

Bankruptcy Prediction

Assigned TA/ Dr. **SAMAR ALI**

Team members and their sections

Ahmed Rabie AbdelRasoul AbdelAzim
(Sec 2)

Ahmed Osama Fathy Mohamed Sheded
(Sec 1)

Ahmed Abdel Wahab AbdelFatah Elsayed
(Sec 2)

Ahmed Awad Salam Mohamed
(Sec 2)

Nour Adel Mohamed Ahmed
(Sec 32)

Omnia Khaled Abdelmged Mohamed
(Sec 6)

Ameera Khaled Ismaeel
(Sec 6)

Bankruptcy Prediction

Bankruptcy prediction is an important problem in finance since successful predictions would allow stakeholders to take early actions to limit their economic losses. Recently, AI models have increasingly been used in bankruptcy prediction. For any bank or financial institution, Bankruptcy prediction is of utmost importance. **The aim** is, therefore, to predict bankruptcy of financial institutions using machine learning classifiers.

- Data From Bankruptcy.csv Is Loaded
 - Manipulated the Data To handle missing Values, data in wrong format, Deal with NaN values with proper imputation techniques, Remove Duplicate records, apply feature scaling (normalization)
 - Applied Machine Learning algorithms like Logistic Regression, Support Vector Machine, Decision Tree to build the classification model to predict bankruptcy
-

Preprocessing

At The beginning we had to figure out a few things about the csv file we had at hand. To do correct preprocessing we had to first locate some of the common causes of data problems and to find each of these causes we used the following functions :

First Our Dataset Contains String of '?' **“WRONG FORMAT”**

```
1 ValueError: could not convert string to float: '?'
```

So we Solve it By Using **na_values** parameter in read_csv() function of Pandas

```
1 df = pd.read_csv("Bankruptcy.csv",na_values='?')
```

Or By Using

```
1 for i in df.index:
2     If df.loc[i,df] = '?':
3         df.drop(i)
4
```

After That Every Cell in our Dataset which Contain '?' Becomes NaN Value.

```
1 print( "NaN VALUES :", df.isnull().sum().sum())
2 NaN VALUES : 12157
```

**So We Deal With This NaN Values By Filling it With the Mean Of DataSet
(Missing Data)**

```
1 df.fillna(df.mean(), inplace = True)
```

Then We Check If Dataset Have Duplicated Values

```
1 print(df.duplicated().sum())
```

After Checking Dataset We Found There are Duplicated Values

**Then We Drop The
Duplicated Records**

```
1 df.drop_duplicates(inplace=True)
```

Feature Scaling


We applied feature Scaling to Normalize the Dataset and Speeding Up the Calculations in the Algorithms.

We Can apply Feature Scaling by 2 ways

1- Standardization

2- normalization

Normalization:

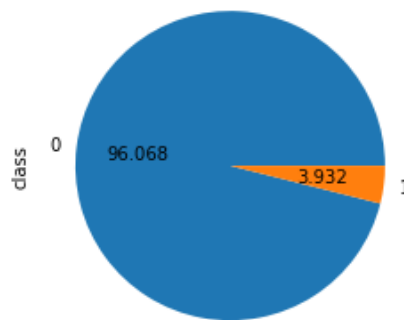


```
1 from sklearn.preprocessing import MinMaxScaler
2 scalar=MinMaxScaler()
3 X_train = scalar.fit_transform(X_train)
4 X_test = scalar.transform(X_test)
```

- First, We Import MinMaxScaler from sklearn package
- Then we fit the Training data (**Normalize The Features**)

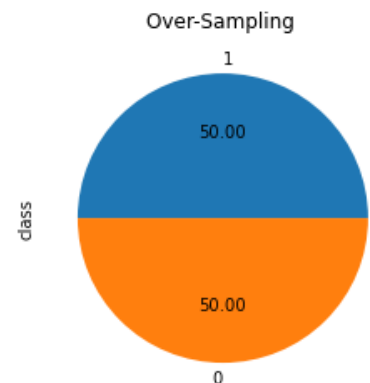
Imbalanced Dataset

We Found That Our Target Label (**Class**) In Our Dataset Is Highly Imbalanced to solve imbalanced datasets problem we applied oversampling Technique – which randomly increases examples from the minority class in the dataset



We Solve It By Using (**Random Over-Sampling Technique**)

```
1 from imblearn.over_sampling import RandomOverSampler
2 ros=RandomOverSampler()
3 X_train, y_train = ros.fit_resample(X_train, y_train)
4 X_test, y_test = ros.fit_resample(X_test, y_test)
```



- First, We Import RandomOverSampler from imblearn package
- Then we fit and resample the Training data and testing data

At this point our data was clean and ready to be fit into our three training models

Classifications

we split our data into input (X) and output (y), having input as all columns in the dataset except “Class” column and Output being The Last column “class”

Input (Features) : All dataset columns except “class” column

Output (Target) : “Class” column

```
1 #Input
2 X = df.drop('class', axis=1)
3 #Output
4 y = df['class']
```

Then We Split Input and output Into Train and Test
(70 % Train , 30% Test)

```
1 # split X and y into training and testing sets
2 X_train, X_test, y_train, y_test = train_test_split( X, y, test_size = 0.3)
```

Logistic Regression:



```
1 from sklearn.linear_model import LogisticRegression
2 Logclassifier = LogisticRegression(max_iter=3000)
3 Logclassifier.fit(X_train, y_train)
4 LogY_pred = Logclassifier.predict(X_test)
5 Logcm = confusion_matrix(y_test, LogY_pred)
6 print ("Confusion Matrix : \n", Logcm)
7 LRacc= accuracy_score(y_test, LogY_pred)
8 print ("\n Accuracy : ", LRacc)
```

First We are training our Logistic Regression model and gave it maximum number of iterations 3000.

Then we use used the Logistic Regression.fit() function and gave it both Of our training datasets : X_train and Y_train

Next we called function Logistic Regression.predict() and gave it the X_test dataset and placed the result in the Y_predict variable

And finally we called the sklearn.metrics functions called confusion_matrix() and accuracy_score() and gave them our Y_test and Y_predict



```
1 Accuracy : 0.9650409276944065
```



```
1 Confusion Matrix :
2 [[2727 205]
3 [ 0 2932]]
```


Support Vector Machine:

```
1 from sklearn import svm
2 svmClassifier = svm.SVC(kernel='linear')
3 svmClassifier.fit(X_train, y_train)
4 svmY_pred = svmClassifier.predict(X_test)
5 SVMcm = confusion_matrix(y_test, svmY_pred)
6 print ("Confusion Matrix : \n", SVMcm)
7 SVMacc=metrics.accuracy_score(y_test, svmY_pred)
8 print("\n Accuracy:",SVMacc)
```

We are training our SVM model

We called the sklearn function (svm.SVC) and decided to use the linear kernel.

Then

After training the model , Then we predict testing data

Then we calculate Confusion matrix and Accuracy

```
1 Accuracy: 0.9795361527967258
```

```
1 Confusion Matrix :
2 [[2812 120]
3 [ 0 2932]]
```

Decision Tree:

```
1 from sklearn.tree import DecisionTreeClassifier
2 Dtreeclassifier = DecisionTreeClassifier()
3 Dtreeclassifier.fit(X_train, y_train)
4 DtreeY_pred = Dtreeclassifier.predict(X_test)
5 Dtreecm = confusion_matrix(y_test, DtreeY_pred)
6 print ("Confusion Matrix : \n", Dtreecm)
7 DTacc= metrics.accuracy_score(y_test, DtreeY_pred)
8 print("\n Accuracy:",DTacc)
```

We are training our Decision Tree model

We called the sklearn function (DecisionTreeClassifier)

Then After training the model , Then we predict testing data

Then we calculate Confusion matrix and Accuracy

```
1 Accuracy: 0.9965893587994543
```

```
1 Confusion Matrix :
2 [[2932  0]
3  [ 20 2912]]
```

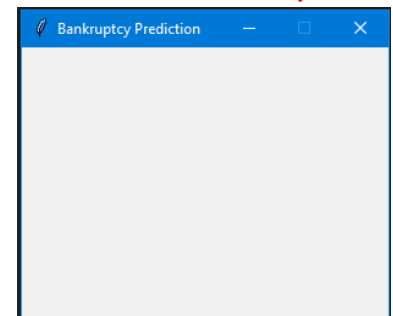
Graphic User Interface

We Use Tkinter GUI

First We Import Tkinter Libraries

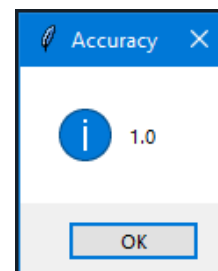
```
1 import tkinter as tk
2 from tkinter import ttk
3 from tkinter.messagebox import showinfo
```

```
1 root = tk.Tk()
2 root.geometry('300x220')
3 root.resizable(False, False)
4 root.title('Bankruptcy Prediction ')
```



We created accuracy variable of type string and make a function named show_accuracy that show accuracy of selected model

```
1 accuracy = tk.StringVar()
2
3 def show_accuracy():
4     showinfo(
5         title='Accuracy ',
6         message=accuracy.get()
7     )
```



We make a title in the Gui window (padding x-axis by 5px – y-axis by 5px)

```
1 label = ttk.Label(text="What's Classification Method ?")
2 label.pack(fill='x', padx=5, pady=5)
```

We make 3 radio buttons and we make for loop to get text and value for each radio button (padding x-axis by 10px – y-axis by 15px)

We make 3 methods of 3 training model each method have name and variable of accuracy of it

```
1 Methods = (('Logistic Regression', LRacc),
2            ('SVM', SVMacc),
3            ('Decision Tree', DTacc)
4            )
```

```
1 # radio buttons
2 for i in Methods:
3     r = ttk.Radiobutton(
4         root,
5         text=i[0],
6         value=i[1],
7         variable=accuracy )
8     r.pack(fill='x', padx=10, pady=15 )
```

We add a button when we click it it shows accuracy for classification we select (padding x-axis by 10px – y-axis by 5px)

```
1 # button
2 button = ttk.Button(
3     root,
4     text=" Show Accuracy ",
5     command=show_accuracy)
6
7 button.pack(fill='x', padx=10, pady=5)
```

