PREDICTING BANKRUPTCY OF COMPANIES

Group 4

Ambika Saini
Prachi Tripathi
Shweta Pandey

Problem Statement

The goal of this project is to understand the different application of classification methods on a given dataset. The dataset is about bankruptcy prediction of Polish companies (link). The data was collected from Emerging Markets Information Service (EMIS), which is a database containing information on emerging markets around the world. The bankrupt companies were analyzed in the period 2000-2012, while the still operating companies were evaluated from 2007 to 2013. To achieve the goal, the dataset taken here is of the 5thYear the data contains financial rates from 5th year of the forecasting period and corresponding class label that indicates bankruptcy status after 1 year. The data contains 5910 instances (financial statements), 410 represents bankrupted companies, 5500 firms that did not bankrupt in the forecasting period.

List of Attributes:

Attr1 net profit / total assets
Attr2 total liabilities / total assets
Attr3 working capital / total assets
Attr4 current assets / short-term liabilities
Attr5 [(cash + short-term securities + receivables - short-term liabilities) / (operating expenses - depreciation)] * 365
Attr6 retained earnings / total assets
Attr7 EBIT / total assets
Attr8 book value of equity / total liabilities
Attr9 sales / total assets
Attr10 equity / total assets
Attr11 (gross profit + extraordinary items + financial expenses) / total assets
Attr12 gross profit / short-term liabilities
Attr13 (gross profit + depreciation) / sales
Attr14 (gross profit + interest) / total assets
Attr15 (total liabilities * 365) / (gross profit + depreciation)
Attr16 (gross profit + depreciation) / total liabilities
Attr17 total assets / total liabilities
Attr18 gross profit / total assets
Attr19 gross profit / sales
Attr20 (inventory * 365) / sales
Attr21 sales (n) / sales (n-1)
Attr22 profit on operating activities / total assets
Attr23 net profit / sales
Attr24 gross profit (in 3 years) / total assets
Attr25 (equity - share capital) / total assets
Attr26 (net profit + depreciation) / total liabilities
Attr27 profit on operating activities / financial expenses
Attr28 working capital / fixed assets
Attr29 logarithm of total assets
Attr30 (total liabilities - cash) / sales
Attr31 (gross profit + interest) / sales
Attr32 (current liabilities * 365) / cost of products sold
Attr33 operating expenses / short-term liabilities
Attr34 operating expenses / total liabilities
Attr35 profit on sales / total assets
Attr36 total sales / total assets
Attr37 (current assets - inventories) / long-term liabilities
Attr38 constant capital / total assets
Attr39 profit on sales / sales
Attr40 (current assets - inventory - receivables) / short-term liabilities
Attr41 total liabilities / ((profit on operating activities + depreciation) * (12/365))
Attr42 profit on operating activities / sales
Attr43 rotation receivables + inventory turnover in days
Attr44 (receivables * 365) / sales
Attr45 net profit / inventory
Attr46 (current assets - inventory) / short-term liabilities
Attr47 (inventory * 365) / cost of products sold

Attr48 EBITDA (profit on operating activities - depreciation) / total assets
Attr49 EBITDA (profit on operating activities - depreciation) / sales
Attr50 current assets / total liabilities
Attr51 short-term liabilities / total assets
Attr52 (short-term liabilities * 365) / cost of products sold)
Attr53 equity / fixed assets
Attr54 constant capital / fixed assets
Attr55 working capital
Attr56 (sales - cost of products sold) / sales
Attr57 (current assets - inventory - short-term liabilities) / (sales - gross profit - depreciation)
Attr58 total costs /total sales
Attr59 long-term liabilities / equity
Attr60 sales / inventory
Attr61 sales / receivables
Attr62 (short-term liabilities *365) / sales
Attr63 sales / short-term liabilities
Attr64 sales / fixed assets
Class - response variable Y:
0 = did not bankrupt; 1 = bankrupt

Overall Summary

We go through a step – by – step process through out this project, the different phases that are seen:

- Data Analysis
- Feature Scaling
- Feature Selection
- Testing Models
- Comparison of Models

The general solution that can be concluded is that amongst the various classification models performed on our dataset the model which gives the most accurate result is the Random Forest model. Further in this report we shall an in-detail comparison of all the models (Logistic Regression, Decision Tree, Random Forest, Naïve Bayes, Support Vector Machine, K Nearest Neighbors and AdaBoost Classifier) and how strong their power is to become an accurate predictive model.

In this report, we are going to see the different stages that were involved and the models implemented to arrive at the most efficient output. The dataset used for this project consists of 5910 rows and 66 columns.

The libraries which were used throughout this project are;

```
# Importing the libraries that have been used in the project

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import pdfkit
from sklearn.linear_model import Lasso
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectFromModel
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, roc_auc_score,roc_curve, auc
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier
```

The first phase of the project is *data analysis* and under this phase with the help of the programming language python the following steps were implemented.

Initially the shape of the dataset was checked, from the dataset the column “id” was dropped since we do not require it for our evaluation. After which the dataset was split into features and target, here columns 1 to 64 were under features and the column “class” was under target. Now moving on to skim the dataset for missing value upon which it was noticed that instead of missing values the dataset consisted of “0” and “?”.

```
# Checking for the missing value before replacing with the  0 and ? marks in the data set

features_with_na=[features for features in year_data.columns if year_data[features].isnull().sum()>1]
print(features_with_na)


```

```
[]
```

```
features_year.replace('0', np.nan, inplace= True)
features_year.replace('?', np.nan, inplace= True)
```

So as seen in the above figure, “0” and “?” are replaced with NaN and the data type of the dataset is converted to float.

```
features_year = features_year.astype(float)

```

Afterwards we check the percentage of NaN values present in each feature to determine the amount of missing values under them.

```
# Here we will check the percentage of nan values present in each feature
# 1 -step make the list of features which has missing values

features_with_na_Year=[features for features in features_year.columns
                       if features_year[features].isnull().sum()>1]

# 2- step print the feature name and the percentage of missing values

print("_________________CHECKING PERCENTAGE MISSING VALUES IN A YEAR _________________")
for feature in features_with_na_Year:
    print(feature, np.round(features_year[feature].isnull().mean(), 4),  ' % missing values')

```

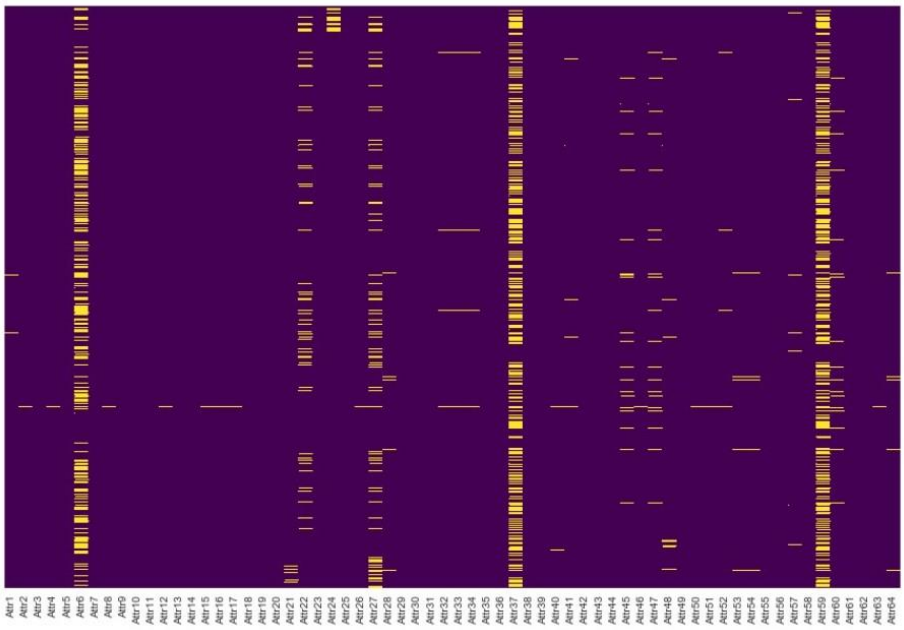
```
_________________
Attr1 0.0068  % missing values
Attr2 0.0032  % missing values
Attr3 0.0007  % missing values
Attr4 0.0037  % missing values
Attr5 0.0024  % missing values
Attr6 0.3853  % missing values
Attr7 0.0014  % missing values
Attr8 0.0032  % missing values
Attr10 0.0007  % missing values
Attr11 0.0014  % missing values
Attr12 0.0037  % missing values
Attr14 0.0014  % missing values
Attr15 0.003  % missing values
Attr16 0.003  % missing values
Attr17 0.0032  % missing values
Attr18 0.0014  % missing values
Attr21 0.0174  % missing values
Attr22 0.092  % missing values
Attr24 0.0254  % missing values
Attr25 0.001  % missing values
Attr26 0.0039  % missing values
Attr27 0.1321  % missing values
Attr28 0.0183  % missing values
Attr29 0.0005  % missing values
Attr32 0.0115  % missing values
Attr33 0.0095  % missing values
Attr34 0.0091  % missing values
Attr35 0.0051  % missing values
Attr36 0.0005  % missing values
Attr37 0.4315  % missing values
Attr38 0.0007  % missing values
Attr40 0.0042  % missing values
Attr41 0.0173  % missing values
Attr45 0.0516  % missing values
Attr46 0.0036  % missing values
Attr47 0.0508  % missing values
Attr48 0.0188  % missing values
Attr50 0.0032  % missing values
Attr51 0.0037  % missing values
Attr52 0.0096  % missing values
Attr53 0.0183  % missing values
Attr54 0.0183  % missing values
Attr57 0.0168  % missing values
Attr59 0.4315  % missing values
Attr60 0.0453  % missing values
Attr61 0.0025  % missing values
Attr63 0.0037  % missing values
Attr64 0.0183  % missing values
```

Visually representing the above output, we take the help of heatmap. A heat map is a two-dimensional representation of data in which values are represented by colors. A simple heat map provides an immediate visual summary of information. More elaborate heat maps allow the viewer to understand complex data sets.

There can be many ways to display heat maps, but they all share one thing in common -- they use color to communicate relationships between data values that would be would be much harder to understand if presented numerically in a spreadsheet.

```
1 plt.subplots(figsize=(15,10)) # Width,Height
2 sns.heatmap(features_year.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1dc95289e10>
```



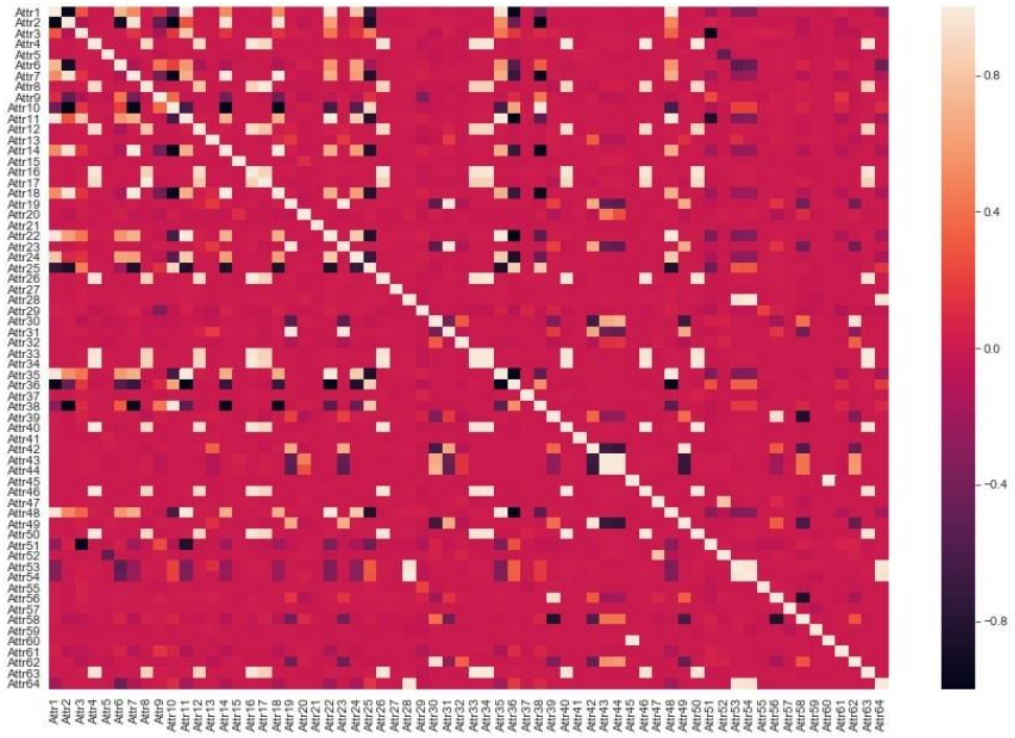
After analyzing the above heatmap we can conclude that;

- The attributes having more yellow spots indicates the amount of missing values.
- The ones with plain purple indicate no missing values.
- Attr37 has the maximum number of missing values.
- Followed by Attr6 and Attr59 having the second highest number of missing values.

Next, we apply the method correlation matrix to observe the dependence between multiple variables at the same time. The result is a table containing the correlation coefficients between each variable and the others which we are going to visualize with the help of heatmap.

```
corr = features_year.corr()
# plot the heatmap
plt.subplots(figsize=(15,10)) # Width,Height
sns.heatmap(corr, xticklabels=corr.columns,yticklabels=corr.columns)

```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1dc956c72e8>
```



With the help of the color bar on the right side of the figure we can see whether the correlation between any two features is positive (lighter color) or negative (darker color).

Once the missing values are taken care of next, we look into handling outliers.

```
Q1 = features_year.quantile(0.25)
Q3 = features_year.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
Attr1        0.065645
Attr2        0.255555
Attr3        0.221558
Attr4        0.864750
Attr5       53.080750
Attr6        0.191454
Attr7        0.073158
Attr8        1.197950
Attr9        0.460575
Attr10       0.250875
Attr11       0.080075
Attr12       0.246310
Attr13       0.061810
Attr14       0.073158
Attr15    1518.180000
Attr16       0.229675
Attr17       1.264750
Attr18       0.073158
Attr19       0.050692
Attr20      37.992000
Attr21       0.218900
Attr22       0.080443
Attr23       0.043570
Attr24       0.205776
Attr25       0.283622
Attr26       0.206380
Attr27       2.223470
Attr28       0.623446
Attr29       0.622900
Attr30       0.203110
                ...
Attr35       0.085502
Attr36       0.813625
Attr37       5.374075
Attr38       0.231610
Attr39       0.058645
Attr40       0.221532
Attr41       0.098224
Attr42       0.055967
Attr43      50.011000
Attr44      33.511000
Attr45       0.423228
Attr46       0.584430
Attr47      40.964500
Attr48       0.086477
Attr49       0.054806
Attr50       0.681805
Attr51       0.228640
Attr52       0.139795
Attr53       0.631430
Attr54       0.601690
Attr55    5613.900000
Attr56       0.062989
Attr57       0.139270
Attr58       0.061160
Attr59       0.328598
Attr60       7.678075
Attr61       3.706500
Attr62      49.041000
Attr63       2.812450
Attr64       3.237825
```

As we now have the IQR scores, it's time to get hold on outliers. The below code will give an output with some "True" and "False" values. The data point where we have "False" that means these values are valid whereas True indicates presence of an outlier.

```
print(features_year < (Q1 - 1.5 * IQR)) |(features_year > (Q3 + 1.5 * IQR))
```

```
features_year = features_year[~((features_year < (Q1 - 1.5 * IQR)) |
                                (features_year > (Q3 + 1.5 * IQR))).any(axis=1)]
features_year.shape
```

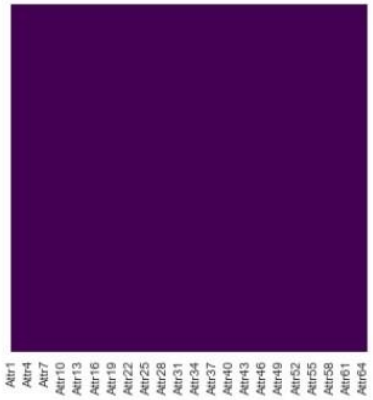
```
(1101, 64)
```

Now we are going to impute the missing values by replacing them with median since the dataset consists of outliers and once again with the aid of heatmap we shall visualize the result.

```
## We will replace by using median since there are outliers

features_year.fillna(features_year.median(), inplace=True)
```

```
plt.subplots(figsize=(5,5)) # Width,Height
sns.heatmap(features_year.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1dc99761cf8>
```



From above heatmap visualization its clearly showing that after imputing now there is no missing value in the data.

Secondly, we move on to the phase of *feature scaling*. In this scenario, this technique is used to standardize the independent features present in the data in a fixed range.

```
feature_scale=[feature for feature in features_year.columns]

from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(features_year[feature_scale])
```

```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

```
scaler.transform(features_year[feature_scale])

```

```
array([[8.41532746e-01, 8.57215818e-01, 7.17890302e-01, ...,
        8.39849684e-04, 5.36449577e-04, 4.41650994e-05],
       [8.41361457e-01, 8.57076593e-01, 7.20097806e-01, ...,
        7.69834532e-04, 6.31610498e-04, 4.48623898e-05],
       [8.41608926e-01, 8.56553570e-01, 7.23529277e-01, ...,
        6.82129593e-04, 8.69558596e-04, 5.90787700e-05],
       ...,
       [8.41047957e-01, 8.58607829e-01, 7.15028535e-01, ...,
        9.84020669e-04, 4.31658731e-04, 8.64766579e-05],
       [8.41175263e-01, 8.57591787e-01, 7.17931227e-01, ...,
        7.07795173e-04, 7.76451884e-04, 2.96000739e-05],
       [8.41181593e-01, 8.57179198e-01, 7.17323407e-01, ...,
        8.26763360e-04, 5.50894463e-04, 4.12128361e-05]])
```

After finishing the mentioned steps so far, we arrive at the third phase which is *feature selection*. Under this phase, we will be taking steps to perform feature selection reduction on the dataset, either to improve estimators' accuracy scores or to boost their performance on very high-dimensional datasets (with the help of sklearn.feature_selection module), using lasso regression model.

```
# For feature slection
# Remember to set the seed, the random state in this function
feature_sel_model = SelectFromModel(Lasso(alpha=0.005, random_state=0))
feature_sel_model.fit(features_year, target)

```

```
C:\Users\ilike\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:476: ConvergenceWarning: O
bjective did not converge. You might want to increase the number of iterations. Duality gap: 173.69204956992263,
tolerance: 0.038155668358714034
  positive)
```

```
SelectFromModel(estimator=Lasso(alpha=0.005, copy_X=True, fit_intercept=True,
                                max_iter=1000, normalize=False, positive=False,
                                precompute=False, random_state=0,
                                selection='cyclic', tol=0.0001,
                                warm_start=False),
                max_features=None, norm_order=1, prefit=False, threshold=None)
```

```
feature_sel_model.get_support()
```

```
array([False, False,  True,  True, False, False, False,  True, False,
       False, False,  True,  True, False, False,  True, False,  True,
       False,  True,  True, False, False, False,  True, False, False,
        True,  True, False,  True, False,  True,  True,  True,  True,
       False, False,  True,  True,  True,  True,  True, False,  True,
        True, False, False, False, False, False,  True,  True,  True,
       False, False,  True, False,  True, False, False, False,  True,
        True])
```

```
# let's print the number of total and selected features
# this is how we can make a list of the selected features

selected_feat = features_year.columns[(feature_sel_model.get_support())]

# let's print some stats
print('total features: {}'.format((features_year.shape[1])))
print('selected features: {}'.format(len(selected_feat)))
#print('features with coefficients shrank to zero: {}'.format(np.sum(sel_.estimator_.coef_ == 0)))
```

```
total features: 64
selected features: 31
```

```
selected_feat
```

```
Index(['Attr3', 'Attr4', 'Attr8', 'Attr12', 'Attr13', 'Attr16', 'Attr18',
       'Attr20', 'Attr21', 'Attr25', 'Attr28', 'Attr29', 'Attr31', 'Attr33',
       'Attr34', 'Attr35', 'Attr36', 'Attr39', 'Attr40', 'Attr41', 'Attr42',
       'Attr43', 'Attr45', 'Attr46', 'Attr52', 'Attr53', 'Attr54', 'Attr57',
       'Attr59', 'Attr63', 'Attr64'],
      dtype='object')
```

After filtering out the selected features from the total number of features we assign "x_1" with the selected features and "y_1" with target.

```
features_year = features_year[selected_feat]
print(features_year.head())
```

```
     Attr3   Attr4    Attr8    Attr12    Attr13    Attr16    Attr18  Attr20  \
0  0.01134  1.0205  0.57752  0.197600  0.096885  0.247420  0.109490  50.199
1  0.23298  1.5998  1.06340 -0.015967  0.037544  0.098825 -0.006202  59.923
2  0.57751  3.6082  3.05900  0.732180  0.165680  0.845390  0.162120  41.508
3  0.26927  1.5222  0.12740 -0.174450  0.084038  0.120840 -0.089951  47.698
4  0.10765  1.2437  0.81682  0.134190  0.055575  0.152970  0.059280  36.074

    Attr21    Attr25    Attr28  Attr29    Attr31  Attr33   Attr34    Attr35  \
0  1.15740  0.320360  0.026093  6.1267  0.077287  2.3498  0.24377  0.135230
1  1.01580  0.080285  0.615450  4.0022  0.000778  3.3779  2.70750 -0.036475
2  1.23620  0.677310  2.872100  4.7622  0.143490  4.4701  0.65878  0.145860
3  1.09420  0.064737  1.251900  4.0153 -0.138650  1.4375  0.83567  0.014027
4  0.99455  0.429200  0.238910  5.8823  0.039129  2.6583  2.13360  0.364200

   Attr36    Attr39    Attr40    Attr41    Attr42  Attr43    Attr45   Attr46  \
0  1.4493  0.095457  0.128790  0.111890  0.095457  127.30  0.452890  0.66883
1  1.2757 -0.028591  0.057810  0.291670  0.001011  171.38 -0.029614  1.06060
2  1.1698  0.129100  1.319600  0.042587  0.129100  163.71  1.013700  3.02800
3  1.2754  0.010998  0.456220  0.149980  0.000000  157.30 -0.539690  1.19900
4  1.5150  0.240400  0.067283  0.198280  0.044632  125.21  0.321770  0.90475

    Attr52   Attr53   Attr54    Attr57    Attr59  Attr63  Attr64
0  0.42557  0.73717  0.73866  0.275430  0.002024  2.5568  3.2597
1  0.29604  1.36140  1.56860 -0.012035  0.152220  3.2841  3.3700
2  0.22371  3.36840  3.36840  0.192290  0.158440  5.1027  5.6188
3  0.69565  0.52538  2.03780 -0.796020  2.878700  2.4735  5.9299
4  0.37618  0.99779  1.13690  0.107160  0.139380  3.4294  3.3622
```

```
## Training and Predicting

x1=features_year
y1=target

```

The fourth phase of the project is quite crucial, *applying different models*, it is from this phase we get a clearer picture on which model would be the most appropriate model to continue forward with in the future when similar situations arrive.

1. Logistic Regression:

   It is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

LOGISTIC REGRESSION

```
x1_train,x1_test,y1_train,y1_test=train_test_split(x1,y1,test_size=0.3,random_state=0)
x1_train.shape
```

```
(4137, 31)
```

```
model_LR= LogisticRegression()
model_LR.fit(x1_train,y1_train)
```

```
C:\Users\ilike\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning: lbfgs faile
d to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
# This will give you positive class prediction probabilities
y_prob = model_LR.predict_proba(x1_test)[:,1]

# This will threshold the probabilities to give class predictions.
y_pred = np.where(y_prob > 0.5, 1, 0)

model_LR.score(x1_test, y_pred)
```

```
1.0
```

```
y_score1_logistic = model_LR.predict_proba(x1_test)[:,1]
```

```
Logistic_cv_score = cross_val_score(model_LR, x1, y1, cv=10, scoring='roc_auc')
```

We split the dataset into training and testing with 70:30 ratio and run the logistic regression model on the training data. Later on, we find the positive class prediction probabilities and set a threshold for the probabilities to give us the class prediction. The score for the logistic model achieved against the testing and predicting variable is 1. We move forward to perform cross validation (CV), the

technique used to test the effectiveness of a machine learning models, it is also a re-sampling procedure used to evaluate a model if we have a limited data.

```
y_score1_logistic = y_score1_logistic.round()
print("=== Confusion Matrix ===\n")
print(confusion_matrix(y1_test, y_score1_logistic))
print('\n')
print("=== Classification Report ===\n")
print(classification_report(y1_test, y_score1_logistic))
print('\n')
print("=== All AUC Scores ===\n")
print(Logistic_cv_score)
print('\n')
print("=== Mean AUC Score ===\n")
print("Mean AUC Score - Logistic: ", Logistic_cv_score.mean())
```

```
=== Confusion Matrix ===

[[1658    7]
 [ 106    2]]


=== Classification Report ===

              precision    recall  f1-score   support

           0       0.94      1.00      0.97      1665
           1       0.22      0.02      0.03       108

    accuracy                           0.94      1773
   macro avg       0.58      0.51      0.50      1773
weighted avg       0.90      0.94      0.91      1773


=== All AUC Scores ===

[0.84292683 0.80177384 0.79605322 0.60337029 0.72829268 0.68922395
 0.71161863 0.81494457 0.7686918  0.74953437]


=== Mean AUC Score ===

Mean AUC Score - Logistic:  0.7506430155210643
```

We compute confusion matrix to evaluate the accuracy of a classification. By definition a confusion matrix C is such that $C_{i,j}$ is equal to the number of observations known to be in group i and predicted to be in group j. Thus, in binary classification, the count of true negatives is $C_{0,0}$ (1658), false negatives is $C_{1,0}$ (106), true positives is $C_{1,1}$ (2) and false positives is $C_{0,1}$ (7).

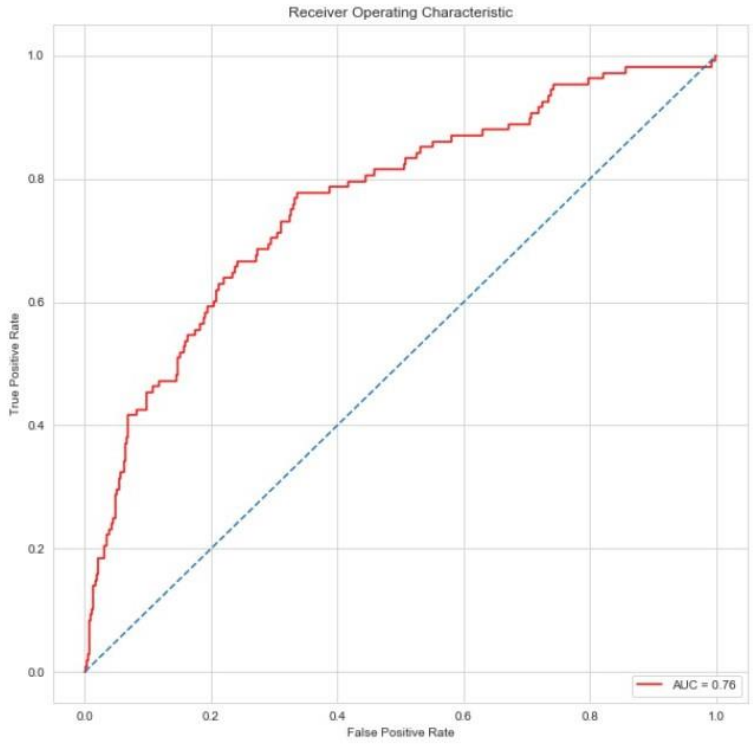
AUC stands for the Area Under the Curve and usually refers to the area under the ROC curve. AUC can be regarded as, the model's capability to distinguish bankrupt companies and normal companies. The value of AUC is supposed to vary from 0 to 1. If the value of AUC is below 0.5, the predicting model fail to identify two kinds of companies. When the value of AUC is between 0.5 and 0.7, model's predicting ability is average. A value from 0.7 to 0.8, a well-predicting model can be defined. If the value of AUC exceeds 0.8, in normal case, the model is almost perfect with strong predicting ability.

```
from sklearn.metrics import roc_curve, auc
false_positive_rate, true_positive_rate, thresholds = roc_curve(y1_test, y_prob)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc
```

```
0.763819374930486
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate,true_positive_rate, color='red',label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

```
Text(0.5, 0, 'False Positive Rate')
```



In the ROC curve, the Y-axis indicates the sensitivity, or the true positive rate that the forecasting result of a company is bankrupt, and it has actually filed a bankruptcy. For instance, ordinate value is 0.5, implying 50 bankrupt companies are forecasted to be bankrupt out of 100 bankrupt companies. The X-axis refers to the 1 minus the specificity, or the false positive rate that a company is predicted to be bankrupt while it does not go bankrupt. The 45-degree line represents a random model without predictive power.

The figure above depicts the ROC curve for logistic regression model, which is above the 45-degree line. In this case, the AUC is 0.76, hence the model can be considered as a well predicting model.

2. Decision Tree:

Decision tree classifiers are utilized as a well-known classification technique in different pattern recognition issues, for example, image classification and character recognition. Decision tree classifiers perform more successfully, specifically for complex classification problems, due to their high adaptability and computationally effective features. Besides, decision tree classifiers exceed expectations over numerous typical supervised classification methods. Generally, a decision tree comprises of three basic segments including a root node, a few hidden nodes, and a lot of terminal nodes (known as leaves).

DECISION TREE

```
# Create feature matrix and target vector
from sklearn.tree import DecisionTreeClassifier

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(x1,y1, test_size=0.3)
```

```
# Create classifier
clf1 = DecisionTreeClassifier();

# Train model
clf1.fit(x1_train, y1_train);

# Get predicted probabilities
y_score1 = clf1.predict_proba(x1_test)[:,1]
```

```
decision_tree_cv_score = cross_val_score(clf1, x1, y1, cv=10, scoring='roc_auc')
```

Now we apply DecisionTreeClassifier to our training dataset and train the model, moving on we obtain the predicted probabilities and the cross-validation score of the model.

```
y_score1 = y_score1.round()
print("=== Confusion Matrix ===")
print(confusion_matrix(y1_test, y_score1))
print('\n')
print("=== Classification Report ===")
print(classification_report(y1_test, y_score1))
print('\n')
print("=== All AUC Scores ===")
print(decision_tree_cv_score)
print('\n')
print("=== Mean AUC Score ===")
print("Mean AUC Score - Decission Tree: ", decision_tree_cv_score.mean())
```

```
=== Confusion Matrix ===
[[1565  100]
 [  66   42]]


=== Classification Report ===
              precision    recall  f1-score   support

           0       0.96      0.94      0.95      1665
           1       0.30      0.39      0.34       108

    accuracy                           0.91      1773
   macro avg       0.63      0.66      0.64      1773
weighted avg       0.92      0.91      0.91      1773



=== All AUC Scores ===
[0.62179601 0.5608204  0.68640798 0.62944568 0.62762749 0.7075388
 0.6267184  0.68004435 0.58157428 0.65383592]


=== Mean AUC Score ===
Mean AUC Score - Decission Tree:  0.6375809312638581
```

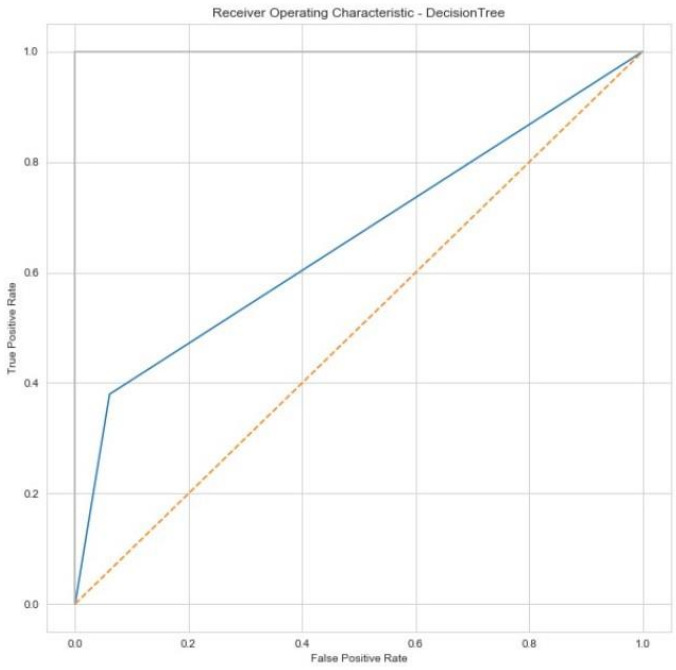
We repeat the steps of computing confusion matrix, where the count of true negatives is $C_{0,0}$ (1565), false negatives is $C_{1,0}$ (66), true positives is $C_{1,1}$ (42) and false positives is $C_{0,1}$ (100).

.

```
# Plot Receiving Operating Characteristic Curve
# Create true and false positive rates
false_positive_rate1, true_positive_rate1, threshold1 = roc_curve(y1_test, y_score1)
print('roc_auc_score for DecisionTree: ', roc_auc_score(y1_test, y_score1))

```

```
roc_auc_score for DecisionTree:  0.6594844844844846
```

```
# Plot ROC curves
plt.subplots(1, figsize=(10,10))
plt.title('Receiver Operating Characteristic - DecisionTree')
plt.plot(false_positive_rate1, true_positive_rate1)
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



After obtaining the AUC for the model which is 0.66, we plot the ROC curve for the model which can be seen in the figure above. Comparing the ROC curve of the decision tree model with the logistic regression model, we can comment that the logistic regression model's ROC curve is better compared to the current one as we can see from the AUC values of the two models. Clearly, more accurate results would be achieved from the previous model.

3. Random Forest:

The random forest is a model made up of many decision trees. Rather than just simply averaging the prediction of trees (which we could call a "forest"), this model uses two key concepts that gives it the name random:

- Random sampling of training data points when building trees
- Random subsets of features considered when splitting nodes

When training, each tree in a random forest learns from a random sample of the data points. The samples are drawn with replacement, known as bootstrapping, which means that some samples will be used multiple times in a single tree. The idea is that by training each tree on different samples, although each tree might have high variance with respect to a particular set of the training

data, overall, the entire forest will have lower variance but not at the cost of increasing the bias.

At test time, predictions are made by averaging the predictions of each decision tree. This procedure of training each individual learner on different bootstrapped subsets of the data and then averaging the predictions is known as bagging, short for bootstrap aggregating.

Random Forest

```
from sklearn import model_selection
from sklearn.ensemble import RandomForestClassifier

# random forest model creation
rfc_model = RandomForestClassifier()
rfc_model.fit(x1_train,y1_train)

```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=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_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

```
# predictions
rfc_predict = rfc_model.predict(x1_test)

# Probabilities
rfc_probs = rfc_model.predict_proba(x1_test)[:, 1]
```

```
rfc_cv_score = cross_val_score(rfc_model, x1, y1, cv=10, scoring='roc_auc')
```

This is the third classification model which is being implemented, after applying the model to the training data we go ahead and generate the probabilities and prediction. Also, find the cross-validation score for random forest model.

```
print("=== Confusion Matrix ===")
print(confusion_matrix(y1_test, rfc_predict))
print('\n')
print("=== Classification Report ===")
print(classification_report(y1_test, rfc_predict))
print('\n')
print("=== All AUC Scores ===")
print(rfc_cv_score)
print('\n')
print("=== Mean AUC Score ===")
print("Mean AUC Score - Random Forest: ", rfc_cv_score.mean())
```

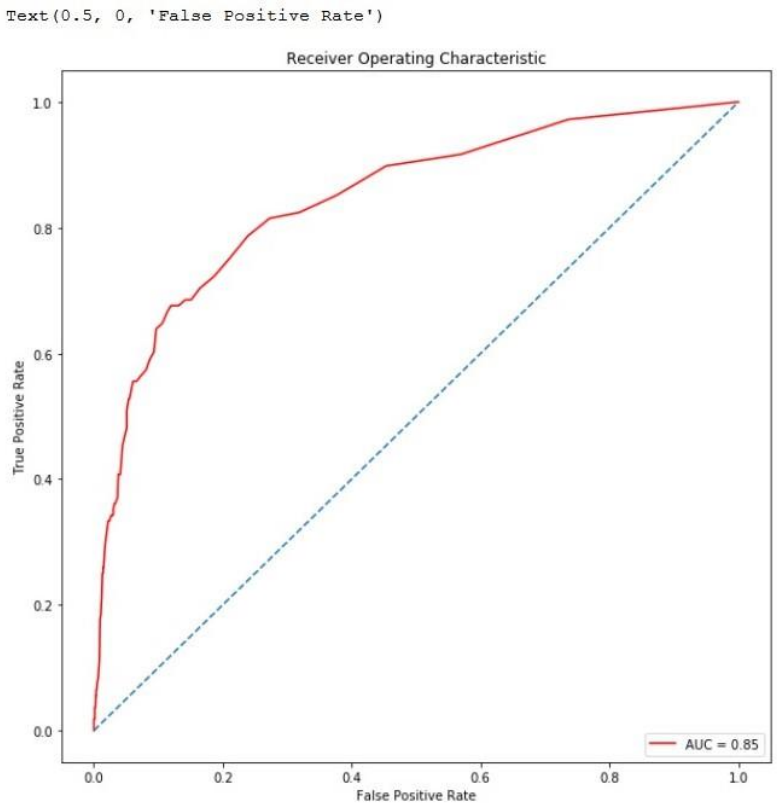
```
=== Confusion Matrix ===
[[1647   18]
 [  88   20]]


=== Classification Report ===
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      1665
           1       0.53      0.19      0.27       108

    accuracy                           0.94      1773
   macro avg       0.74      0.59      0.62      1773
weighted avg       0.92      0.94      0.93      1773



=== All AUC Scores ===
[0.88241685 0.88689579 0.86283814 0.79312639 0.86059867 0.81104213
 0.75017738 0.8616408  0.90031042 0.87676275]


=== Mean AUC Score ===
Mean AUC Score - Random Forest:  0.8485809312638581
```

As we can see from the above figure, the confusion matrix gives us the count of true negatives is $C_{0,0}$ (1647), false negatives is $C_{1,0}$ (88), true positives is $C_{1,1}$ (20) and false positives is $C_{0,1}$ (18). Then we obtain the classification report and mean AUC score from all pairwise class comparisons, here is 0.85 (approx.).

```
from sklearn.metrics import roc_curve, auc
false_positive_rate_rf, true_positive_rate_rf, thresholds = roc_curve(y1_test, rfc_probs)
auc_roc_rf = auc(false_positive_rate_rf, true_positive_rate_rf)
auc_roc_rf
```

```
0.8456984762540318
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate_rf,true_positive_rate_rf, color='red',label = 'AUC = %0.2f' % auc_roc_rf)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```



The ROC curve for the random forest is slightly above the 45-degree line as compared to the logistic regression model. The AUC being 0.85, states that it is a good predicting model and can give better results than the logistic regression model.

4. Naïve Bayes Classifier:

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Naive bayes

```
from sklearn.naive_bayes import GaussianNB
model_naive = GaussianNB()
model_naive.fit(x1_train, y1_train)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

```
# This will give you positive class prediction probabilities
y_prob_navie = model_naive.predict_proba(X_test)[:,1]

# This will threshold the probabilities to give class predictions.
y_pred_navie = np.where(y_prob_navie > 0.5, 1, 0)

model_naive.score(x1_test, y_pred_navie)
```

```
0.9379582628313593
```

```
print("Number of mislabeled points from %d points : %d"
% (x1_test.shape[0],(y1_test!= y_pred_navie).sum()))
```

```
Number of mislabeled points from 1773 points : 1616
```

```
navie_cv_score = cross_val_score(model_naive, x1, y1, cv=10, scoring='roc_auc')
```

Like the previous models, we apply the model to the training dataset and find out the probabilities and predictions. For this model we find out the quantity of mislabeled points which is 1616 out of 1773 points and we continue getting the cross-validation score for the model.

```
print("=== Confusion Matrix ===")
print(confusion_matrix(y1_test, y_pred_navie))
print('\n')
print("=== Classification Report ===")
print(classification_report(y1_test, y_pred_navie))
print('\n')
print("=== All AUC Scores ===")
print(navie_cv_score)
print('\n')
print("=== Mean AUC Score ===")
print("Mean AUC Score - Navie Bayes: ", navie_cv_score.mean())
```

```
=== Confusion Matrix ===
[[  52 1613]
 [   3  105]]


=== Classification Report ===
              precision    recall  f1-score   support

           0       0.95      0.03      0.06      1665
           1       0.06      0.97      0.12       108

    accuracy                           0.09      1773
   macro avg       0.50      0.50      0.09      1773
weighted avg       0.89      0.09      0.06      1773
```

```
=== All AUC Scores ===
[0.76252772 0.81645233 0.73374723 0.65095344 0.72682927 0.64784922
 0.72594235 0.72062084 0.74638581 0.76321508]


=== Mean AUC Score ===
Mean AUC Score - Navie Bayes:  0.7294523281596452
```

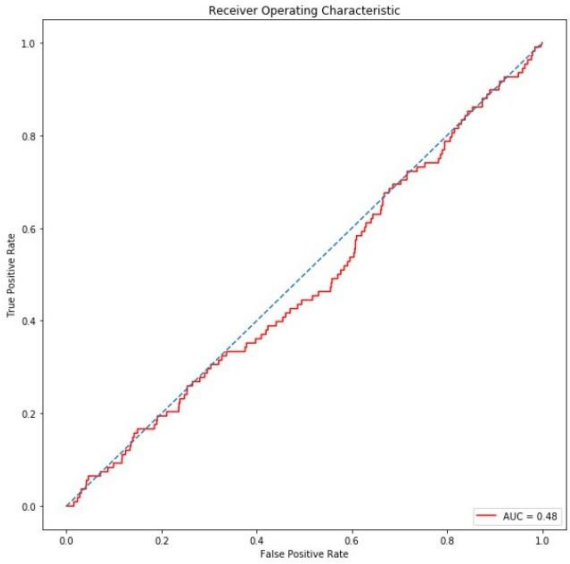
From the figure it can be noticed the confusion matrix outcome for this model is quite different from the previous models. Here, the count of true negatives is $C_{0,0}$ (52), false negatives is $C_{1,0}$ (3), true positives is $C_{1,1}$ (105) and false positives is $C_{0,1}$ (1613).

```
from sklearn.metrics import roc_curve, auc
false_positive_rate_navie, true_positive_rate_navie, thresholds = roc_curve(y1_test, y_prob_navie)
auc_roc_navie = auc(false_positive_rate_navie, true_positive_rate_navie)
auc_roc_navie
```

```
0.47997441886330766
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate_navie,true_positive_rate_navie, color='red',label = 'AUC = %0.2f' % auc_roc_navi
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

```
Text(0.5, 0, 'False Positive Rate')
```



For Naïve Bayes' model the area under the ROC curve is 0.48. This is a bad output as the value of AUC is below 0.5, the predicting model fails to identify two kinds of companies- bankrupt and normal.

5. Support Vector Machine:

Support Vector Machine, abbreviated as SVM can be used for both regression and classification tasks. But it is widely used in classification objectives. The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points. To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e. the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence. Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

Support Vector Machine

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

svm_model= SVC()
```

```
# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}

svm_model_grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)

# fitting the model for grid search
svm_model_grid.fit(x1_train, y1_train)
```

```
# print best parameter after tuning
print(svm_model_grid.best_params_)

# print how our model looks after hyper-parameter tuning
print(svm_model_grid.best_estimator_)
```

```
{'C': 0.1, 'gamma': 1, 'kernel': 'rbf'}
SVC(C=0.1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=1, kernel='rbf', max_iter=-1,
    probability=False, random_state=None, shrinking=True, tol=0.001,
    verbose=False)
```

```
y_pred_svm= svm_model_grid.predict(x1_test)
print(metrics.accuracy_score(y_pred_svm,y1_test))
```

```
0.9390862944162437
```

```
svm_cv_score = cross_val_score(svm_model_grid, x1, y1, cv=10, scoring='roc_auc')
```

The model is applied to the training dataset once the parameter range for grid search is defined, then we obtain the best parameter after tuning along with the model's status after hyper-parameter tuning. Like in the previous steps, we find the cross-validation score of the model.

```
print("=== Confusion Matrix ===")
print(confusion_matrix(y1_test, y_pred_svm))
print('\n')
print("=== Classification Report ===")
print(classification_report(y1_test, y_pred_svm))
print('\n')
print("=== All AUC Scores ===")
print(svm_cv_score)
print('\n')
print("=== Mean AUC Score ===")
print("Mean AUC Score - SVM: ", svm_cv_score.mean())
```

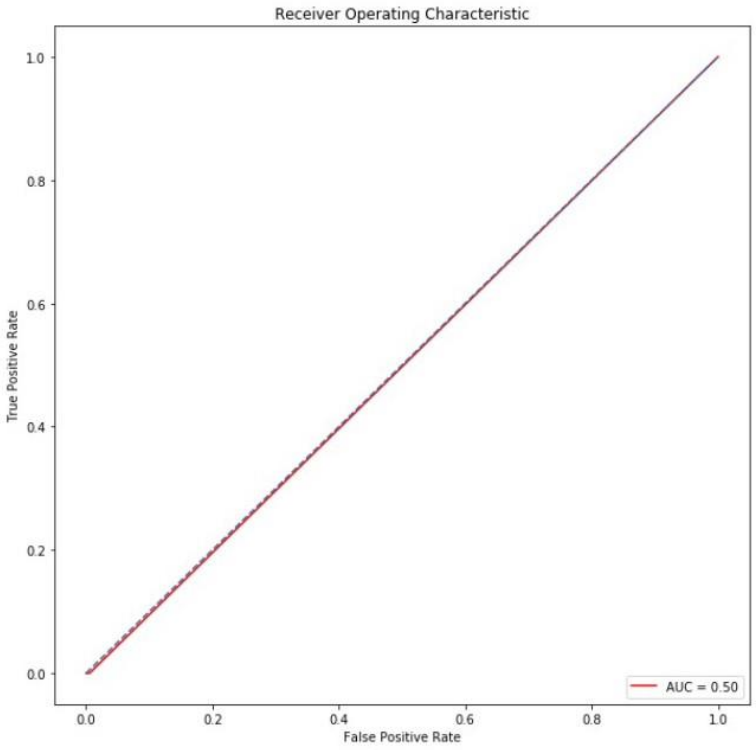
```
=== Confusion Matrix ===
[[1665    0]
 [ 108    0]]


=== Classification Report ===
              precision    recall  f1-score   support

           0       0.94      1.00      0.97      1665
           1       0.00      0.00      0.00       108

    accuracy                           0.94      1773
   macro avg       0.47      0.50      0.48      1773
weighted avg       0.88      0.94      0.91      1773


=== All AUC Scores ===
[0.61578714 0.58046563 0.55662971 0.53028825 0.55121951 0.61658537
 0.53995565 0.60492239 0.56543237 0.61113082]


=== Mean AUC Score ===
Mean AUC Score - SVM:  0.5772416851441242
```

The confusion matrix gives us the count of true negatives is $C_{0,0}$ (1665), false negatives is $C_{1,0}$ (108), true positives is $C_{1,1}$ (0) and false positives is $C_{0,1}$ (0).

```
from sklearn.metrics import roc_curve, auc
false_positive_rate_svm, true_positive_rate_svm, thresholds = roc_curve(y_test, y_pred)
auc_roc_svm = auc(false_positive_rate_svm, true_positive_rate_svm)
auc_roc_svm
```

```
0.49725943970767356
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate_svm,true_positive_rate_svm, color='red',label = 'AUC = %0.2f' % auc_roc_svm)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')

```



Since the area under the ROC curve is 0.50, we have a complete overlap between the results from the bankrupt companies and the results from the normal companies, we have a worthless output. A worthless output has a discriminating ability equal to flipping a coin.

6. K Nearest Neighbors:

KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. To evaluate any technique, we generally look at 3 important aspects:

- Ease to interpret output
- Calculation time
- Predictive Power

KNN algorithm fairs across all parameters of considerations. It is commonly used for its easy of interpretation and low calculation time. KNN works by finding the distances between a query and all the examples in the data, selecting the

specified number examples (K) closest to the query, then votes for the most frequent label.

K Nearest Neighbors

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x1_train, y1_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
y_pred_Knn = classifier.predict(X_test)
```

```
y_prob_knn = classifier.predict_proba(x1_test)[:,1]
```

```
KNN_cv_score = cross_val_score(model_naive, x1, y1, cv=10, scoring='roc_auc')

```

Under this model we set the parameter n_neighbors at 5 and then we apply the model to the training dataset. Later we obtain the probabilities and prediction of the model and eventually find the cross-validation score for the same.

```
print("=== Confusion Matrix ===")
print(confusion_matrix(y1_test, y_pred_Knn))
print('\n')
print("=== Classification Report ===")
print(classification_report(y1_test, y_pred_Knn))
print('\n')
print("=== All AUC Scores ===")
print(KNN_cv_score)
print('\n')
print("=== Mean AUC Score ===")
print("Mean AUC Score - KNN: ", KNN_cv_score.mean())
```

```
=== Confusion Matrix ===
[[1644   21]
 [ 107    1]]


=== Classification Report ===
              precision    recall  f1-score   support

           0       0.94      0.99      0.96      1665
           1       0.05      0.01      0.02       108

    accuracy                           0.93      1773
   macro avg       0.49      0.50      0.49      1773
weighted avg       0.88      0.93      0.90      1773


=== All AUC Scores ===
[0.76252772 0.81645233 0.73374723 0.65095344 0.72682927 0.64784922
 0.72594235 0.72062084 0.74638581 0.76321508]


=== Mean AUC Score ===
Mean AUC Score - KNN:  0.7294523281596452
```

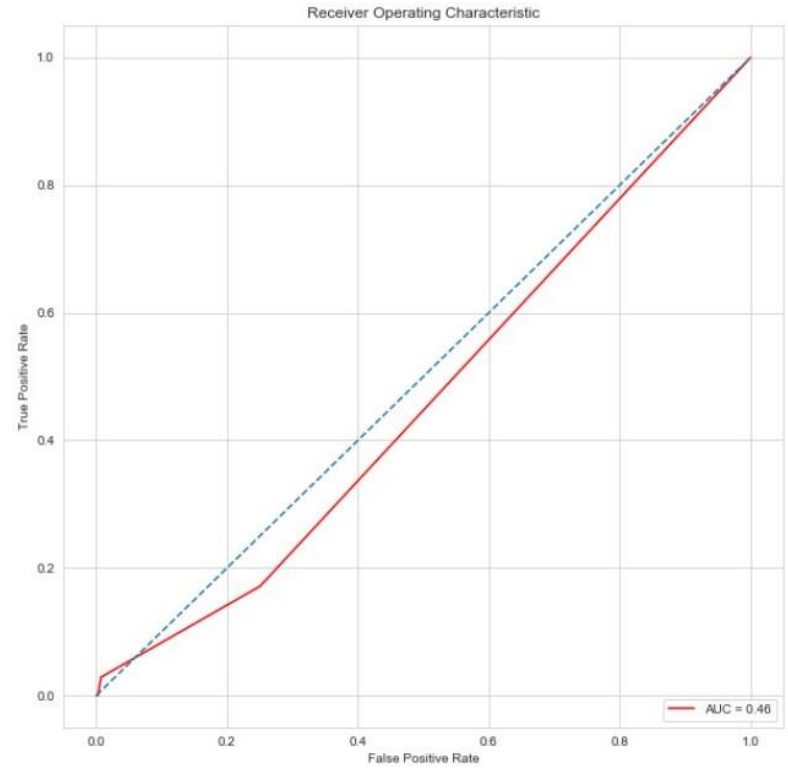
The confusion matrix gives us the count of true negatives is $C_{0,0}$ (1644), false negatives is $C_{1,0}$ (107), true positives is $C_{1,1}$ (1) and false positives is $C_{0,1}$ (21).

```
from sklearn.metrics import roc_curve, auc
false_positive_rate_knn, true_positive_rate_knn, thresholds = roc_curve(y_test, y_prob_knn)
auc_roc_Knn = auc(false_positive_rate_knn, true_positive_rate_knn)
auc_roc_Knn
```

```
0.46334932054356515
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate_knn,true_positive_rate_knn, color='red',label = 'AUC = %0.2f' % auc_roc_Knn)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')

```

```
Text(0.5, 0, 'False Positive Rate')
```



The KNN model is so far the worse model since value of AUC is 0.46 which is below 0.5. Therefore, the predicting model again fails to identify whether a company is bankrupt or not.

7. AdaBoost Classifier:

Ada-boost or Adaptive Boosting is one of ensemble boosting classifier proposed by Yoav Freund and Robert Schapire in 1996. It combines multiple classifiers to increase the accuracy of classifiers. AdaBoost is an iterative ensemble method. AdaBoost classifier builds a strong classifier by combining multiple poorly performing classifiers so that you will get high accuracy strong classifier. The basic concept behind Adaboost is to set the weights of classifiers and training the data sample in each iteration such that it ensures the accurate predictions of unusual observations.

AdaBoost Classifier

```
from sklearn.ensemble import AdaBoostClassifier
```

```
abc = AdaBoostClassifier(n_estimators=50,learning_rate=1)

# Train Adaboost Classifer
model_adaboost = abc.fit(x1_train, y1_train)
```

```
#Predict the response for test dataset
y_pred_adaBoost = model_adaboost.predict(x1_test)

y_prob_adaBoost = model_adaboost.predict_proba(x1_test)[:,1]
```

```
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_adaBoost))
```

```
Accuracy: 0.9007332205301748
```

```
adaboost_cv_score = cross_val_score(model_naive, x1, y1, cv=10, scoring='roc_auc')
```

We fit the model to the training dataset and subsequently find the prediction of response and probability for the model. The accuracy of the model is said to be 90%. Afterwards we apply cross-validation to get cv score of the model.

```
print("=== Confusion Matrix ===")
print(confusion_matrix(y1_test, y_pred_adaBoost))
print('\n')
print("=== Classification Report ===")
print(classification_report(y1_test, y_pred_adaBoost))
print('\n')
print("=== All AUC Scores ===")
print(adaboost_cv_score)
print('\n')
print("=== Mean AUC Score ===")
print("Mean AUC Score - AdaBoost: ", adaboost_cv_score.mean())
```

```
=== Confusion Matrix ===
[[1643   22]
 [  72   36]]


=== Classification Report ===
              precision    recall  f1-score   support

           0       0.96      0.99      0.97      1665
           1       0.62      0.33      0.43       108

    accuracy                           0.95      1773
   macro avg       0.79      0.66      0.70      1773
weighted avg       0.94      0.95      0.94      1773


=== All AUC Scores ===
[0.76252772 0.81645233 0.73374723 0.65095344 0.72682927 0.64784922
 0.72594235 0.72062084 0.74638581 0.76321508]


=== Mean AUC Score ===
Mean AUC Score - AdaBoost:  0.7294523281596452
```

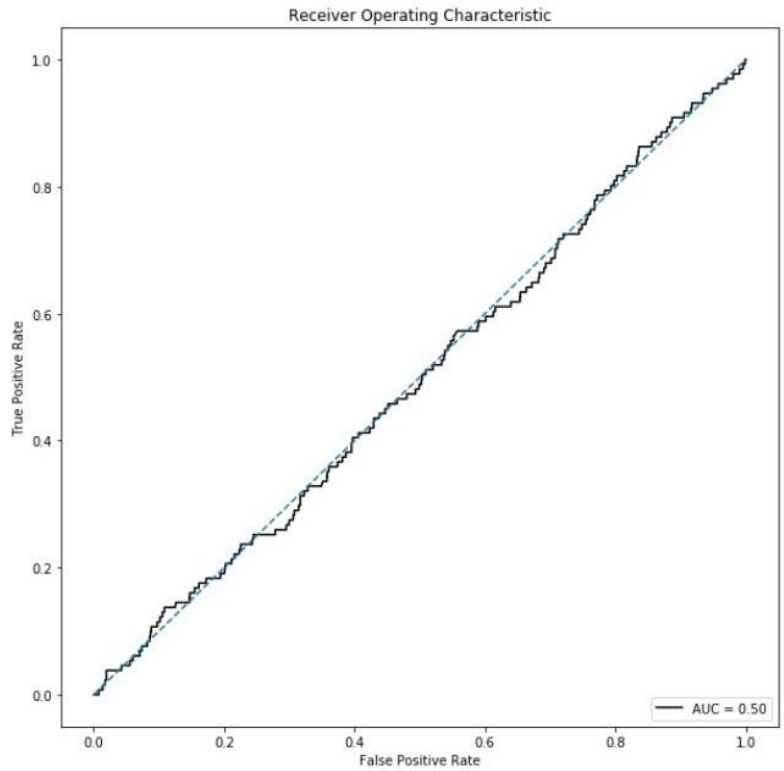
The confusion matrix gives us the count of true negatives is $C_{0,0}$ (1643), false negatives is $C_{1,0}$ (72), true positives is $C_{1,1}$ (36) and false positives is $C_{0,1}$ (22). Then we obtain the classification report and mean AUC score from all pairwise class comparisons, here is 0.73 (approx.).

```
from sklearn.metrics import roc_curve, auc
false_positive_rate_ada, true_positive_rate_ada, thresholds = roc_curve(y_test, y_prob_adaBoost)
auc_roc_ada = auc(false_positive_rate_ada, true_positive_rate_ada)
auc_roc_ada
```

```
0.5170139051453719
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate_ada,true_positive_rate_ada, color='black',label = 'AUC = %0.2f' % auc_roc_ada)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],linestyle='--')
plt.axis('tight')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')

```

```
Text(0.5, 0, 'False Positive Rate')
```



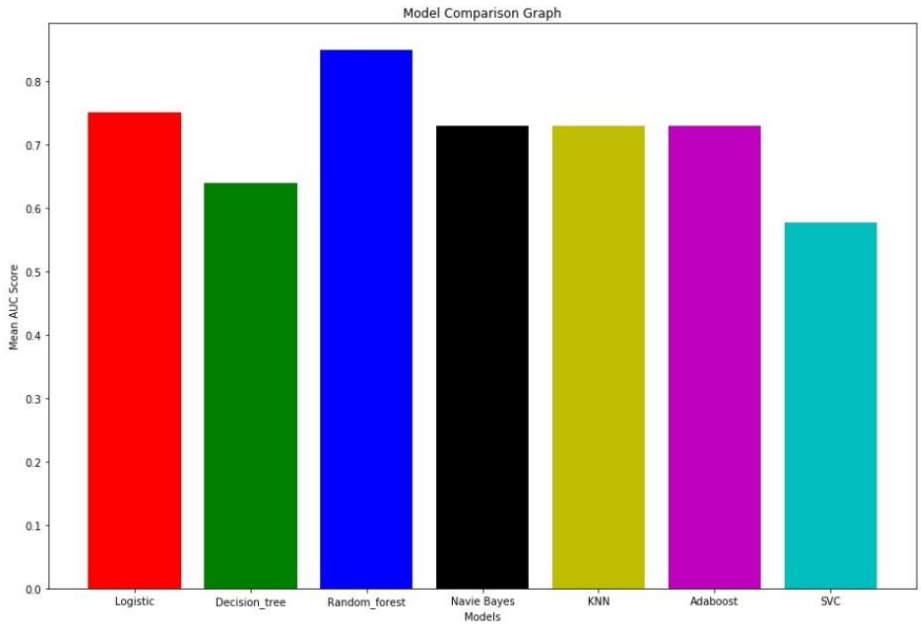
Similar to the SVM model, this model also is a poor predictive model as the area under the ROC curve is 0.5. And when AUC is 0.5, it means model has no class separation capacity whatsoever.

Conclusion

In this project we have achieved our goal of testing different classification methods – Logistic Regression, Decision Tree, Random Forest, Naïve Bayes, Support Vector Machine, K Nearest Neighbors and AdaBoost Classifier. In our final phase, the journey of finding the best model amongst the seven different models comes to an end, we have arrived at the decision that Random Forest model is the best classification technique among them all.

As from the results achieved in this project, we can say that area under the ROC curve was the highest in the Random Forest model (as seen in the two figures below) than the rest. The AUC value helps us in understanding the predictive power a model has in differentiating whether a company is bankrupt or just a normal company. Hence the model can give us more accurate result after analyzing all the various attributes.

As we can see apart from Random Forest model the other model which comes close to it giving good result is the Logistic Regression model, whose AUC value is 0.76. While there are some models who perform poorly like- Naïve Bayes, Support Vector Machine, K Nearest Neighbors and AdaBoost Classifier. The issue that is seen in the model with no capacity of accurate prediction is that their AUC values, in most cases, fall below 0.5.



Lessons Learnt

Selection of data set is very important before selecting the dataset, we didn't understand the importance of the domain knowledge. For us it's was a pretty challenging task to understand the features contribution which lead to company bankrupt. So, it is very important that we should at least have little bit of domain Knowledge otherwise it just becomes like, so far so good, if output is coming then its good, and would lose interest. So, it is very important that we should choose the data that we should at least interpret the impact of features. Too large dataset gave hazy results.

Training the data is time consuming, so we have to apply the techniques in ways that it will reduce the time complexity. Which is very much important as per the industry perspective many times we don't even realize, which ultimately leads to the escalated costs.

Bibliography

1. https://google.com
2. https://scikit-learn.org/stable/
3. https://towardsdatascience.com
4. https://analyticsvidhya.com
5. https://youtube.com
6. https://stackexchange.com/
7. https://stackoverflow.com/
8. https://www.quora.com/