

Decision Trees and Ensemble Methods: Random Forest

Main Sources

- Hands-on Machine Learning with Scikit-Learn & Tensor Flow – Aurelien Geron
- Data Science and Analytics with Python – Jesus Rogel-Salazar
- Machine Learning in Python – Michael Bowles
- Python Machine Learning – Sepastian Raschka & Vahid Mirjalili

Outline

➤ Decision Trees

- ❖ Basics
- ❖ Theoretical Background
- ❖ Advantages
- ❖ Limitations
- ❖ Mathematical Approaches
- ❖ Pruning
- ❖ Evaluation
- ❖ Case Study

➤ Ensemble

- ❖ Random Forest
- ❖ Example
- ❖ Practice

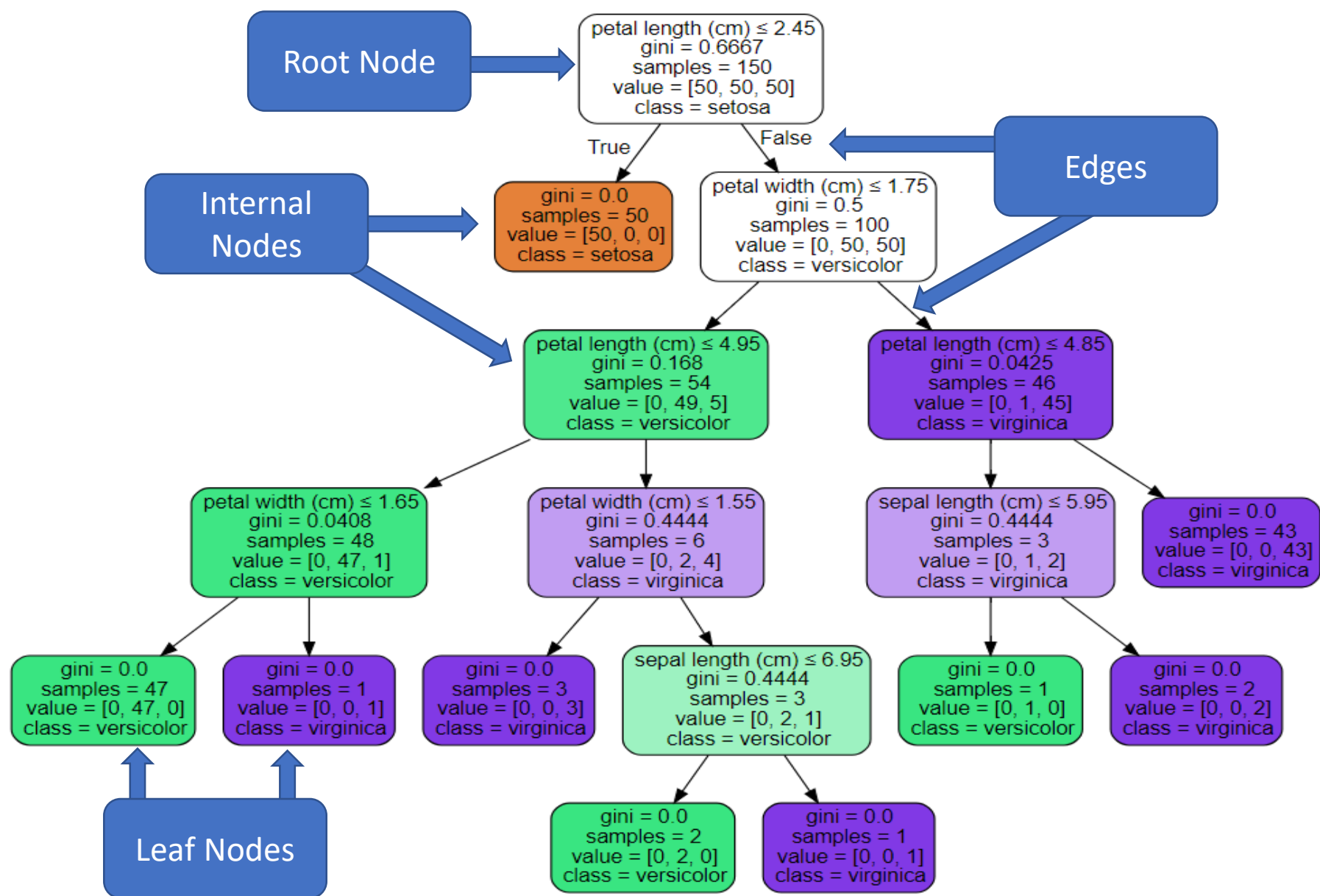
Decision Trees: Basics

- Decision trees are a hierarchical technique
 - ❖ Meaning that a series of decisions are made until a predetermined metric is met
 - Model is built such that a sequence of ordered decisions concerning values of data features results in assigning class labels
- Nonparametric
 - ❖ Due to the number of parameters is not pre-determined as is the case with linear models that have pre-determine parameters thus limiting their **degrees of freedom**
 - ❖ No assumptions need to be met concerning parameters or distributions
- Best recognized through graphs produced
 - ❖ Type of Acyclic graph - are used to model probabilities, connectivity, and causality. A “graph” in this sense means a structure made from nodes and edges
 - ❖ Trees consist of nodes and edges defined by decisions rules applied to the data features

Decision Trees: Basics

- Tree begins with a **Root Node** that has no incoming edges and two or more outgoing edges
- **Internal Node** – Has one incoming edge and two or more outgoing and represent test conditions at every given level
- **Leaf Node** – One incoming edge and no outgoing edges

Decision Trees: Graph Example



Decision Trees: Theoretical Background

- One possible approach that is commonly referenced for building out decision trees is Hunt's algorithm (Hunt, E.B., J. Marin, & P.J. Stone, 1966). Hunt's includes the basic steps below.
 1. If node n is pure, such that all the data instances N_n belong to class A , the process is complete and we've generated a leaf node
 2. If node n is not pure, we need to keep splitting. This necessitates a split criteria or test condition that allows for the partition. This results in an internal node (child node).
 3. The test condition is then run and we assign each of the instances N_n to one of the two child nodes created from node n
 4. These steps are applied repeatedly to each child node
- This process requires a test condition, which can take several forms
 - ❖ Gini coef, error rate, entropy

Decision Trees: Theoretical Background: CART Algorithm (cost function)

- Builds off of the Hunt's paradigm, Classification and Regression Tree (CART)
- Can produce either classification or regression based trees and is used by both sklearn in Python and the CART package in R
 - ❖ Can be used on numerical or categorical data
- Default for classification is Gini index for split identification
- First splits the training data in two subsets using a single feature k and a threshold t_k
 - ❖ Searches through all possible pairs (k, t_k) to identify the split that produces the purest subsets, based on weighted average of information gain.
- Stops once it cannot find a split that reduces impurity or by a pre-determine hyperparameter (`max_depth`).

Decision Trees: Theoretical Background

- Uses **recursive binary splitting** - Considering every possible partition of space is computationally infeasible, a **greedy** approach is used to divide the space, called recursive binary splitting.
- **Greedy algorithm** because at each step of the tree building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in the future.
- Trees can be regression or classification based, but in both instances will use **recursive binary splitting**
 - ❖ The difference is that in regression based trees we are predicting the actual class whereas in classification we are generating the probability of class inclusion as the determinate of the splitting
 - ❖ This probability measure that drives the splitting for classification comes in two forms Gini Index or Entropy

Decision Trees: Advantages

- Simple to understand and to interpret. Trees can be visualized.
- Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed.
- The cost of using the tree (i.e., predicting data) is **logarithmic** in the number of data points used to train the tree.
 - ❖ Inverse operation of exponential

Decision Trees: Advantages

- Able to handle both numerical and categorical data. Other techniques are usually specialized in analyzing datasets that have only one type of variable.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model. ROC/Hit Rate/Error Rate

Decision Trees: Limitations

- Decision-tree learners can create over-complex trees that do not generalize the data well. This is called **overfitting**.
 - ❖ Compare terminal nodes to data points, use the depth for each depth 6^2 equals 64 terminal nodes, if you have 100 data points that's a lot of single data terminal nodes
- Mechanisms such as pruning setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an **ensemble** or Random Forest.

Decision Trees: Limitations

- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts.
 - ❖ Consequently, practical decision-tree learning algorithms are based on **heuristic algorithms** such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to **balance** the dataset prior to fitting with the decision tree

Decision Trees: Definitions

- Overfitting – model becomes overly complex and as a result is predicting noise or the space between features (random error) instead of the true relationship. It is in theory possible to create a leaf node for every data point.
- Ensemble methods – Process of running numerous models and codifying them using a decision rule to choose the optimal model result – example is majority vote on feature inclusion
- Heuristic algorithms – approaches designed for operational efficiency generating a approximation to the ideal result but does not guarantee the best model

Decision Trees: Balancing Dataset

➤ Imbalanced classes often considers imbalanced to mean a minority class of 10% to 20%. In reality, datasets can get far more imbalanced than this, examples:


1. About 2% of credit card accounts are defrauded per year¹. (Most fraud detection domains are heavily imbalanced.)
2. Medical screening for a condition is usually performed on a large population of people without the condition, to detect a small minority with it (e.g., HIV prevalence in the USA is ~0.4%).
3. The conversion rates of online ads has been estimated to lie between 10^{-3} to 10^{-6} .
4. Factory production defect rates typically run about 0.1%.
5. Intrusion or threat detection, .0001% of network traffic

Decision Trees: Balancing Dataset

- Working on real world problems will almost certainly include unbalanced datasets. Making standard algorithms function inefficiently.
 - ❖ Solution is to **oversample** the minority target or **under-sample** the majority to create a more balanced training dataset
- **Oversampling** – Most commonly used, can result in a synthetic reduction in the variance associated with the variable but as a positive it essentially duplicates the number of errors, i.e. if there's a single false positive and it's included five times you get four additional errors.

Decision Trees: Node split mode

- Decision Trees can use several different types of node split criteria depending on the data or data scientist's preference
 - ❖ Regression/MSE – Continuous data
 - ❖ Entropy – Binary data splits
 - ❖ Gini Coefficient – Most common approach
- Let's take a look at each approach



Both Entropy and Gini Coefficient use Information Gain to determine variable split criteria

Mathematical Approaches: Regression/MSE

- Works to identify the split point in the data set that minimizes **mean squared error (MSE)** point
- The average of each of the groups is the term that minimizes the mean squared error
 - ❖ MSE – is the average of the difference between the prediction and actual values

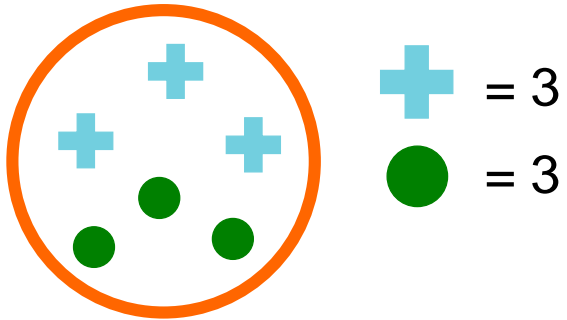
$$\sqrt{\frac{\sum_{t=1}^n (Y_t - \hat{Y}_t)^2}{n}}$$

- The decision tree algorithm searches through all variables and all possible split points to identify the point that minimizes error

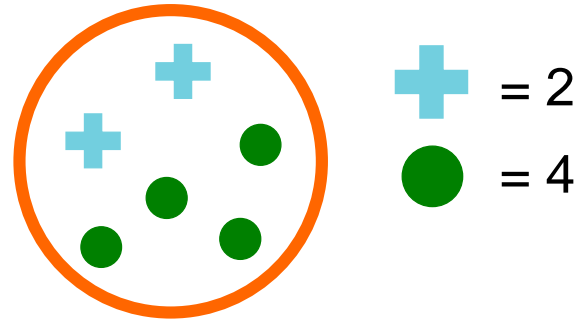
Decision Trees: Mathematical Approaches: Classification, Entropy

- The formula for entropy is below, where P_i is the probability that a random selection would have a state i

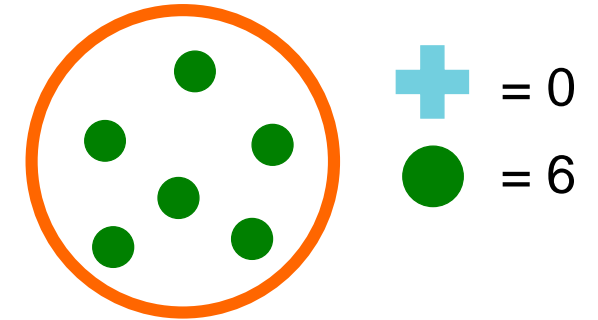
$$\text{Entropy} = \text{sum}(-P_i * \log_2 P_i)$$



$$(3/6 \log_2 3/6) - (3/6 \log_2 3/6) = 1$$



$$(2/6 \log_2 2/6) - (4/6 \log_2 4/6) = 0.92$$



$$(6/6 \log_2 6/6) = 0$$

Decision Trees: Mathematical Approaches: Classification, Entropy

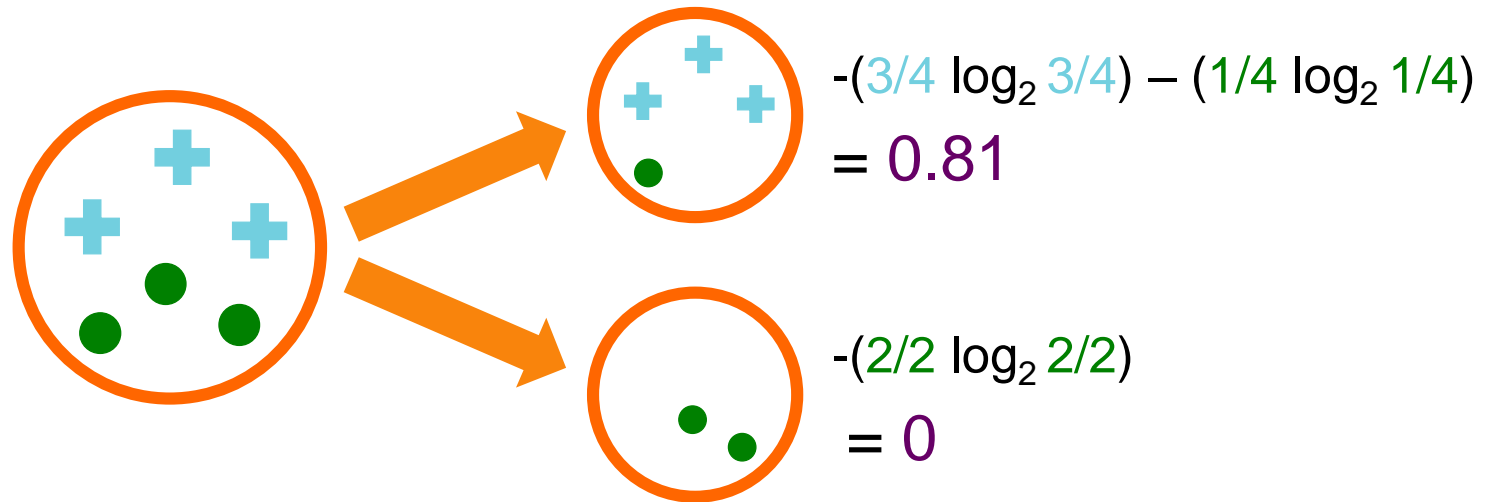
- Information gain helps us understand how important an attribute is in the data
- We can use it to decide how to order the nodes of the decision tree

Information gain = **entropy (parent)** – **average entropy (children)**

entropy (parent)

average entropy (children)

$$-(\frac{3}{6} \log_2 \frac{3}{6}) - (\frac{3}{6} \log_2 \frac{3}{6}) \\ = 1$$



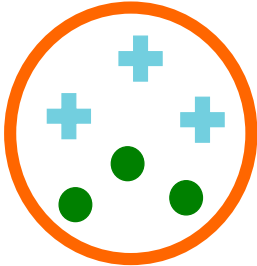
Decision Trees: Mathematical Approaches: Classification, Entropy

- In order to calculate the average entropy for the split, we need to weigh the split by the number of data points in each node. So we create a weighted average of the entropy of the children nodes.

Information gain = **entropy (parent)** – **average entropy (children)**

entropy (parent)

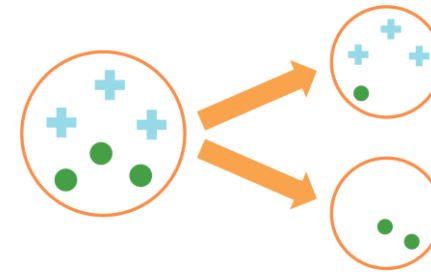
= 1



Weighted average entropy (children)

= $(2/6 * 0) + (4/6 * 0.81)$

= 0.54

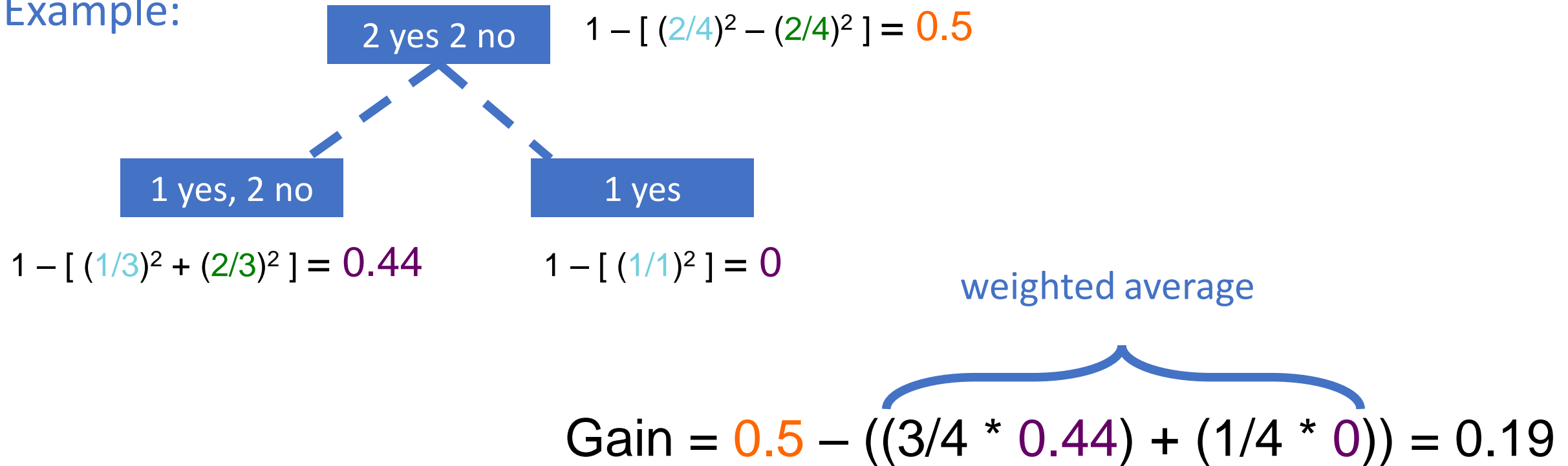


Decision Trees: Mathematical Approaches: Classification, Gini Coefficient

➤ Gini

- ❖ Gini Impurity = $1 - \sum[(P_i)^2]$
- ❖ P_i – Represents the probability that a random selection would have state i .
- ❖ Same mathematical process as entropy

➤ Example:



Decision Trees: Code example: Entropy

```
#Importing packages
import pandas as pd
import numpy as np
import scipy
import matplotlib.pyplot as plt
from sklearn import tree
import sklearn as sk
from IPython.display import Image
import pydotplus
import graphviz
```

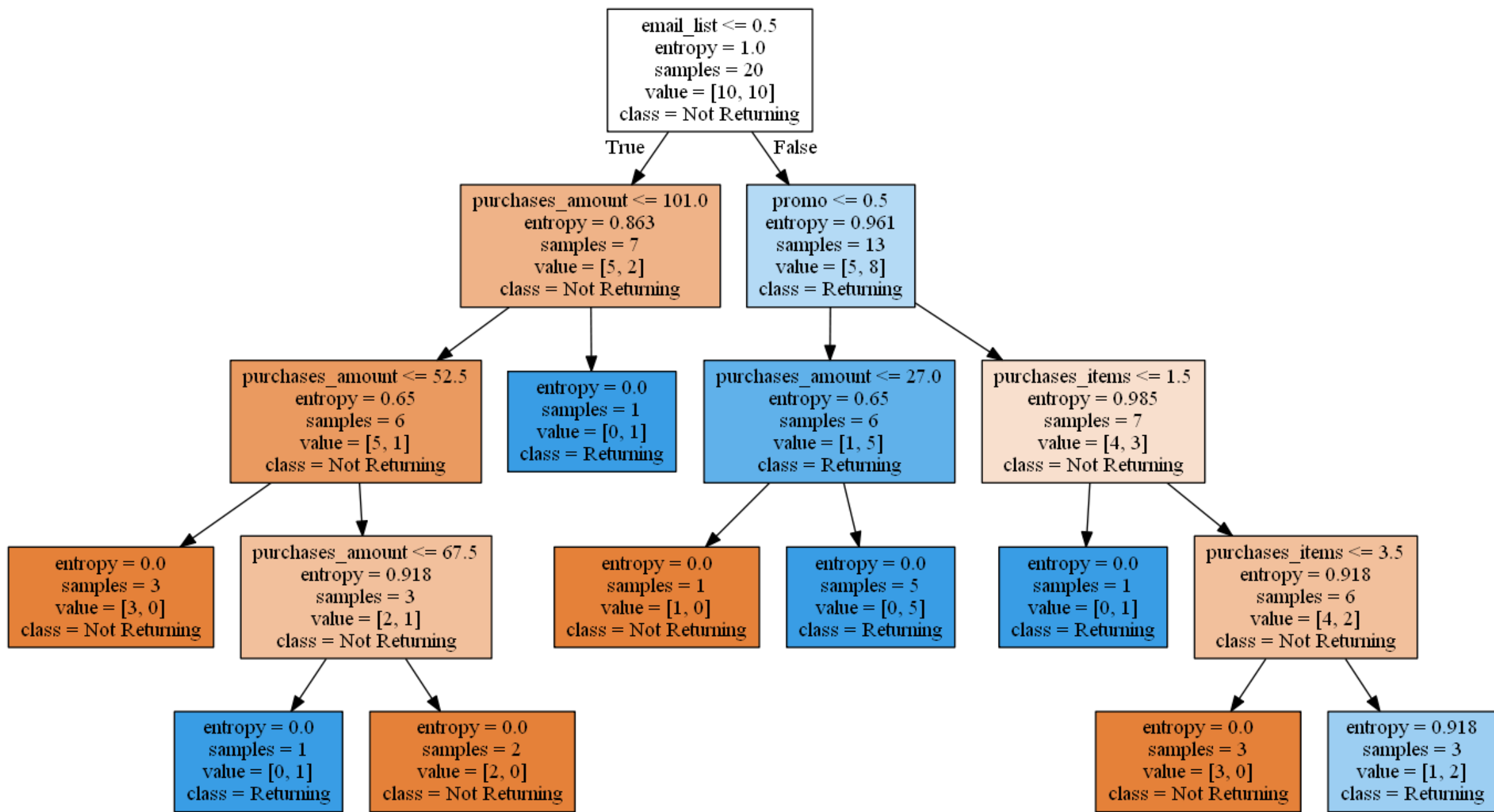
Decision Trees: Code example: Entropy

```
#Creating Data
customers = pd.DataFrame()
customers['purchases_amount'] = [105, 65, 89, 99, 149, 102, 34, 120, 129, 39,
                                20, 30, 109, 40, 55, 100, 23, 20, 70, 10]
customers['purchases_items'] = [1, 4, 5, 4, 7, 1, 2, 10, 6, 5,
                                1, 3, 2, 1, 5, 10, 3, 3, 1, 1]
customers['promo'] = [1, 1, 0, 1, 0, 0, 0, 0, 0, 1,
                     1, 1, 1, 0, 1, 1, 1, 0, 1, 1]
customers['email_list'] = [1, 0, 1, 1, 1, 0, 1, 1, 1, 1,
                           0, 1, 1, 0, 1, 0, 1, 1, 0, 0]
customers['checkouts'] = [1, 5, 3, 3, 1, 2, 4, 4, 1, 1,
                          1, 1, 2, 4, 1, 1, 2, 1, 1, 1]
repeat_customers = pd.DataFrame()
repeat_customers['repeat'] = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                              0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```


Decision Trees: Code example: Entropy

```
#Launch the Tree and fit the tree
decision_tree = tree.DecisionTreeClassifier(
    criterion='entropy',    --- Setting the split criteria
    max_features=1, --- number of features used to split per node
    max_depth=4, --- depth of the tree
    random_state = 1000 --- normally wouldn't use but simply creates the same tree
    for everyone for comparison purposes
)
decision_tree.fit(customers, repeat_customers) --- fits the tree
sk.tree.export_graphviz(decision_tree) --- produces the results
```

Decision Trees: Example and Practice



Decision Trees: Code example: Exercise and Practice

- Using the same data remove the random state and compare results
 - ❖ Are they the same?
 - ❖ If not why?
- Adjust the depth of the tree and compare results

Decision Trees: Overfitting and Hyper-parameters

- Decision trees are often prone to overfitting one solution is utilize the hyper-parameters to control the depth of the tree or limit the expansion of leaf nodes
- Another option is to use a ensemble method via bagging or what's known as Random Forest
 - ❖ Will discuss further later in the day

Decision Trees: Hyper-parameter tuning (Pruning)

- **Minimum samples for a node split** Minimum number of samples (or observations) which are required in a node to be considered for splitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample. It should be tuned using cross validation.
- **Minimum samples for a terminal node (leaf)** The minimum number of samples (or observations) required in a terminal node or leaf. For imbalanced class problems, a lower value should be used since regions dominant with samples belonging to minority class will be much smaller in number.
- **Maximum depth of tree (vertical depth)** The maximum depth of trees, lower values prevent a model from learning relations which might be highly specific to the particular sample. It should be tuned using cross validation.

Decision Trees: Hyper-parameter tuning (Pruning)

- **Maximum number of terminal nodes** Also referred as *number of leaves*. Since binary trees are created, a depth of n would produce a maximum of 2^n leaves.
- **Maximum features to consider for split** The number of features to consider (selected randomly) while searching for a best split. A typical value is the square root of total number of available features. A higher number typically leads to over-fitting but is dependent on the problem as well.
- There's actually many more, the documentation is here:
 - ❖ <http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

Decision Trees: Pruning Code Example

➤ Go to Spyder

Decision Trees: Prediction

- You can use the built in prediction function in sklearn to implement our models
 - ❖ `clf.predict([[5,1.5,2,2]])`
 - ❖ `tree_clf_reg_full.predict([[5,1.5,2,2]])`
- Seems like we have different results, let's look at model evaluation next.
- Quick Exercise
 - ❖ Use different ranges of each of the variables to generate different predictions
 - Petal Length
 - Petal Width
 - Sepal Length
 - Sepal Width

Break 15 Minutes

Decision Trees: Evaluation: Receiver Operating Curve or Area Under the Curve

- ROC and AUC are two very common approaches for measuring the performance of classification models
- Essentially both these measure the misclassification error rate associated with your model
- A Confusion Matrix is a good tool for understanding how accurate your model is classifying and can be used to build ROC

Decision Trees: Evaluation: Confusion Matrix and threshold

- Let's use intruder detection as an example
- Say we have 135 emails entering our system and we are trying to detect whether they are fraudulent or not
 - ❖ We use lots of criteria – source, subject, if they came from a prince...
- Generate probability measures as a result of our tree algorithm to determine the likelihood that any one of these emails is fraudulent
- The cutoff point that is predetermined in the tree at 50% but can be modified as an input to the `confusion_matrix()` function by adjusting the `sample_weight` parameter

Actual Class	Predictive Class		
		Positive Fraud	Negative Not Fraud
	Positive Fraud	True Positive 10	False Negative 7
	Negative Not Fraud	False Positive 22	True Negative 96

Decision Trees: Evaluation: Confusion Matrix and threshold

➤ Let's consider the extremes: what if we set the threshold to 0?

❖ Means that everything is captured as Fraud and no ever gets a email again!

Actual Class	Predictive Class		
		Positive Fraud	Negative Not Fraud
	Positive Fraud	True Positive 17	False Negative 0
	Negative Not Fraud	False Positive 118	True Negative 0

➤ Let's consider the extremes: what if we set the threshold to 100?

❖ Means nothing is fraud and now everyone is getting rich off of Arabian princes

Decision Trees: Evaluation: Confusion Matrix and threshold

- Let's consider the extremes: what if we set the threshold to 100?
 - ❖ Means nothing is fraud and now everyone is getting rich off of Arabian princes

Actual Class	Predictive Class		
		Positive Fraud	Negative Not Fraud
	Positive Fraud	True Positive 0	False Negative 17
	Negative Not Fraud	False Positive 0	True Negative 118

Decision Trees: Evaluation: Confusion Matrix and threshold

➤ We can also predict the accuracy of our model using this information:

❖ True Positive Rate (TPR) = $10/(10+22) = .45$

❖ False Positive Rate (FDR) = $7/(7+96) = .06$

Actual Class	Predictive Class		
		Positive Fraud	Negative Not Fraud
	Positive Fraud	True Positive 10	False Negative 7
	Negative Not Fraud	False Positive 22	True Negative 96

➤ These two data points can then be used to begin to develop a Receiver Operating Characteristic Curve or ROC curve

❖ True Positive Rate (TPR) = $10/(10+22) = .45 = y\text{-axis}$

❖ False Positive Rate (FDR) = $7/(7+96) = .06 = x\text{-axis}$

Decision Trees: Evaluation: Confusion Matrix: Code Example

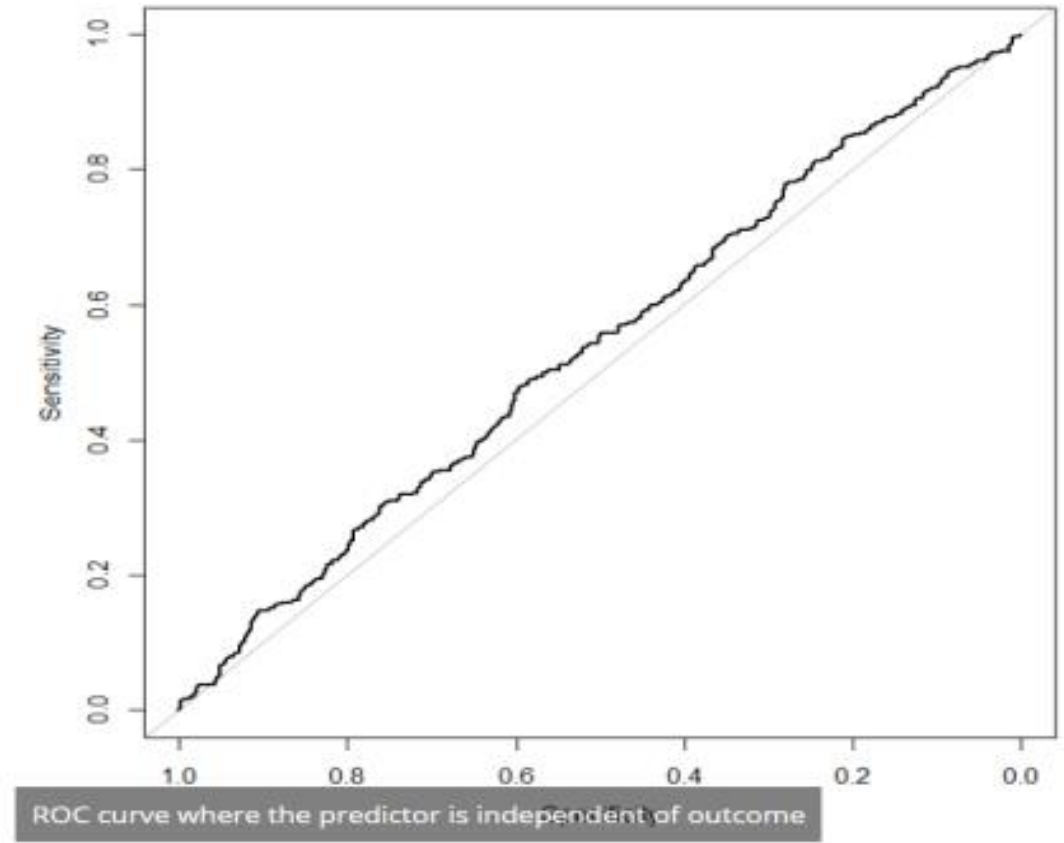
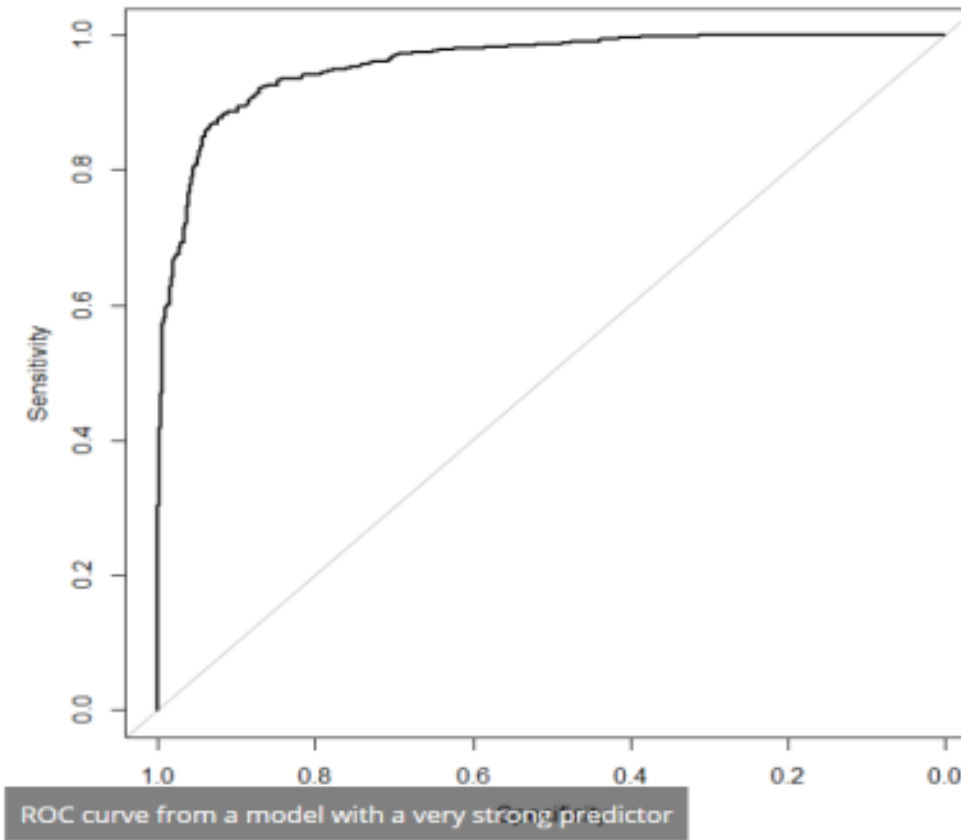
- Below we are using the two decision trees we developed to generate confusion matrix to evaluate our models.
- We first use the original data to predict then compare the results to the original data

```
clfpred = clf.predict(iris.data)
sk.metrics.confusion_matrix(y,clfpred)

regpred = tree_clf_reg_full.predict(iris.data)
sk.metrics.confusion_matrix(y,regpred)
```

Decision Trees: Evaluation: ROC

- ROC curve is essentially a graphical representation of the adjusted threshold values of the confusion matrix, below are two examples



Decision Trees: Evaluation: Code ROC

#The ROC is best used in a 2-demensional format so we will pull forward a previous example focused on customer prediction

```
customerpred = decision_tree.predict(customers)
```

```
#Here we see the confusion matrix  
sk.metrics.confusion_matrix(repeat_customers, customerpred)
```

#The roc_curve has three outputs fpr:false positive rate, tpr: true positive rate and thresholds, below we are developing output for each

```
false_positive_rate, true_positive_rate, thresholds =  
roc_curve(repeat_customers, customerpred)
```

Here we are develop the area under the curve

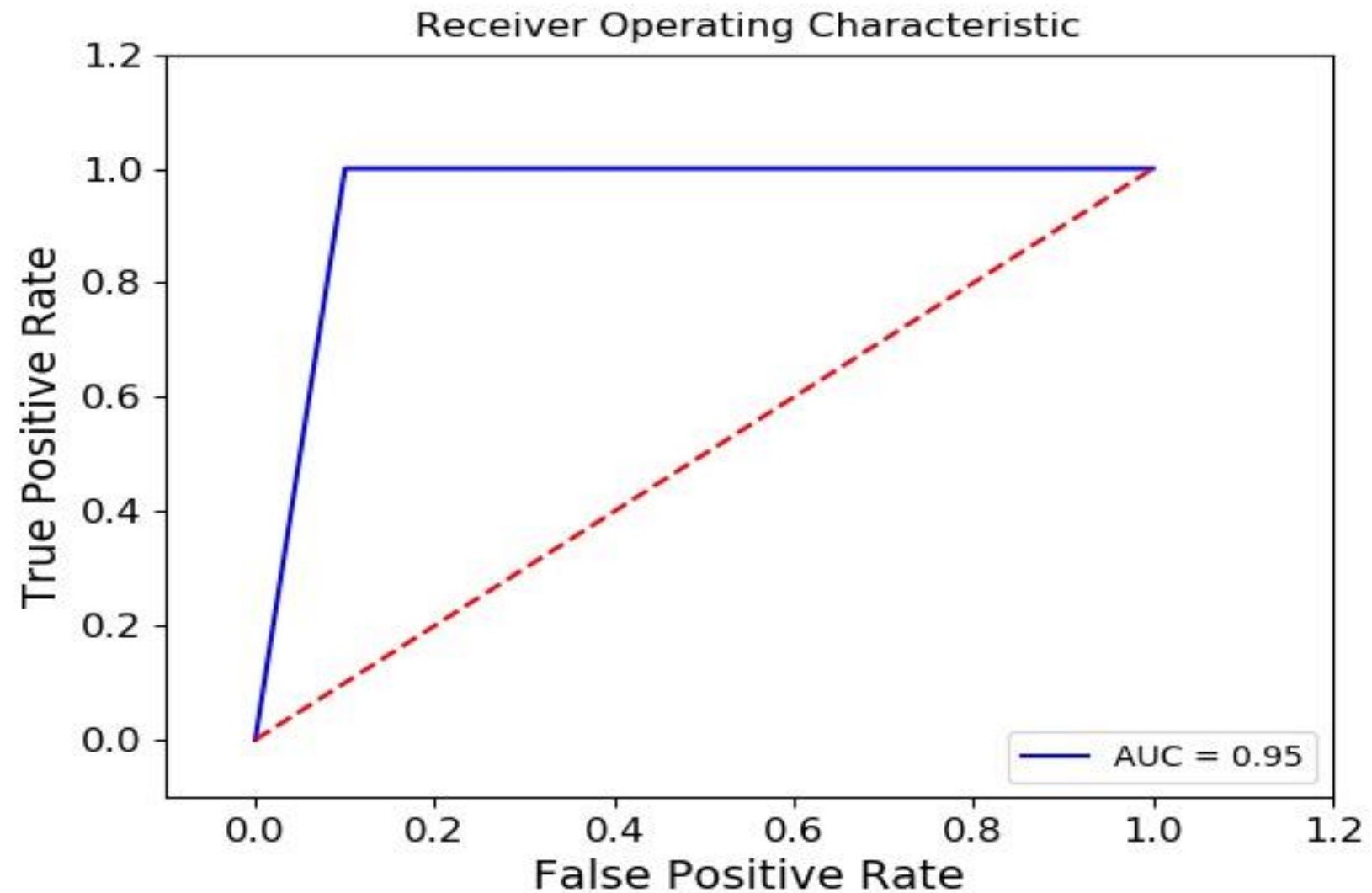
```
roc_auc = auc(false_positive_rate, true_positive_rate)
```

- Here we use the variables we created in the previous slide to display the output graphically

```
#The code below plots these outputs
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate, true_positive_rate, 'b',
label='AUC = %0.2f'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1], 'r--')
plt.xlim([-0.1,1.2])
plt.ylim([-0.1,1.2])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

- Here we use the variables we created in the previous slide to display the output graphically

```
#The code below plots these outputs
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate, true_positive_rate, 'b',
label='AUC = %0.2f'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1], 'r--')
plt.xlim([-0.1,1.2])
plt.ylim([-0.1,1.2])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Break: 15 Minutes

Case Study Example

- Review Case Study and Discuss (time permitting)

Outline

➤ Decision Trees

- ❖ Basics
- ❖ Theoretical Background
- ❖ Advantages
- ❖ Limitations
- ❖ Mathematical Approaches
- ❖ Pruning
- ❖ Evaluation
- ❖ Case Study

➤ Ensemble

- ❖ Random Forest
- ❖ Example
- ❖ Practice

Ensemble Methods

Ensemble Methods

- Ensemble methods are more or less aggregated predictions of many different algorithms in order to increase predictive accuracy.
- Often in solving a machine learning problem building a ensemble model comes at the end of the process after trying several different approaches you can combine them into one all knowing predictor!
- We will focus on Random Forrest, a ensemble of decision tress but also discuss bagging and boosting

Ensemble Methods

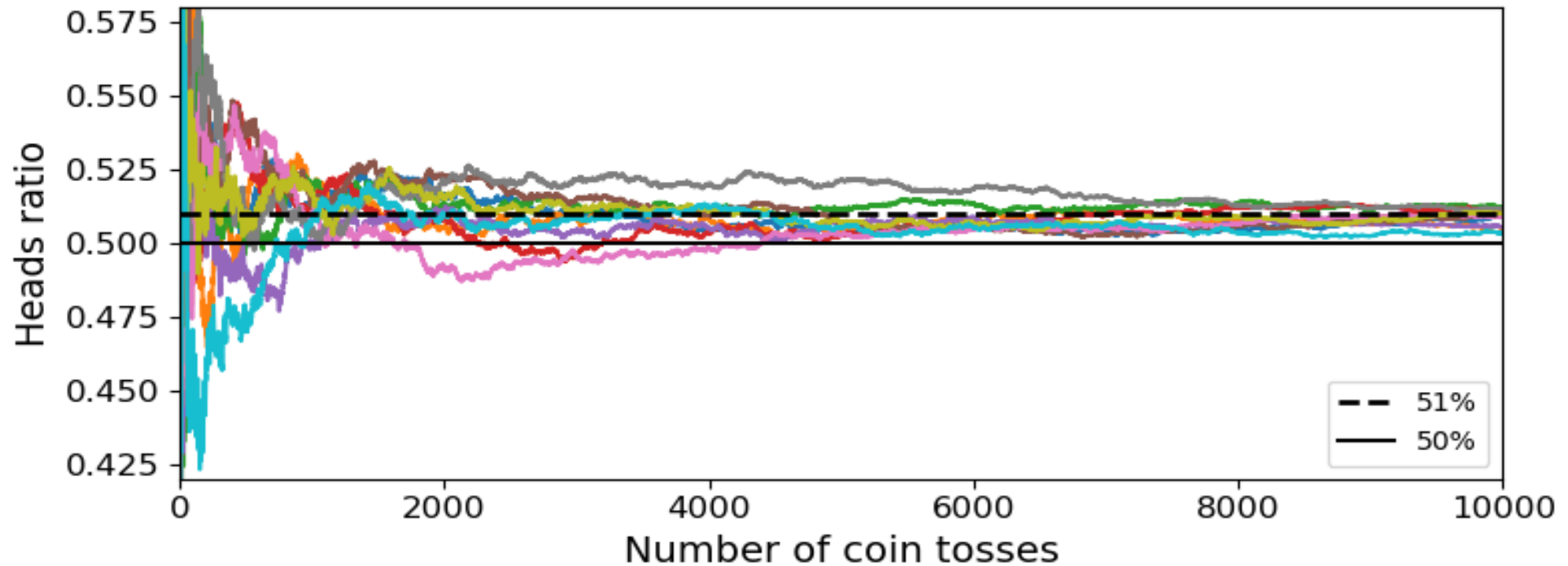
- Example: suppose we have developed several different classifiers a KNN model, Logistic Regression, an SVM and a Decision Tree.
- We could use the majority vote process to build the final classification of our data points, based on below what would be the outcome for this single data point?

Model	KNN	Logistic Regression	Support Vector Machine	Decision Tree
Prediction	1	0	1	1

- This is called **hard voting**, another approach is **soft voting (discuss later)**
- This process often works better than using single **weak learners** – or algorithms that predict only slightly better than random guessing

- Why do the models work better together?
 - ❖ Essentially scale increases the probability of finding a majority vote.
 - ❖ As an example if we have a bias coin flip that gives us 51% chance of heads and 49% chance of tails the more we toss the coin the higher the probability of getting a majority vote for heads.
 - 1,000 = 75% chance
 - 10,000 = 97% chance
- Works the same way for model building, the reliability of the results simply increase with scale.
 - ❖ Works best if the models are perfectly independent, meaning the error terms don't correlate which is hard when using one approach on a single dataset, that's why combining approaches can sometimes result in better outcomes

Ensemble Methods: Probabilities Converge (Code Available)



Ensemble Methods: Ensemble

- **Soft voting** is used for algorithms that produce probabilistic outputs or those that have a hyperparameter that can be modified to support probability outputs (such as SVM, probability hyperparameter to TRUE).
 - ❖ The class is generated using the highest class probability as the metric
 - ❖ Replacing voting = hard to voting = soft in the VotingClassifier() function

Ensemble Method: Code Example

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons

#Load in the models we will be using
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

#Use the train test function on the moon dataset
X, y = make_moons(n_samples=500, noise=0.30, random_state=1)
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=1)

#Next we are establishing the variables for each of our models
log_clf = LogisticRegression(random_state=1)
rnd_clf = RandomForestClassifier(random_state=1)
svm_clf = SVC(random_state=1)
```

Ensemble Method: Code Example

```
#Next we are just developing a voting classifier as we discussed that
uses hard voting to develop an ensemble method
voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='hard')
voting_clf.fit(X_train, y_train)
```

```
#Now let's see how we did
from sklearn.metrics import accuracy_score # This metric calculates
the error rate for each of our models
```

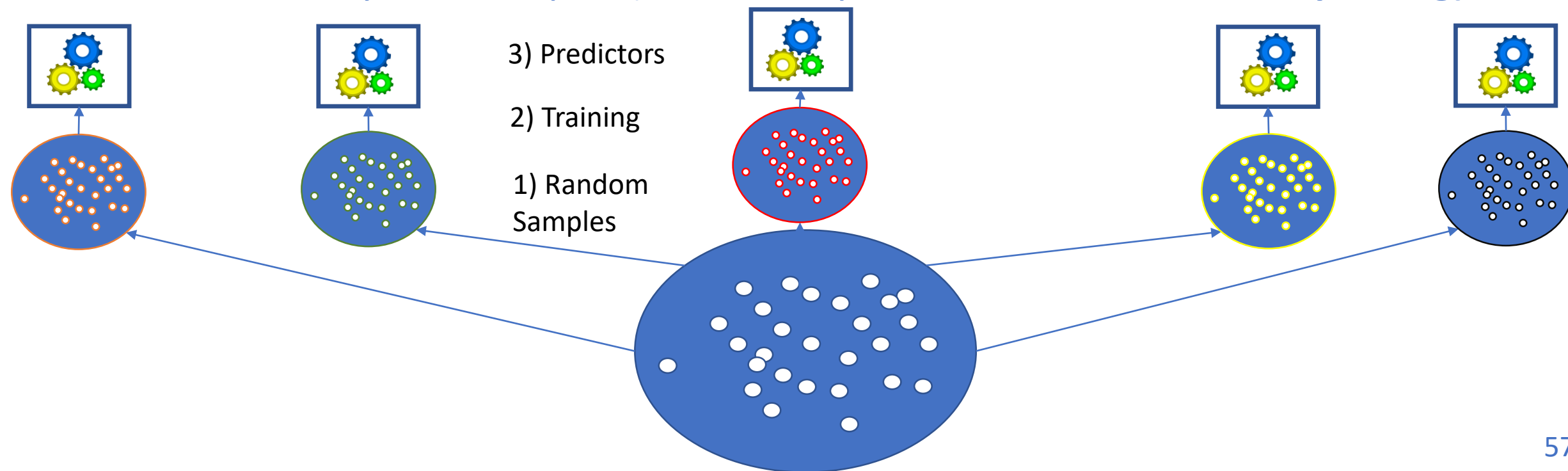
```
for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

Ensemble Methods: Random Forest – power in numbers

- Ensemble methods – Essentially instead of building a single tree we are going to build a whole bunch
- We can limit the growth of the trees but don't have to use any of the hyper-parameters
- Set the number of trees grown and track the error classification rate of our algorithm
- Can be used for again for both Regression or Classification
- Random Forest uses **bagging**
 - ❖ **Bagging** – Bootstrap Aggregation – selecting subsets of the data with replacement to generate unique trees
 - ❖ **Boosting** – Another form of dataset selection that is commonly used in ensemble methods

Ensemble Methods: Bagging

- As discussed on method is to use several methods and combine them to produce a more powerful outcome
- You can also you the same technic but re-sample the data numerous times to produce different results, one such method is **Bagging**
- **Bagging** – is sampling with replacement, meaning that the entire dataset is available for every sub-sample. (without replacement is often called **pasting**)



Ensemble Methods: Random Forest

Sampling with replacement

Sampling without replacement



Obs	X1	X2	Y1	Y2
1	2.5	3.6	4.8	3.7
2	2.8	4.7	-2.8	7.1
3	5.8	9.7	9.1	13

Obs	X1	X2	Y1	Y2
1	2.5	3.6	4.8	3.7
2	2.8	4.7	-2.8	7.1
1	2.5	3.6	4.8	3.7

Obs	X1	X2	Y1	Y2
2	2.8	4.7	-2.8	7.1
1	2.5	3.6	4.8	3.7
3	5.8	9.7	9.1	13

Ensemble Methods: Random Forest

- If we keep building decision trees on the same dataset, we would essentially get the same decision trees every time

Obs	X1	X2	Y1	Y2
1	2.5	3.6	4.8	3.7
2	2.8	4.7	-2.8	7.1
3	5.8	9.7	9.1	13



Obs	X1	X2	Y1	Y2
2	2.8	4.7	-2.8	7.1
1	2.5	3.6	4.8	3.7
3	5.8	9.7	9.1	13



Obs	X1	X2	Y1	Y2
3	5.8	9.7	9.1	13
1	2.5	3.6	4.8	3.7
2	2.8	4.7	-2.8	7.1



Ensemble Methods: Random Forest

- Use a subset of attributes generated by bagging to build original data sets to make decision trees

Obs	X1	X2	Y1	Y2
1	2.5	3.6	4.8	3.7
1	2.5	3.6	4.8	3.7
3	5.8	9.7	9.1	13



Obs	X1	X2	Y1	Y2
2	2.8	4.7	-2.8	7.1
1	2.5	3.6	4.8	3.7
2	2.8	4.7	-2.8	7.1



Obs	X1	X2	Y1	Y2
3	5.8	9.7	9.1	13
3	5.8	9.7	9.1	13
2	2.8	4.7	-2.8	7.1



Ensemble Methods: Bagging

- Typically once the sub-samples have been gathered the new classes are generated through hard voting for classification or the average for regression
 - ❖ However if the algorithm produces a probabilistic output, like decision trees, soft voting is used.

Ensemble Methods: Coding Example

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

#This is for classification if we want to run the same process on regression
it's BaggingRegressor
bag_clf = BaggingClassifier(DecisionTreeClassifier(),
n_estimators=500,max_samples=100, bootstrap=True,n_jobs=-1)
#n_estimators = number of tree classifiers
#max_samples= number of times each of those classifiers is trained using
random sampling with replacement
#bootstrap = tells the baggingclassifier to use with replacement
#n_jobs = tells the classifier the number of CPUs to use for the classifier,
-1 tells to use all available
bag_clf.fit(X_train, y_train) #No we run the data from the moon dataset
y_pred=bag_clf.predict(X_test) #Generate predictions using the test dataset
print(accuracy_score(y_test, y_pred)) #Calculate the accuracy score
```

Ensemble Methods: Random Forest

- Instead of building the ensemble from the ground up using the bagging method we can also just use the Random Forest classifier
- Typically all the hyperparameters are the same as the DecisionTreeClassifier and the hyperparameters of the bagging classifier
- Extra Randomness is also included as the Random Forest classifier as it searches for the best node split among all the random subsets of features
 - ❖ This creates greater tree diversity but trades higher **variance** for lower **bias**, which is often the case when increasing the complexity of the method
- **Bias** – is underfitting a model, meaning you're missing important relationships between variables.
- **Variance** – is overfitting the model, meaning the model is measuring the spaces between data points or the **noise** versus

Ensemble Methods: Random Forest: Code Example

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf=RandomForestClassifier(n_estimators=500,
max_leaf_nodes=16,n_jobs=-1)

#n_estimators = number of tree classifiers
#max_leaf_nodes = complexity of the model, limits the terminal leaf
nodes

rnd_clf.fit(X_train, y_train)

y_pred_rf=rnd_clf.predict(X_test)
```


Ensemble Methods: Random Forest: Variable Importance

- One of the really nice features associated with Random Forrest is it's ability to select variables that are most "important" to training the model.
- This is done by determining which variables are closest to the root of the tree and then moving outward
- Random Forrest does this at scale and the output is generated as a part of the training process
 - ❖ Sklearn calculates the average depth at which each feature is utilized and ranks them accordingly

Ensemble Methods: Random Forest: Variable Importance: Code Example

```
iris = datasets.load_iris()
# Fit the model using a different random forest classifier
ext = ExtraTreesClassifier()
ext.fit(iris.data, iris.target)

# display the relative importance of each attribute
print(ext.feature_importances_)

# Below will align the feature importance with the variable being used and
print the result

for name, importance in zip(iris["feature_names"], ext.feature_importances_):
    print(name, "=", importance)
```