ECON7960 User Experience and A/B Test

Hong Kong Baptist University

Topic 7: Random Forest Algorithms and their Applications

In previous 6 topics, we cover

Data Decision, Strategy and Design in Enhancing User Experience

A/B Simple Hypothesis Testing Steps and Underlying Statistical Principles

User Behaviour Patterns and User Metrics - Cognitive or Habit behaviours

A Quick Review

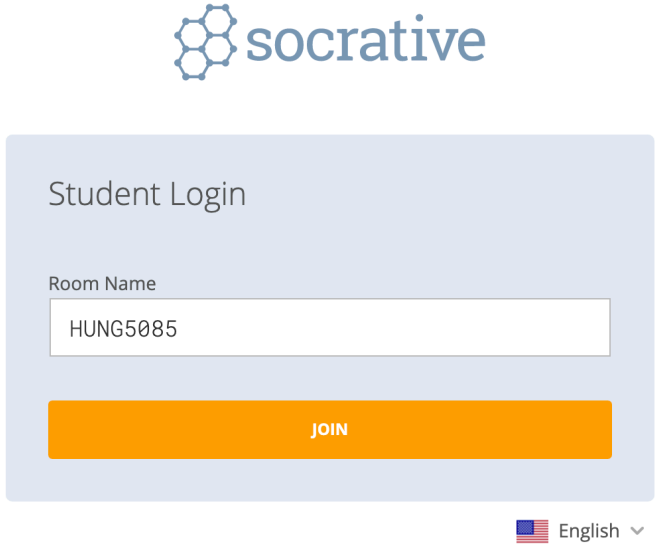
What did we learn last week

Please use your phone to download an app or web http://www.socrative.com ("SOCRATIVE" student version) and open it, you should see

Enter the Room Name "HUNG5085"

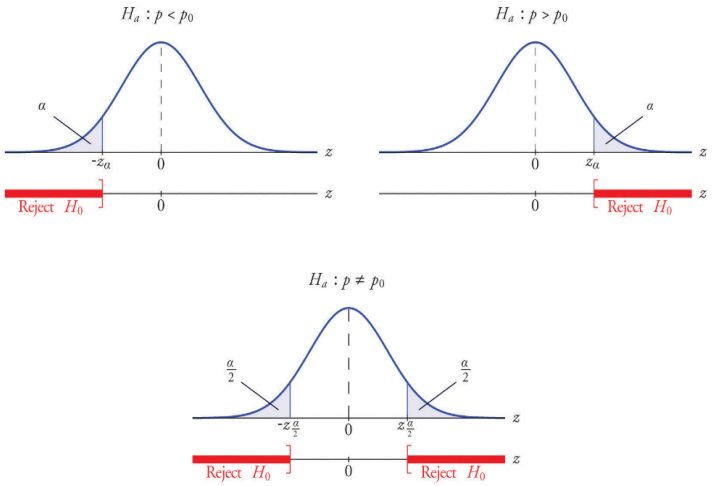
The quiz will start at 6:30pm before the lecture and lasted for 20 minutes.

Keep the "SOCRATIVE" apps open during the lecture

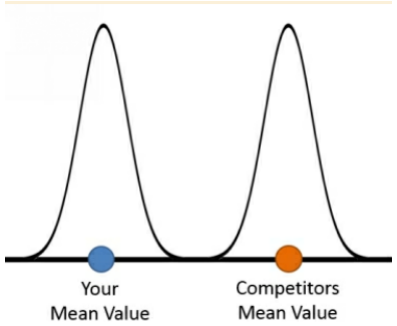


Alternative Hypothesis	Rejection region of α significance level
$H_a: \mu > \mu_0$	$z_{1-\alpha} \leq z$
$H_a: \mu < \mu_0$	$z_{\alpha} \geq z$
$H_a: \mu \neq \mu_0$	$z_{\alpha/2} \geq z \ or$
	$z_{1-\alpha/2} \leq z$

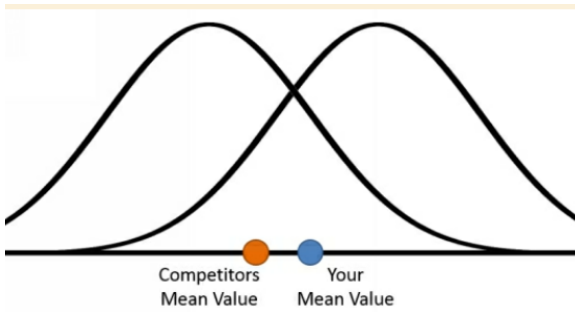
The rejection region



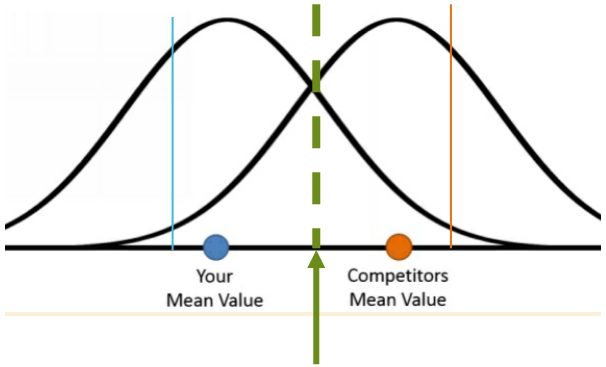
A/B Test is to able to separate the two estimate implications



If the distribution is like these, it will be **easily** to distinguish them



If the distribution is like these, it will be **difficult** to distinguish them



Able to separate from two distributions

Decision Tree

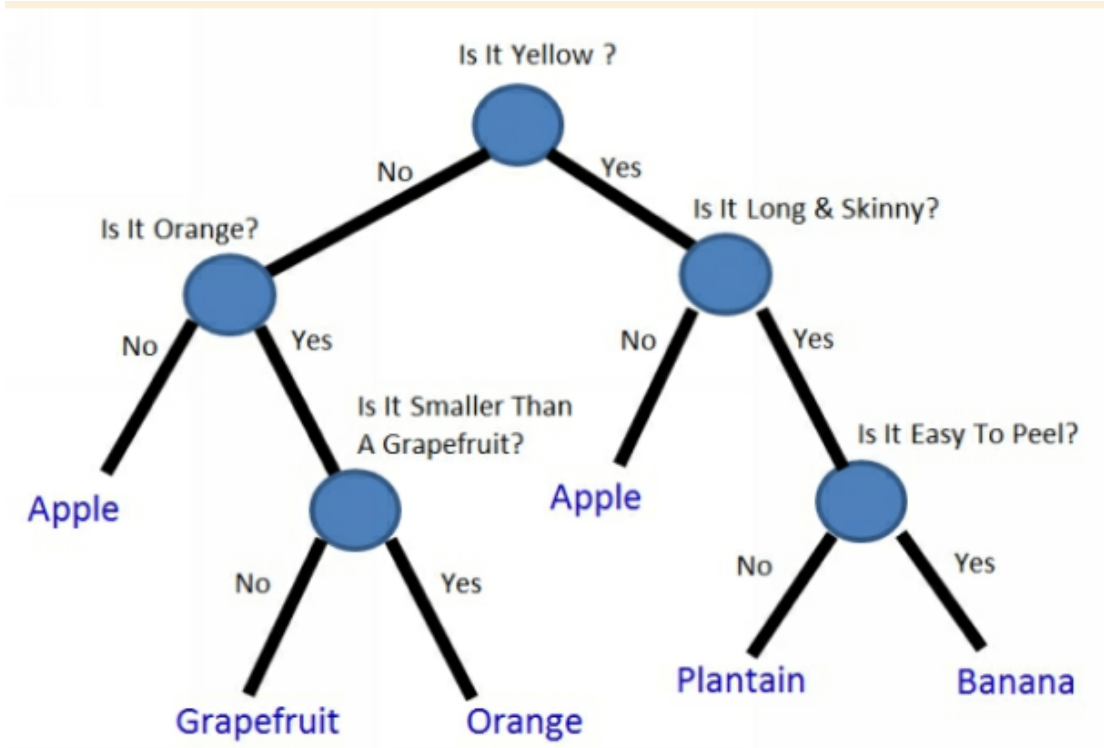
Random Forests are kinds of machine learning algorithms and are simply a collection of Decision Trees that have been generated using a random subset of data.

It is a way like follow the flow chart of questions to determine what type of variables are.

For example: classify fruits

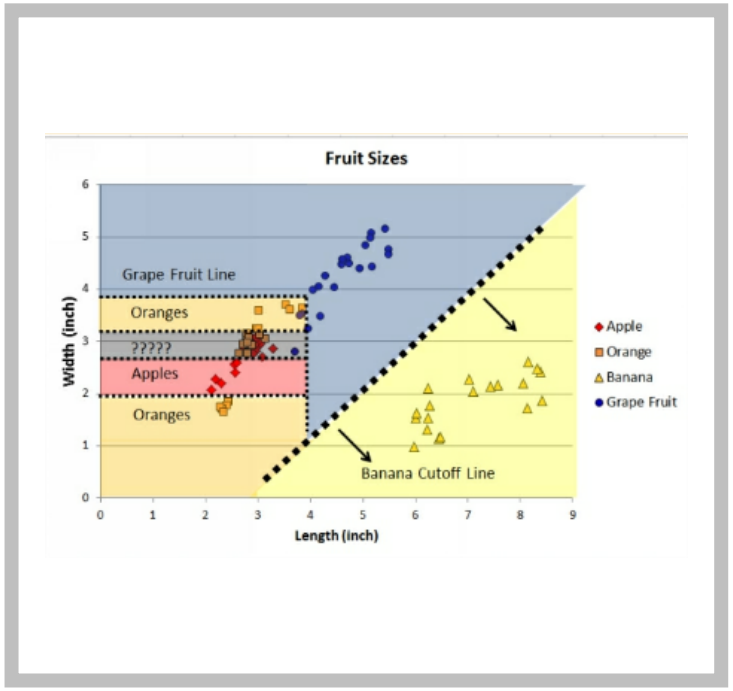
- Is it yellow?
- If so, is it long and skinny?
- If so, is it easy to peel?
- Then it is a banana

Like the diagram to follow through making a decision.



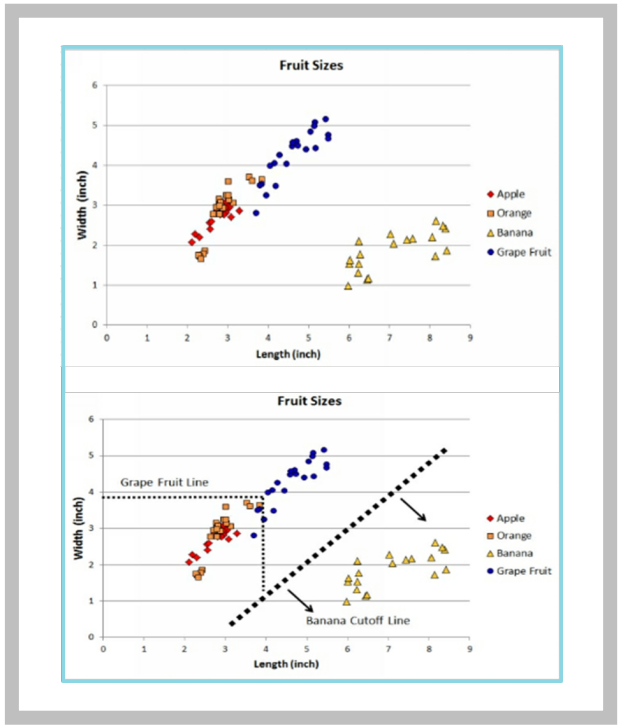
A graphic presentation of how decision tree make classification:

The plot was created by generating a decision tree and then testing thousands of length, width combinations and shading each location based on its result from the decision tree. There is also a coarser plot of a few hundred dots generated the same way. Hartshorn, Scott.



A partial separation
By 2 dimensional features

A complete separation:
By 2 dimensional features

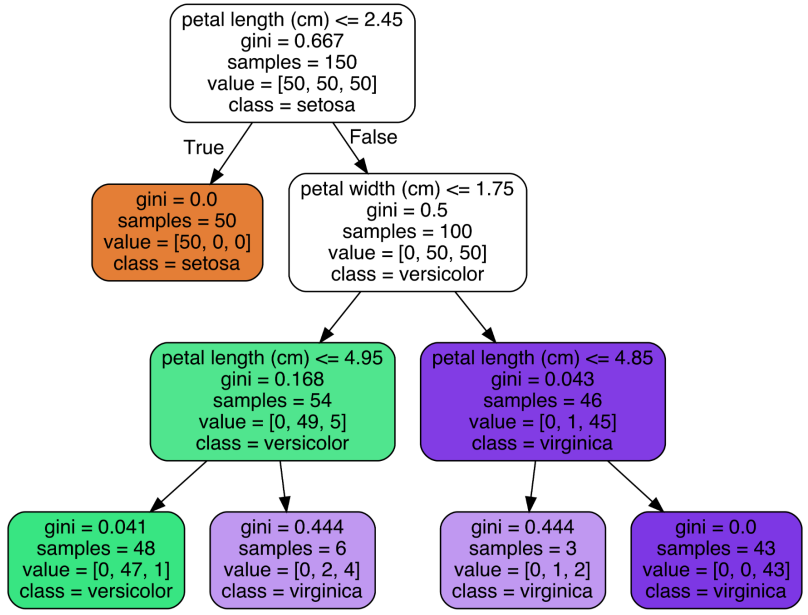


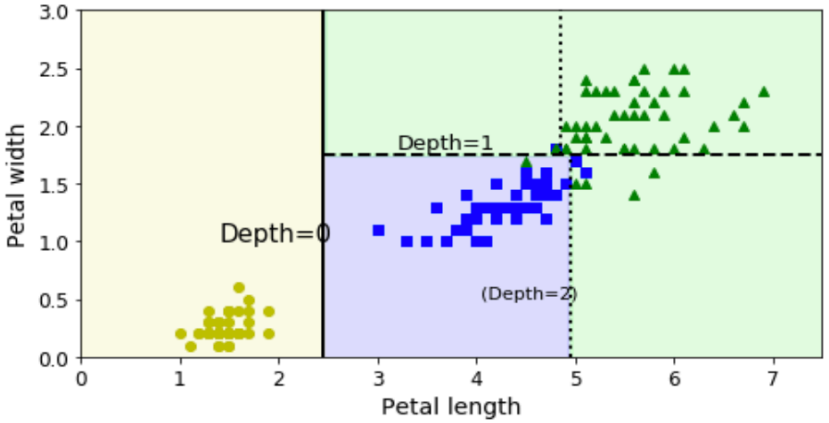
LAB 1: the laboratory is let get similar with Random Forest Algorithms

Example: A Decision Tree

Python Code

```
Class =  ['setosa' 'versicolor' 'virginica'] Features =  ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Scores =  0.9733333333333334
```





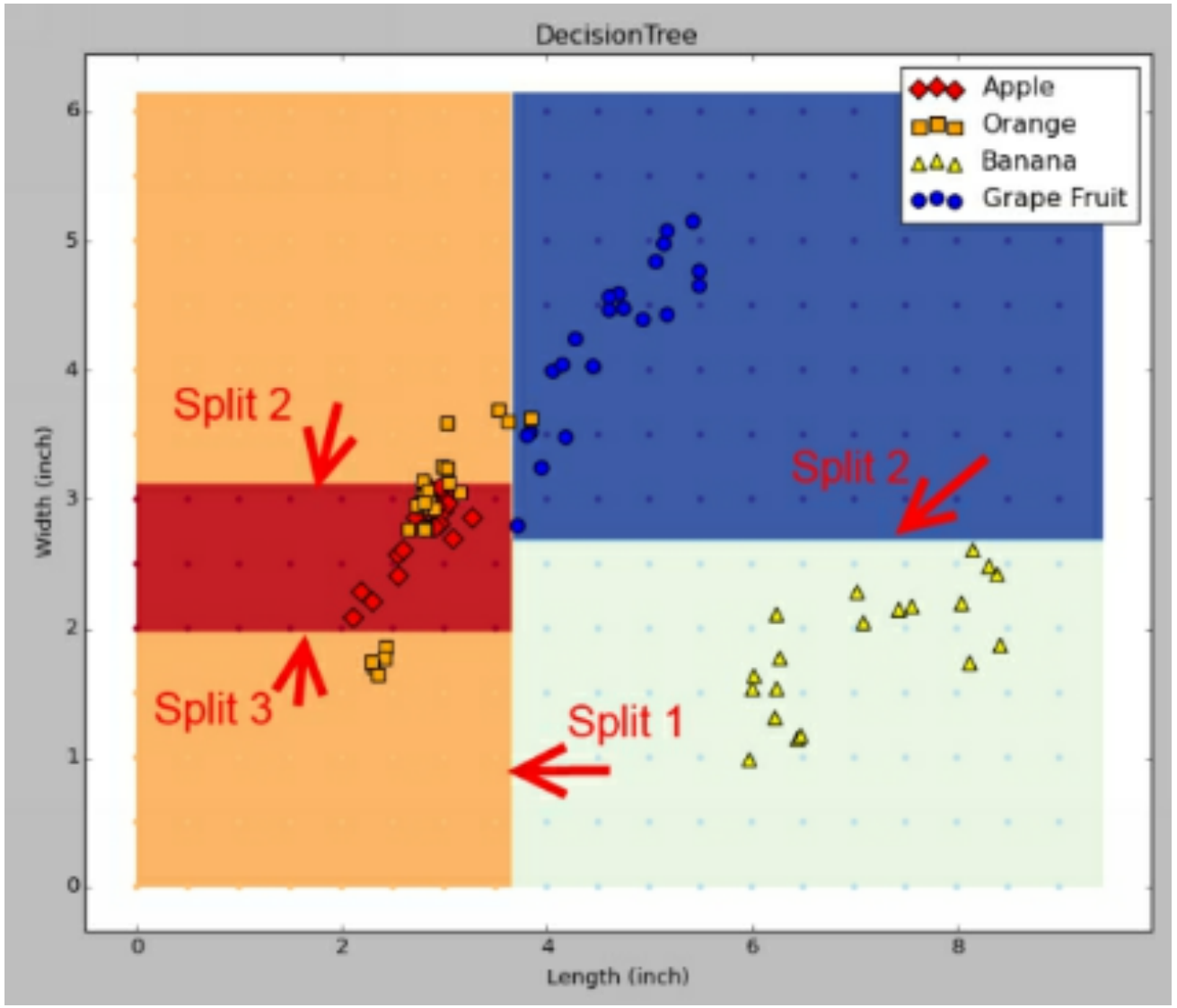

```python
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris()
X = iris.data[:, 2:] # petal length and width
y = iris.target



tree_clf = DecisionTreeClassifier(max_depth=3, random_state=42)
tree_clf.fit(X, y)

from sklearn.tree import export_graphviz
# Export as dot file
export_graphviz(tree_clf, out_file='tree_clf.dot',
                feature_names = iris.feature_names[2:],
                class_names = iris.target_names,
                rounded = True, filled = True)

# Convert to png using system command (requires Graphviz)
from subprocess import call
call(['dot', '-Tpng', 'tree_clf.dot', '-o', 'tree_clf.png', '-Gdpi=600'])
scores = tree_clf.score(X, y)
print(f"Scores = ",scores)
# Display in jupyter notebook
from IPython.display import Image
Image(filename = 'tree_clf.png')
```

Decision Tree: Pros and Cons

- Pros
  - Little data preparation, no need feature rescaling or centering
  - Simple efficiency parameter of impurity, Gini impurity, 1-sum(probabilities within groups)^2
  - Many package can generate decision tree with simple code (Scikit-learn produces only binary node trees whereas algorithims like ID3 generate Decision Trees with more than 2 nodes)
- Cons
  - A optimal tree NP-Complete problem O(exp(m)) time, making it intractable even fairly small training sets (m is number of instance)
  - Computational complexity can be reduced by (set presort= True)
  - Overfitting: regularization hypermeters (max_depth, min_samples_split, min_samples_leaf, and Mex_features)reduce the problem
  - Instability or sensitivity to training set details

A Decision Tree overfit example

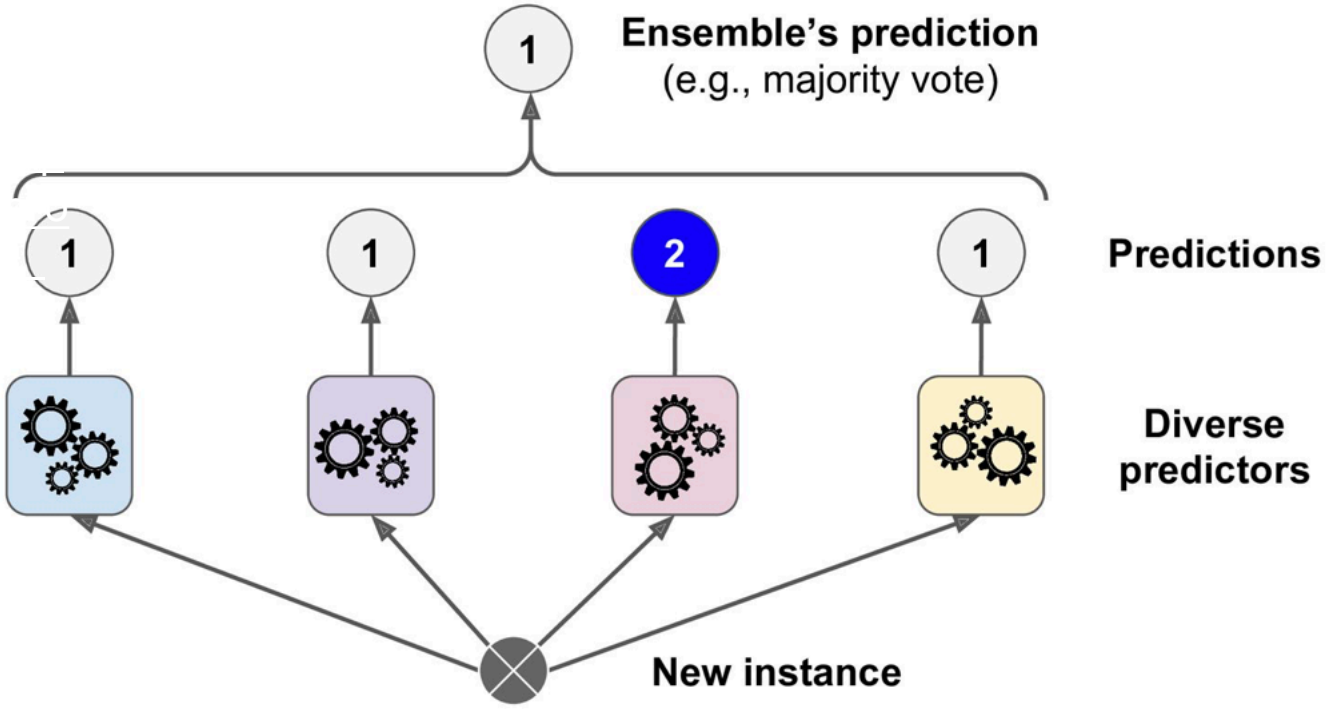
Decision tree is a kind of step-wise (layers) decision algorithm



4 step-wise single feature decision to separate the four classes

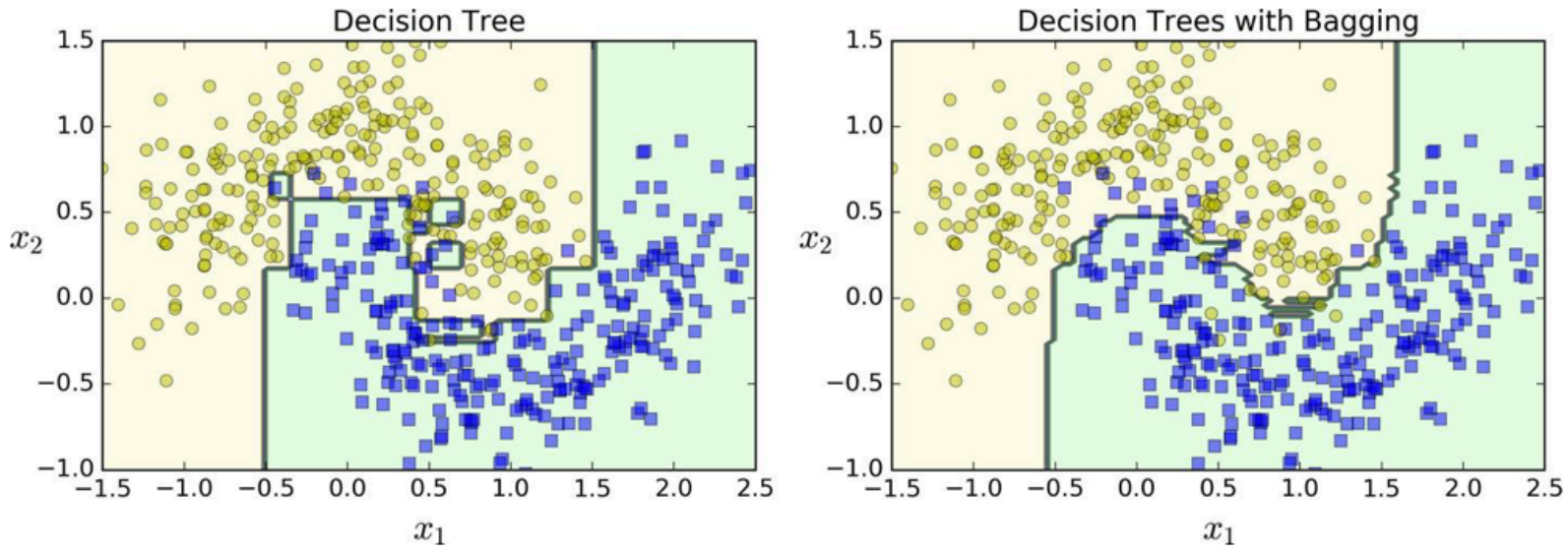
Random Forests and Voting

- Suppose you learn so many ML algorithms, how you could get a good results, one of the way is to average the output of many ML models, Random Forests apply this principle to compare multiple decision trees



A very simple way to create an even better classifier is to aggregate the predictions of each classifier and predict the class that gets the most votes. This majority-vote classifier is called a hard-voting classifier

Compares the decision boundary of a single Decision Tree with the decision boundary of a bagging (sampling with replacement) ensemble of 500 trees both trained on the moons dataset. As you can see, the ensemble's predictions will likely generalize much better than the single Decision Tree's predictions: the ensemble has a comparable bias but a smaller variance (it makes roughly the same number of errors on the training set, but the decision boundary is less irregular).

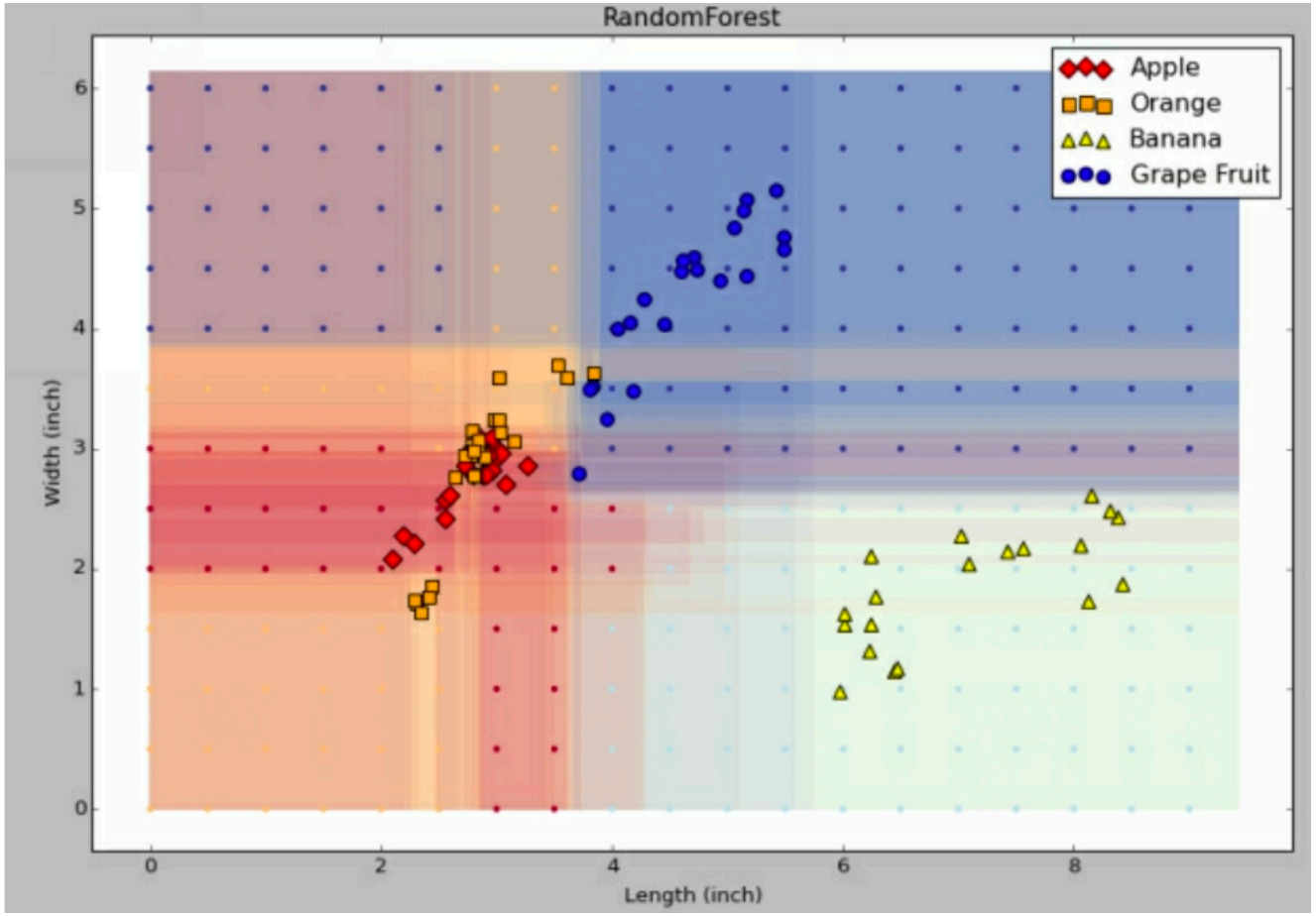


Random Forest Generation Algorithm

- A Random Forest is made up of a number of decision trees, each of which has been generated slightly differently.
- A Random Forest is an ensemble of Decision Trees, generally trained via the bagging method (or sometimes pasting), typically with max_samples set to the size of the training set. The RandomForestClassifier class, which is more convenient and optimized for Decision Trees (similarly, there is a RandomForestRegressor class for regression tasks).
- The Random Forest algorithm introduce extra-randomness when growing decision trees, instead of searching for the very best features when splitting a node, it search for the best features amongst a random subset of feature.
- Still with decision tree problem
  - The prediction is based on features set in the algorithm. Even the algorithm only used 16 of the 20 features (for instance), it likely identified which features it was used in their ranks in the array to determine lead to decisions.
  - If drop features, even ones that were not used, the implementations of algorithm will be affected.

How is a Random Forest Different

- A decision tree will not smooth out anomalies, random Forest attempt to fix the problem by using multiple decision tree and averaging the result. The algorithms generate their decision trees using subsets of the full data that are randomly selected.
- The most obvious difference between the random forest plot and the decision tree plot is that the colors are not pure anymore.



Randomness in a Random Tree

- Sampling
  - Bootstrapping technique is applied. Each of the trees uses a set of data that is the same size of the original dataset.
  - By assigning each data point with an equal chance being drawn to the sub sample. When a data point is picked to be included in the sample, it will be placed back to the original sample, to have the same change to be picked again.
  - If one run this random sampling enough times, it can be calculated as $1-(1-1/N)^N$, N= 1000, this equals 63.2%
- Criteria Selection
  - A random forest adds randomness to a decision tree is deciding which feature to split the tree on.
  - Any any given branch in the decision tree, only a subset of the features are available for it to classify on.
  - In many algorithm, it will use the square root of the number of features as the maximum features that it will look on any given branch.
- Number of estimator (number of trees) = 10 by default, recommend 100

Out of Bag Errors and Cross-Validation

- One can run the 10% set aside through classifier and find out if those predictions match the actual values.
- Since out of bag data for a given tree, one can use that data to check the quality of each tree. The average error over the tree will be our out of bag error.
  - For example, let's say that we have 10 data points in our Random Forest and that it is made up of 15 trees. On average each of our points will be present in approximately 2/3 of the trees, which means they will be out of bag in approximately 1/3 of the trees.
  - That means we expect any given data point to be out of bag for 5 trees, although some will likely be out of bag for only 3 or 4 trees, and others will be out of bag for 6 or 7.
  - Now imagine that we have 3 different categories we are trying to classify our data points into, category A, B, C. For each of the 10 data points, we "Predict" them, i.e. try to classify them using the Random Forest, but we only include 4 trees where they are out of bag for that prediction.
    - Point 1 is actually category A. It is out of bag for 5 trees. When tested on those 5 trees, 3 trees predict category A, 1 predicts B, 1 predicts C. CORRECT
    - Point 2 is actually category A. It is out of bag for 7 trees. The results are 3 A, 2 B, and 2 C. The most common category here is also A, even though it is not a majority. CORRECT
    - Point 3 is category B. It is out of bag for 3 points. The results are 1 B, 2 C. INCORRECT
    - Point 4 is category B. It is out of bag for 4 points, the results are 2 B, 2 C. This one is difficult because there is a tie, so it depends on how the software is implemented. UNKNOWN
  - OUT Of BAG Validation is approximately 67.5%

The most frequent cross validation I have seen is to set aside 10% of the data 10 times and train on the other 90% each of those times, averaging the results. If you do this, all the data gets set aside once, and cross-validation multiple times will give you a consistent way of identifying any improvements in your model.

Hartshorn, Scott. Machine Learning With Random Forests And Decision Trees

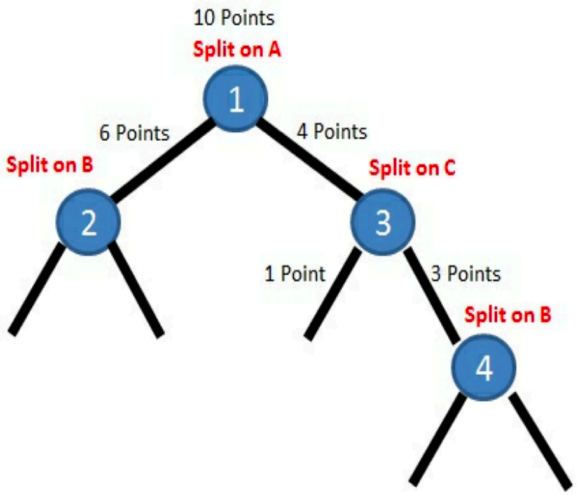
Feature Importance

- One of the interesting things that can be done with Random Forests is to tell one the relative importance of different features in your data.
- When one starts to get a large number of features, it is useful to know which ones are the most important so you can focus your attention on them.
- Intuitively one probably knows that subjectively the metrics of the UX is the least important feature of all, but what about the important and the most cost-efficiency boost one.

Idea behind the function to Extract Important Features

- For a single tree in the Random Forest, the algorithm works as follows
  - Start with an array that is the same size as the number of features in your model. Initialize that array to zero.
  - Begin to traverse the tree, with the data that was used to build the tree.
  - Whenever you reach a branch in the tree, determine what feature that branch operated on.
  - Recall that every branch splits on one and only one feature. Additionally determine how many data points reached this branch, as opposed to following a different path down the tree
  - Calculate the information gain after branching as opposed to before the branch. This is true regardless of the method used to determine information gain (e.g. Gini, Entropy, MSE, …)
  - Multiply the information gain by the number of data points that reached this branch, and add that product to the array at whatever feature is being split on Once all the information gain for all the branches are summed, normalize the array

Illustration: how the algorithm figure out the feature importance



Feature Importance

Split #	# of Data Points	Split on Which Feature	Information Gain	# of Nodes * Information Gain
1	10	A	0.26	2.6
2	6	B	0.4	2.4
3	4	C	0.3	1.2
4	3	B	0.1	0.3

Importance

Feature	Importance	Normalized
A	2.60	0.40
B	2.70	0.42
C	1.20	0.18

Final Random Forest Tips

- Use cross-validation or out of bag error to judge how good the fit is
- Better data features are the most powerful tool you have to improve your results. They trump everything else Use feature importance to determine where to spend your time
- Increase the number of trees until the benefit levels off Set the random seed before generating your Random Forest so that you get reproducible results each time
- Use a predict() or a predict_proba() where it makes sense, depending on if you want just the most likely answer, or you want to know the probability of all the answers
- Investigate limiting branching either by
  - Number of splits
  - Number of data points in a branch to split
  - Number of data points in the final resulting leaves
- Investigate looking at different numbers of features for each split to find the optimum. The default number of features in many implementations of Random Forest is the square root of the number of features. However other options, such as using a percentage of the number of features, the base 2 logarithm of the number of features, or other values can be good options.

Hartshorn, Scott. Machine Learning With Random Forests And Decision Trees

Clustering: Unsupervised Learning

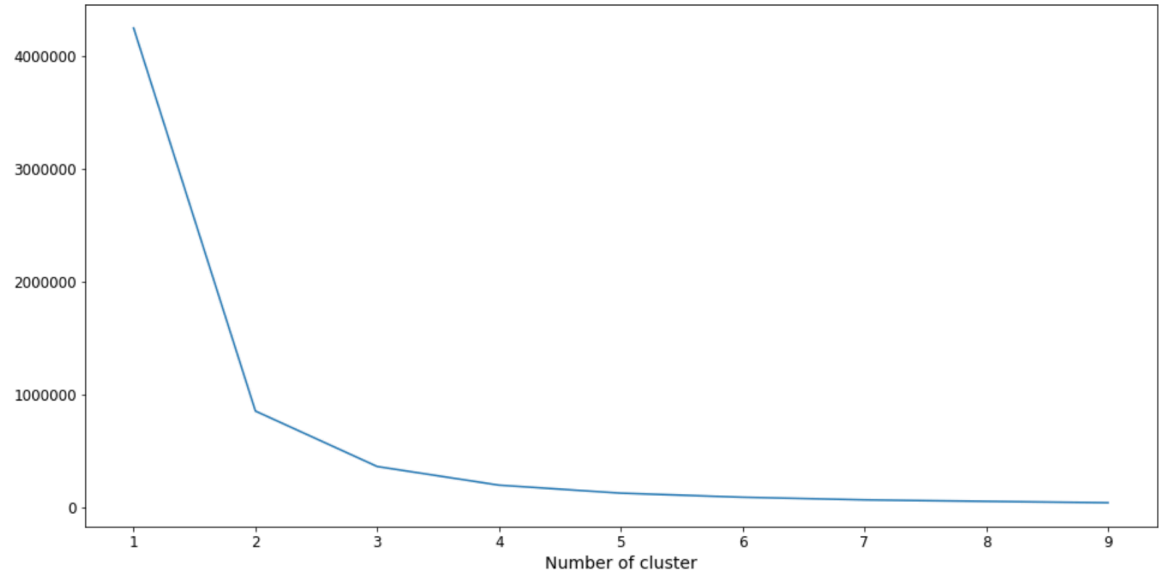
- Unsupervised Learning is said to embody the essence of Artificial Intelligence. That's because there's not much human supervision or intervention. As a result, the algorithms are left on their own to discover things from data. This is especially the case in Clustering wherein the goal is to reveal organic aggregates or "clusters" in data.
- The method is to look for structures in the data. "Dividing" the dataset into groups wherein members have some similarities or proximities. For example, each ecommerce customer might belong to a particular persona group (e.g. browse ring and purchase pattern).
- One way to make sense of data through Clustering is by K-Means. It's one of the most popular Clustering algorithms because of its simplicity. It works by partitioning objects into k clusters (number of clusters we specified) based on feature similarity.

How to run the Clustering Calculation

```
def order_cluster(cluster_field_name, target_field_name,df,ascending):
    new_cluster_field_name = 'new_' + cluster_field_name
    df_new = df.groupby(cluster_field_name)[target_field_name].mean().reset_index()
    df_new = df_new.sort_values(by=target_field_name,ascending=ascending).reset_index(drop=True)
    df_new['index'] = df_new.index
    df_final = pd.merge(df,df_new[[cluster_field_name,'index']], on=cluster_field_name)
    df_final = df_final.drop([cluster_field_name],axis=1)
    df_final = df_final.rename(columns={"index":cluster_field_name})
    return df_final
```

The number of clusters is often arbitrary, there are ways to find that optimal number. One such way is through the Elbow Method and WCSS (within-cluster sums of squares).

WCSS



Usually Visual Presentation