Big Data & Predictive Analytics
Lecture notes on Random Forest

School of Informatics
University of Leicester

1 Introduction

Classification is a type of problems that are frequent in predictive analytics. Random forest is one of the most powerful algorithm to solve this type of problem. What is more is that this algorithm scale well with big data (large-scale datasets). As the name suggests, it builds up a forest of random trees with each tree representing a *decision tree*. A decision tree is a supervised learning algorithm to classify an outcome that is a categorical or discrete variable. This can can be viewed as as set `if-then` rules encoded as a tree.

As an example, assume we would like to predict the temperature at Leicester in 3 days time. For any given day of the year, there is a wide range values of temperatures at Leicester from say -10^oC to 40^oC. However asking questions like:

- Which season are we in?
- What is the temperature today?
- What is the historical average temperature for that particular day?

will enables us to narrow down a good estimate of the temperature on that day. This gives us some kind of decision tree. We may repeat this process by changing the order of the questions for example to build up various decision trees; each leading to an estimate of the temperature on that day. We can predict the temperature of that day by **averaging** all the estimates obtained by the various decision trees. This is the idea of the **random forest algorithm** in a nutshell. Note that in this example we are predicting a numerical value so we are doing a regression. In Section 3, we will use an example for classification. The building block of the random forest algorithm is decision tree; another important concept is **bootstrapping**.

2 Bootstrapping estimation

The Bootstrapping method relies on **repeated re-sampling with replacement** of the initial sample as the basis of estimation. An example of bootstrap method to estimate the population mean consists of the following steps:

1. Select a random sample of size n from a population of size N.
2. Re-sample the initial sample by selecting n values with replacement from the initial sample, and compute the sample means for this re-sample.
3. Repeat the step 2 m number of times to produce m re-samples and their associated means.
4. Construct the distribution of the obtained m sample means.
5. Sort the obtained sample means in increasing order.
6. Find the values that exclude the smallest $\frac{\alpha}{2} \times 100\%$ of means and the largest $\frac{\alpha}{2} \times 100\%$ of the means. These values become the lower and upper limits of the bootstrap confidence interval estimate of the population mean with $100(1-\alpha)\%$ confidence.

This gives us an algorithm to estimate a confidence interval in the case where the population cannot be assumed to be normally distributed. This algorithm can be implemented in a programming language such as Python in order to carry out the bootstrapping method.

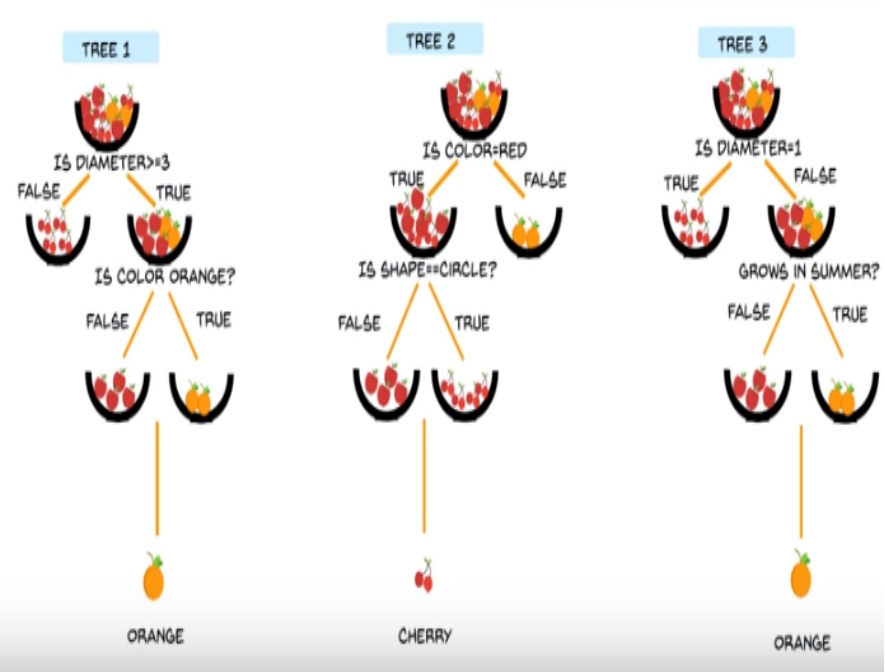
Bootstrap Example: Consider the original dataset formed by

						Mean
1	2	3	4	5	6	3.5
5	2	3	1	4	**5**	3.33
4	2	**4**	3	**4**	1	3.00
6	**2**	4	5	3	**2**	3.67
6	**6**	**1**	2	3	**1**	3.17
...						...

We repeat this process say 30 times to get 30 mean values to be sorted and get the confidence interval as described in the algorithm.

3 Decision trees

A decision tree is a supervised learning algorithm that can be used for both regression and classification. In a decision tree, the root node is at the top and the leaves (terminal nodes) are at the bottom. An internal node has at least one child. A leaf node is assigned a class or a target value. There are different ways of splitting a dataset into branch-like segments. The following figure taken from `https://www.youtube.com/watch?v=eM4uJ6XGnSM` illustrates examples of decision trees:



If we want to recognize the type of fruit in a bucket, we can successively ask questions about the colour, the shape, the diameter, etc. Depending on which order we ask those questions, we get different decision trees. The *majority vote* will gives us the fruit type. To increase reliability of our prediction, we can construct more trees with various types of questions.

1. **Entropy** is a measure of randomness or unpredictability in the dataset. It can be used to measure impurity or heterogeneity in the random forest. A zero entropy points to homogeneous trees while higher entropy indicates more heterogeneity. For the decision tree, we need to minimize entropy by adding nodes that result in information gain.
2. **Information gain** is a measure of decrease in entropy when the dataset is split.
3. **Gini Index** is a measure of purity (homogeneity) with zero meaning homogeneous trees in the forest and one meaning heterogeneous ones. This can be used to find the best split of tree nodes by calculating the Gini index of all the candidate variables. A variable with a higher Gini index is selected for the split.

4 Random Forest

Random Forest is an **ensemble** learning method for decision trees. An *ensemble method* combines multiple learning algorithms (possibly weak in terms of predictive accuracy) in order to obtain a predictive algorithm with higher accuracy. The final result is an aggregation of the results from each learning algorithm. Why this is a good idea? If the base learning algorithms are classifiers, and if we have 25 base classifiers; each classifier with an error rate of $\epsilon = 0.3$. Assuming independence among classifiers, the probability that the ensemble classifier makes a wrong prediction can be calculated using the binomial distribution as follows:

$$\sum_{k=13}^{25} \epsilon^k (1-\epsilon)^{n-k}$$

In general, Random forest works as follows:

- **Training**: given a dataset S, at each iteration i, a training set S_i is sampled from S by using Bootstrapping (sampling with replacement) with some randomization.
- **Classification**: a classifier C_i is learned for each S_i so that for an unseen sample X, each classifier C_i returns its class prediction and and the **class with most votes** is assigned to X.
- **Regression**: a regressor R_i is learned for each S_i so that for an unseen sample X, each regressor R_i returns its predicted value and the average value over all predictions is assigned to X.

For the rest of these notes, we focus on the Random forest algorithm for classification.

4.1 The Algorithm

The Random forest algorithm for classification can be described as follows:

```
1  Given a training set S
2  for i = 1 to k
3        do
4            Build subset S_i of size n by sampling with replacement from S
5            Build a decision tree T_i from S_i
6            for each node:
7                 do
8                     Choose best split from random subset of F features
9            Each tree grows to the largest extend, and no pruning
10 Make predictions according to majority vote of the set of k trees.
```

This algorithm loops a specified number k of times wherein k can be understood as the the number of trees in the forest. For each iteration, a random sample of size n with replacement is built-up, then a random subset of the dataset's features is selected in order to create a decision tree classifier. The decision tree is let to grow without any kind of pruning.

The combination of many decision trees lead to a non transparent 'black-box' classifier with no compact representation. But the Random forest algorithm has the following advantages among others:

- It provides an improved accuracy over many of the current classification algorithms.
- It runs efficiently on large datasets; this is more adequate for big data analytics.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.
- It can deal effectively with datasets when a large proportion of the data are missing by maintaining accuracy.

4.2 Using Random Forest in Python

Consider the publicly available `iris` dataset, which has information about flower species, their petal and sepal dimensions. The aim is to classify flowers according to their species by using features such as their petal and sepal dimensions.

1. Let us first use a decision three classifier from within Python

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

file = 'iris.csv'
df = pd.read_csv(file)
df['Species'].unique()
```

This shows us that there are 3 different species in the dataset:

```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

We can then split the data into training and test sets and them build the classifier as follows:

```
train, test = train_test_split(df, test_size=0.25)
features = df.columns.tolist()
predictors = features[0:4]
target = features[4]
# build the model
dt = DecisionTreeClassifier(min_samples_split=2, random_state=0)
dt.fit(train[predictors], train[target])
```

This outputs the following:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

Now we can test our classifier on unseen data as follows:

```
# test the model
preds = dt.predict(test[predictors])
pd.crosstab(test[target], preds, rownames = ['Actual'], colnames = ['Predicted'])
```

We get the following classification result:

Predicted / Actual	setosa	versicolor	virginica
setosa	13	0	0
versicolor	0	15	2
virginica	0	0	8

This gives us a predictive accuracy of about 0.947.

2. Let us now use the Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
#creates the classifier
crf = RandomForestClassifier(n_jobs=2, n_estimators=10, random_state=0)
#train the data
crf.fit(train[predictors],y)
# test the model
preds = crf.predict(test[predictors])
pd.crosstab(test[target], preds, rownames = ['Actual'], colnames = ['Predicted'])
```

This gives us similar result as in using decision tree classifier. This is a small dataset though and it did not enables us to notice the power of Random forest. Nonetheless, we have have learned how to use both tools for classification.

We can also view the predicted probabilities of given observations as follows (below the first 2 observations in the test set):

```
crf.predict_proba(test[predictors])[0:2]
array([[0., 1., 0.],
       [0., 1., 0.]])
```

Through majority vote, `versicolor` has been predicted in both observations. Finally, the created classifier looks like below:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=2,
            oob_score=False, random_state=0, verbose=0, warm_start=False)
```

4.3 Evaluation

If we use Random forest for classification, then metrics such as accuracy or `oob_score` (*out-of-bag* score) should be used to assess how good is the resulting classifier. The out-of-bag score is a way to validate the Random forest model. To some extent, it is similar to the cross-validation score. In the case of regression, we may be interested into the mean squared error: *the sum of squared differences between the predicted and actual values divided by the number of data points*. Other metrics of the Random forest regressor include the coefficient of determination r^2 or the `oob_score`.

```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_jobs=2, oob_score=True, n_estimators=10)
rf.fit(X,Y)
print(rf.oob_score_) #oob_score
#Mean squared error
df['pred'] = rf.oob_prediction_
(n,m) = df.shape
error = sum((df['pred']-df['actual'])**2)/n
```