Boosting techniques in Python

By Eshwar. Kumar

Introduction

- In machine learning, **Boosting** is an ensemble meta-algorithm for primarily reducing bias(error), and also variance in supervised learning.

- It is one of the machine learning algorithms that converts weak learners to strong ones.

- There are three main types of Boosting:
  1. Gradient boost
  2. Adaboost
  3. XG Boost(Extreme gradient boosting)

- Let us apply these boosting techniques in a supervised Machine learning dataset "**titanic**" further.

- Before that first let us upload the dataset and do usual EDA (exploratory data analysis).

First download the necessary packages

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
import seaborn as sns
%matplotlib inline
```

Upload the dataset “titanic.csv”

In [42]:
```
titanic = pd.read_csv('titanic.csv')
```

In [43]:
```
titanic.head(10)
```

Out[43]:

	pclass	survived	name	sex	age	sibsp	parch	fare	embarked
0	1	1	Allen, Miss. Elisabeth Walton	female	29.0	0	0	211.3375	S
1	1	0	Allison, Miss. Helen Loraine	female	2.0	1	2	151.5500	S
2	1	0	Allison, Mr. Hudson Joshua Creighton	male	30.0	1	2	151.5500	S
3	1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0	1	2	151.5500	S
4	1	1	Anderson, Mr. Harry	male	48.0	0	0	26.5500	S
5	1	0	Andrews, Mr. Thomas Jr	male	39.0	0	0	0.0000	S
6	1	1	Appleton, Mrs. Edward Dale (Charlotte Lamson)	female	53.0	2	0	51.4792	S
7	1	0	Astor, Col. John Jacob	male	47.0	1	0	227.5250	C
8	1	1	Astor, Mrs. John Jacob (Madeleine Talmadge Force)	female	18.0	1	0	227.5250	C
9	1	1	Aubart, Mme. Leontine Pauline	female	24.0	0	0	69.3000	C

Analyze the dataset

In [4]:
```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1257 entries, 0 to 1256
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   pclass    1257 non-null   int64  
 1   survived  1257 non-null   int64  
 2   name      1257 non-null   object 
 3   sex       1257 non-null   object 
 4   age       996 non-null    float64
 5   sibsp     1257 non-null   int64  
 6   parch     1257 non-null   int64  
 7   fare      1257 non-null   float64
 8   embarked  1257 non-null   object 
dtypes: float64(2), int64(4), object(3)
memory usage: 88.5+ KB
```

In [5]:
```
titanic.shape # specifies the rows and columns of the dataset i.e the dimensions
```

Out[5]: (1257, 9)

In [6]:
```
titanic.describe() # DO a descriptive stats which we do in Excel by passing the describe function
```

Out[6]:

	pclass	survived	age	sibsp	parch	fare
count	1257.000000	1257.000000	996.000000	1257.000000	1257.000000	1257.000000
mean	2.310263	0.382657	29.070783	0.501989	0.377884	32.720896
std	0.831791	0.486229	12.819750	1.056616	0.863035	51.127788
min	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000
25%	2.000000	0.000000	21.000000	0.000000	0.000000	7.895800
50%	3.000000	0.000000	28.000000	0.000000	0.000000	14.400000
75%	3.000000	1.000000	37.000000	1.000000	0.000000	31.000000
max	3.000000	1.000000	60.000000	8.000000	9.000000	512.329200

Treatment of Missing values(NaN)

Treatment of missing values('NaN')

In [47]:
```
titanic.isnull().sum()
```

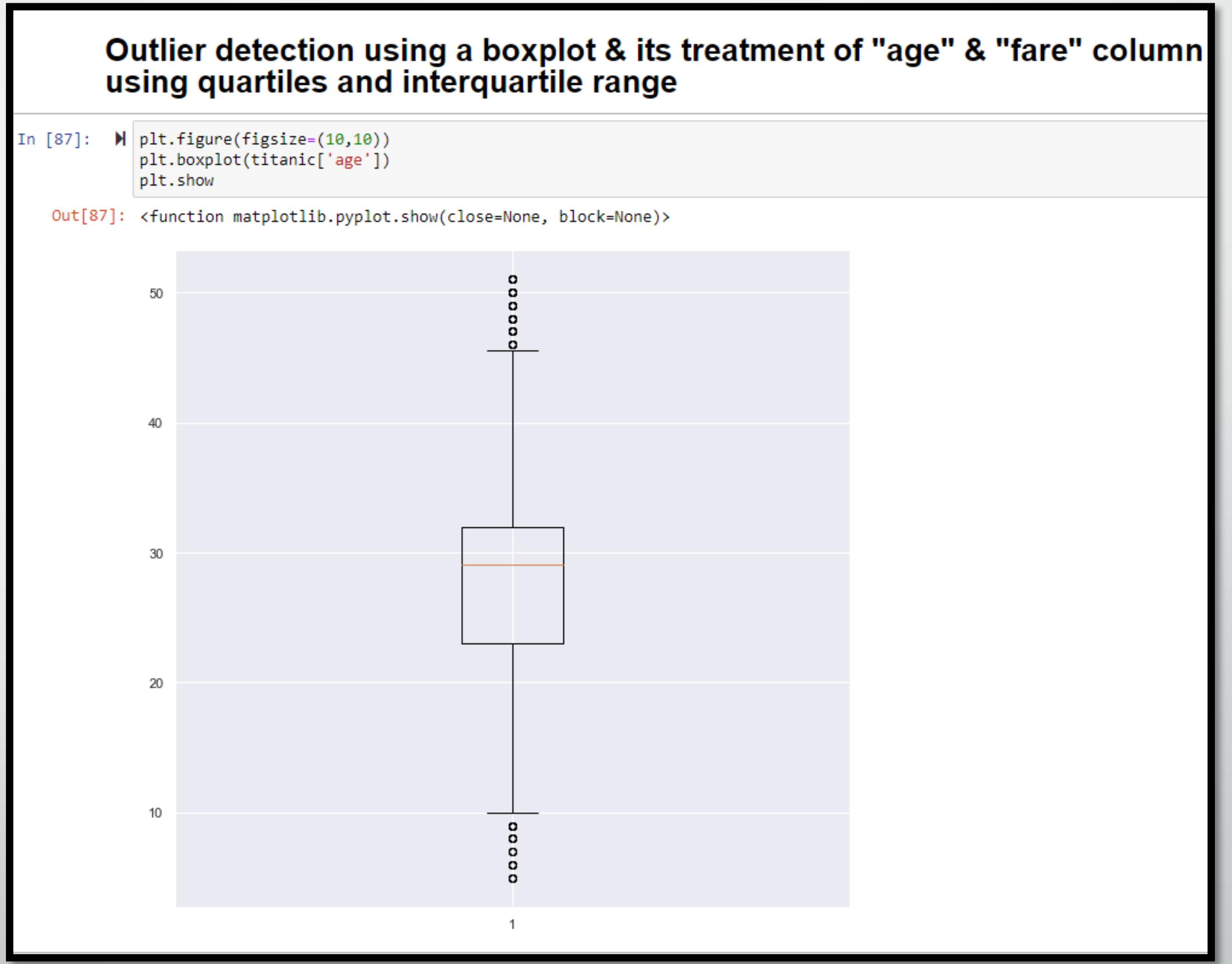
Out[47]:
```
pclass        0
survived      0
name          0
sex           0
age         261
sibsp         0
parch         0
fare          0
embarked      0
dtype: int64
```

In [48]:
```
titanic["age"]=titanic['age'].fillna(titanic['age'].mean())
titanic.isnull().sum()
```

Out[48]:
```
pclass      0
survived    0
name        0
sex         0
age         0
sibsp       0
parch       0
fare        0
embarked    0
dtype: int64
```

Boxplot before outlier treatment of age

- Outlier treatment using boxplot for “**age**” column in the titanic dataset.
- Boxplot normally have box at the center and whiskers at the end.
- This boxplot analyses outliers denoted by dots above & below the whiskers in the graph before treating it.



Outlier treatment of “age” using quartile, interquartile-range, upper & lower bound

In [50]:

```
''' Detection '''
# IQR
Q1_age = np.percentile(titanic['age'], 25,
                   interpolation = 'midpoint')

Q3_age = np.percentile(titanic['age'], 75,
                   interpolation = 'midpoint')

IQR_age = Q3_age - Q1_age

# Upper bound
upper_age_bound = Q3_age+1.5*IQR_age
# Lower bound
lower_age_bound = Q1_age-1.5*IQR_age

print(Q1_age)
print(Q3_age)
print("Maximum of age:",max(titanic['age']))
print("Minimum of age:",min(titanic['age']))
print("Upper age bound:",upper_age_bound)
print("Lower age bound:",lower_age_bound)
```

```
22.0
34.0
Maximum of age: 60.0
Minimum of age: 1.0
Upper age bound: 52.0
Lower age bound: 4.0
```

In [51]:

```
outlierFilter_lower=titanic['age']>4
titanic = titanic[outlierFilter_lower]
outlierFilter_higher=titanic['age']<52
titanic = titanic[outlierFilter_higher]
```

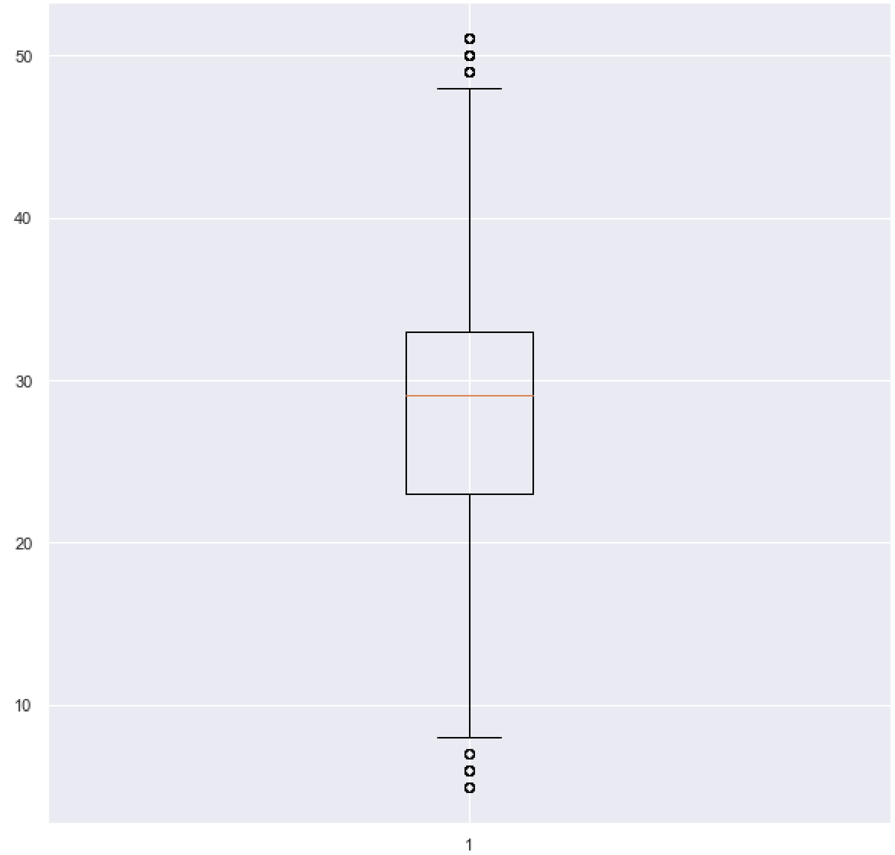
Boxplot after outlier treatment of age

- Sometimes it is not necessary to remove all the outliers. But you can keep some few.
- Outlier treatment sometimes don't require to entirely eliminate outliers completely.
- There are no hard and fast criteria of how many outliers are acceptable in a dataset.
- Sometimes it is 5% for a small dataset and 20-25% of large dataset. Still there are no fixed assumptions.

In [52]:

```
plt.figure(figsize=(10,10))
plt.boxplot(titanic['age'])
plt.show
```

Out[52]: `<function matplotlib.pyplot.show(close=None, block=None)>`

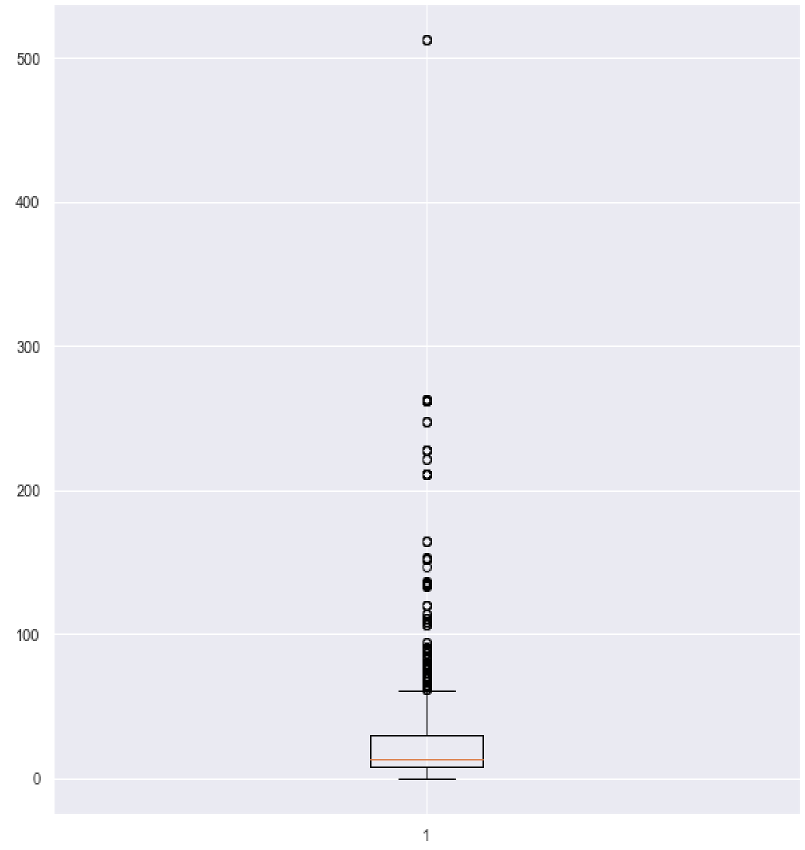


Boxplot before outlier treatment of “fare”

- In this case there are only upper outliers but in huge numbers.
- There are no lower outliers.
- We will move on for treating it….

```
In [53]:  plt.figure(figsize=(10,10))
          plt.boxplot(titanic['fare'])
          plt.show
```

```
Out[53]: <function matplotlib.pyplot.show(close=None, block=None)>
```



Outlier treatment of “fare” using quartile, interquartile-range & upper bound

In [54]:
```
''' Detection '''
# IQR
Q1_fare = np.percentile(titanic['fare'], 25,
                   interpolation = 'midpoint')

Q3_fare = np.percentile(titanic['fare'], 75,
                   interpolation = 'midpoint')

IQR_fare = Q3_fare - Q1_fare

# Upper bound
upper_fare_bound = Q3_fare+1.5*IQR_fare
# Lower bound
lower_fare_bound = Q1_fare-1.5*IQR_fare
```

In [55]:
```
print(Q1_fare)
print(Q3_fare)
print("Maximum of fare:",max(titanic['fare']))
print("Minimum of fare:",min(titanic['fare']))
print("Upper fare bound:",upper_fare_bound)
print("Lower fare bound:",lower_fare_bound)
```

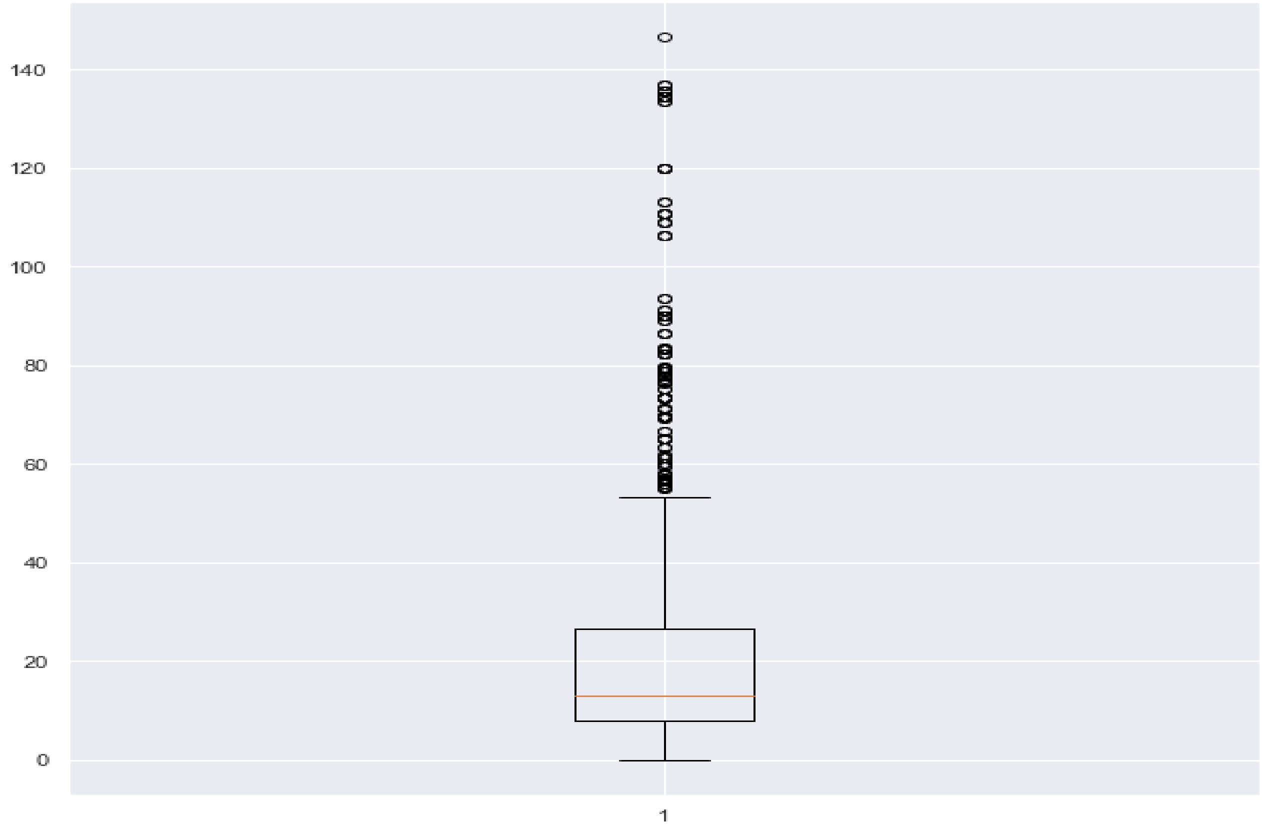
```
7.8958
29.4125
Maximum of fare: 512.3292
Minimum of fare: 0.0
Upper fare bound: 61.68755
Lower fare bound: -24.37925
```

In [56]:
```
outlierFilter_higher=titanic['fare']<150
titanic = titanic[outlierFilter_higher]
```

Boxplot after outlier treatment of fare

In [57]:
```
plt.figure(figsize=(10,10))
plt.boxplot(titanic['fare'])
plt.show
```

Out[57]: <function matplotlib.pyplot.show(close=None, block=None)>



Force to drop “fare” column!

There are cases where the outliers cannot be removed to a huge extent when they are the maximum in a column of variables. So they have to be removed. In this case we have to discard 'fare' column of titanic

In [58]:
```
titanic = titanic.drop(columns=['fare'])
```

In [59]:
```
titanic.head()
```

Out[59]:

	pclass	survived	name	sex	age	sibsp	parch	embarked
4	1	1	Anderson, Mr. Harry	male	48.000000	0	0	S
5	1	0	Andrews, Mr. Thomas Jr	male	39.000000	0	0	S
9	1	1	Aubart, Mme. Leontine Pauline	female	24.000000	0	0	C
10	1	1	Barber, Miss. Ellen "Nellie"	female	26.000000	0	0	S
11	1	0	Baumann, Mr. John D	male	29.070783	0	0	S

Drop the unnecessary columns like “**name**” in this case.

```
In [60]: titanic = titanic.drop(columns = ['name'])

In [61]: titanic.head()

Out[61]:
```

	pclass	survived	sex	age	sibsp	parch	embarked
4	1	1	male	48.000000	0	0	S
5	1	0	male	39.000000	0	0	S
9	1	1	female	24.000000	0	0	C
10	1	1	female	26.000000	0	0	S
11	1	0	male	29.070783	0	0	S

Reindexing the columns

```
In [62]: colnames =["survived","pclass","sex","age","sibsp","parch","embarked"]

In [63]: titanic = titanic.reindex(columns = colnames)

In [64]: titanic.head()
```

Out[64]:

	survived	pclass	sex	age	sibsp	parch	embarked
4	1	1	male	48.000000	0	0	S
5	0	1	male	39.000000	0	0	S
9	1	1	female	24.000000	0	0	C
10	1	1	female	26.000000	0	0	S
11	0	1	male	29.070783	0	0	S

Replace variables like 1 and 0 to have categorical variables in columns

Replace all the variables with 1 and 0s wherever there are categorical observations

In [65]:
```
titanic.loc[titanic.sibsp > 0,'sibsp']=1
titanic.loc[titanic.parch > 0,'parch']=1
titanic['sex'] = (titanic['sex'] == 'male').astype(int)
titanic['embarked'] = (titanic['embarked'] == 'S').astype(int)
```

In [66]:
```
titanic.head()
```

Out[66]:

	survived	pclass	sex	age	sibsp	parch	embarked
4	1	1	1	48.000000	0	0	1
5	0	1	1	39.000000	0	0	1
9	1	1	0	24.000000	0	0	0
10	1	1	0	26.000000	0	0	1
11	0	1	1	29.070783	0	0	1

Fix target variable and independent variables denoted by Y & X respectively.

```
In [68]:  X = titanic[["pclass","sex","age","sibsp","parch","embarked"]]
          Y = titanic['survived']
```

Cross validation

```
In [69]:
import sklearn.model_selection as model_selection
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, Y, train_size=0.75, random_state=101)
print ("X_train: ", X_train)
print ("y_train: ", y_train)
print("X_test: ", X_test)
print ("y_test: ", y_test)
```

```
X_train:       pclass  sex        age  sibsp  parch  embarked
740        3    0  24.000000      0      0         0
308        2    1  18.000000      0      0         1
149        1    0  16.000000      0      1         0
812        3    0  29.070783      0      0         1
18         1    1  26.000000      0      0         0
...      ...  ...        ...    ...    ...       ...
684        3    1  24.000000      0      0         0
1099       3    1  10.000000      1      1         0
110        1    1  50.000000      1      0         1
710        3    1  34.000000      1      1         1
983        3    0  29.070783      0      0         0

[842 rows x 6 columns]
y_train:  740     0
308     0
149     1
812     0
18      1
       ..
684     0
1099    0
110     1
710     0
983     1
Name: survived, Length: 842, dtype: int64
X_test:       pclass  sex        age  sibsp  parch  embarked
1149       3    1  20.000000      0      0         1
372        2    0  34.000000      0      1         1
322        2    1  23.000000      0      0         1
51         1    0  14.000000      1      1         1
851        3    1  33.000000      0      0         1
...      ...  ...        ...    ...    ...       ...
1090       3    1  29.070783      0      0         1
1142       3    1  29.070783      1      0         0
440        2    0  24.000000      1      0         1
```

Run the logistic model(download necessary packages wherever necessary)

```
In [70]:  from sklearn.linear_model import LogisticRegression
          from sklearn.metrics import classification_report
          from sklearn.metrics import confusion_matrix

In [71]:  model = LogisticRegression()

In [72]:  model.fit(X_train, y_train)

Out[72]:  LogisticRegression()

In [73]:  predictions = model.predict(X_test)
```

In Logistic regression the equation known as Sigmoid function is shown as:

f(y)=[1/(1-e^-y)]……. Where, y = a+bx1+cx2+dx3.

Simplifying the sigmoid function we get the results as **p= [e^y/(1+e^y)]**.

Let us move on to the Boosting algorithms!

Gradient boosting

Gradient Boosting

In [92]:
```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import GradientBoostingClassifier #For Classification
```

In [93]:
```
# Let us encode true and false to number value 0 and 1
LE=LabelEncoder()
titanic['pclass']=LE.fit_transform(titanic['pclass'])
titanic['sex']=LE.fit_transform(titanic['sex'])
titanic['embarked']=LE.fit_transform(titanic['embarked'])
```

In [94]:
```
GB=GradientBoostingRegressor(n_estimators=2)
GB.fit(X,Y)
Y_predict=GB.predict(X) #ages predicted by model with 2 estimators
Y_predict
```

Out[94]:
```
array([0.35481904, 0.35481904, 0.47496453, ..., 0.40886382, 0.31369908,
       0.31369908])
```

In [95]:
```
from sklearn.model_selection import KFold
kf = KFold(n_splits=5,random_state=42,shuffle=True)
for train_index,val_index in kf.split(X):
    X_val = X.iloc[val_index]
    y_val = Y.iloc[val_index]
```

In [96]:
```
gradient_booster = GradientBoostingClassifier(learning_rate=0.1)
gradient_booster.get_params()
```

Out[96]:
```
{'ccp_alpha': 0.0,
 'criterion': 'friedman_mse',
 'init': None,
 'learning_rate': 0.1,
 'loss': 'deviance',
 'max_depth': 3,
 '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,
 'n_estimators': 100,
 'n_iter_no_change': None,
 'presort': 'deprecated',
 'random_state': None,
 'subsample': 1.0,
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm_start': False}
```

Gradient boosting (contd..)

```
In [97]: gradient_booster.fit(X_train,y_train)
         print(classification_report(y_val,gradient_booster.predict(X_val)))

              precision    recall  f1-score   support

           0       0.82      0.94      0.88       145
           1       0.86      0.61      0.71        79

    accuracy                           0.83       224
   macro avg       0.84      0.78      0.79       224
weighted avg       0.83      0.83      0.82       224


In [98]: print('Accuracy:', accuracy_score(y_val,gradient_booster.predict(X_val)))

         Accuracy: 0.8258928571428571
```

The concept of Mean square error in boosting

In Boosting,If you increase the number of estimator the MSE decreases¶

In [99]:

```
#Following code is used to find out MSE of prediction with Gradient boosting algorithm having estimator 2.
MSE_2=(sum((Y-Y_predict)**2))/len(Y)
print('MSE for two estimators :',MSE_2)
```

```
MSE for two estimators : 0.19879291207608302
```

In [100]:

```
GB=GradientBoostingRegressor(n_estimators=3)
GB.fit(X,Y)
Y_predict=GB.predict(X) #ages predicted by model with 3 estimators
Y_predict
MSE_2=(sum((Y-Y_predict)**2))/len(Y)
print('MSE for two estimators :',MSE_2)
```

```
MSE for two estimators : 0.18761649146810688
```

In [101]:

```
GB=GradientBoostingRegressor(n_estimators=50)
GB.fit(X,Y)
Y_predict=GB.predict(X) #ages predicted by model with 50 estimators
Y_predict
MSE_2=(sum((Y-Y_predict)**2))/len(Y)
print('MSE for two estimators :',MSE_2)
```

```
MSE for two estimators : 0.12852817695790641
```

Adaboost

AdaBoost

In [102]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import LabelEncoder
```

In [103]:
```
ad=AdaBoostClassifier()
```

In [104]:
```
pred = ad.fit(X_train, y_train).predict(X_test)
```

In [105]:
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.78      0.80       185
           1       0.61      0.65      0.63        96

    accuracy                           0.74       281
   macro avg       0.71      0.71      0.71       281
weighted avg       0.74      0.74      0.74       281
```

In [106]:
```
accuracy_score(y_test,pred)
```

Out[106]: 0.7366548042704626

In [ ]:

XG boosting

XG Boosting

```
[107]:
import xgboost as xgb
from xgboost import XGBClassifier
model = XGBClassifier()

# fit the model with the training data
model.fit(X_train,y_train)


# predict the target on the train dataset
predict_XGboost_train = model.predict(X_train)
print('\nTarget on train data',predict_XGboost_train)

# Accuray Score on train dataset
accuracy_train = accuracy_score(y_train,predict_XGboost_train)
print('\naccuracy_score on train dataset : ', accuracy_train)

# predict the target on the test dataset
predict_XGBoost_test = model.predict(X_test)
print('\nTarget on test data',predict_XGBoost_test)

# Accuracy Score on test dataset
accuracy_test = accuracy_score(y_test,predict_XGBoost_test)
print('\naccuracy_score on test dataset : ', accuracy_test)
```

```
[22:03:46] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Expli
citly set eval_metric if you'd like to restore the old behavior.

Target on train data [1 0 1 0 1 1 1 0 1 0 1 1 0 0 1 0 1 0 1 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0
 1 0 0 1 1 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 0 1 0 0 1 1
 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 1 1 0 1 0 0 1 0 1 0 0 1 0 0 1
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0
 0 0 1 1 1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1 0 1 0 1 1 0 0 1 0 1 1 1 0 0 1
 0 1 0 0 1 1 1 1 1 0 0 0 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 1 0 0 0 0
 0 0 0 0 1 0 1 0 0 0 1 0 1 1 0 0 0 0 1 1 1 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0
 0 1 0 0 1 0 1 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0
 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 1 0 0 1 0 1 1 0
 1 0 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 1 0 1 0 1
 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 0 0 1 0
 0 1 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 0 0 1 0 0 0 0
 0 0 1 0 1 1 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1
 1 1 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 1 1 0 0 0 1
 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1
 0 0 1 0 0 0 0 0 1 0 1 0 1 0 1 1 1 0 1 0 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 1 1
 1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 1
 1 0 1 0 0 1 0 0 1 1 1 1 1 1 0 1 1 1 0 0 0 1 0 0 1 0 1 0 1 1 0 0 1 1 0 0 1
 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 0 0 0 1 0 0 0 0 1 1 1 0 0 1 0 0 0 0
 0 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0
 1 0 0 1 0 0 1 0 0 1 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1]

accuracy_score on train dataset :  0.8871733966745843

Target on test data [0 1 0 1 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1
 0 0 0 0 1 1 0 0 1 0 1 0 0 0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 1
 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0
 1 0 0 1 0 0 1 0 1 0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0 0 1 0
 0 0 1 1 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 1 1 0 0 0 0 0 0 1 0 1 1 1
 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 1 1 0 0 1 1 0
 1 1 1 1 0 0 1 0 0 0 1 1 0 0 0 0 1 0 0 1 0 1]

accuracy_score on test dataset :  0.7829181494661922
```

Summing up all the results!

Comparing the classification outcome of **accuracy** results of all boosting techniques:

```
print(classification_report(y_val,gradient_booster.predict(X_val)),"-----------------------------------------------------------Gra
print()
print(classification_report(y_test,pred),"-----------------------------------------------------------Adaboost Classification repor
print()
print(classification_report(y_test, predict_XGBoost_test)),"-----------------------------------------------------------XGboost cla
```

```
              precision    recall  f1-score   support

           0       0.82      0.94      0.88       145
           1       0.86      0.61      0.71        79

    accuracy                           0.83       224
   macro avg       0.84      0.78      0.79       224
weighted avg       0.83      0.83      0.82       224
 -----------------------------------------------------------Gradient boosting classifiction report
              precision    recall  f1-score   support

           0       0.81      0.78      0.80       185
           1       0.61      0.65      0.63        96

    accuracy                           0.74       281
   macro avg       0.71      0.71      0.71       281
weighted avg       0.74      0.74      0.74       281
 -----------------------------------------------------------Adaboost Classification report
              precision    recall  f1-score   support

           0       0.83      0.85      0.84       185
           1       0.69      0.66      0.67        96

    accuracy                           0.78       281
   macro avg       0.76      0.75      0.76       281
weighted avg       0.78      0.78      0.78       281

: (None,
  '-----------------------------------------------------------XGboost classification report')
```

Results interpretation

- Gradient boosting gives the best accuracy with 0.83 (83%).
- Followed by XG-boost with 0.78 (78%).
- Finally Adaboost with 0.74 (74%).

Thank you