Tree Methods

Let’s learn something!

Python and Spark

* A very powerful group of algorithms falls under the “Tree Methods” title.
* We’ll discuss decision trees, random forests, and gradient boosted trees!

Tree Methods

- Imagine that I play Tennis every Saturday and I always invite a friend to come with me.
- Sometimes my friend shows up, sometimes not.
- For him it depends on a variety of factors, such as: weather, temperature, humidity, wind etc..
- I start keeping track of these features and whether or not he showed up to play with me.

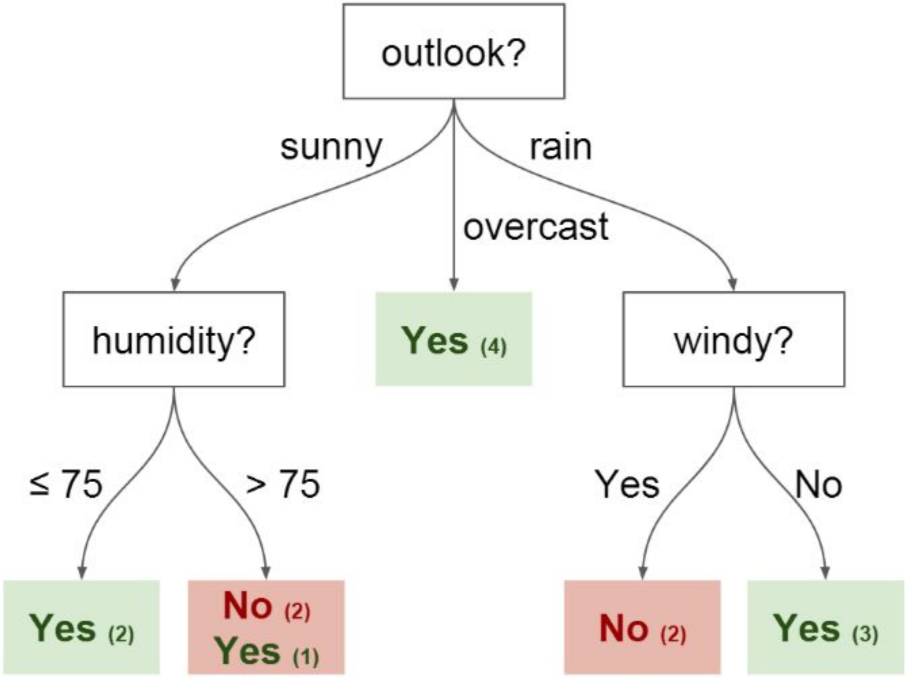
Tree Methods

Temperature	Outlook	Humidity	Windy	Played?
Mild	Sunny	80	No	Yes
Hot	Sunny	75	Yes	**No**
Hot	Overcast	77	No	Yes
Cool	Rain	70	No	Yes
Cool	Overcast	72	Yes	Yes
Mild	Sunny	77	No	**No**
Cool	Sunny	70	No	Yes
Mild	Rain	69	No	Yes
Mild	Sunny	65	Yes	Yes
Mild	Overcast	77	Yes	Yes
Hot	Overcast	74	No	Yes
Mild	Rain	77	Yes	**No**
Cool	Rain	73	Yes	**No**
Mild	Rain	78	No	Yes

Tree Methods

I want to use this data to predict whether or not he will show up to play.

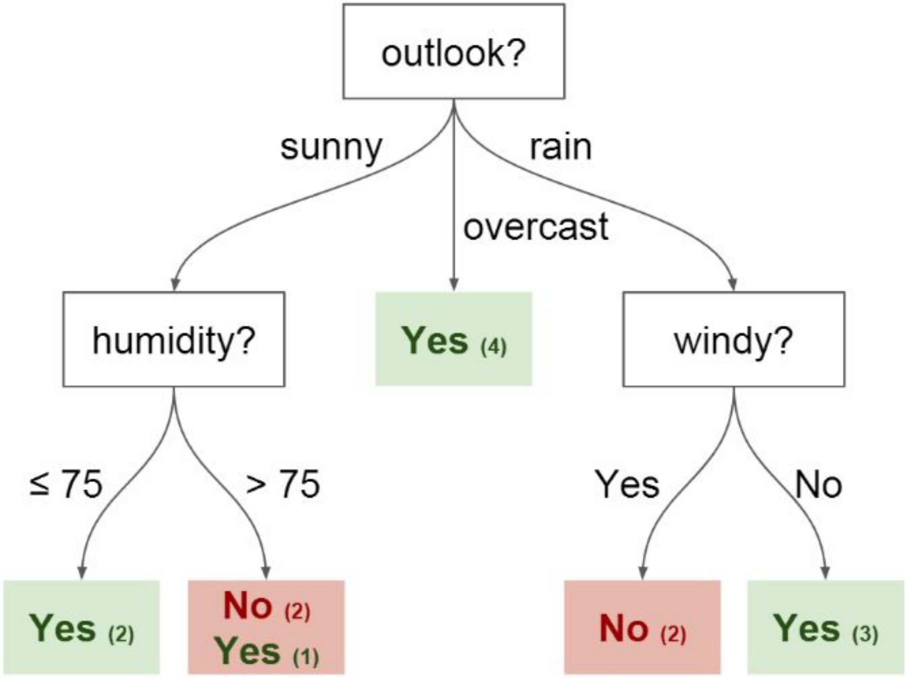
An intuitive way to do this is through a Decision Tree



Tree Methods

In this tree we have:

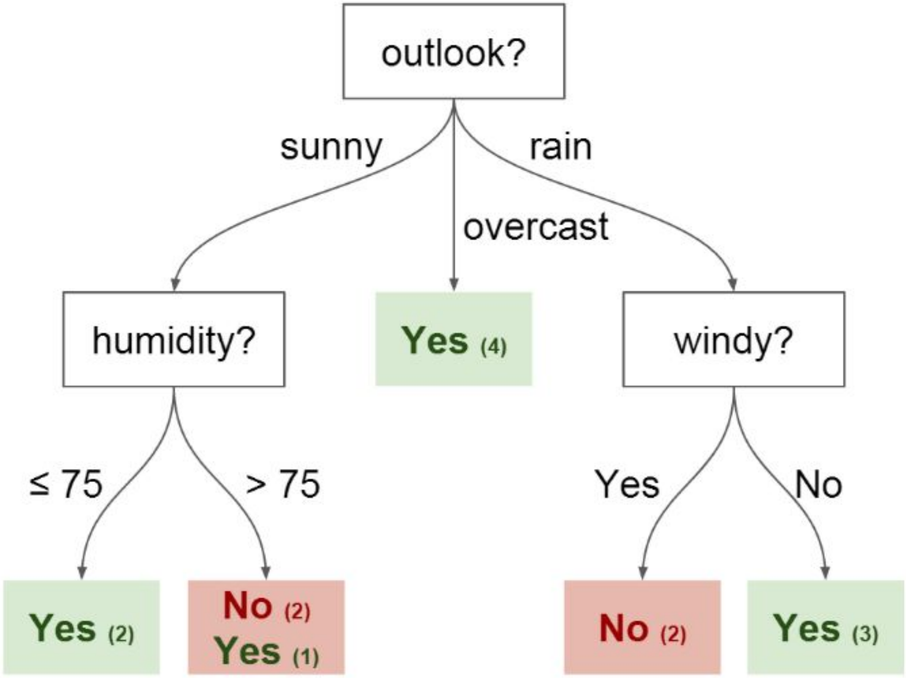
- Nodes
  - Split for the value of a certain attribute
- Edges
  - Outcome of a split to next node



Tree Methods

In this tree we have:

- Root
  - The node that performs the first split
- Leaves
  - Terminal nodes that predict the outcome



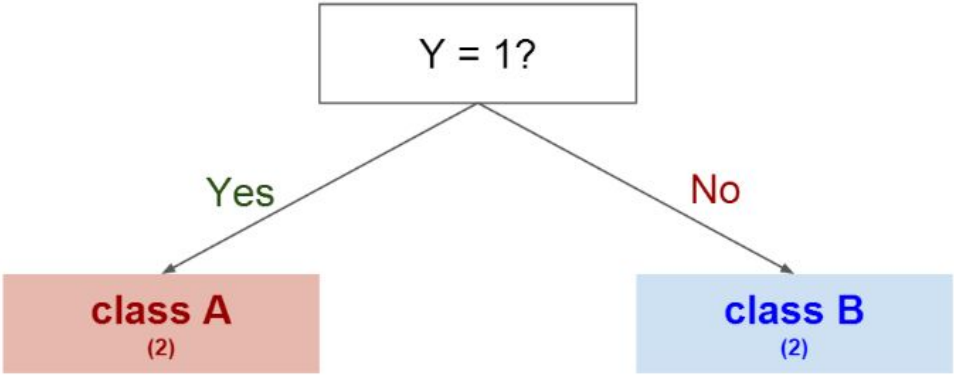
Intuition Behind Splits

Imaginary Data with 3 features (X,Y, and Z) with two possible classes.

X	Y	Z	Class
1	1	1	A
1	1	0	A
0	0	1	B
1	0	0	B

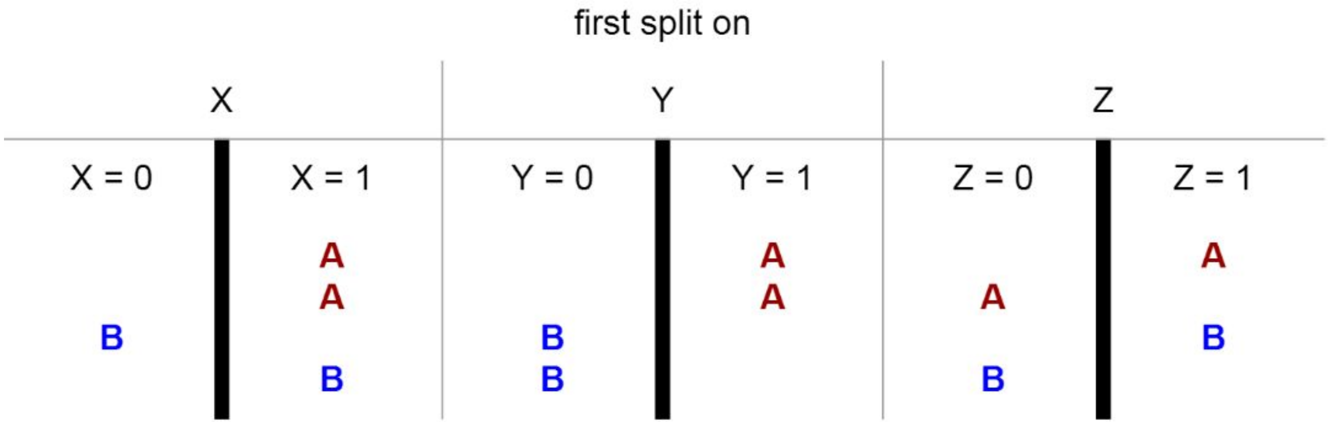
Intuition Behind Splits

Splitting on Y gives us a clear separation between classes



Intuition Behind Splits

We could have also tried splitting on other features first:



Intuition Behind Splits

Entropy and Information Gain are the Mathematical Methods of choosing the best split. Refer to reading assignment.

$$Entropy:$$

$$H(S) = -\sum_i p_i(S)\, log_2\, p_i(S)$$

$$Information\ Gain:$$

$$IG(S, A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{S} H(S_v)$$

Random Forests

To improve performance, we can use many trees with a random sample of features chosen as the split.

- A new random sample of features is chosen for **every single tree at every single split**.
- Works for both classification and regression tasks!

Random Forests

What's the point?

- Suppose there is **one very strong feature** in the data set. Most of the trees will use that feature as the top split, resulting in an ensemble of similar trees that are **highly correlated**.

Random Forests

What's the point?

- Averaging highly correlated quantities does not significantly reduce variance.
- By randomly leaving out candidate features from each split, **Random Forests "decorrelates" the trees**, such that the averaging process can reduce the variance of the resulting model.

Python and Spark

- Finally, let’s discuss Gradient Boosted Trees!

Python and Spark

- Gradient boosting involves three elements:
  - A loss function to be optimized.
  - A weak learner to make predictions.
  - An additive model to add weak learners to minimize the loss function.

Python and Spark

- Loss Function
  - A loss function in basic terms is the function/equation you will use to determine how “far off” your predictions are.

Python and Spark

- Loss Function
  - For example, regression may use a squared error and classification may use logarithmic loss.
  - We won't have to deal with this directly using Spark.

Python and Spark

- Weak Learner
  - Decision trees are used as the weak learner in gradient boosting.
  - It is common to constrain the weak learners: such as a maximum number of layers, nodes, splits or leaf nodes.

Python and Spark

- Additive Model
  - Trees are added one at a time, and existing trees in the model are not changed.
  - A gradient descent procedure is used to minimize the loss when adding trees.

Python and Spark

- So what is the most intuitive way to think about all this if Spark does this all for us?
- Here is a nice way to think about it in 3 “easy” steps

Python and Spark

1. Train a weak model **m** using data samples drawn according to some weight distribution

Python and Spark

2. Increase the weight of samples that are misclassified by model m, and decrease the weight of samples that are classified correctly by model m

Python and Spark

3. Train next weak model using samples drawn according to the updated weight distribution.

Python and Spark

- In this way, the algorithm always trains models using data samples that are "difficult" to learn in previous rounds, which results an ensemble of models that are good at learning different "parts" of training data.

Python and Spark

- Basically “boosting” weights of samples that were difficult to get correct.

Python and Spark

- The real details of gradient boosting lies in the mathematics, which you may or may not be ready for depending on your background.
- The full details are at the end of Chapter 8 of ISLR!

Python and Spark

- Spark handles all of this “under the hood” for you, so you can use the defaults if you want, or dive into ISLR and begin to play around with the parameters!
- Let’s continue!