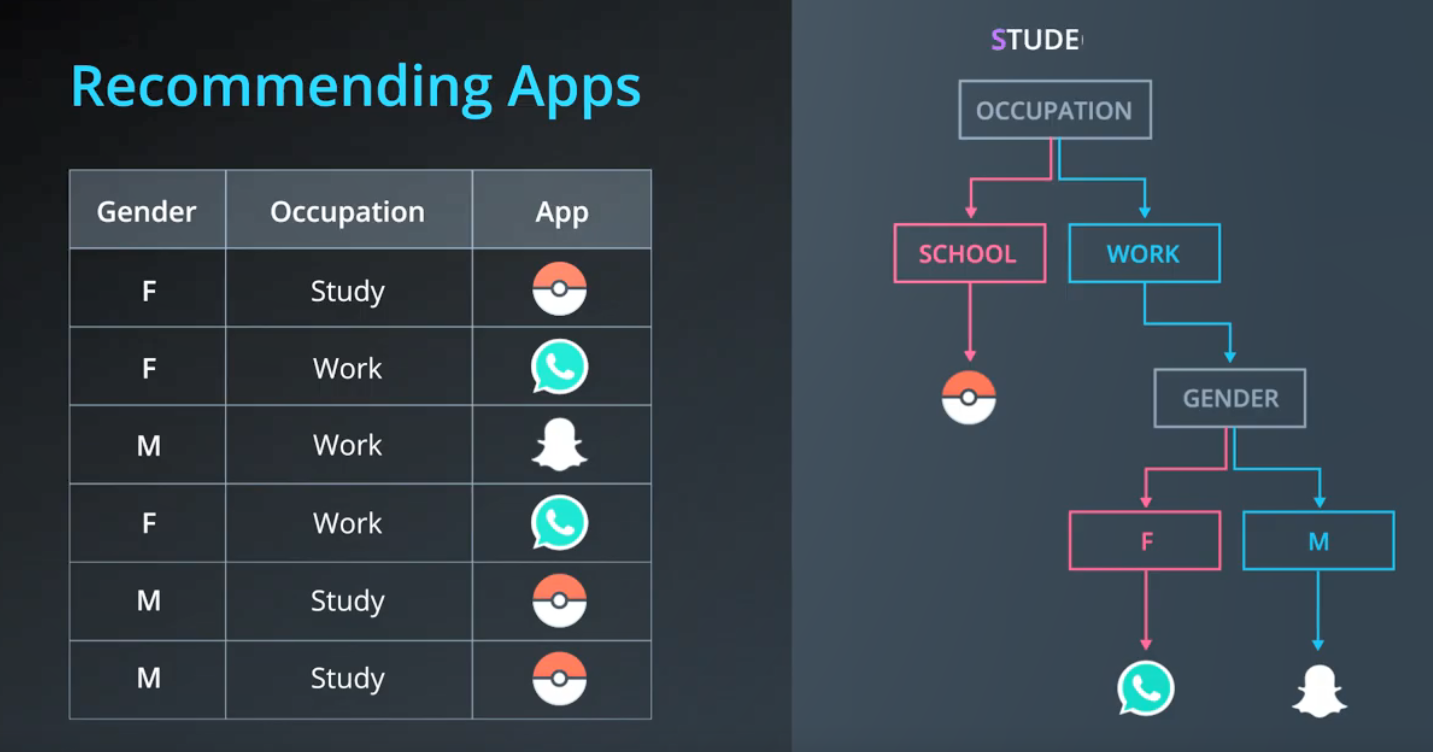
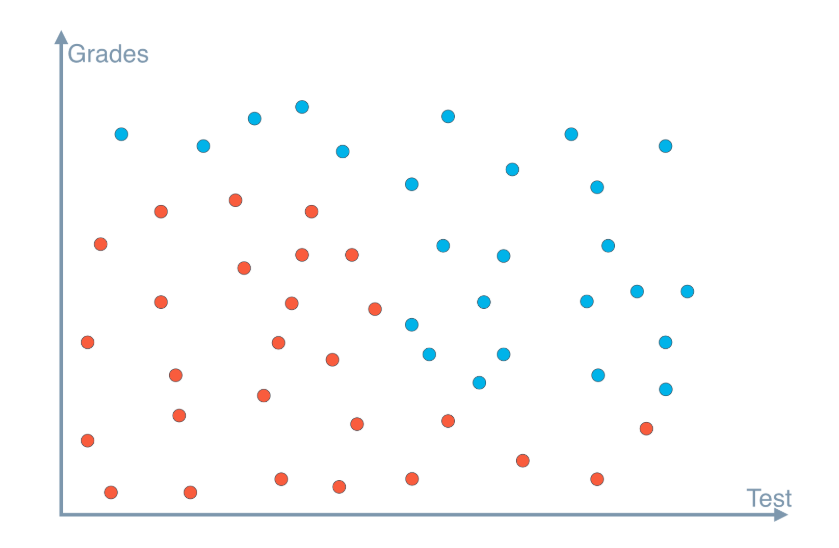
**Decision Trees**

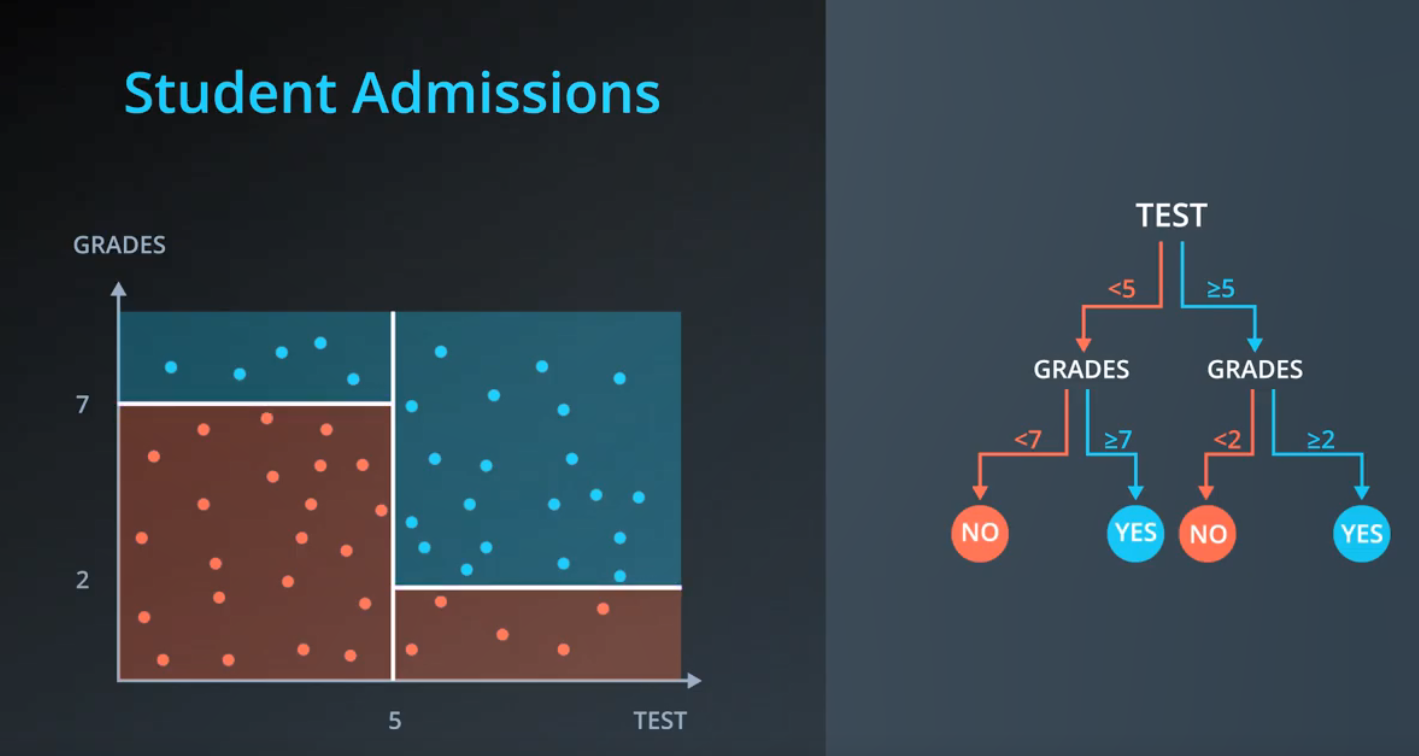
Decision trees will ask our data a question and then use the answer to ask a new question and so on and so forth. Our model will learn more and more about the data throught his way until it is comfortable making a prediction.

If we have the following table we could make predictions on what app we should recommend someone. We can see that occupation has a stronger affect on what app we would take so we make the decision to look at that first and then take gender into account only if you “work” instead of “study”. We can do this through human eye but teaching a computer to do this is a different matter.



In above example we have categorical features. We can also make decision trees based off of continuous features, such as the example below where we have the score in a test and grades and blue and red points saying whether they are accepted or rejected. Between Grades and Tests, which one determines student acceptance better? This question also equates to, what would cut the data better, a vertical line or a horizontal line? With the answer being a vertical line.

This means the better feature to separate this data is Test with a node of 5. See below to how we break this down through several decision tree questions:



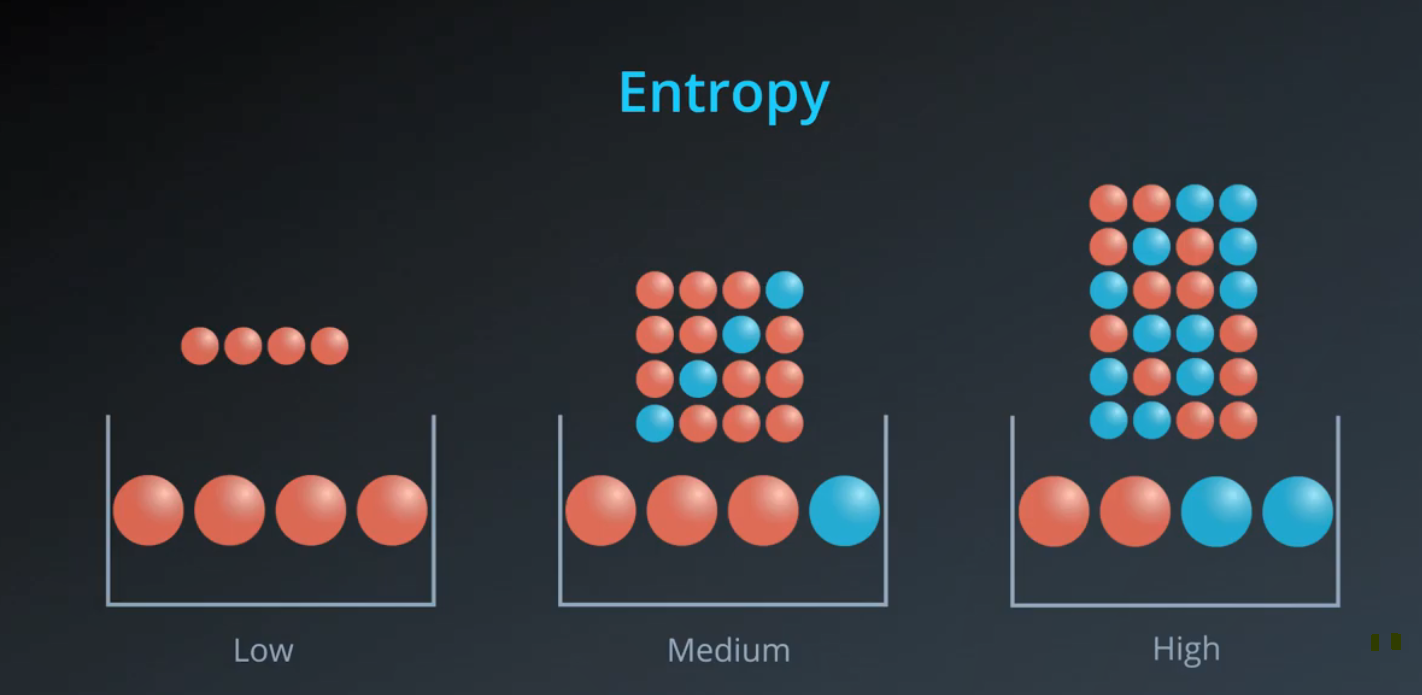
In this way we create questions that have a threshold that needs to be breached rather then a direct yes/no question.

**Entropy**

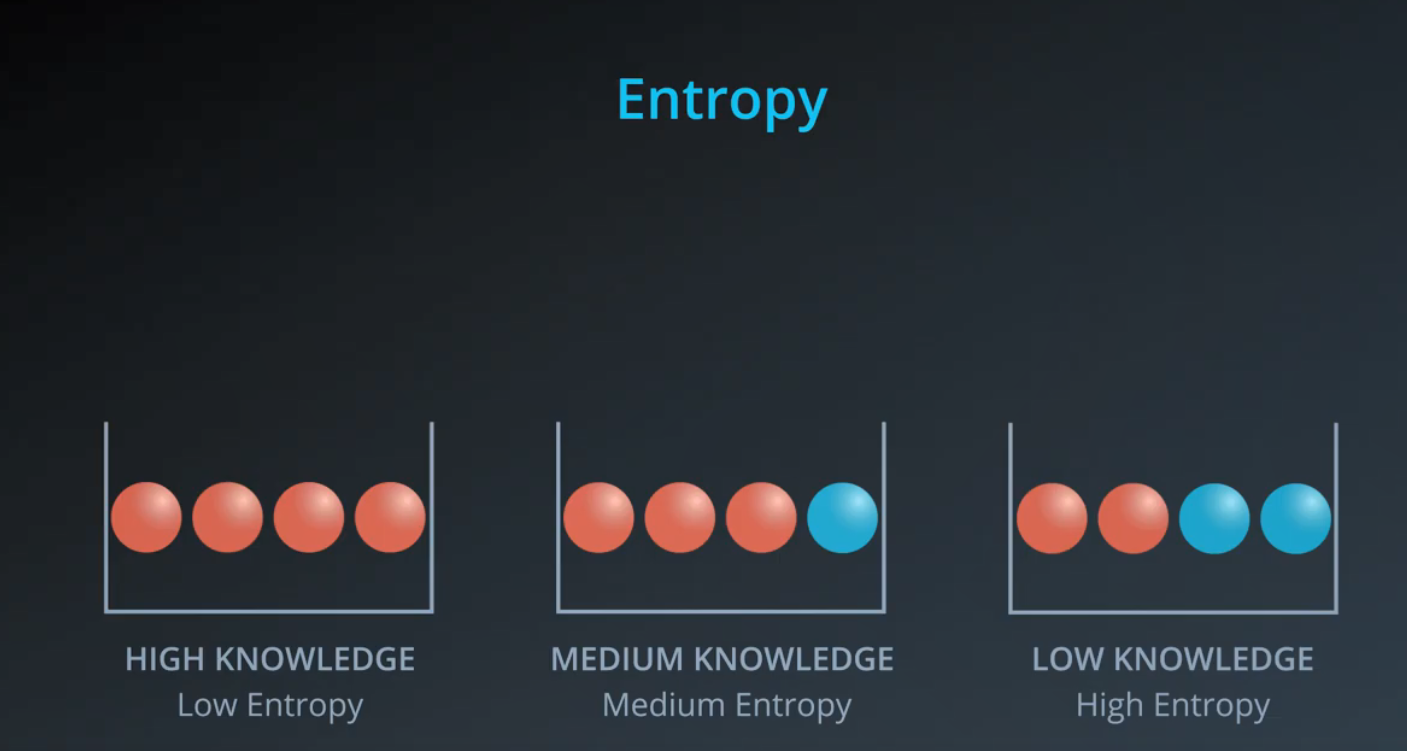
Entropy comes from physics. It defines how much freedom a particle has to move around, a good way to understand it is to use the three states of water. Ice would have a low entropy as it can’t move, water a medium as it can move a bit and steam a high entropy as it has a lot of room to move around:



In Probability, we can say that the more rigid or consistant a set is the lower the entropy it is. It’s not an exact definition but the more rigid a set is the less entropy you will have:

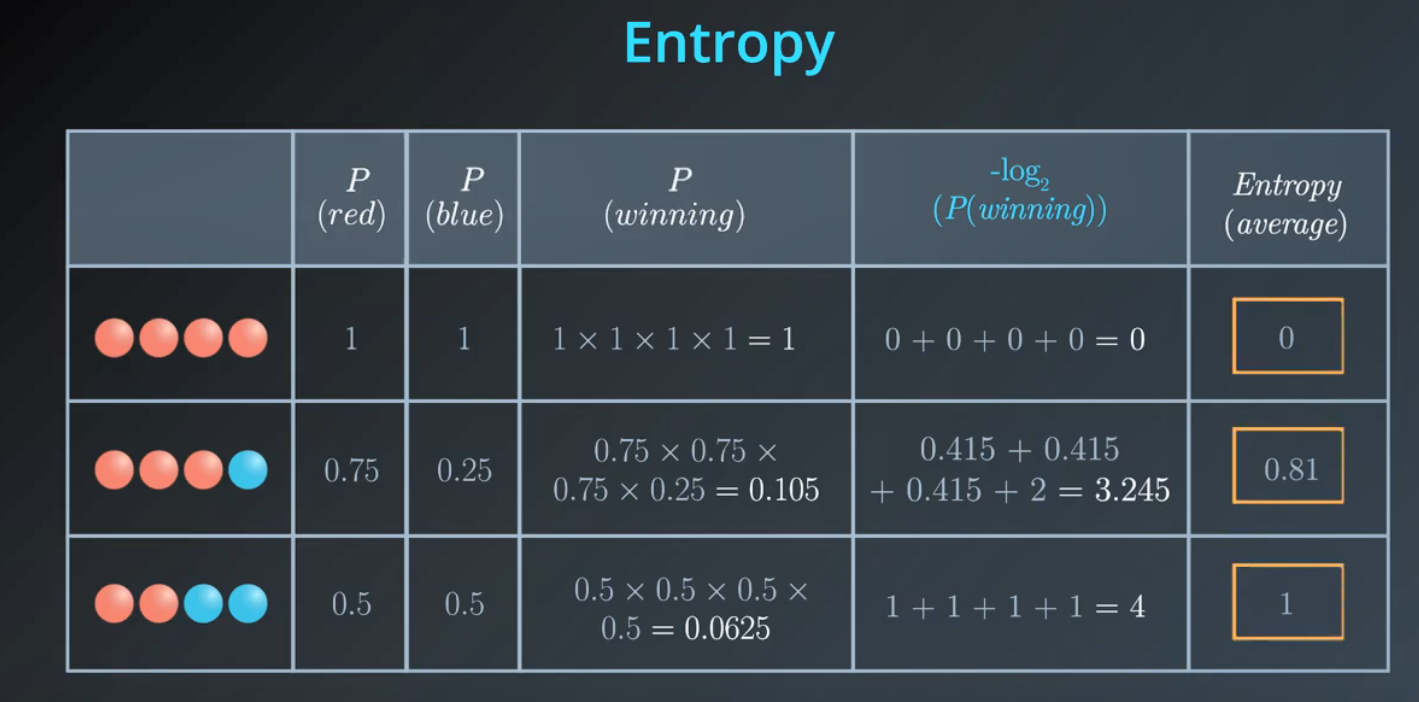


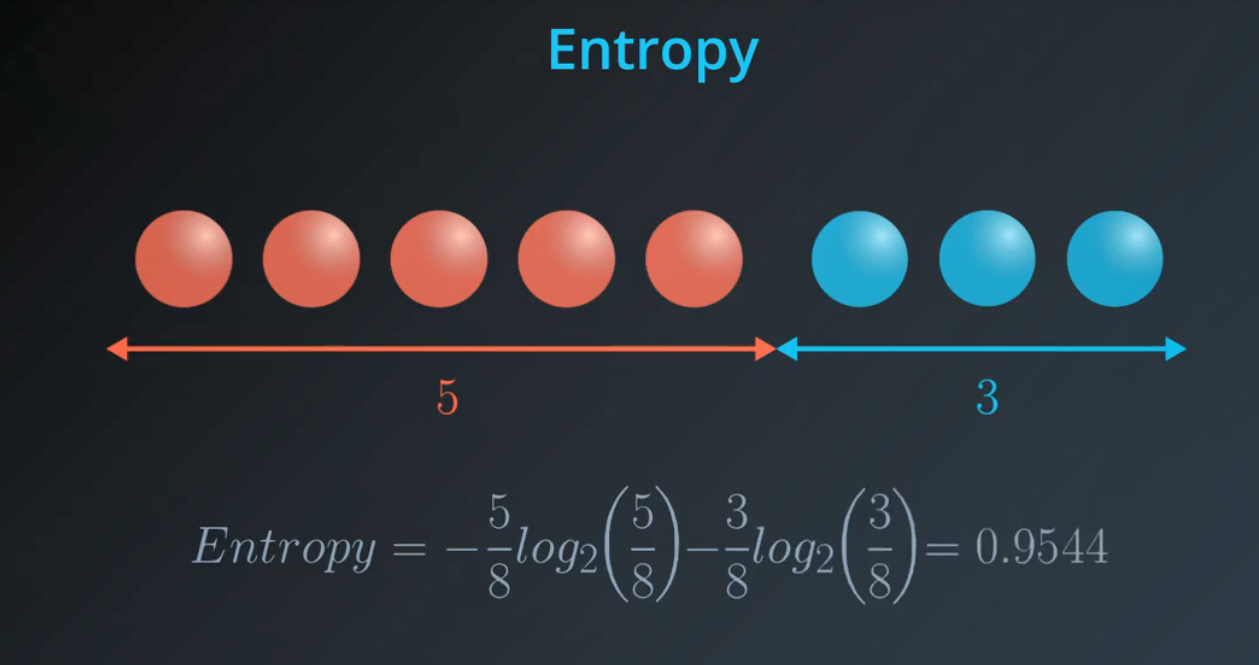
Entropy and Knowledge tend to be opposite:



**Formula for Entropy**

In normal probability we would use fractions and multiply them together to get the probability of an action happening. With Entropy we want to steer clear of products so we will use log on the products as: log(ab) = log(a) + log(b). This ouwld give us a negative number so we use –log(Probability)





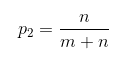
**Multi-class Entropy**

Last time, you saw this equation for entropy for a bucket with m*m* red balls and n*n* blue balls:

[[https://s3.amazonaws.com/video.udacity-data.com/topher/2018/May/5b046ed6_screen-shot-2018-05-22-at-12.25.34-pm/screen-shot-2018-05-22-at-12.25.34-pm.png](https://classroom.udacity.com/nanodegrees/nd009-ent/parts/e2907194-a763-44ca-9a5c-976bdc957119/modules/435676be-33e4-4ec8-877a-e4ee1f1904ca/lessons/6a868a0f-2f09-41af-9133-4783d084c344/concepts/0ddc9466-18ff-44c2-bce8-ecfa89a8ba14)](https://classroom.udacity.com/nanodegrees/nd009-ent/parts/e2907194-a763-44ca-9a5c-976bdc957119/modules/435676be-33e4-4ec8-877a-e4ee1f1904ca/lessons/6a868a0f-2f09-41af-9133-4783d084c344/concepts/0ddc9466-18ff-44c2-bce8-ecfa89a8ba14)

We can state this in terms of probabilities instead for the number of red balls as p\_1*p*1​ and the number of blue balls as p\_2*p*2​:

[[https://s3.amazonaws.com/video.udacity-data.com/topher/2018/May/5b046fae_screen-shot-2018-05-22-at-12.27.22-pm/screen-shot-2018-05-22-at-12.27.22-pm.png](https://classroom.udacity.com/nanodegrees/nd009-ent/parts/e2907194-a763-44ca-9a5c-976bdc957119/modules/435676be-33e4-4ec8-877a-e4ee1f1904ca/lessons/6a868a0f-2f09-41af-9133-4783d084c344/concepts/0ddc9466-18ff-44c2-bce8-ecfa89a8ba14)](https://classroom.udacity.com/nanodegrees/nd009-ent/parts/e2907194-a763-44ca-9a5c-976bdc957119/modules/435676be-33e4-4ec8-877a-e4ee1f1904ca/lessons/6a868a0f-2f09-41af-9133-4783d084c344/concepts/0ddc9466-18ff-44c2-bce8-ecfa89a8ba14)

[[](https://classroom.udacity.com/nanodegrees/nd009-ent/parts/e2907194-a763-44ca-9a5c-976bdc957119/modules/435676be-33e4-4ec8-877a-e4ee1f1904ca/lessons/6a868a0f-2f09-41af-9133-4783d084c344/concepts/0ddc9466-18ff-44c2-bce8-ecfa89a8ba14)](https://classroom.udacity.com/nanodegrees/nd009-ent/parts/e2907194-a763-44ca-9a5c-976bdc957119/modules/435676be-33e4-4ec8-877a-e4ee1f1904ca/lessons/6a868a0f-2f09-41af-9133-4783d084c344/concepts/0ddc9466-18ff-44c2-bce8-ecfa89a8ba14)

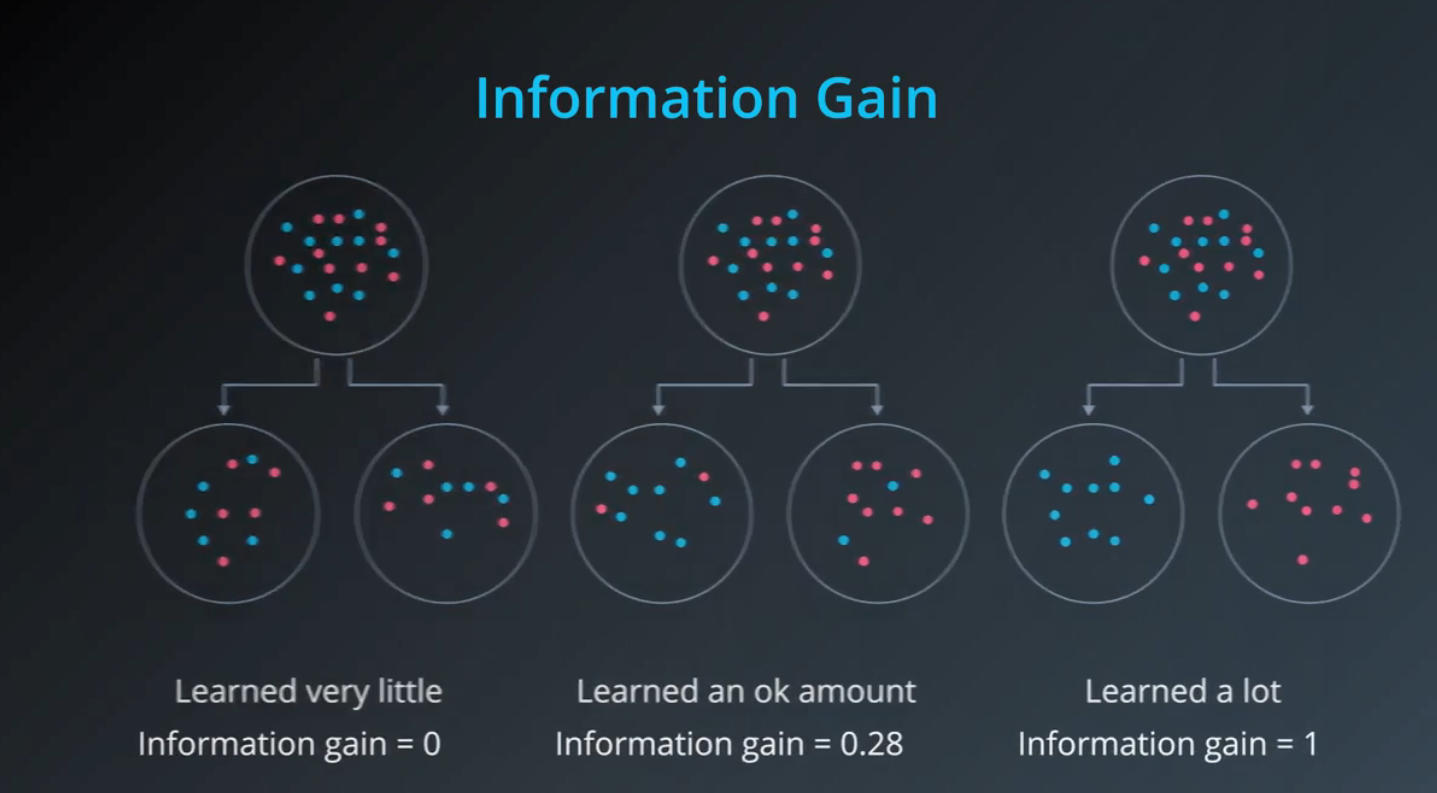
entropy = -p\_1\log\_2(p\_1)-p\_2\log\_2(p\_2)*entropy*=−*p*1​log2​(*p*1​)−*p*2​log2​(*p*2​)

This entropy equation can be extended to the multi-class case, where we have three or more possible values:

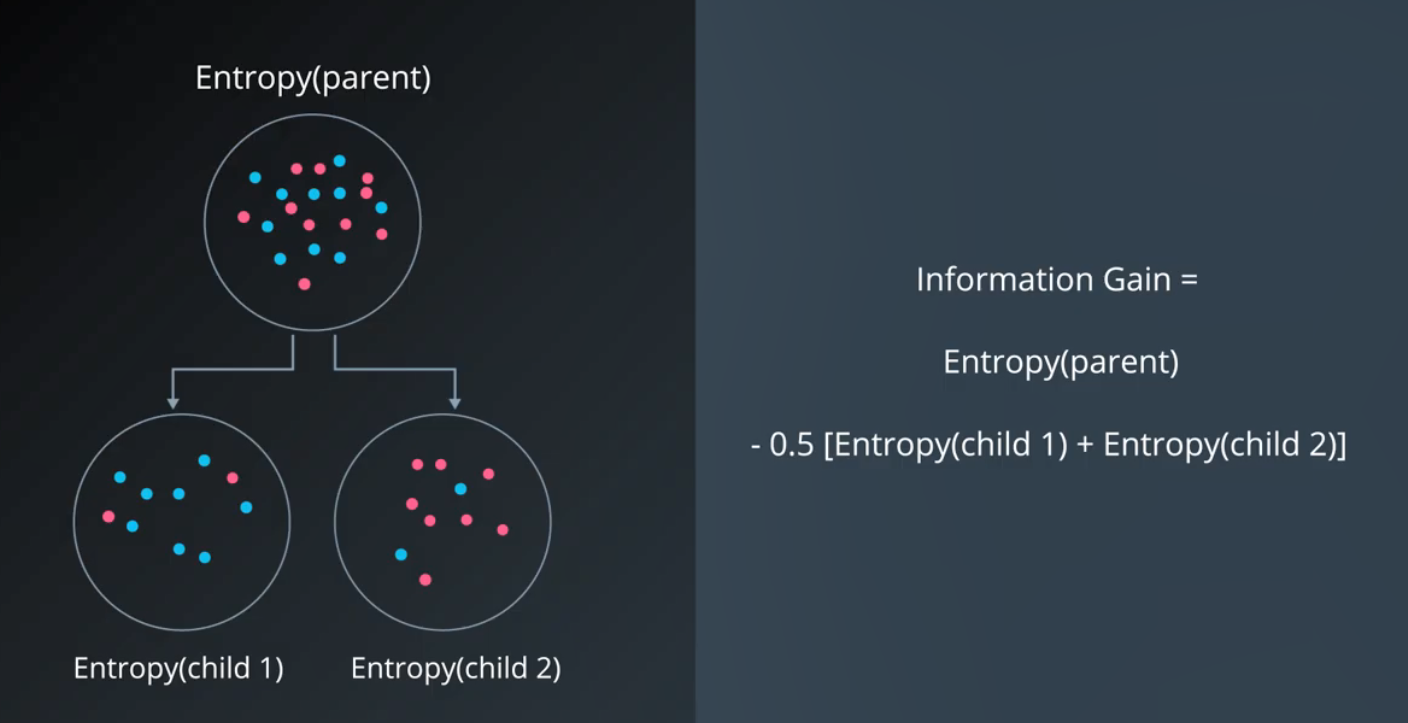
entropy = -p\_1\log\_2(p\_1) - p\_2\log\_2(p\_2) - ... - p\_n\log\_2(p\_n) = -\sum\limits\_{i=1}^n p\_i\log\_2(p\_i)*entropy*=−*p*1​log2​(*p*1​)−*p*2​log2​(*p*2​)−...−*pn*​log2​(*pn*​)=−*i*=1∑*n*​*pi*​log2​(*pi*​)

The minimum value is still 0, when all elements are of the same value. The maximum value is still achieved when the outcome probabilities are the same, but the upper limit increases with the number of different outcomes. (For example, you can verify the maximum entropy is 2 if there are four different possibilities, each with probability 0.25.)

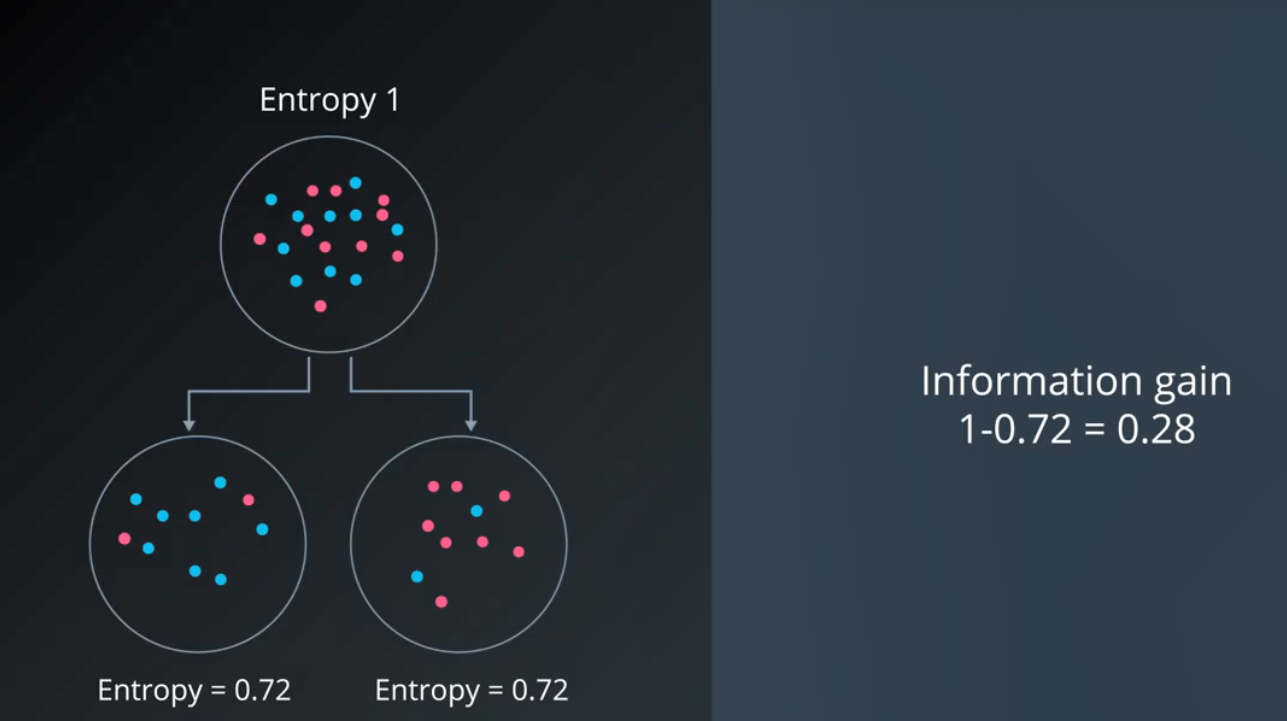
**Learning how to calculate information gained**

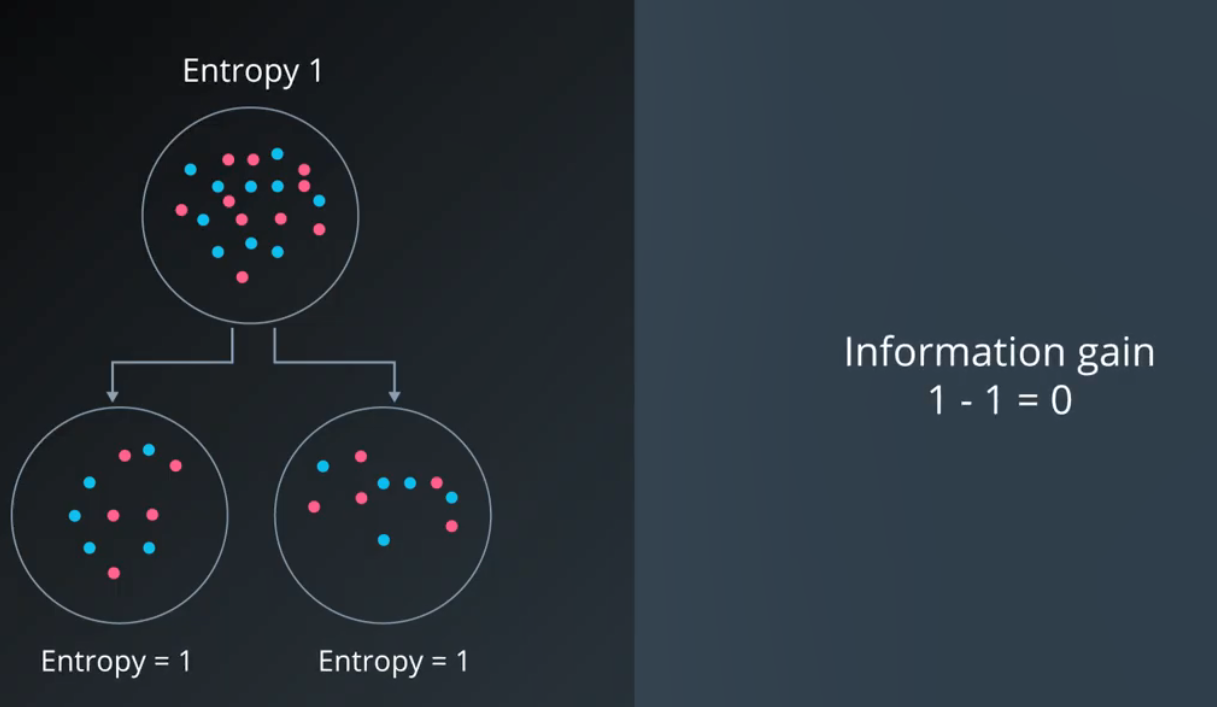


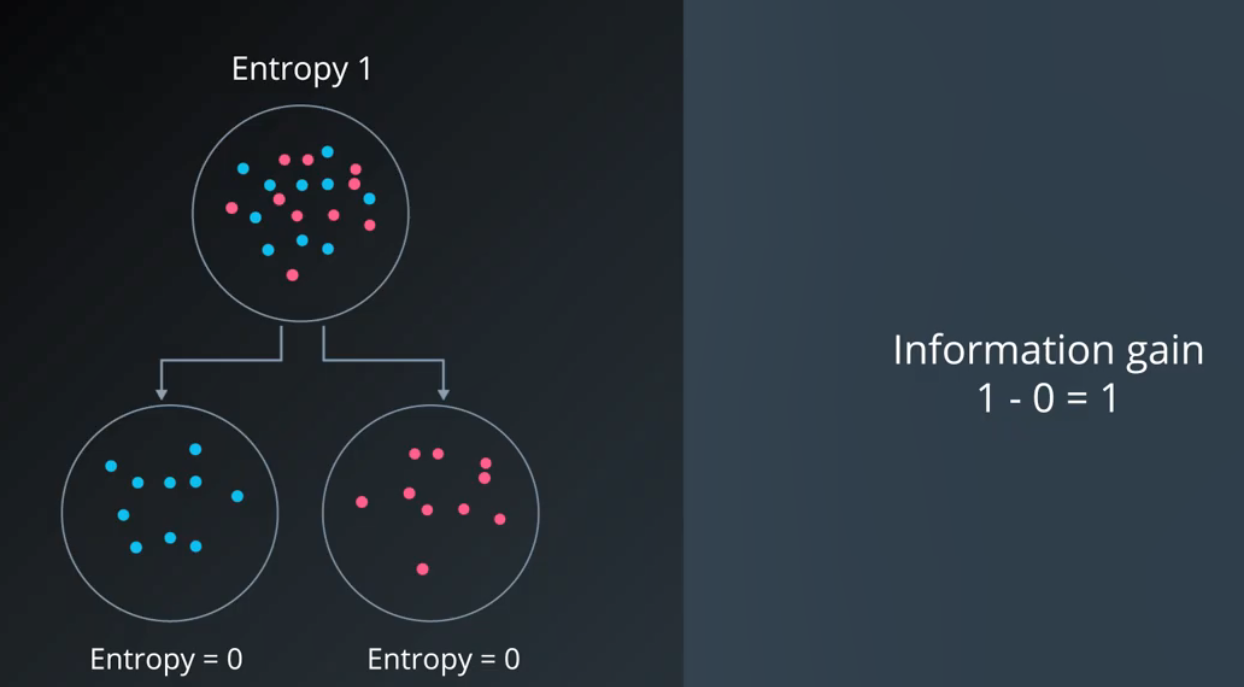
Information gained is the difference between the entropy of the parent and the average entropy of the children:



Entropy and information gained examples:

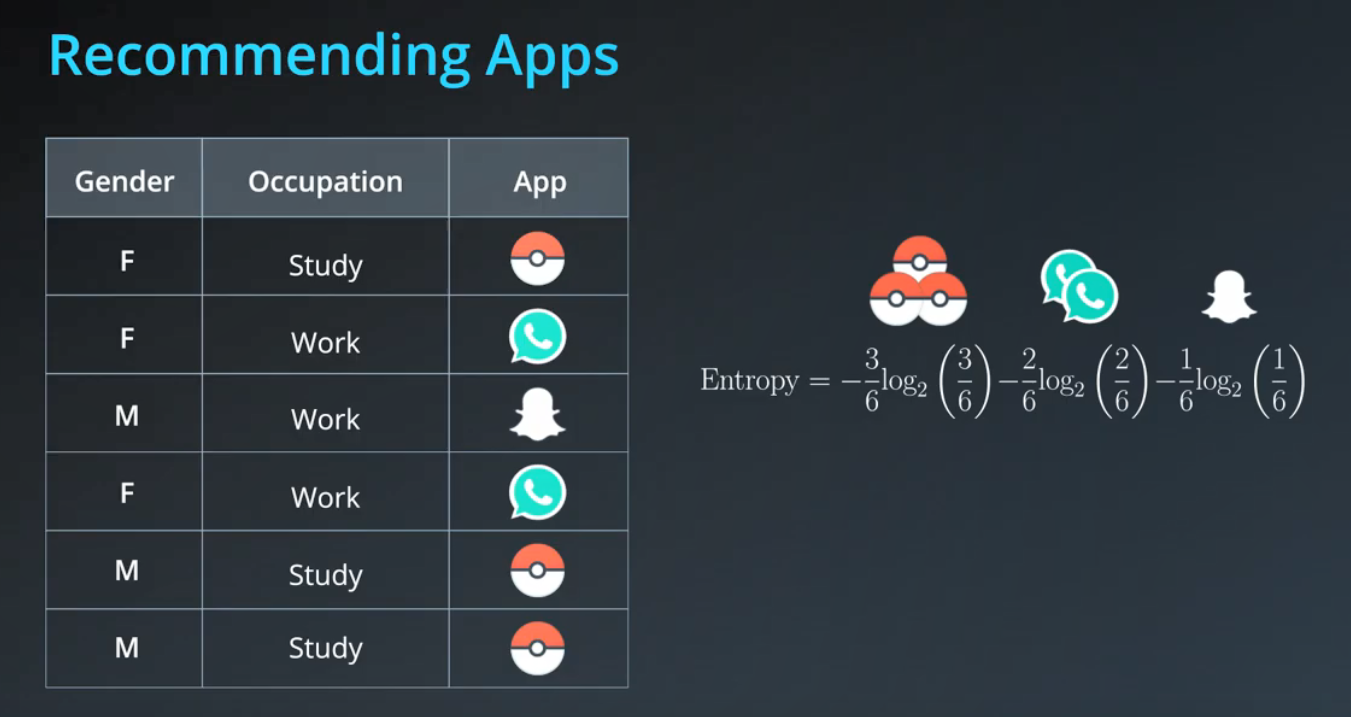




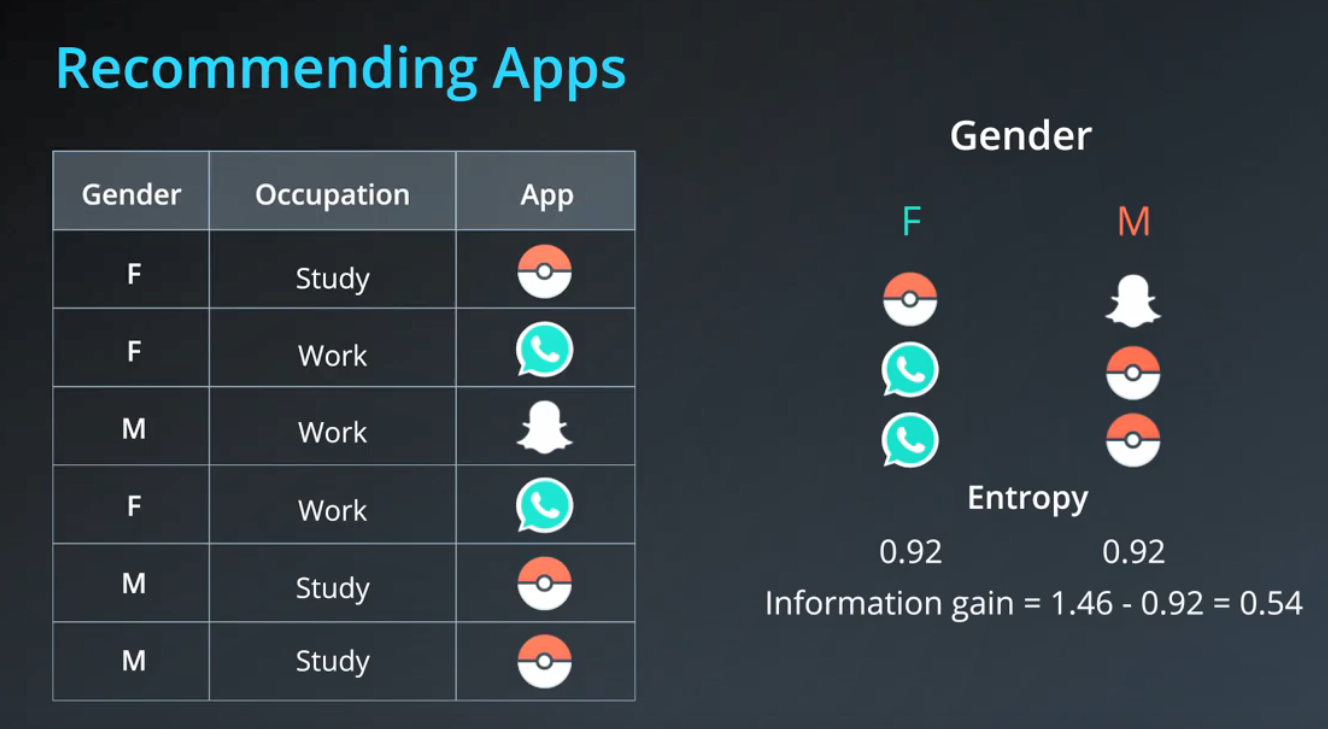


**How To Maximise Information Gain**

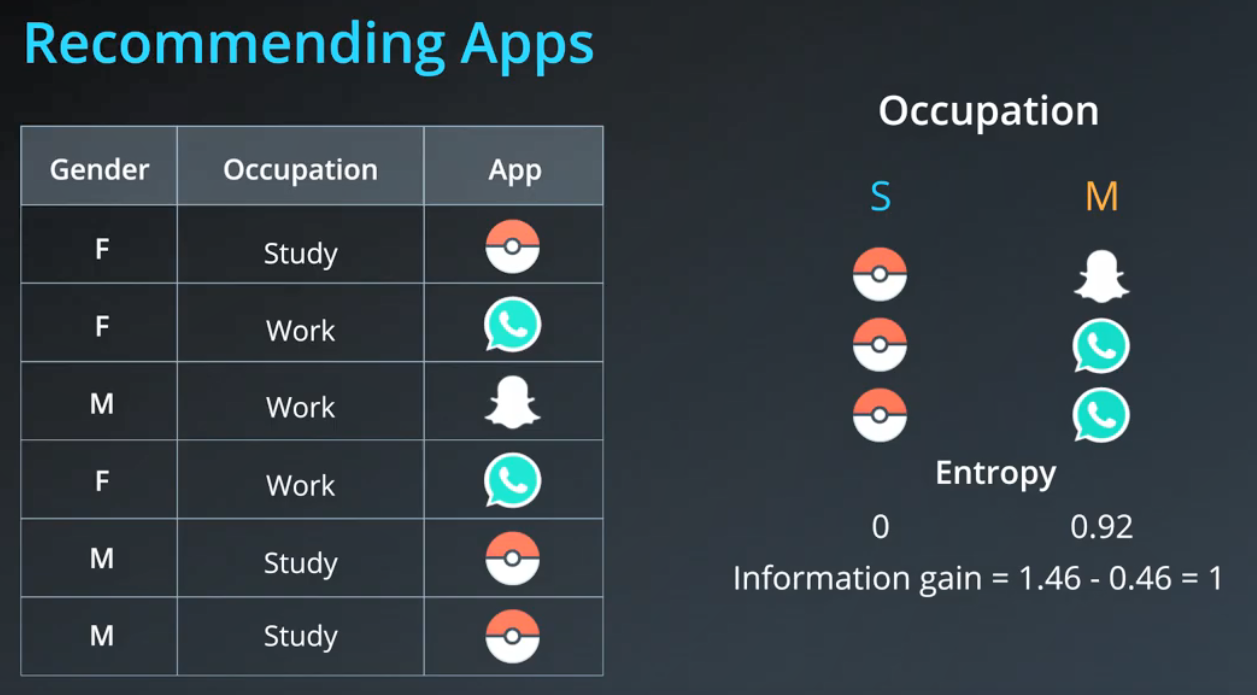
Take our original example where we are trying to recommend apps based off of gender and occupation. If we relate this to entropy we would get the following:



Split by gender:



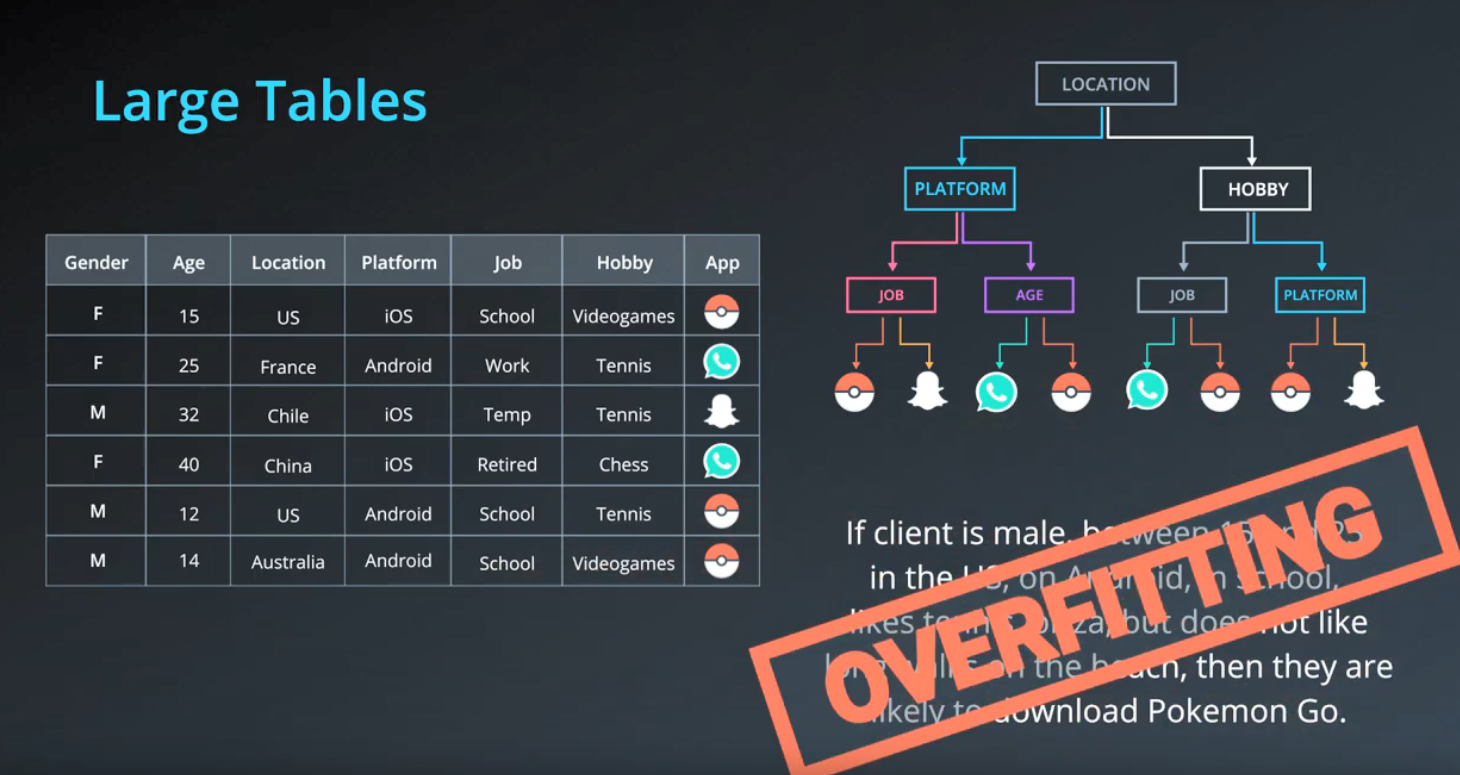
Split by occupation:



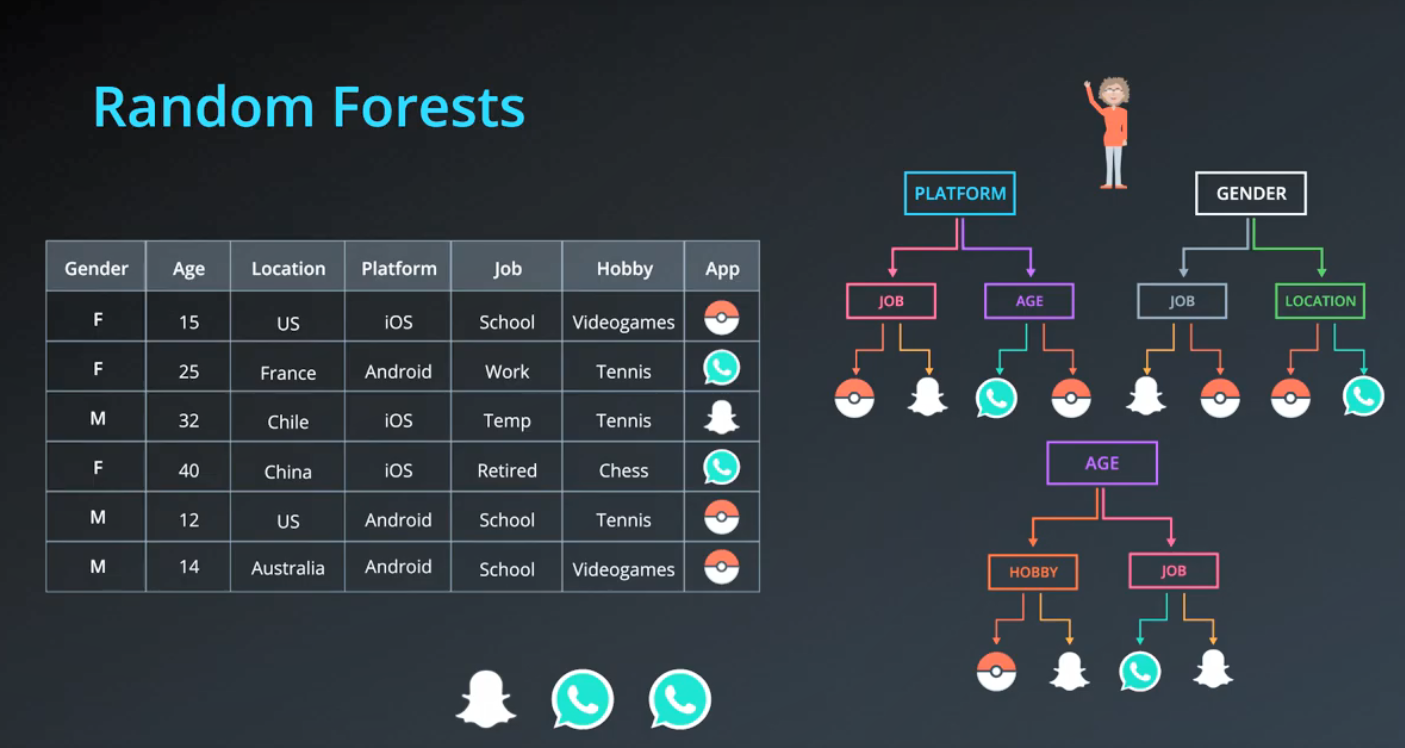
So splitting by the gender gives us an information gain of 0.54 and splitting us by occupation gave us a knowledge gain of 1.

**Random Forests**

Decision Trees tend to overfit data, such as below:



We can sole this using a theory called Random Forests. We would take some random columns from this data and build a decision tree based on that. We would do the same for different random selections of columns. Then we would bring our outcome through each of these decision trees and take the answer that appears most often:



# Hyperparameters for Decision Trees

In order to create decision trees that will generalize to new problems well, we can tune a number of different aspects about the trees. We call the different aspects of a decision tree "hyperparameters". These are some of the most important hyperparameters used in decision trees:

### Maximum Depth

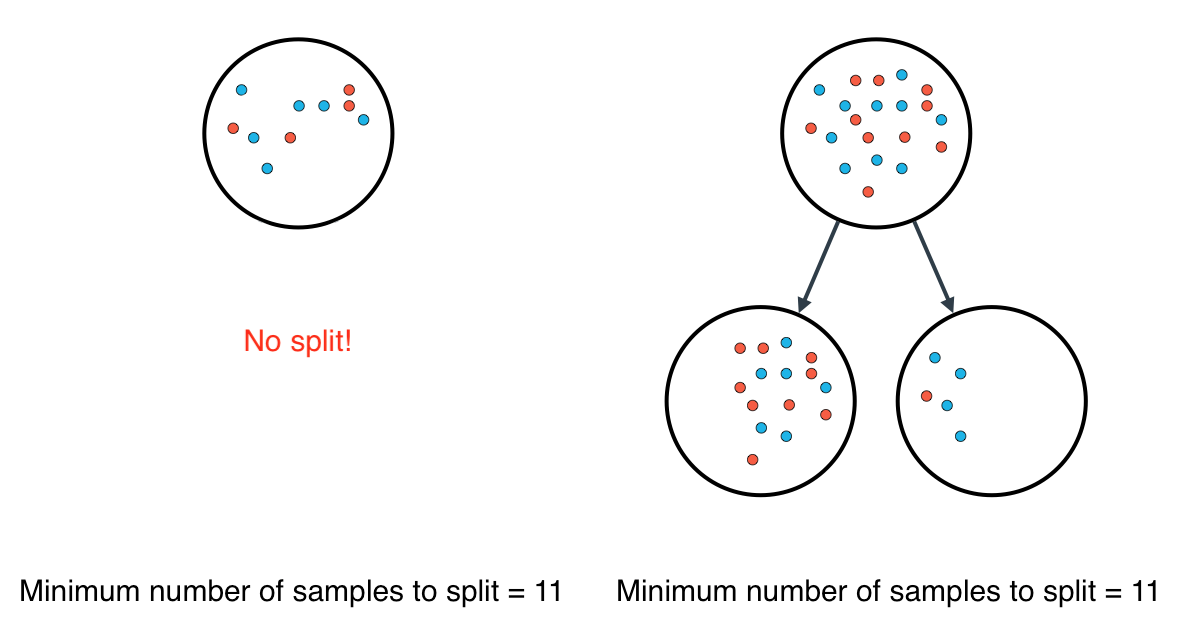
The maximum depth of a decision tree is simply the largest possible length between the root to a leaf. A tree of maximum length k*k* can have at most 2^k2*k* leaves.

[[](https://classroom.udacity.com/nanodegrees/nd009-ent/parts/e2907194-a763-44ca-9a5c-976bdc957119/modules/435676be-33e4-4ec8-877a-e4ee1f1904ca/lessons/6a868a0f-2f09-41af-9133-4783d084c344/concepts/a750d064-6240-47e7-87de-6e41dab807c5)](https://classroom.udacity.com/nanodegrees/nd009-ent/parts/e2907194-a763-44ca-9a5c-976bdc957119/modules/435676be-33e4-4ec8-877a-e4ee1f1904ca/lessons/6a868a0f-2f09-41af-9133-4783d084c344/concepts/a750d064-6240-47e7-87de-6e41dab807c5)

[Maximum depth of a decision tree](https://classroom.udacity.com/nanodegrees/nd009-ent/parts/e2907194-a763-44ca-9a5c-976bdc957119/modules/435676be-33e4-4ec8-877a-e4ee1f1904ca/lessons/6a868a0f-2f09-41af-9133-4783d084c344/concepts/a750d064-6240-47e7-87de-6e41dab807c5)

### Minimum number of samples to split

A node must have at least min\_samples\_split samples in order to be large enough to split. If a node has fewer samples than min\_samples\_split samples, it will not be split, and the splitting process stops.

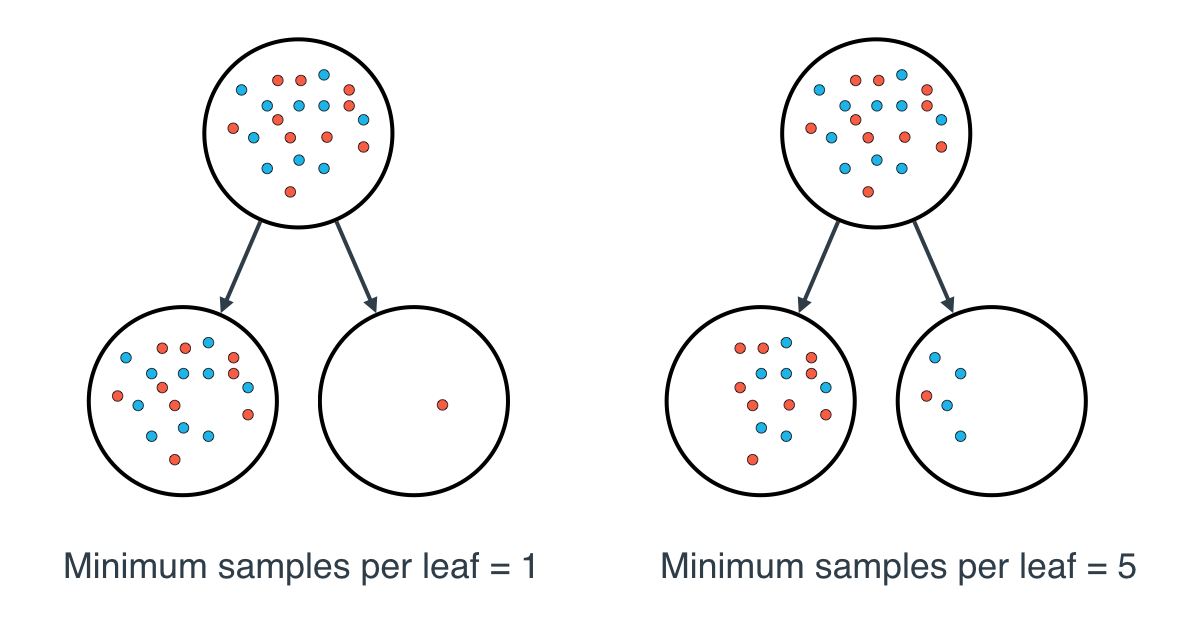
[[](https://classroom.udacity.com/nanodegrees/nd009-ent/parts/e2907194-a763-44ca-9a5c-976bdc957119/modules/435676be-33e4-4ec8-877a-e4ee1f1904ca/lessons/6a868a0f-2f09-41af-9133-4783d084c344/concepts/a750d064-6240-47e7-87de-6e41dab807c5)](https://classroom.udacity.com/nanodegrees/nd009-ent/parts/e2907194-a763-44ca-9a5c-976bdc957119/modules/435676be-33e4-4ec8-877a-e4ee1f1904ca/lessons/6a868a0f-2f09-41af-9133-4783d084c344/concepts/a750d064-6240-47e7-87de-6e41dab807c5)

[Minimum number of samples to split](https://classroom.udacity.com/nanodegrees/nd009-ent/parts/e2907194-a763-44ca-9a5c-976bdc957119/modules/435676be-33e4-4ec8-877a-e4ee1f1904ca/lessons/6a868a0f-2f09-41af-9133-4783d084c344/concepts/a750d064-6240-47e7-87de-6e41dab807c5)

However, min\_samples\_split doesn't control the minimum size of leaves. As you can see in the example on the right, above, the parent node had 20 samples, greater than min\_samples\_split = 11, so the node was split. But when the node was split, a child node was created with that had 5 samples, less than min\_samples\_split = 11.

### Minimum number of samples per leaf

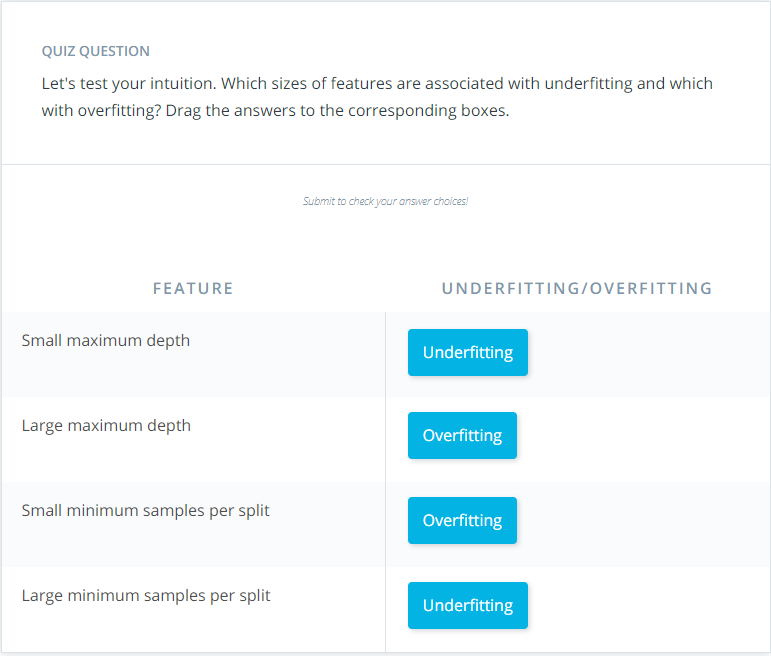
When splitting a node, one could run into the problem of having 99 samples in one of them, and 1 on the other. This will not take us too far in our process, and would be a waste of resources and time. If we want to avoid this, we can set a minimum for the number of samples we allow on each leaf.

[[](https://classroom.udacity.com/nanodegrees/nd009-ent/parts/e2907194-a763-44ca-9a5c-976bdc957119/modules/435676be-33e4-4ec8-877a-e4ee1f1904ca/lessons/6a868a0f-2f09-41af-9133-4783d084c344/concepts/a750d064-6240-47e7-87de-6e41dab807c5)](https://classroom.udacity.com/nanodegrees/nd009-ent/parts/e2907194-a763-44ca-9a5c-976bdc957119/modules/435676be-33e4-4ec8-877a-e4ee1f1904ca/lessons/6a868a0f-2f09-41af-9133-4783d084c344/concepts/a750d064-6240-47e7-87de-6e41dab807c5)

[Minimum number of samples per leaf](https://classroom.udacity.com/nanodegrees/nd009-ent/parts/e2907194-a763-44ca-9a5c-976bdc957119/modules/435676be-33e4-4ec8-877a-e4ee1f1904ca/lessons/6a868a0f-2f09-41af-9133-4783d084c344/concepts/a750d064-6240-47e7-87de-6e41dab807c5)

This number can be specified as an integer or as a float. If it's an integer, it's the minimum number of samples allowed in a leaf. If it's a float, it's the minimum percentage of samples allowed in a leaf. For example, 0.1, or 10%, implies that a particular split will not be allowed if one of the leaves that results contains less than 10% of the samples in the dataset.

If a threshold on a feature results in a leaf that has fewer samples than min\_samples\_leaf, the algorithm will not allow that split, but it may perform a split on the same feature at a different threshold, that does satisfy min\_samples\_leaf.



# Decision Trees in sklearn

In this section, you'll use decision trees to fit a given sample dataset.

Before you do that, let's go over the tools required to build this model.

For your decision tree model, you'll be using scikit-learn's [Decision Tree Classifier](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html) class. This class provides the functions to define and fit the model to your data.

>>> **from** sklearn.tree **import** DecisionTreeClassifier

>>> model = DecisionTreeClassifier()

>>> model.fit(x\_values, y\_values)

In the example above, the model variable is a decision tree model that has been fitted to the data x\_values and y\_values. Fitting the model means finding the best tree that fits the training data. Let's make two predictions using the model's predict() function.

>>> print(model.predict([ [0.2, 0.8], [0.5, 0.4] ]))

[[ 0., 1.]]

The model returned an array of predictions, one prediction for each input array. The first input, [0.2, 0.8], got a prediction of 0.. The second input, [0.5, 0.4], got a prediction of 1..

### Hyperparameters

When we define the model, we can specify the hyperparameters. In practice, the most common ones are

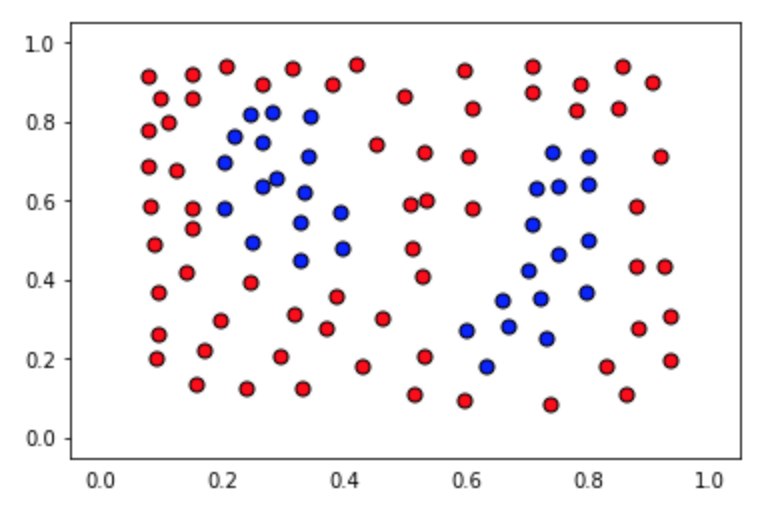
* max\_depth: The maximum number of levels in the tree.
* min\_samples\_leaf: The minimum number of samples allowed in a leaf.
* min\_samples\_split: The minimum number of samples required to split an internal node.

For example, here we define a model where the maximum depth of the trees max\_depth is 7, and the minimum number of elements in each leaf min\_samples\_leaf is 10.

>>> model = DecisionTreeClassifier(max\_depth = 7, min\_samples\_leaf = 10)

## Decision Tree Quiz

In this quiz, you'll be given the following sample dataset, and your goal is to define a model that gives 100% accuracy on it.

[[](https://classroom.udacity.com/nanodegrees/nd009-ent/parts/e2907194-a763-44ca-9a5c-976bdc957119/modules/435676be-33e4-4ec8-877a-e4ee1f1904ca/lessons/6a868a0f-2f09-41af-9133-4783d084c344/concepts/e964511f-92b0-4d11-a3df-3a1b1262cc8d)](https://classroom.udacity.com/nanodegrees/nd009-ent/parts/e2907194-a763-44ca-9a5c-976bdc957119/modules/435676be-33e4-4ec8-877a-e4ee1f1904ca/lessons/6a868a0f-2f09-41af-9133-4783d084c344/concepts/e964511f-92b0-4d11-a3df-3a1b1262cc8d)

The data file can be found under the "data.csv" tab in the quiz below. It includes three columns, the first 2 comprising of the coordinates of the points, and the third one of the label.

The data will be loaded for you, and split into features X and labels y.

### You'll need to complete each of the following steps:

**1. Build a decision tree model**

* Create a decision tree classification model using scikit-learn's [DecisionTree](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html" \t "_blank) and assign it to the variablemodel.

**2. Fit the model to the data**

* You won't need to specify any of the hyperparameters, since the default ones will yield a model that perfectly classifies the training data. However, we encourage you to play with hyperparameters such as max\_depth and min\_samples\_leaf to try to find the simplest possible model.

**3. Predict using the model**

* Predict the labels for the training set, and assign this list to the variable y\_pred.

**4. Calculate the accuracy of the model**

* For this, use the function sklearn function [accuracy\_score](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html" \t "_blank). A model's **accuracy** is the fraction of all data points that it correctly classified.

When you hit **Test Run**, you'll be able to see the boundary region of your model, which will help you tune the correct parameters, in case you need them.

**Note:** This quiz requires you to find an accuracy of 100% on the training set. This is like memorizing the training data! A model designed to have 100% accuracy on training data is unlikely to generalize well to new data. If you pick very large values for your parameters, the model will fit the training set very well, but may not generalize well. Try to find the smallest possible parameters that do the job—then the model will be more likely to generalize well. (This aspect of the exercise won't be graded.)

*# Import statements*

*from sklearn.tree import DecisionTreeClassifier*

*from sklearn.metrics import accuracy\_score*

*import pandas as pd*

*import numpy as np*

*# Read the data.*

*data = np.asarray(pd.read\_csv('data.csv', header=None))*

*# Assign the features to the variable X, and the labels to the variable y.*

*X = data[:,0:2]*

*y = data[:,2]*

*# TODO: Create the decision tree model and assign it to the variable model.*

*# You won't need to, but if you'd like, play with hyperparameters such*

*# as max\_depth and min\_samples\_leaf and see what they do to the decision*

*# boundary.*

*model = DecisionTreeClassifier(max\_depth = 2)*

*# TODO: Fit the model.*

*model.fit(X, y)*

*# TODO: Make predictions. Store them in the variable y\_pred.*

*y\_pred = model.predict(X)*

*# TODO: Calculate the accuracy and assign it to the variable acc.*

*acc = accuracy\_score(y, y\_pred)*

*///////////////////////////////////////////////////////////////*