Project Report

On

Bankruptcy Prediction Using Machine Learning Techniques

Submitted in partial fulfillment for the award of

Post Graduate Diploma in Big Data Analytics (PG-DBDA)

From CDAC-ACTS(Chennai)

Guided by:

Ms.N. Rekha

Submitted By:

Kartik Chaudhari (+917263054292)

CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

This is to certify that

Kartik Chaudhari (+917263054292)

Have successfully completed their project on

Bankruptcy Prediction Using Machine Learning Techniques

Under the guidance of Ms.N. Rekha

ACKNOWLEDGEMENT

This project **"Bankruptcy Prediction Using Machine Learning Techniques"** was a great learning experience for us and we are submitting this work to CDAC (Pune).

We all are very glad to mention the name of **Ms.N Rekha** and **Mr. Harikrishnan** for his valuable guidance to work on this project. His guidance and support helped us to overcome various obstacles and intricacies during the course of project work.

We are highly grateful to **Mr. L.R.Prakash** (Senior Director and Center Head, Acts Chennai) for his guidance and support whenever necessary while doing this course Post Graduate Diploma in Big Data Analytics (PG-DBDA) through C-DAC ACTS, Pune.

Our most heartfelt thanks goes to **Mrs. Sunandha Duraisamy and Mrs. Sumitra** (Course Coordinator, PG-DBDA) who gave all the required support and kind coordination to provide all the necessities like required hardware, internet facility and extra Lab hours to complete the project and throughout the course up to the last day here in C-DAC ACTS, Chennai.

From:

Kartik Chaudhari
(+917263054292)

TABLE OF CONTENTS

1. INTRODUCTION

In their development, almost all companies go through debt phases to finance their activities. Some manage to develop, emerge and reach maturity while others have much more difficulty and go bankrupt. Everything starts from a situation of financial distress that translates into an abnormally high debt and can end in total bankruptcy.

The financial institutions that grant loans to businesses try to predict, thanks to the financial variables of the businesses, if they will be able to repay the loans in time. Predicting the failure of a company can therefore be very important for both sides.

On the side of the companies, it would allow to evaluate the internal dynamics and to see where the policies in place lead the company in order to direct new policies, new investments, etc.

For financial institutions, it would allow them to make appropriate lending decisions. The model of predicting the failure of a company was introduced in 1968 by Altman. According to Sun, Li, Huang & He in 2014, mild financial distress can be expressed as a temporary cash flow difficulty, such as the concepts of insolvency, default and the most serious is the bankruptcy of the company.

Therefore, the objective of our work is to use Under-sampling and Over-Sampling techniques and machine learning models to build predictive models of company bankruptcy using its financial information.

2. **Abstract**

Our study aims at predicting business failure. It was conducted using data from 6819 companies and 96 variables. After cleaning the database, we ended up with 93 variables. Also, given the imbalance observed in the classes of the variable of interest, we rebalanced the classes in the learning curve before estimating our different models. From our analysis, it appears that high values of the debt ratio and of the current liabilities to assets ratio are reasons that justify the bankruptcy of firms. On the other hand, high or increasing net income to total assets and return on assets allow firms to avoid bankruptcy and perform better. Moreover, the estimation of different models revealed that the random forest is the one that best predicts whether firms will fail or not with an accuracy of 99%, an estimated precision of about 88%, a recall of about 92% and an f1-score of 85%.

3. Literature Review

There are several ambiguities about the actual definition of business failure among the various authors who have worked on the subject. Some authors define it simply as the failure of the company, others as financial distress. Thus, for Altman and Hotchkiss in 2006, there are four terms to describe unsuccessful firms: failure, insolvency, default and bankruptcy. They describe failure as the rate of return realized on invested capital that is dramatically and persistently lower than the prevailing rates on equivalent risk- adjusted investments; insolvency as a firm's inability to meet its current obligations; default as a firm's failure to meet an obligation, particularly the payment of a loan or appearance in a lower court; and finally, they believe that there are two types of bankruptcy: one that relates to the net worth of the business and the second that refers to the formal declaration of the business in a federal district court accompanied by a petition either to liquidate its assets or to attempt a reorganization program.

But this definition of bankruptcy by Altman and Hotchkiss does not meet with unanimous approval because for Ross, Westfield and Jaffe in 1999, there are three types of bankruptcy: legal bankruptcy, technical bankruptcy and accounting bankruptcy. Legal bankruptcy just means that the company goes to court to obtain a declaration of bankruptcy. Technical bankruptcy which describes the situation in which a company cannot fulfill the contract within the stipulated time and accounting bankruptcy which refers to the situation in which a company simply has negative net book assets.

In a somewhat similar sense, Korol & Korodi in 2011, study so-called early warning signals to predict bankruptcy risk. The different authors had different approaches according to their time and the study techniques developed in their time Altman was one of the first to introduce a prediction model with a 5-factor discriminant analysis. The main models developed at the beginning were multivariate discriminant analysis, logit and probit models. As statistical techniques improved, new models were developed with the rise of artificial intelligence, the neural network became a widely used tool. For example, Pan (2012) wanted to optimize the general regression neural network model by applying the algorithm and obtained good convergence results that indicate the good predictive ability of the model. For example, models such as the support vector machine were introduced in 2005 by Shin, Lee & Kim, 2005; Min & Lee.

4. Materials and Methods

4.1. Methodology:

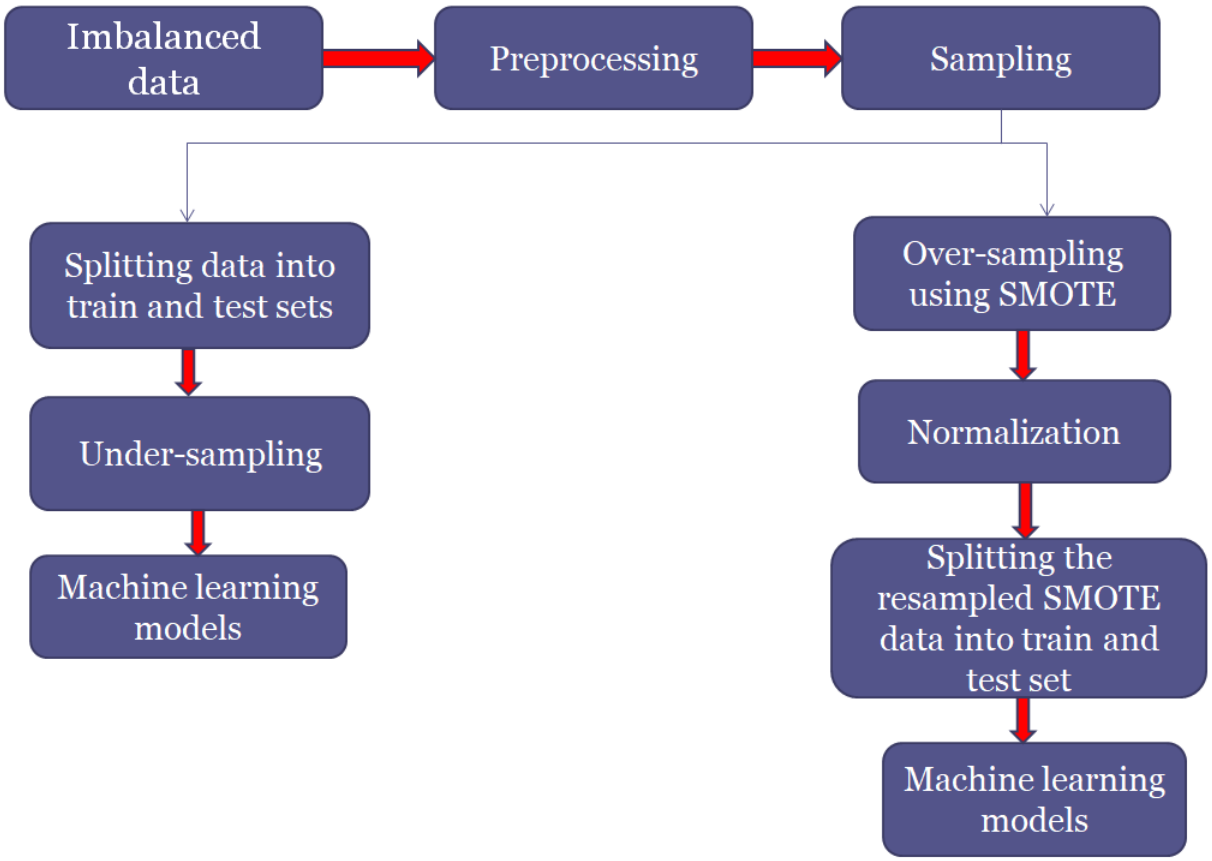


Fig 4.1.1: Bankruptcy Prediction

4.2. Data description:

A company faces bankruptcy when they are unable to pay off their debts. The Taiwan Economic Journal for the years 1999 to 2009 has listed the details of company bankruptcy based on the business regulations of the Taiwan Stock Exchange. The Taiwan Stock Exchange was established in 1961 and began operating as a stock exchange on 9 February 1962. It is a financial institution located in Taipei, Taiwan. It has over 900 listed companies. The data includes a majority of numerical attributes that help understand the possibility of bankruptcy. It contains 6819 companies, 96 attributes, two categories. There is a huge imbalance in the database with 96.774% non-bankrupt firms and 3.226% bankrupt firms. Firms in bankruptcy and not in bankruptcy are marked separately as '1' and '0'. Data is from:

https://archive.ics.uci.edu/ml/datasets/Taiwanese+Bankruptcy+Prediction

4.3. Appraisal of the database:

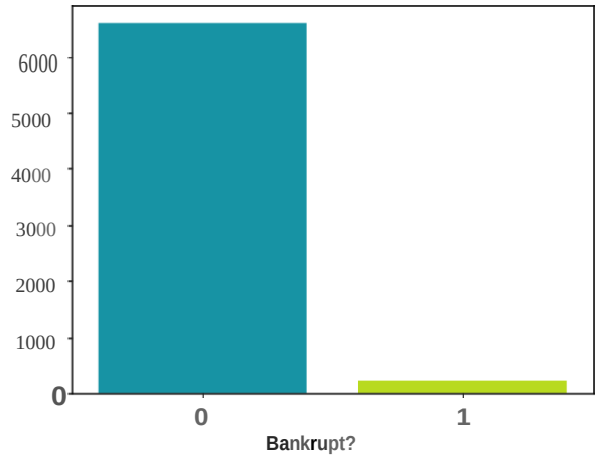
The database initially contains 95 explanatory variables and the dependent variable, which is whether or not the firm is bankrupt. A first exploration of the data through the verification of the missing data showed us that there are no missing values in the database. Then, the exploration of the types of variables revealed that we have 3 categorical variables including the dependent variable and 93 quantitative variables. Through univariate analyses on the qualitative variables of the database, we noticed that apart from the dependent variable, the two others qualitative explanatory variables present very unbalanced classes. All (100%) of the observations are found in a single class for one and more than 99% of the observations are found in a single class for the second variable. We therefore removed these variables from our analysis. Also, since the dependent variable has unbalanced classes, we rebalanced the classes only on the train set within this variable before estimating the different models.

4.4. Descriptive statistics:

Univariate analysis

We notice the imbalance between the modalities of the variable of interest, hence the use of a rebalancing technique. In fact, about 3% of the companies in our database have gone bankrupt and 97% are not bankrupt

Figurel: Distribution of firms by business failure



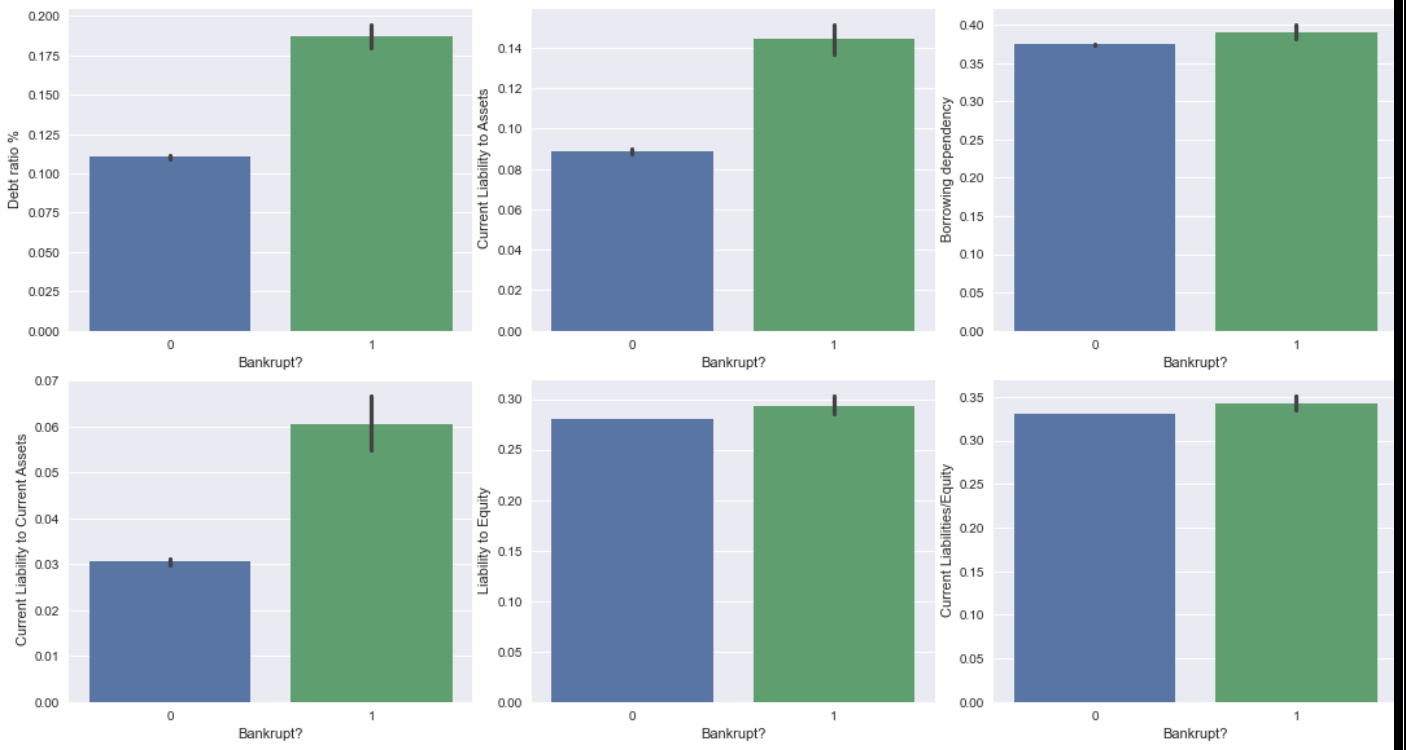
Variable	Outcome	Count	Percent
Bankrupt?	0	6599	96.77
	1	220	3.23

Source : Realized by the author

Bivariate analysis

We note a strong positive relationship between the features "Debt Ratio_%, Current_Liability_To_Assets, Current_Liability_ To_ Current Assets" and bankruptcy. Indeed, these indicators are much higher in companies that have gone bankrupt than in those that have not.

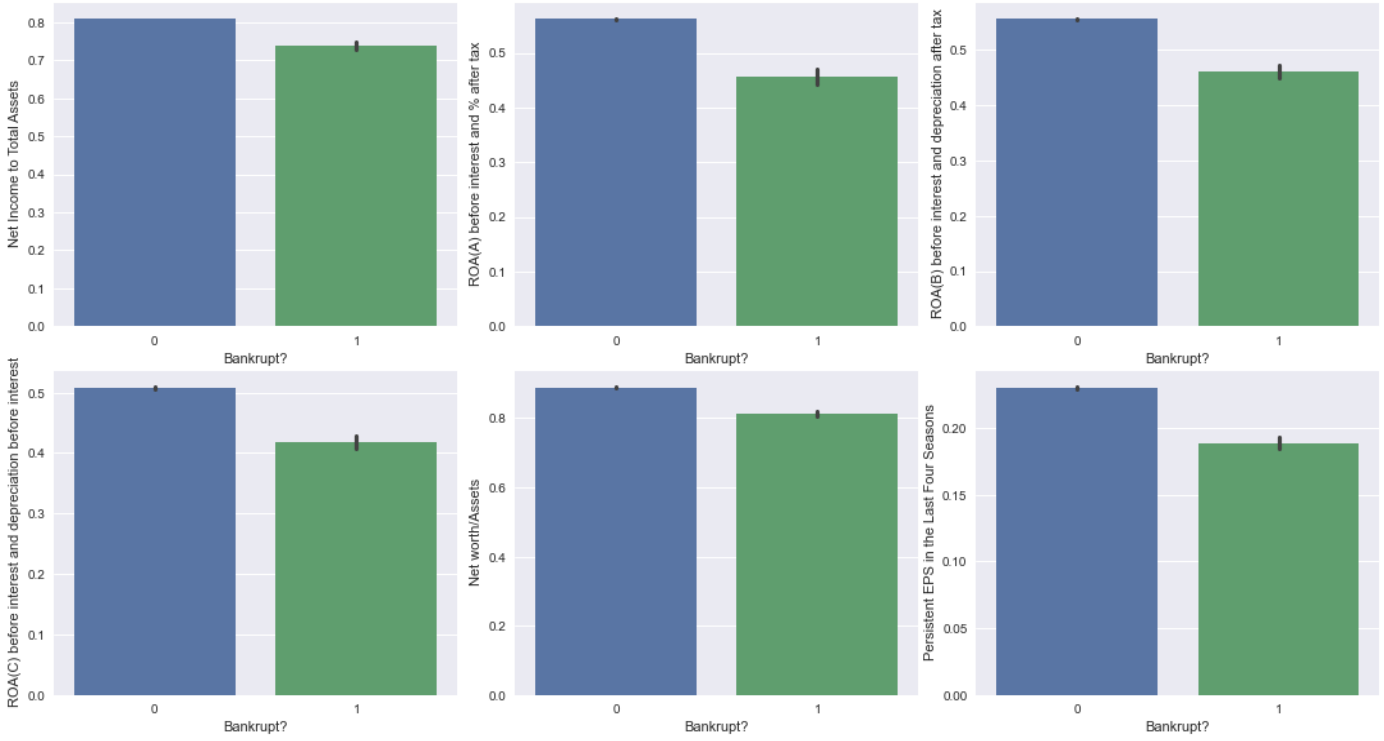
Figure2: Positive correlation with the target attributes



Source Realized by the author

Furthermore, we observe a negative relationship between bankruptcy and features "_ROA(C)_before_interest_and_depreciation_before_interest,_ROA(B)_before_interest_and_depreciation_after_tax,_Net_Income_to_Total_Assets". Indeed, these indicators are lower in firms that have gone bankrupt compared to those that have not.

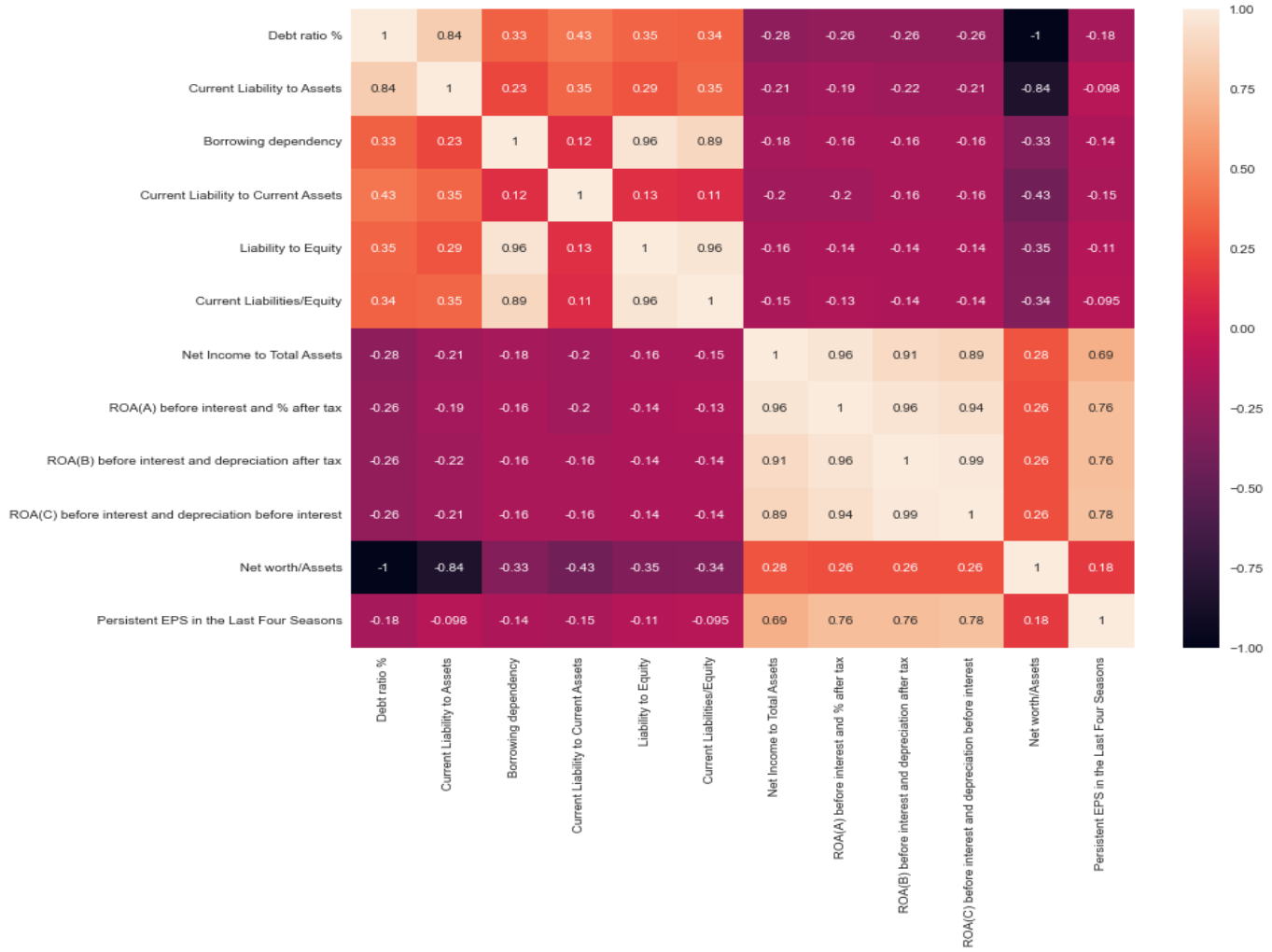
Figure 3: Negative correlation with the target attribute



Source: Realized by the author

Looking at the correlation matrix, we notice a strong linear relationship between some of our explanatory variables, suggesting problems of multicollinearity.

Figure 4: Correlation matrix



Source : Realized by the authors

4.5. Database rebalancement:

As we have 96.774% non-bankrupt firms and 3.226% bankrupt firms, we have to rebalance the database.

In the presence of unbalanced data, resampling is a solution often used to overcome the problem. There are two types of methods: subsampling and oversampling. When there are two unbalanced classes, the preferred operation is oversampling, i.e. generating observations in the underrepresented class until it reaches the second class. This technique is preferred because in the case of under sampling, observations that may contain important information for future analyses are removed. SMOTE is an oversampling technique based on the K-Nearest Neighbors algorithm and allows to generate synthetic data from the existing ones in the dataset. First, the number of observations to be created is defined, an instance is randomly selected and the algorithm is iterated until the classes are rebalanced. At the level of each new observation, values are assigned by multiplying the distance between the point that was used to generate it and it by a value between O and 1 and this value is added to the original vector.

5. SYSTEM REQUIREMENTS

5.1. Hardware Requirements:

- Platform – Windows 10
- RAM – 8 GB of RAM,
- Peripheral Devices – Mouse, Keyboard, Monitor
- A network connection for data recovering over network.

5.2. Software Requirements:

- Python 3.7
- Jupyter Notebook
- Machine Learning

6. Methods and Results

6.1. Under-Sampling:

Under sampling refers to a group of techniques designed to balance the class distribution for a classification dataset that has a skewed class distribution.

An imbalanced class distribution will have one or more classes with few examples (the minority classes) and one or more classes with many examples (the majority classes). It is best understood in the context of a binary (two-class) classification problem where class 0 is the majority class and class 1 is the minority class.

Under sampling techniques remove examples from the training dataset that belong to the majority class in order to better balance the class distribution, such as reducing the skew from a 1:100 to a 1:10, 1:2, or even a 1:1 class distribution. This is different from oversampling that involves adding examples to the minority class in an effort to reduce the skew in the class distribution.

Under sampling methods can be used directly on a training dataset that can then, in turn, be used to fit a machine learning model. Typically, under sampling methods are used in conjunction with an oversampling technique for the minority class, and this combination often results in better performance than using oversampling or under sampling alone on the training dataset. The simplest under sampling technique involves randomly selecting examples from the majority class and deleting them from the training dataset. This is referred to as random under sampling. Although simple and effective, a limitation of this technique is that examples are removed without any concern for how useful or important they might be in determining the decision boundary between the classes. This means it is possible or even likely, that useful information will be deleted.

Taking a Sample from the Data

In [22]:
```
#Take sample to balance the data
bankrupt_sample = df[df['Bankrupt?'] == 0][0:220]
non_bankrupt_sample = df[df['Bankrupt?'] == 1]

#creating a new data frame
new_df = pd.concat([bankrupt_sample,non_bankrupt_sample],axis = 0)
new_df.head()
```

Out[22]:

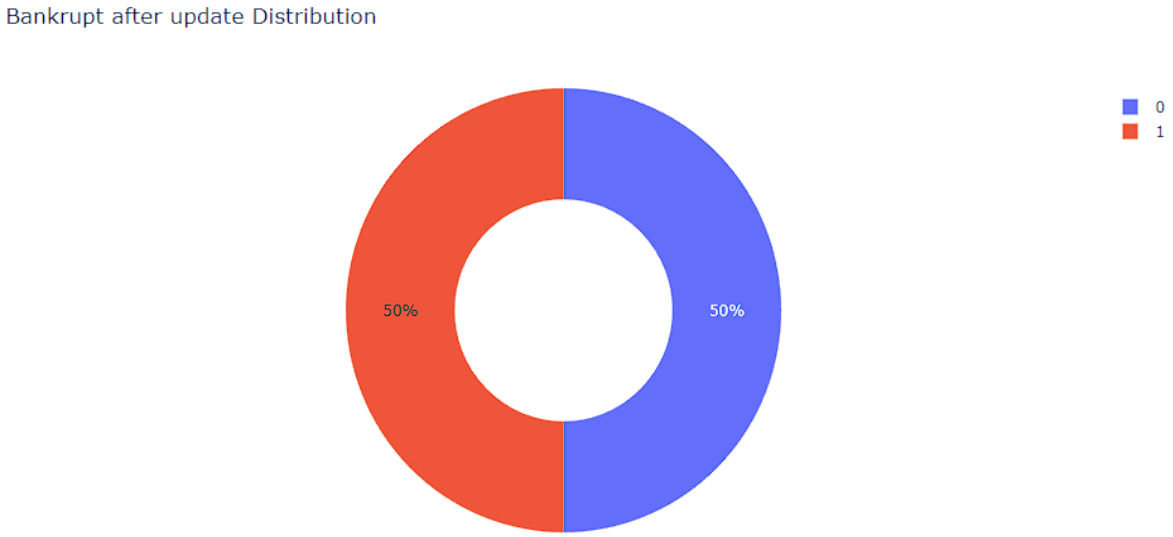
	Bankrupt?	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Realized Sales Gross Margin	Operating Profit Rate	Pre-tax net Interest Rate	After-tax net Interest Rate	Non-industry income and expenditure/revenue	...	Net Income to Total Assets	Total assets to GNP price	No-credit Interval
6	0	0.390923	0.445704	0.436158	0.619950	0.619950	0.998993	0.797012	0.808960	0.302814	...	0.736619	0.018372	0.623655
7	0	0.508361	0.570922	0.559077	0.601738	0.601717	0.999009	0.797449	0.809362	0.303545	...	0.815350	0.010005	0.623843
8	0	0.488519	0.545137	0.543284	0.603612	0.603612	0.998961	0.797414	0.809338	0.303584	...	0.803647	0.000824	0.623977
9	0	0.495686	0.550916	0.542963	0.599209	0.599209	0.999001	0.797404	0.809320	0.303483	...	0.804195	0.005798	0.623865
10	0	0.482475	0.567543	0.538198	0.614026	0.614026	0.998978	0.797535	0.809460	0.303759	...	0.814111	0.076972	0.623687

5 rows × 96 columns

Ploting Balanced Bankrupt data

```
In [23]: print("The new shape for our data",new_df.shape)
         # previewing the 'Bankrupt' column

         print(new_df['Bankrupt?'].value_counts())
         labels = new_df['Bankrupt?'].unique()
         values = new_df['Bankrupt?'].value_counts().sort_values(ascending = True)

         fig = go.Figure(data = [ go.Pie(labels = labels, values = values, hole = 0.5)])

         fig.update_layout(title_text = "Bankrupt after update Distribution")
         fig.show()
```

```
The new shape for our data (440, 96)
0    220
1    220
Name: Bankrupt?, dtype: int64
```



6.2. SMOTE (Synthetic Minority Over-Sampling Technique):

A problem with imbalanced classification is that there are too few examples of the minority class for a model to effectively learn the decision boundary.

One way to solve this problem is to oversample the examples in the minority class. This can be achieved by simply duplicating examples from the minority class in the training dataset prior to fitting a model. This can balance the class distribution but does not provide any additional information to the model.

An improvement on duplicating examples from the minority class is to synthesize new examples from the minority class.

This technique was described by Nitesh Chawla, et al. in their 2002 paper named for the technique titled "SMOTE: Synthetic Minority Over-sampling Technique."

SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line.

Specifically, a random example from the minority class is first chosen. Then k of the nearest neighbors for that example are found (typically $k=5$). A randomly selected neighbor is chosen and a synthetic example is created at a randomly selected point between the two examples in feature space.

This procedure can be used to create as many synthetic examples for the minority class as are required. As described in the paper, it suggests first using random undersampling to trim the number of examples in the majority class, then use SMOTE to oversample the minority class to balance the class distribution.

The approach is effective because new synthetic examples from the minority class are created that are plausible, which is, are relatively close in feature space to existing examples from the minority class.

A general downside of the approach is that synthetic examples are created without considering the majority class, possibly resulting in ambiguous examples if there is a strong overlap for the classes.

SMOTE Technique

- **SMOTE** actually creates as many synthetic examples for minority class as are requirred so that finally two target class are well represented. It does so by synthesising samples that are close to the feature space ,for the minority target class.

In [44]:
```
sm = SMOTE(random_state=123)
X_sm , y_sm = sm.fit_resample(X,y)

print(f'''Shape of X before SMOTE:{X.shape}
Shape of X after SMOTE:{X_sm.shape}''',"\n\n")

print(f'''Target Class distributuion before SMOTE:\n{y.value_counts(normalize=True)}
Target Class distributuion after SMOTE :\n{y_sm.value_counts(normalize=True)}''')
```

```
Shape of X before SMOTE:(6819, 95)
Shape of X after SMOTE:(13198, 95)


Target Class distributuion before SMOTE:
0    0.967737
1    0.032263
Name: Bankrupt?, dtype: float64
Target Class distributuion after SMOTE :
1    0.5
0    0.5
Name: Bankrupt?, dtype: float64
```

6.3. Random Forest Classifier:

The random forest or decision tree forest is a particularly efficient algorithm in terms of predictions in the field of machine learning. It is composed of several decision trees that each produce an estimate whose assembly allows to obtain a global estimate. It is based on the bagging system and the final estimate is made from a classification method that consists in choosing the category that comes up most often.

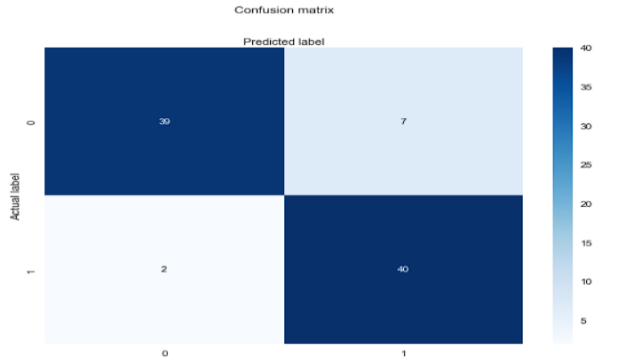
To apply the random forest, we have to choose the number of decision trees and the number of variables to implement.

For Under-Sampling:

Build Random Forest Classifier Model

```
# create the classifier with n_estimators = 100

clf = RandomForestClassifier(n_estimators=100, random_state = 777)

# fit the model to the training set
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
print('Model Accuracy is: ',accuracy_score(y_pred,y_test))

report = classification_report(y_test,y_pred)
print("-"*100)
print(report)

cm = confusion_matrix(y_test,clf.predict(X_test))
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

sns.heatmap(cm,annot=True,fmt="",cmap='Blues')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.savefig('cmrf.png')
```

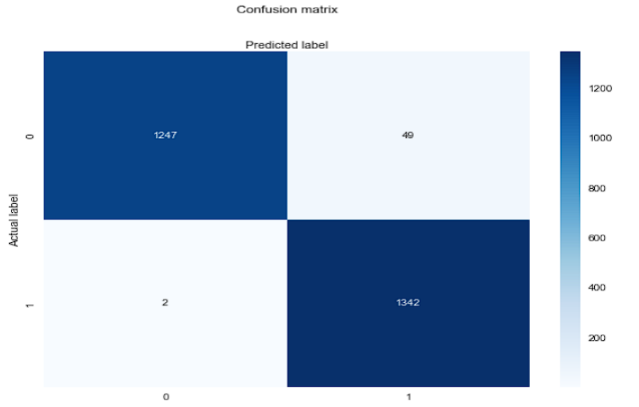
```
Model Accuracy is:  0.8977272727272727
----------------------------------------------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      0.85      0.90        46
           1       0.85      0.95      0.90        42

    accuracy                           0.90        88
   macro avg       0.90      0.90      0.90        88
weighted avg       0.90      0.90      0.90        88
```



For SMOTE:

```
# create the classifier with n_estimators = 100
clf = RandomForestClassifier(n_estimators=100, random_state = 777)

# fitting the model to the training set
clf.fit(X_train, y_train)

yrf_sm_pred = clf.predict(X_test)
print('Random Forest Model Accuracy is: ',accuracy_score(yrf_sm_pred, y_test))

report = classification_report(y_test, yrf_sm_pred)
print("-"*100)
print(report)

cm = confusion_matrix(y_test,clf.predict(X_test))
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

sns.heatmap(cm,annot=True,fmt="",cmap='Blues')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.savefig('cm_smrf.png')
```

```
Random Forest Model Accuracy is:  0.9806818181818182
----------------------------------------------------------------------------------------------------
              precision    recall  f1-score   support

           0       1.00      0.96      0.98      1296
           1       0.96      1.00      0.98      1344

    accuracy                           0.98      2640
   macro avg       0.98      0.98      0.98      2640
weighted avg       0.98      0.98      0.98      2640
```



The accuracy of the model is about 98%. This implies that the model, which seems satisfactory, very well predicted 98% of the companies. Also, the precision of the modelis about 96%, which is satisfactory. Digging a little deeper, we observe that the recall is about 96%, which is not bad since it implies that the model predicted very well or correctly classified 96% of the firms that actually failed. Further on, we also note thatthe f1-score associated with the model is about 98%. This model seems very satisfactoryin view of its performance.

6.4. Logistic Regression:

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). In other words, the logistic regression model predicts P(Y=1) as a function of X.

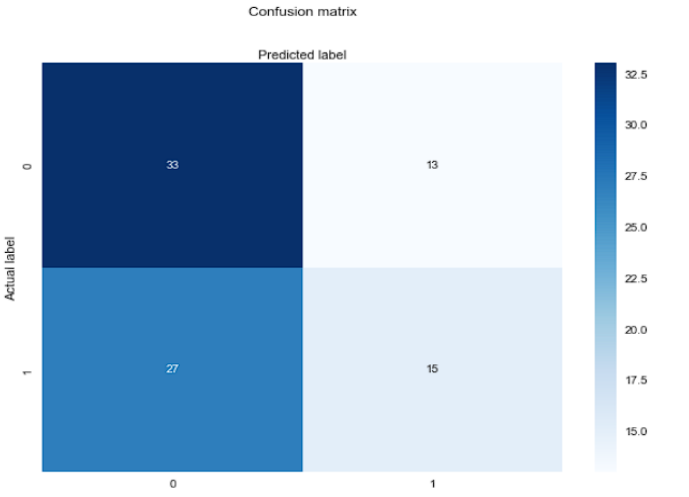
Logistic Regression Assumptions:

- Binary logistic regression requires the dependent variable to be binary.
- For a binary regression, the factor level 1 of the dependent variable should represent the desired outcome.
- Only the meaningful variables should be included.
- The independent variables should be independent of each other. That is, the model should have little or no multicollinearity.
- The independent variables are linearly related to the log odds.
- Logistic regression requires quite large sample sizes.

For Under-Sampling:

```
lr_model = LogisticRegression()

# fit the model to the training set
lr_model.fit(X_train,y_train)

accuracy_lr = lr_model.score(X_test,y_test)
print("Logistic Regression accuracy is :",accuracy_lr)
lr_pred= lr_model.predict(X_test)

report = classification_report(y_test,lr_pred)
print("-"*100)
print(report)

cm = confusion_matrix(y_test,lr_model.predict(X_test))
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

sns.heatmap(cm,annot=True,fmt="d",cmap = "Blues")
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.savefig('cmlr.png')
```

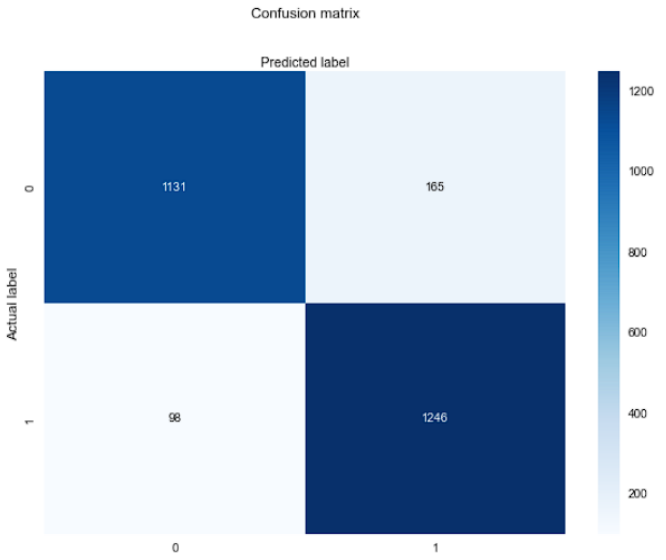
```
Logistic Regression accuracy is : 0.5454545454545454
----------------------------------------------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.55      0.72      0.62        46
           1       0.54      0.36      0.43        42

    accuracy                           0.55        88
   macro avg       0.54      0.54      0.53        88
weighted avg       0.54      0.55      0.53        88
```



For SMOTE:

```
lr_model = LogisticRegression()

# fit the model to the training set
lr_model.fit(X_train,y_train)

accuracy_lr = lr_model.score(X_test,y_test)
print("Logistic Regression accuracy is :",accuracy_lr)
lr_pred= lr_model.predict(X_test)

report = classification_report(y_test,lr_pred)
print("-"*100)
print(report)

cm = confusion_matrix(y_test,lr_model.predict(X_test))
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

sns.heatmap(cm,annot=True,fmt="d",cmap = "Blues")
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.savefig('cm_smlr.png')
```

```
Logistic Regression accuracy is : 0.9003787878787879
----------------------------------------------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.92      0.87      0.90      1296
           1       0.88      0.93      0.90      1344

    accuracy                           0.90      2640
   macro avg       0.90      0.90      0.90      2640
weighted avg       0.90      0.90      0.90      2640
```



6.5. Decision Tree Classifier:

Decision Tree algorithm belongs to the family of supervised learning algorithms. The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by **learning simple decision rules** inferred from prior data (training data).

In Decision Trees, for predicting a class label for a record we start from the **root** of the tree. We compare the values of the root attribute with the record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

Decision trees classify the examples by sorting them down the tree from the root to some leaf/terminal node, with the leaf/terminal node providing the classification of the example.

Each node in the tree acts as a test case for some attribute, and each edge descending from the node corresponds to the possible answers to the test case. This process is recursive in nature and is repeated for every sub-tree rooted at the new node.

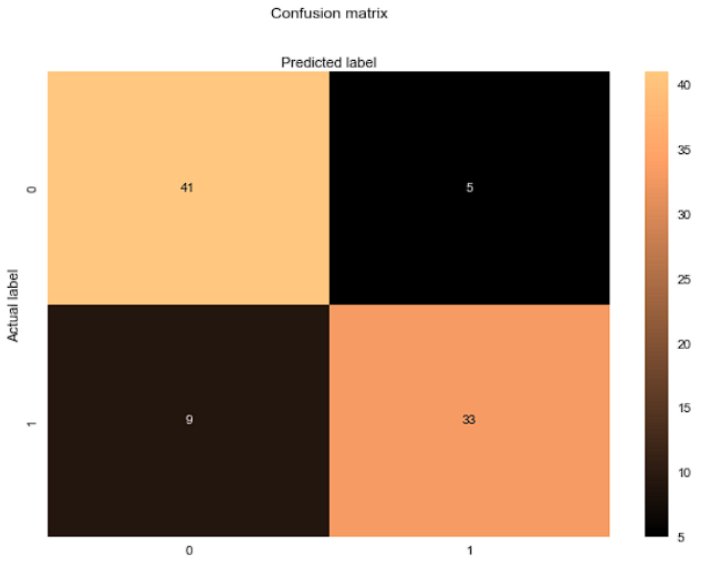
Assumptions while creating Decision Tree:

- In the beginning, the whole training set is considered as the **root.**
- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- Records are **distributed recursively** on the basis of attribute values.
- Order to placing attributes as root or internal node of the tree is done by using some statistical approach.

For Under-Sampling:

```
dt = DecisionTreeClassifier()

# fit the model to the training set
dt.fit(X_train,y_train)

ydt_pred = dt.predict(X_test)
accuracy = dt.score(X_test,y_test)
print("Decision Tree accuracy is :",accuracy)

report = classification_report(y_test,ydt_pred)
print("-"*100)
print(report)

cm = confusion_matrix(y_test,dt.predict(X_test))
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

sns.heatmap(cm,annot=True,fmt="d", cmap = 'copper')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.savefig('cmdcst.png')
```

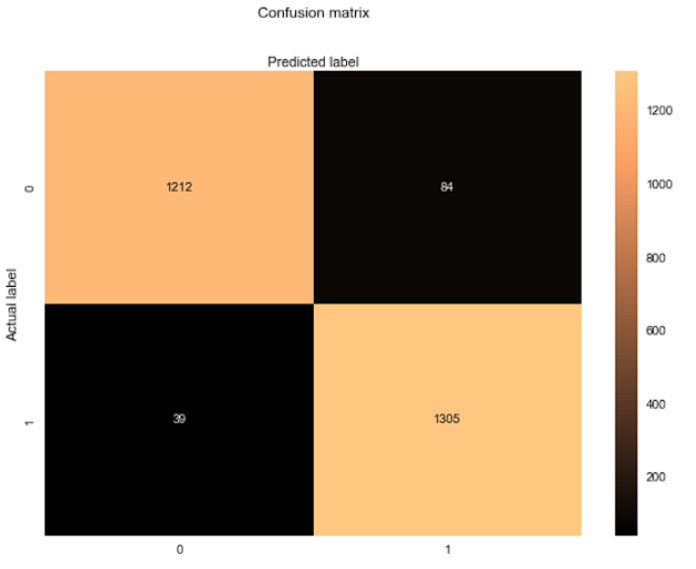
```
Decision Tree accuracy is : 0.8409090909090909
----------------------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.82      0.89      0.85        46
           1       0.87      0.79      0.82        42

    accuracy                           0.84        88
   macro avg       0.84      0.84      0.84        88
weighted avg       0.84      0.84      0.84        88
```



For SMOTE:

```
dt = DecisionTreeClassifier()

# fit the model to the training set
dt.fit(X_train,y_train)

ydt_sm_pred = dt.predict(X_test)
accuracy = dt.score(X_test,y_test)
print("Decision Tree accuracy is :",accuracy)

report = classification_report(y_test, ydt_sm_pred)
print("-"*100)
print(report)

cm = confusion_matrix(y_test,dt.predict(X_test))
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

sns.heatmap(cm,annot=True,fmt="d", cmap = 'copper')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.savefig('cm_smdcst.png')
```

```
Decision Tree accuracy is : 0.9534090909090909
----------------------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.97      0.94      0.95      1296
           1       0.94      0.97      0.95      1344

    accuracy                           0.95      2640
   macro avg       0.95      0.95      0.95      2640
weighted avg       0.95      0.95      0.95      2640
```



6.6. Ada Boost Classifier:

Ada-boost or Adaptive Boosting is one of ensemble boosting classifier which 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. Any machine learning algorithm can be used as base classifier if it accepts weights on the training set. Adaboost should meet two conditions:

The classifier should be trained interactively on various weighed training examples.

In each iteration, it tries to provide an excellent fit for these examples by minimizing training error.

It works in the following steps:

Initially, Adaboost selects a training subset randomly.

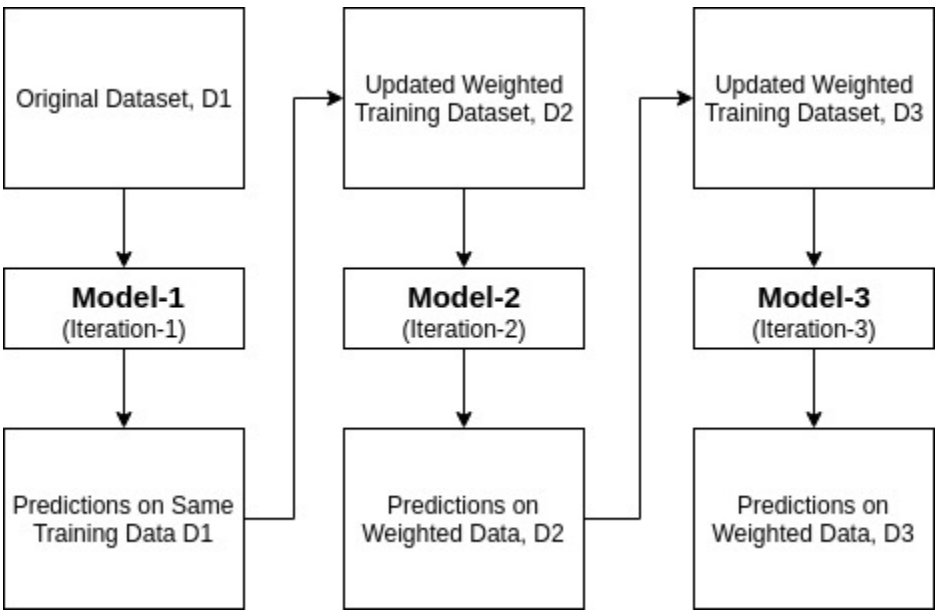
It iteratively trains the AdaBoost machine learning model by selecting the training set based on the accurate prediction of the last training.

It assigns the higher weight to wrong classified observations so that in the next iteration these observations will get the high probability for classification.

Also, It assigns the weight to the trained classifier in each iteration according to the accuracy of the classifier. The more accurate classifier will get high weight.

This process iterate until the complete training data fits without any error or until reached to the specified maximum number of estimators.

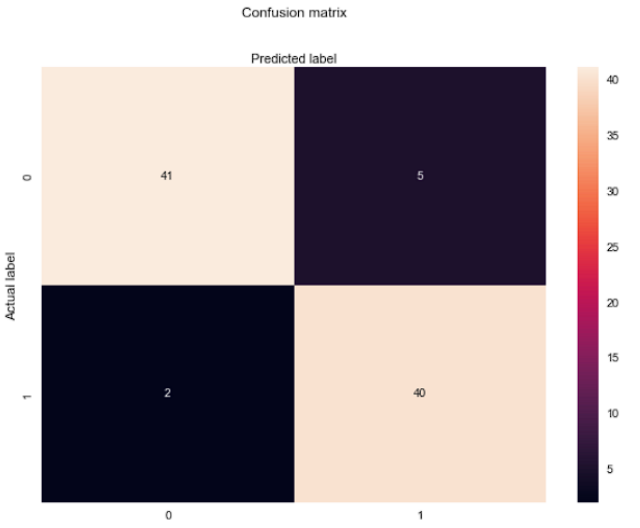
To classify, perform a "vote" across all of the learning algorithms you built.



For Under-Sampling:

```
abc = AdaBoostClassifier(n_estimators = 50, learning_rate = 1, random_state = 0)

# fit the model to the training set
abc.fit(X_train, y_train)

y_pred_abc = abc.predict(X_test)
print("AdaBoost Classifier Model Accuracy:", accuracy_score(y_test, y_pred_abc))

report = classification_report(y_test,y_pred_abc)
print("-"*100)
print(report)

cm = confusion_matrix(y_test,abc.predict(X_test))
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

sns.heatmap(cm,annot=True,fmt="d")
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.savefig('cmadabc.png')
```

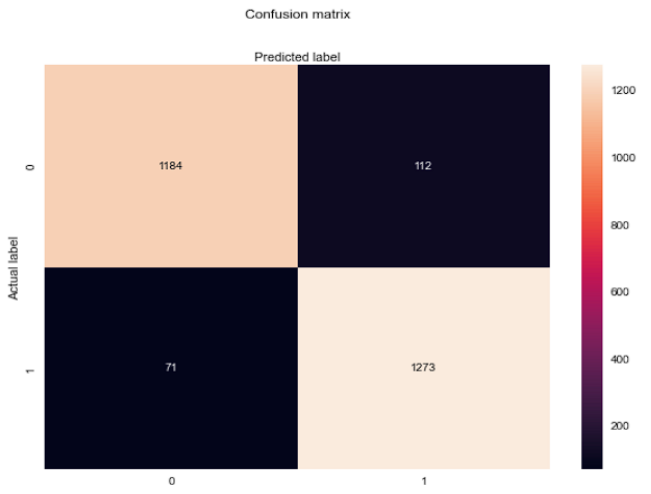
```
AdaBoost Classifier Model Accuracy: 0.9204545454545454
----------------------------------------------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      0.89      0.92        46
           1       0.89      0.95      0.92        42

    accuracy                           0.92        88
   macro avg       0.92      0.92      0.92        88
weighted avg       0.92      0.92      0.92        88
```



For SMOTE:

```
abc = AdaBoostClassifier(n_estimators = 50,
                         learning_rate = 1,
                         random_state = 0)
# fit the model to the training set
abc.fit(X_train, y_train)

y_pred_abc_sm = abc.predict(X_test)
print("AdaBoost Classifier Model Accuracy:", accuracy_score(y_test, y_pred_abc_sm))

report = classification_report(y_test, y_pred_abc_sm)
print("-"*100)
print(report)

cm = confusion_matrix(y_test,abc.predict(X_test))
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

sns.heatmap(cm,annot=True,fmt="d")
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.savefig('cm_smadabc.png')
```

```
AdaBoost Classifier Model Accuracy: 0.9306818181818182
----------------------------------------------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.94      0.91      0.93      1296
           1       0.92      0.95      0.93      1344

    accuracy                           0.93      2640
   macro avg       0.93      0.93      0.93      2640
weighted avg       0.93      0.93      0.93      2640
```



6.7. Gradient Boosting Classifier:

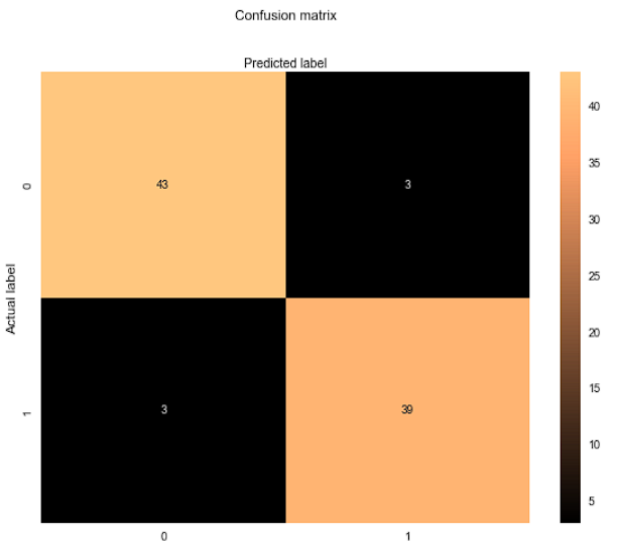
Gradient boosting classifiers are specific types of algorithms that are used for classification tasks, as the name suggests.

Features are the inputs that are given to the machine learning algorithm, the inputs that will be used to calculate an output value. In a mathematical sense, the features of the dataset are the variables used to solve the equation. The other part of the equation is the *label* or target, which are the classes the instances will be categorized into. Because the labels contain the target values for the machine learning classifier, when training a classifier you should split up the data into training and testing sets. The training set will have targets/labels, while the testing set won't contain these values. Gradient boosting classifiers are the AdaBoosting method combined with weighted minimization, after which the classifiers and weighted inputs are recalculated. The objective of Gradient Boosting classifiers is to minimize the loss, or the difference between the actual class value of the training example and the predicted class value.

For Under-Sampling:

```
gb = GradientBoostingClassifier()

# fit the model to the training set
gb.fit(X_train, y_train)

gb_pred = gb.predict(X_test)
print("Gradient Boosting Classifier:", accuracy_score(y_test, gb_pred))

report = classification_report(y_test, gb_pred)
print("-"*100)
print(report)

cm = confusion_matrix(y_test,gb.predict(X_test))
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

sns.heatmap(cm,annot=True,fmt="d", cmap = 'copper')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.savefig('cmgbc.png')
```

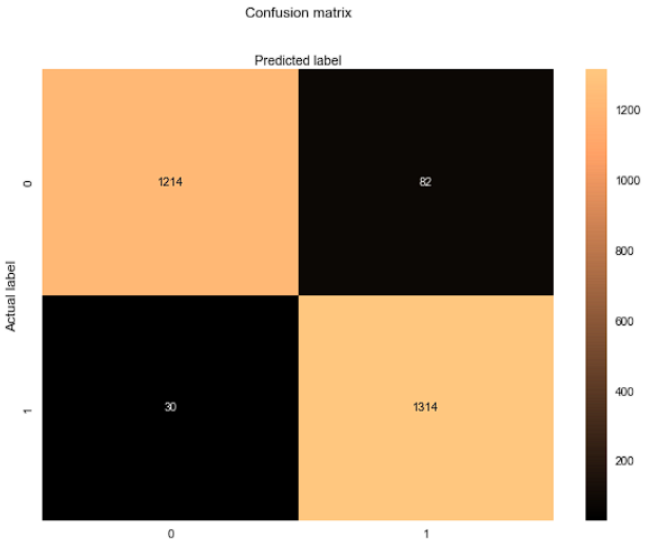
```
Gradient Boosting Classifier: 0.9318181818181818
----------------------------------------------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        46
           1       0.93      0.93      0.93        42

    accuracy                           0.93        88
   macro avg       0.93      0.93      0.93        88
weighted avg       0.93      0.93      0.93        88
```



For SMOTE:

```
gb = GradientBoostingClassifier()

# fit the model to the training set
gb.fit(X_train, y_train)

gb_sm_pred = gb.predict(X_test)
print("Gradient Boosting Classifier Accuracy is: ", accuracy_score(y_test, gb_sm_pred))

report = classification_report(y_test, gb_sm_pred)
print("-"*100)
print(report)

cm = confusion_matrix(y_test,gb.predict(X_test))
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

sns.heatmap(cm,annot=True,fmt="d", cmap = 'copper')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.savefig('cm_smgbc.png')
```

```
Gradient Boosting Classifier Accuracy is:  0.9575757575757575
----------------------------------------------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.98      0.94      0.96      1296
           1       0.94      0.98      0.96      1344

    accuracy                           0.96      2640
   macro avg       0.96      0.96      0.96      2640
weighted avg       0.96      0.96      0.96      2640
```



6.8. Voting Classifier:

A Voting Classifier is a machine learning model that trains on an ensemble of numerous models and predicts an output (class) based on their highest probability of chosen class as the output.

It simply aggregates the findings of each classifier passed into Voting Classifier and predicts the output class based on the highest majority of voting. The idea is instead of creating separate dedicated models and finding the accuracy for each them, we create a single model which trains by these models and predicts output based on their combined majority of voting for each output class. Voting Classifier supports two types of votings.

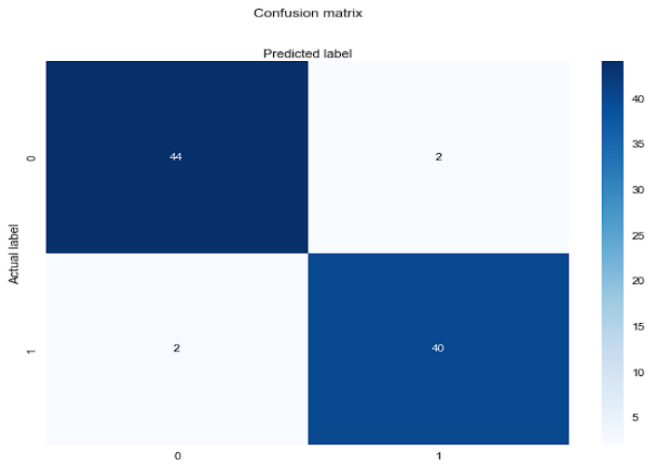
Hard Voting: In hard voting, the predicted output class is a class with the highest majority of votes i.e. the class which had the highest probability of being predicted by each of the classifiers. Suppose three classifiers predicted the output class(A, A, B), so here the majority predicted A as output. Hence A will be the final prediction.

Soft Voting: In soft voting, the output class is the prediction based on the average of probability given to that class. Suppose given some input to three models, the prediction probability for class A = (0.30, 0.47, 0.53) and B = (0.20, 0.32, 0.40). So the average for class A is 0.4333 and B is 0.3067, the winner is clearly class A because it had the highest probability averaged by each classifier.

For Under-Sampling:

```
clf1 = GradientBoostingClassifier()
clf2 = LogisticRegression()
clf3 = AdaBoostClassifier()
eclf1 = VotingClassifier(estimators=[('gbc', clf1), ('lr', clf2), ('abc', clf3)], voting='soft')

eclf1.fit(X_train, y_train)
predictions = eclf1.predict(X_test)

print("Voting Accuracy Score is:", accuracy_score(y_test, predictions))
report = classification_report(y_test, predictions)
print("-"*100)
print(report)

cm = confusion_matrix(y_test,eclf1.predict(X_test))
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

sns.heatmap(cm,annot=True,fmt="d", cmap = 'Blues')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.savefig('cmvc.png')
```

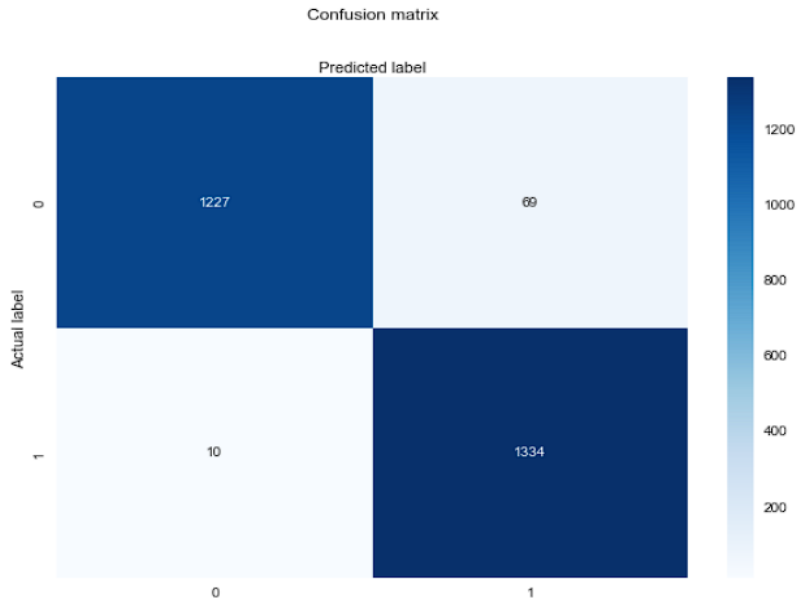
```
Voting Accuracy Score is: 0.9545454545454546
----------------------------------------------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        46
           1       0.95      0.95      0.95        42

    accuracy                           0.95        88
   macro avg       0.95      0.95      0.95        88
weighted avg       0.95      0.95      0.95        88
```



For SMOTE:

```
clf1 = RandomForestClassifier()
clf2 = LogisticRegression()
clf3 = DecisionTreeClassifier()
clf4 = AdaBoostClassifier()
clf5 = GradientBoostingClassifier()
eclf2 = VotingClassifier(estimators=[('rf', clf1), ('lr', clf2), ('dtc', clf3), ('abc', clf4), ('gbc', clf5)], voting='soft')
eclf2.fit(X_train, y_train)
predictions = eclf2.predict(X_test)
print("Voting Accuracy Score is:", accuracy_score(y_test, predictions))
report = classification_report(y_test, predictions)
print("-"*100)
print(report)

cm = confusion_matrix(y_test,eclf2.predict(X_test))
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

sns.heatmap(cm,annot=True,fmt="d", cmap = 'Blues')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.savefig('cm_smvc.png')
```

```
Voting Accuracy Score is: 0.9700757575757576
--------------------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.99      0.95      0.97      1296
           1       0.95      0.99      0.97      1344

    accuracy                           0.97      2640
   macro avg       0.97      0.97      0.97      2640
weighted avg       0.97      0.97      0.97      2640
```



6.9. Sigmoid Calibration:

In this technique we use a slight variation of sigmoid function to fit out distribution of predicted probabilities to the distribution of probability observed in training data. We actually perform logistic regression on the output of the model with respect to the actual label. The mathematical function used is shown below:

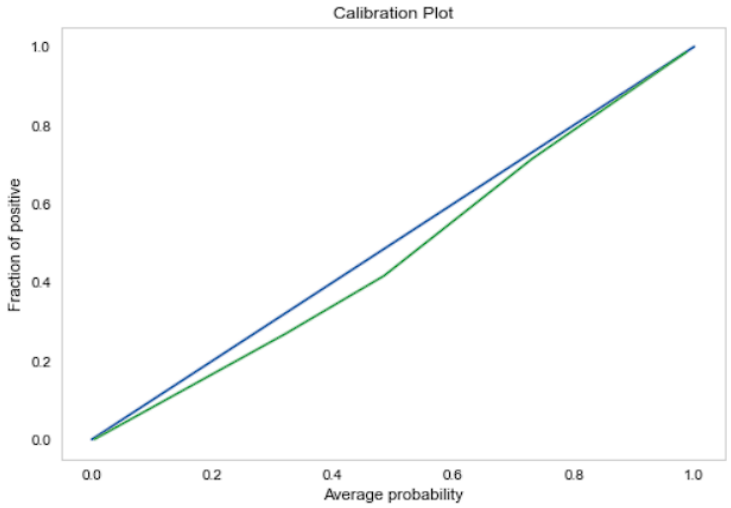
$$\mathrm{P}(y=1|x) = \frac{1}{1+\exp(Af(x)+B)},$$

Platt Scaling Formula

i.e., a logistic transformation of the classifier scores ***f*(*x*)**, where ***A*** and ***B*** are two scalar parameters that are learned by the algorithm.

The parameters ***A*** and ***B*** are estimated using a **maximum likelihood** method that optimizes on the same training set as that for the original classifier *f*.

```
rfc = RandomForestClassifier()
calibrated_sigmoid_model = CalibratedClassifierCV(rfc, method = 'sigmoid', cv = 5)
calibrated_sigmoid_model.fit(X_train, y_train)

prob = calibrated_sigmoid_model.predict_proba(X_test)[:,1]
fop, apv = calibration_curve(y_test, prob, n_bins = 5, normalize = True)

plt.plot([0,1],[0,1])
plt.plot(apv,fop)
plt.grid()
plt.xlabel("Average probability")
plt.ylabel("Fraction of positive")
plt.title("Calibration Plot")
plt.show()
psig = calibrated_sigmoid_model.predict(X_test)
print("calibrated sigmoid model Accuracy Score is: ", accuracy_score(psig, y_test))
```



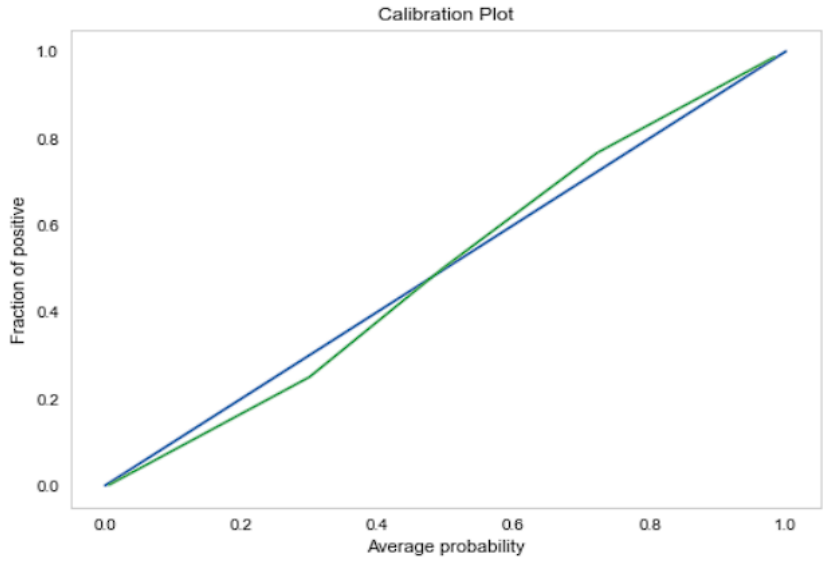

```
calibrated sigmoid model Accuracy Score is:  0.9840909090909091
```

6.10. Isotonic Calibration:

In general, isotonic regression fits a non-decreasing line to a sequence of points in such a way as to make the line as close to the original points as possible. When applied to the problem of calibration, the technique aims to perform the regression on the original calibration curve. The main advantage of using this method over Platt scaling is the fact that it doesn't require the curve to be S-shaped. Its downside, however, is its sensitivity to outliers (it can overfit quite easily) and thus works bests for large datasets - it's often recommended only if the calibration set has more than a thousand examples.

Let's look at how the isotonic regression model fits to the calibration curve for the analysed random forest model:

```
calibrated_isotonic_model = CalibratedClassifierCV(rfc, method = 'isotonic', cv = 5)
calibrated_isotonic_model.fit(X_train, y_train)

prob = calibrated_isotonic_model.predict_proba(X_test)[:,1]
fop, apv = calibration_curve(y_test, prob, n_bins = 5, normalize = True)

plt.plot([0,1],[0,1])
plt.plot(apv,fop)
plt.grid()
plt.xlabel("Average probability")
plt.ylabel("Fraction of positive")
plt.title("Calibration Plot")
plt.show()
piso = calibrated_isotonic_model.predict(X_test)
print("calibrated isotonic model Accuracy Score is: ",accuracy_score(piso,y_test))
```




```
calibrated isotonic model Accuracy Score is:  0.9852272727272727
```

7. Results

Accuracy Score:

Machine learning model accuracy is the measurement used to determine which model is best at identifying relationships and patterns between variables in a dataset based on the input, or training, data. The better a model can generalize to 'unseen' data, the better predictions and insights it can produce, which in turn deliver more business value.

Model Accuracy Score: 0.9852272727272727

8. FUTURE SCOPE

In this section, we intend to discuss the future work in bankruptcy prediction. So far, in our experiments, we have dealt with the synthetic features—an arithmetic combination of core econometric features. But it is also possible to gather more core features and hence synthesize more synthetic features by varying the arithmetic operations performed on these core features. Also, it is possible to synthesize more features considering the current synthetic features as base features. Doing so may result in better prediction of bankruptcy, but it has to be thoroughly validated by domain experts, as to whether such highly complex synthetic features would be meaningful, in terms of financial economics. It is also feasible to reduce the dimensionality of features. The features replicated/synthetically created in such manner might actually bear significant impact in the prediction, had the data not been so much sparse. So, the takeaway is that, if the data to be collected in the future, pertinent to bankruptcy prediction, is made sure to less sparse, it is possible to apply all the techniques mentioned in the future scope so far, and hence obtain better predictive models.

9. Conclusion

The biggest disappointment a company can experience is bankruptcy. The present study, which aims to predict business failure, is based on business data collected by the Taiwan. Economic Newspaper. A total of 6819 companies are considered for this study. From the results of this study, it can be seen that high values of debt ratio or loan dependency are indicators of poor business health and lead to business failure. At the same time, a high level of return on assets is a factor that stimulates the growth of a firm. Furthermore, through this study, we estimated parametric logistic regression model to predict the probability of a categorical dependent variable, and then non-parametric methods such as random forest, decision tree, adaptive boosting, and gradient boosting to predict the bankruptcy or not of firms in several aspects and voting classifier to be able to choose the optimal model.

The results of the analyses show that non-parametric models perform much better than parametric models in predicting whether a firm will fail or not. Two models clearly stand out for their very satisfactory performance. They are the random forest with a precision of 96%, a recall of 95% and an f1-score of 98% and the gradient boosting with an precision of 94%, a recall of 98% and an f1-score of 96%. Finally, the random forest was selected as the best model.

References

1. N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, pages 321– 357, 2002.

2. H. Han, W.-Y. Wang, and B.-H. Mao. Borderline-smote: a new over-sampling method in imbalanced data sets learning. In International Conference on Intelligent Computing, pages 878–887. Springer, 2005.

3. R. C. Prati, G. E. Batista, and M. C. Monard. Data mining with imbalanced class distributions: concepts and methods. In Indian International Conference Artificial Intelligence, pages 359–376, 2009.

4. Chawla, N., Bowyer, K., Hall, L., & Kegelmeyer, P. (2000). SMOTE: Synthetic Minority Over-sampling TEchnique. In International Conference of Knowledge Based Computer Systems, pp. 46–57. National Center for Software Technology, Mumbai, India, Allied Press.