



# Northeastern University

## College of Professional Studies

### **Module 3 Assignment**

**Shivam Chauhan**

January 31, 2020

**Class: ALY 6020 – Predictive Analytics**

**Professor: Dr. Marco Montes de Oca**

**Topic: Fashion items recognition with random forests and gradient boosting.**

# Introduction

Decision Tree learning is a supervised learning method which can be used as a classification tree to predict classes or as a regression tree to predict values. Decision Tree's goal is to create a model that predicts the value of a target variable based on several input variables. It partitions the target variable based on rules which are derived from "if else" rule.

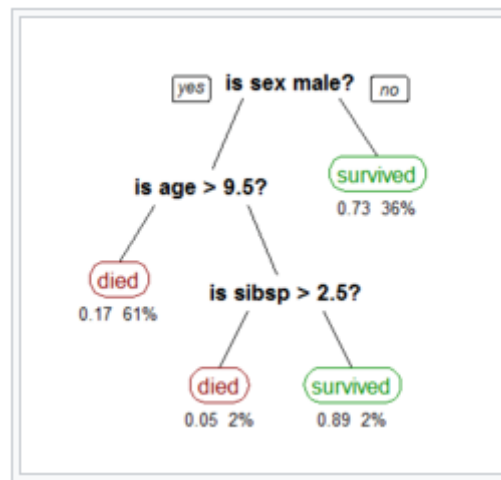


Fig: Example of a decision tree showing who survived or died based on sex, age, and sibsp.

There are some disadvantages of using the decision tree. Decision trees are prone to overfitting, it gives most optimal solution but not globally optimal solution. This means it gives different results when we make a different decision tree on same data. Due to this reason they do not have same predictive accuracy or better accuracy than other models.

To overcome these problems, we use ensemble methods. Ensemble learning is a machine learning paradigm where multiple models (often called "weak learners") are trained to solve the same problem and combined to get better results (Medium, 2019). The main aim of using ensemble methods is to create a model which has low bias and low variance.

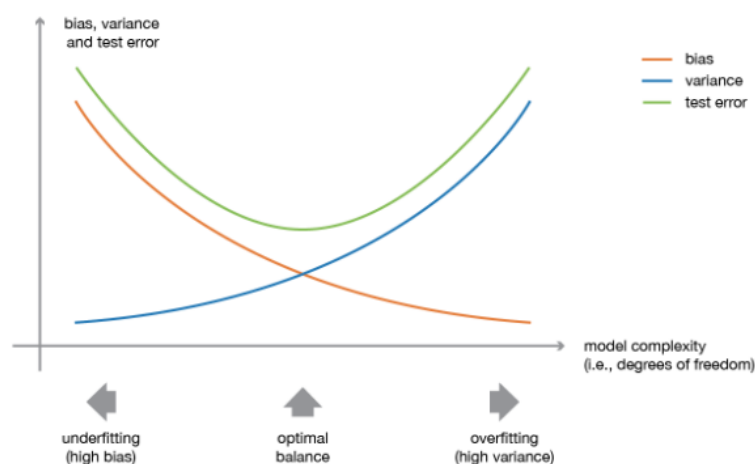
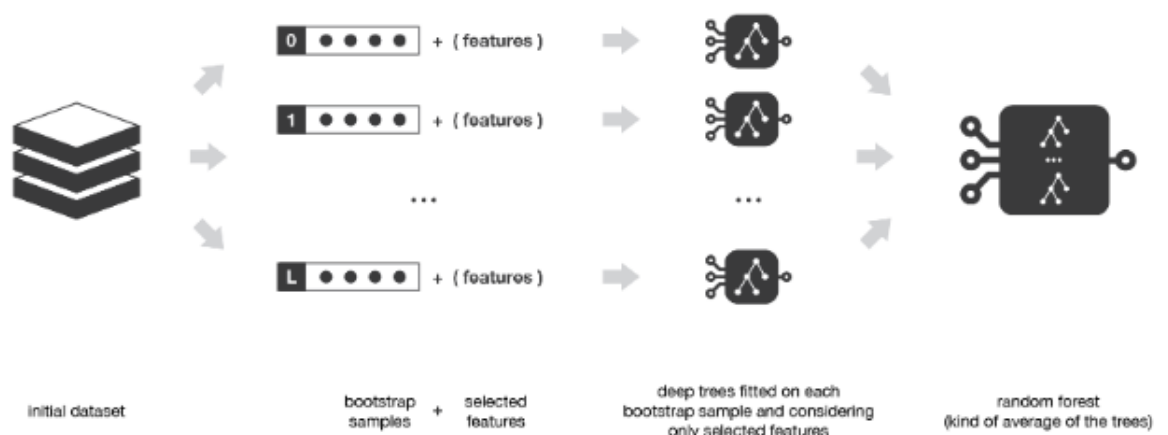


Illustration of the bias-variance tradeoff.

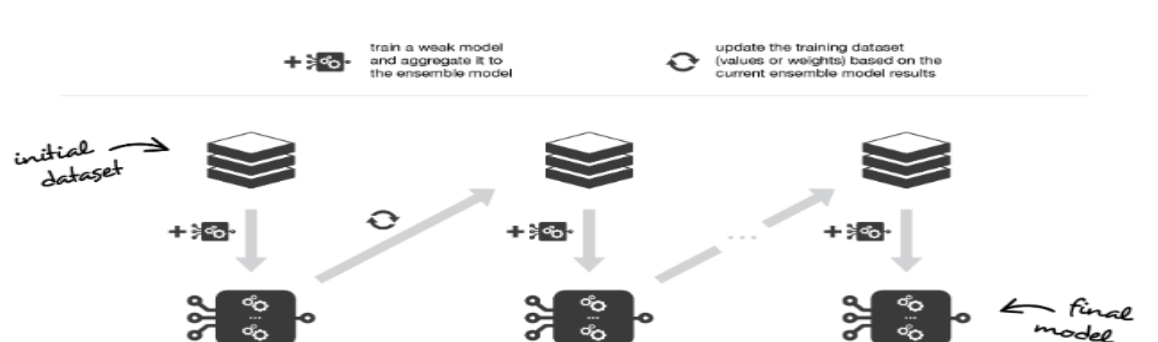
There is different type of ensemble techniques:

1. **Bagging:** In this method, many weak learners like different decision trees are randomly created with random samples and then the results are the average of results predicted by different decision trees. All these trees or weak learners are independent from each other. One of the examples of bagging algorithm is Random Forest.

The random forest approach is a bagging method where deep trees, fitted on bootstrap samples, are combined to produce an output with lower variance (Medium, 2019). So, it tries to overcome problem of a Low bias and High Variance.

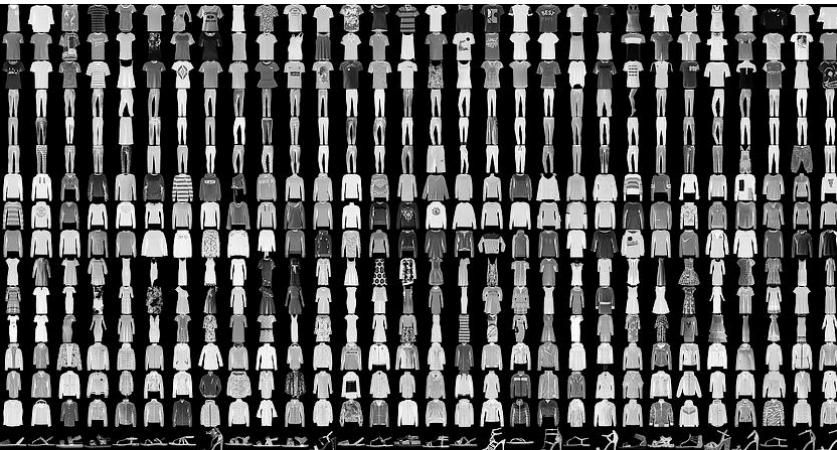




2. **Boosting:** In this method, weak learners are created in sequence. Each weak learner learns from the previous weak learners. This means it tries to improve the predicted output based on the different weak models' predictions. One of the Boosting technique is Gradient Boosting. In gradient boosting, the ensemble model we try to build is also a weighted sum of weak learners. It tries to minimize the loss/cost function which is the squared error between the desired output and predicted output.



Boosting consists in, iteratively, fitting a weak learner, aggregate it to the ensemble model and "update" the training dataset to better take into account the strengths and weakness of the current ensemble model when fitting the next base model.

The main objective of this assignment is to make a prediction model using Random Forest and Gradient Boosting that recognize the fashion items available to us as the data of images which is encoded as a row of 784 integer values between 0 and 255 indicating the brightness of each pixel. Data files contains thousands of 28x28 grayscale images. The label associated with each image is encoded as an integer value between 0 and 9.

Label	Description	
0	T-shirt/top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandal	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boot	

## Data Preparation

```
#library used
import pandas as pd
import os
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.ensemble import GradientBoostingClassifier
```

```

# defining the column names
name=[]
for i in range(785):
    name.append(i)

#Loading the data and looking into the data type
df = pd.read_csv("fashion_train.csv" , names = name)
df_test = pd.read_csv("fashion_test.csv", names = name)

"""Preparing the train and test data for modeling """
x_train = np.array(df.iloc[:,1:])
x_test = np.array(df_test.iloc[:,1:])
y_train = np.array(df[0])
y_test = np.array(df_test[0])

```

## Random Forest

Random forest with 1 tree:

```

In [51]: """ Random forest with 1 tree """
...: rfc1 = RandomForestClassifier(n_estimators= 1, random_state=0)
...: rfc1.fit(x_train,y_train)
...: y_pred1 = rfc1.predict(x_test)
...: accuracy_1tree = accuracy_score(y_test,y_pred1)
...: accuracy_1tree
...:
...:
Out[51]: 0.7585

```

Random forest with 10 trees:

```

In [12]: rfc10 = RandomForestClassifier(n_estimators= 10,
random_state=0)
...:
...: rfc10.fit(x_train,y_train)
...:
...: y_pred10 = rfc10.predict(x_test)
...:
...:
...: from sklearn.metrics import accuracy_score
...:
...: accuracy_10tree =
accuracy_score(y_test,y_pred10)
...: accuracy_10tree
Out[12]: 0.8529

```

Random forest with 100 trees:

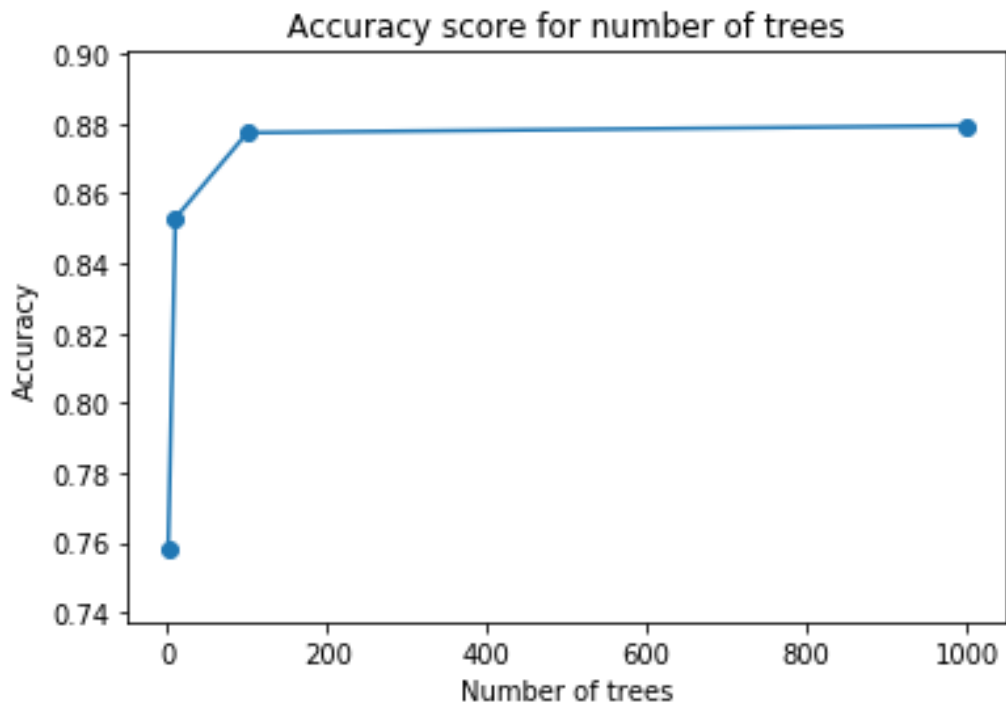
```
In [13]: rfc100 = RandomForestClassifier(n_estimators=
100, random_state=0)
...:
...: rfc100.fit(x_train,y_train)
...:
...: y_pred100 = rfc100.predict(x_test)
...:
...:
...: from sklearn.metrics import accuracy_score
...:
...: accuracy_100tree =
accuracy_score(y_test,y_pred100)
...: accuracy_100tree
...:
Out[13]: 0.8774
```

Random forest with 1000 trees:

```
In [14]: rfc1000 = RandomForestClassifier(n_estimators=
1000, random_state=0)
...:
...: rfc1000.fit(x_train,y_train)
...:
...: y_pred1000 = rfc1000.predict(x_test)
...:
...:
...: from sklearn.metrics import accuracy_score
...:
...: accuracy_1000tree =
accuracy_score(y_test,y_pred1000)
...: accuracy_1000tree
...:
Out[14]: 0.8794
```

Number of Trees in Random Forest	Accuracy %
1	75.8 %
10	85.2 %
100	87.7%
1000	87.9%

```
"""Plotting no. of trees and their accuracy for Random Forest"""
plot=plt.scatter(y= [accuracy_1tree,accuracy_10tree,accuracy_100tree, 0.8794], x=[1,10,100,1000])
plt.plot([1,10,100,1000],[accuracy_1tree,accuracy_10tree,accuracy_100tree, 0.8794 ] )
plt.title('Accuracy score for number of trees')
plt.xlabel('Number of trees')
plt.ylabel('Accuracy')
```



In this case we can see that accuracy got better when we used 100 trees but then it got stable and does not increased more from 100 to 1000 trees. In this case result from 100 trees are appropriate to conclude the accuracy of Random Forest.

## Gradient boosting

Gradient boosting using 1 tree:

```
In [20]: gbc1 =
GradientBoostingClassifier(n_estimators=1)
...:
...: gbc1.fit(x_train,y_train)
...:
...: y_pred_gbl = gbc1.predict(x_test)
...:
...: acc_gbl = accuracy_score(y_test,y_pred_gbl)
...: acc_gbl
...:
Out[20]: 0.7191
```

Gradient boosting using 10 trees:

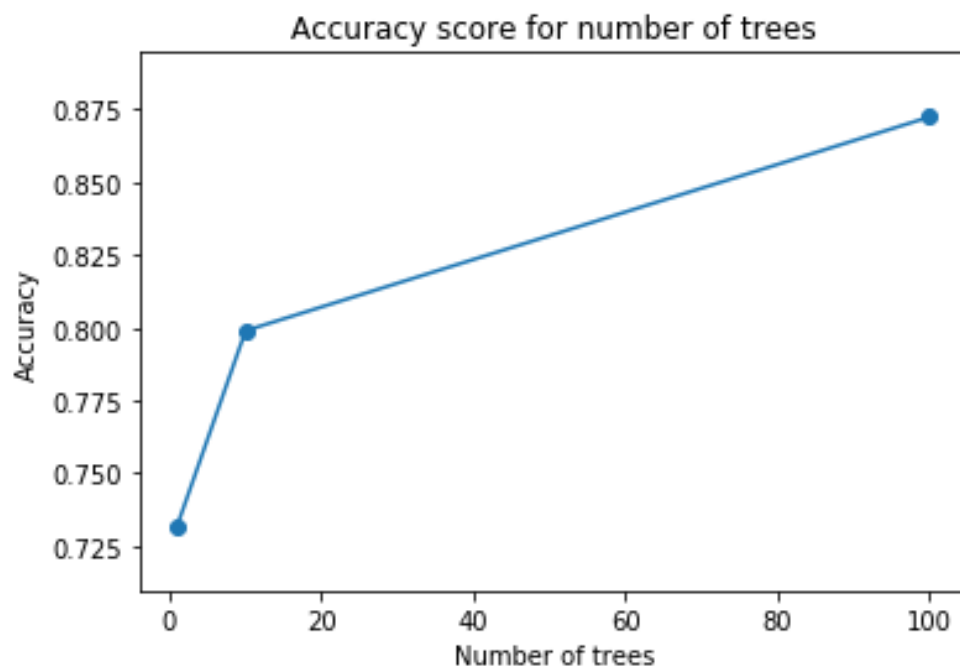
```
In [4]: gbc10 =
GradientBoostingClassifier(n_estimators=10)
...:
...: gbc10.fit(x_train,y_train)
...:
...: y_pred_gbl0 = gbc10.predict(x_test)
...:
...: acc_gbl0 = accuracy_score(y_test,y_pred_gbl0)
...: acc_gbl0
...:
Out[4]: 0.7947
```

Gradient boosting using 100 trees:

```
In [5]: gbc100 =  
GradientBoostingClassifier(n_estimators=100)  
...:  
...: gbc100.fit(x_train,y_train)  
...:  
...: y_pred_gbc100 = gbc100.predict(x_test)  
...:  
...: acc_gbc100 = accuracy_score(y_test,y_pred_gbc100)  
...: acc_gbc100  
Out[5]: 0.8668
```

Number of Trees in Gradient Boosting	Accuracy %
1	71.9 %
10	79.4 %
100	87.6%

```
"""Plotting no. of trees and their accuracy for Gradient Boosting"""  
plot=plt.scatter(y= [acc_gbc1,acc_gbc10,acc_gbc100], x=[1,10,100])  
plt.plot([1,10,100],[acc_gbc1,acc_gbc10,acc_gbc100] )  
plt.title('Accuracy score for number of trees')  
plt.xlabel('Number of trees')  
plt.ylabel('Accuracy')
```



For gradient decent we can also see the accuracy increased when we used 100 trees. Due to lack of computational capacity, I could not use 1000 trees on entire data. However, we will compare 1000 trees on a subset in later part.



## Model Comparison

This is for the entire train set. These models are trained using all 60,000 rows of data and tested on 10,000 data.

Models	Accuracy %
Logistic Regression (HW2)	83.84 %
Random Forest (100 trees)	87.7%
Gradient Boosting (100 trees)	87.5%

After comparing these 3 models we can see best model worked on the entire data was Random Forest with highest accuracy of 87.7%. Gradient decent was close and the difference between the accuracy for both is approximately 0.2%. Logistic Regression when compared to these models lack behind and was not as good model to predict fashion items.

The reason behind saying Random Forest as a good model is also because of its computational speed along with the higher accuracy. Random Forest is much faster to train than Gradient boosting.

Now let's see what happens when we have a small dataset and use 1000 trees.

## Stratify sampling

Stratified sampling to test how 1000 trees will work if we decrease the training size.

```
from sklearn.model_selection import train_test_split

#creating a new sample dataset
stratified_sample, _ = train_test_split(df, test_size=0.9, stratify=df[[0]])
#checking if it is evenly divided
df.iloc[:,0].value_counts()
stratified_sample.iloc[:,0].value_counts()
#creating train and test data
df_y = stratified_sample.iloc[:,0]
df_x = stratified_sample.iloc[:,1:]

x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, stratify=df_y, test_size = 0.3)
```

Only 1/10<sup>th</sup> of the entire data set was used to build these models and test it.

The Sample was creates using stratify sampling on the Label column. The proportion of the label/classes was same as it was in the original dataset.

Now we will train Random Forest and Gradient Boosting using 1000 trees and test it.

## Random Forest on small dataset

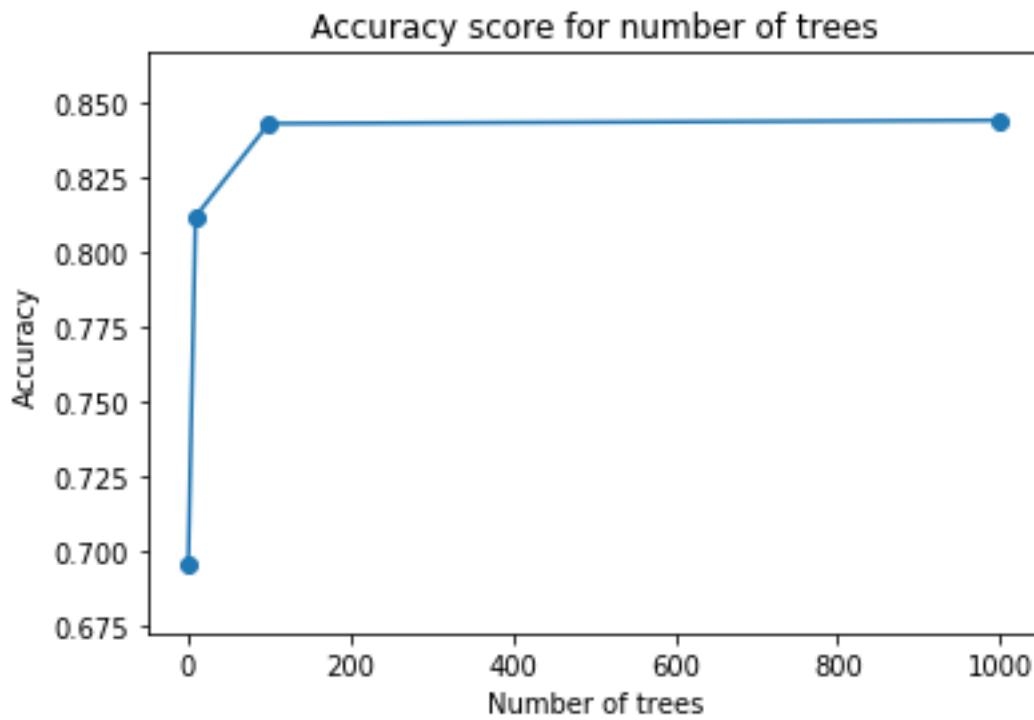
```
In [26]: """ Random forest with 1 tree """
...: rfc1 = RandomForestClassifier(n_estimators= 1, random_state=0)
...: rfc1.fit(x_train,y_train)
...: y_pred1 = rfc1.predict(x_test)
...: accuracy_1tree = accuracy_score(y_test,y_pred1)
...: accuracy_1tree
...:
Out[26]: 0.695

In [27]: rfc10 = RandomForestClassifier(n_estimators= 10, random_state=0)
...: rfc10.fit(x_train,y_train)
...: y_pred10 = rfc10.predict(x_test)
...: accuracy_10tree = accuracy_score(y_test,y_pred10)
...: accuracy_10tree
...:
Out[27]: 0.8116666666666666

In [28]: rfc100 = RandomForestClassifier(n_estimators= 100, random_state=0)
...: rfc100.fit(x_train,y_train)
...: y_pred100 = rfc100.predict(x_test)
...: accuracy_100tree = accuracy_score(y_test,y_pred100)
...: accuracy_100tree
...:
Out[28]: 0.8427777777777777

In [29]: rfc1000 = RandomForestClassifier(n_estimators= 1000,
random_state=0)
...: rfc1000.fit(x_train,y_train)
...: y_pred1000 = rfc1000.predict(x_test)
...: accuracy_1000tree = accuracy_score(y_test,y_pred1000)
...: accuracy_1000tree
...:
Out[29]: 0.8438888888888889

In [31]: """Plotting no. of trees and their accuracy for Random Forest"""
...: plot =plt.scatter(y= [0.695,
0.8116666666666666,0.8427777777777777,0.8438888888888889], x=[1,10,100,1000])
...: plt.plot([1,10,100,1000],[0.695,
0.8116666666666666,0.8427777777777777,0.8438888888888889] )
...: plt.title('Accuracy score for number of trees')
...: plt.xlabel('Number of trees')
...: plt.ylabel('Accuracy')
...:
...:
```



Number of Tress in Random Forest for our stratified sample	Accuracy %
1	69.5 %
10	81.2 %
100	84.3%
1000	84.4%

So random forest worked well for small dataset, but accuracy didn't increase much from 100 trees to 1000 trees.

## Gradient Boosting on small dataset

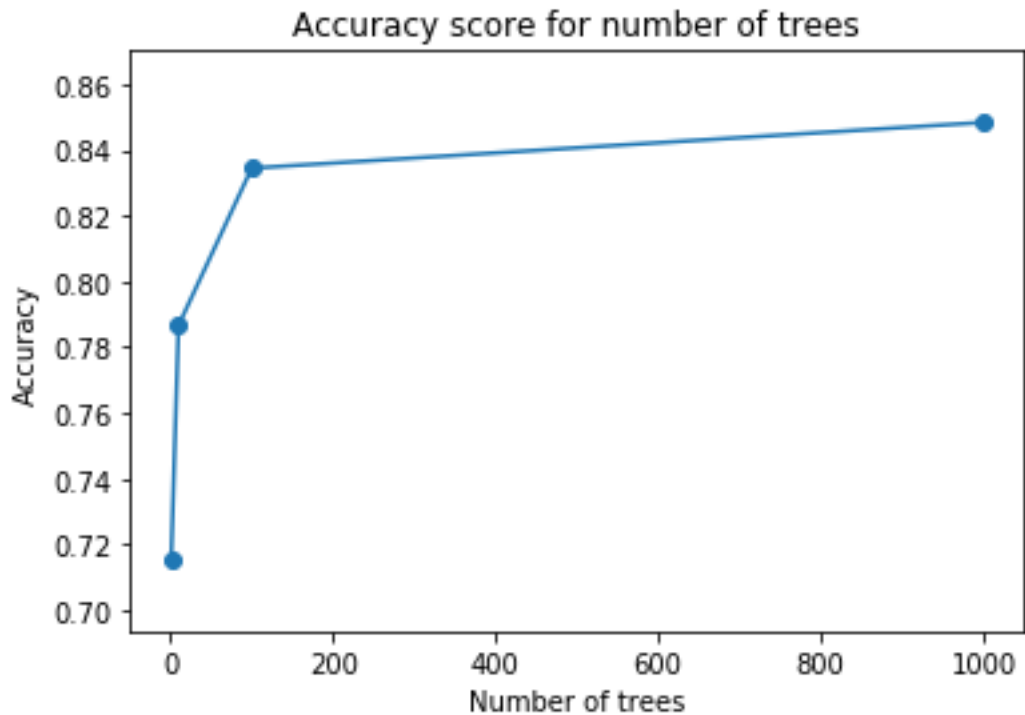
```
In [20]: gbc1 = GradientBoostingClassifier(n_estimators=1)
...: gbc1.fit(x_train,y_train)
...: y_pred_gbl = gbc1.predict(x_test)
...: acc_gbl = accuracy_score(y_test,y_pred_gbl)
...: acc_gbl
...:
Out[20]: 0.7155555555555555

In [21]: gbc10 = GradientBoostingClassifier(n_estimators=10)
...: gbc10.fit(x_train,y_train)
...: y_pred_gbl0 = gbc10.predict(x_test)
...: acc_gbl0 = accuracy_score(y_test,y_pred_gbl0)
...: acc_gbl0
Out[21]: 0.7866666666666666

In [22]: gbc100 = GradientBoostingClassifier(n_estimators=100)
...: gbc100.fit(x_train,y_train)
...: y_pred_gbl00 = gbc100.predict(x_test)
...: acc_gbl00 = accuracy_score(y_test,y_pred_gbl00)
...: acc_gbl00
...:
Out[22]: 0.8344444444444444

In [23]: gbc1000 = GradientBoostingClassifier(n_estimators=1000)
...: gbc1000.fit(x_train,y_train)
...: y_pred_gbl000 = gbc1000.predict(x_test)
...: acc_gbl000 = accuracy_score(y_test,y_pred_gbl000)
...: acc_gbl000
...:
Out[23]: 0.8483333333333334

In [32]: """Plotting no. of trees and their accuracy for Random Forest"""
...: plot=plt.scatter(y=
[0.7155555555,0.7866666666,0.8344444444,0.848333333], x=[1,10,100,1000])
...: plt.plot([1,10,100,1000],
[0.7155555555,0.7866666666,0.8344444444,0.848333333] )
...: plt.title('Accuracy score for number of trees')
...: plt.xlabel('Number of trees')
...: plt.ylabel('Accuracy')
...:
```



Number of Tress in Gradient Boosting for our stratified sample	Accuracy %
1	71.5 %
10	78.7 %
100	83.4%
1000	84.8%

We can observe that Gradient Boosting worked very well for small data set. Accuracy also increased by 1.4% which are great numbers. This shows that model got better and better when we use a greater number of trees. In this case it might be better to use more than 1000 trees if more computational speed is available.

### Model Comparison

These modes are trained on a subset of the original data. Subset is of 6000 data rows and they are divided into 70% train data and 30% test data.

Models	Accuracy %
Random Forest (1000 trees)	84.4%
Gradient Boosting (1000 trees)	84.8%

In this case, when we used small data, Gradient Boosting shows better accuracy and for small data computational speed will not be a big challenge.

## Conclusion

In conclusion, for a large dataset, Random Forest is a good model to predict the fashion items. Random Forest. In this case Random forest was fast to generate results and it is also simple algorithm for multi class classification. The only disadvantages for random forest are we do not get the rules behind which it classifies the objects as it takes average of all the trees. Models like logistic regression does only binary classification but they provide the probability that image belong to a class or not. Gradient Boosting worked well for small data set with 1000 trees, however it takes a lot of time to build models with higher trees. From this experiment we saw Gradient Boosting is giving better results when we are increasing the number of trees. On the other hand, Random Forest accuracy get's constant after 100 trees.

### Reference:

Ensemble methods: bagging, boosting and stacking. (2019). Medium. Retrieved 1 February 2020, from <https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205>

### Attachment:



final assignment.py