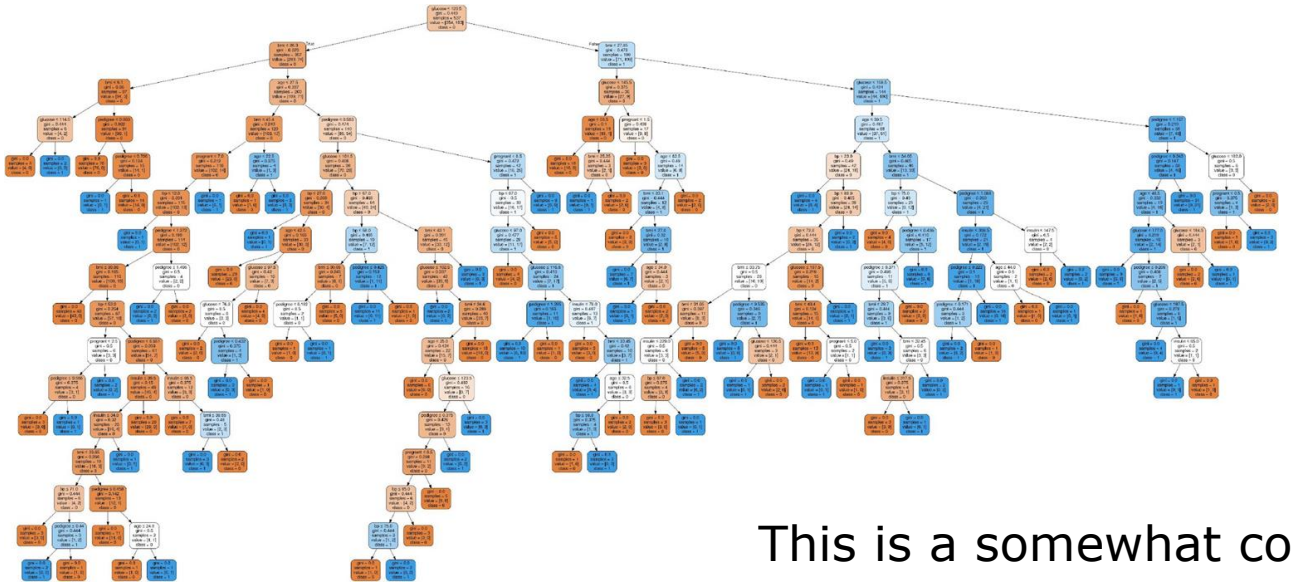
- Cuz it’s a tree like structure so you will lose some relationships between features.

They use greedy learning algorithms, because we usually use greedy traversal when dealing with a tree.

- This is a downside because it is not guaranteed to find the global optimal model.

ANY WAY TO IMPROVE THE ABOVE DOWNSIDES?

Sure! Using them in a Random Forest (We will talk about that in a bit) helps mitigate some of this issues.



This is a somewhat complex tree!

License

This material is originally made by Hongjun Wu for the course CSE416: Introduction to Machine Learning in the Spring 2020 quarter taught by Dr. Valentina Staneva, at University of Washington Paul G. Allen School of Computer Science and Engineering.

It was originally made for educational purpose, in a section taught by teaching assistants to help students explore material in more depth.

Any other materials used are cited in the Credits section.

This material is licensed under the Creative Commons License.

Anyone, especially other educators and students, are welcomed and strongly encouraged to study and use this material.