COMPANY BANKRUPTCY PREDICTION

PREDICTING WHETHER A COMPANY IS GOING TO BE BANKRUPT

- RITIK PRAKASH NAYAK

A CLASSIFICATION TASK

INDEX

A BRIEF INTRODUCTION TO THE DATA AND THE PROBLEM STATEMENT

Prediction of bankruptcy is a phenomenon of increasing interest to firms who stand to lose money because on unpaid debts. Since computers can store huge dataset pertaining to bankruptcy making accurate predictions from them before hand is becoming important.

The data were collected from the Taiwan Economic Journal for the years 1999 to 2009. Company bankruptcy was defined based on the business regulations of the Taiwan Stock Exchange.

In this project you will use various classification algorithms on bankruptcy dataset to predict bankruptcies with satisfying accuracies long before the actual event.

Attribute Information

Updated column names and description to make the data easier to understand (Y = Output feature, X = Input features)

Y - Bankrupt?: Class label 1 : Yes , 0: No

X1 - ROA(C) before interest and depreciation before interest: Return On Total Assets(C)

X2 - ROA(A) before interest and % after tax: Return On Total Assets(A)

X3 - ROA(B) before interest and depreciation after tax: Return On Total Assets(B)

X4 - Operating Gross Margin: Gross Profit/Net Sales

X5 - Realized Sales Gross Margin: Realized Gross Profit/Net Sales

X6 - Operating Profit Rate: Operating Income/Net Sales

X7 - Pre-tax net Interest Rate: Pre-Tax Income/Net Sales

X8 - After-tax net Interest Rate: Net Income/Net Sales

X9 - Non-industry income and expenditure/revenue: Net Non-operating Income Ratio

X10 - Continuous interest rate (after tax): Net Income-Exclude Disposal Gain or Loss/Net Sales

X11 - Operating Expense Rate: Operating Expenses/Net Sales

X12 - Research and development expense rate: (Research and Development Expenses)/Net Sales

X13 - Cash flow rate: Cash Flow from Operating/Current Liabilities

X14 - Interest-bearing debt interest rate: Interest-bearing Debt/Equity

X15 - Tax rate (A): Effective Tax Rate

X16 - Net Value Per Share (B): Book Value Per Share(B)

X17 - Net Value Per Share (A): Book Value Per Share(A)

X18 - Net Value Per Share (C): Book Value Per Share(C)

X19 - Persistent EPS in the Last Four Seasons: EPS-Net Income

X20 - Cash Flow Per Share

X21 - Revenue Per Share (Yuan ¥): Sales Per Share

X22 - Operating Profit Per Share (Yuan ¥): Operating Income Per Share

X23 - Per Share Net profit before tax (Yuan ¥): Pretax Income Per Share

X24 - Realized Sales Gross Profit Growth Rate

X25 - Operating Profit Growth Rate: Operating Income Growth

X26 - After-tax Net Profit Growth Rate: Net Income Growth

X27 - Regular Net Profit Growth Rate: Continuing Operating Income after Tax

X28 - Continuous Net Profit Growth Rate: Net Income-Excluding Disposal Gai

X29 - Total Asset Growth Rate: Total Asset Growth

X30 - Net Value Growth Rate: Total Equity Growth

X31 - Total Asset Return Growth Rate Ratio: Return on Total Asset Growth

X32 - Cash Reinvestment %: Cash Reinvestment Ratio

X33 - Current Ratio

X34 - Quick Ratio: Acid Test

X35 - Interest Expense Ratio: Interest Expenses/Total Revenue

X36 - Total debt/Total net worth: Total Liability/Equity Ratio

X37 - Debt ratio %: Liability/Total Assets

X38 - Net worth/Assets: Equity/Total Assets

X39 - Long-term fund suitability ratio (A): (Long-term Liability+Equity)/Fixed Assets

X40 - Borrowing dependency: Cost of Interest-bearing Debt

X41 - Contingent liabilities/Net worth: Contingent Liability/Equity

X42 - Operating profit/Paid-in capital: Operating Income/Capital

X43 - Net profit before tax/Paid-in capital: Pretax Income/Capital

X44 - Inventory and accounts receivable/Net value: (Inventory+Accounts Receivables)/Equity

X45 - Total Asset Turnover

X46 - Accounts Receivable Turnover

X47 - Average Collection Days: Days Receivable Outstanding

X48 - Inventory Turnover Rate (times)

X49 - Fixed Assets Turnover Frequency

X50 - Net Worth Turnover Rate (times): Equity Turnover

X51 - Revenue per person: Sales Per Employee

X52 - Operating profit per person: Operation Income Per Employee

X53 - Allocation rate per person: Fixed Assets Per Employee

X54 - Working Capital to Total Assets

X55 - Quick Assets/Total Assets

X56 - Current Assets/Total Assets

X57 - Cash/Total Assets

X58 - Quick Assets/Current Liability

X59 - Cash/Current Liability

X60 - Current Liability to Assets

X61 - Operating Funds to Liability

AI

X62 - Inventory/Working Capital

X63 - Inventory/Current Liability

X64 - Current Liabilities/Liability

X65 - Working Capital/Equity

X66 - Current Liabilities/Equity

X67 - Long-term Liability to Current Assets

X68 - Retained Earnings to Total Assets

X69 - Total income/Total expense

X70 - Total expense/Assets

X71 - Current Asset Turnover Rate: Current Assets to Sales

X72 - Quick Asset Turnover Rate: Quick Assets to Sales

X73 - Working capitcal Turnover Rate: Working Capital to Sales

X74 - Cash Turnover Rate: Cash to Sales

X75 - Cash Flow to Sales

X76 - Fixed Assets to Assets

X77 - Current Liability to Liability

X78 - Current Liability to Equity

X79 - Equity to Long-term Liability

X80 - Cash Flow to Total Assets

X81 - Cash Flow to Liability

X82 - CFO to Assets

X83 - Cash Flow to Equity

X84 - Current Liability to Current Assets

X85 - Liability-Assets Flag: 1 if Total Liability exceeds Total Assets, 0 otherwise

X86 - Net Income to Total Assets

X87 - Total assets to GNP price

X88 - No-credit Interval

X89 - Gross Profit to Sales

X90 - Net Income to Stockholder's Equity

X91 - Liability to Equity

X92 - Degree of Financial Leverage (DFL)

X93 - Interest Coverage Ratio (Interest expense to EBIT)

X94 - Net Income Flag: 1 if Net Income is Negative for the last two years, 0 otherwise

X95 - Equity to Liability

a. Variables with Maximum Value Greater Than 1.0

```
{' Accounts Receivable Turnover': 9740000000.0,
 ' Allocation rate per person': 9570000000.0,
 ' Average Collection Days': 9730000000.0,
 ' Cash Turnover Rate': 10000000000.0,
 ' Cash/Current Liability': 9650000000.0,
 ' Current Asset Turnover Rate': 10000000000.0,
 ' Current Ratio': 2750000000.0,
 ' Fixed Assets Turnover Frequency': 9990000000.0,
 ' Fixed Assets to Assets': 8320000000.0,
 ' Interest-bearing debt interest rate': 990000000.0,
 ' Inventory Turnover Rate (times)': 9990000000.0,
 ' Inventory/Current Liability': 9910000000.0,
 ' Long-term Liability to Current Assets': 9540000000.0,
 ' Net Value Growth Rate': 9330000000.0,
 ' Quick Asset Turnover Rate': 10000000000.0,
 ' Quick Assets/Current Liability': 8820000000.0,
 ' Quick Ratio': 9230000000.0,
 ' Research and development expense rate': 9980000000.0,
 ' Revenue Per Share (Yuan ¥)': 3020000000.0,
 ' Revenue per person': 8810000000.0,
 ' Total Asset Growth Rate': 9990000000.0,
 ' Total assets to GNP price': 9820000000.0,
 ' Total debt/Total net worth': 9940000000.0}
```

b. Normalizing the Variables

```
## A function for normalization

def normalization(df, columns):
  for col in columns:
    maximum = df[col].max()
    minimum = df[col].min()
    n = maximum - minimum
    df[col] = (maximum - df[col]) / n
  return df
```

Why normalize?

Since most of the variables are in the range of 0 and 1, it makes more sense to normalize the variables than standardize, because standardization doesn't restrict the variables in that range. So, the scales become different. We'll see which variables to normalize soon.

CLASS DISTRIBUTION IN THE TARGET VARIABLE

```
print(df['Bankrupt?'].value_counts())
print('--------------------------------------')
print(df['Bankrupt?'].value_counts(normalize = True))
```

```
0    6599
1     220
Name: Bankrupt?, dtype: int64
--------------------------------------
0    0.967737
1    0.032263
Name: Bankrupt?, dtype: float64
```

Why are classes not evenly distributed?

It is the ideal situation, as one will expect only a fraction of the companies to be bankrupt.

MACHINE LEARNING

Train - Validation Split

1. The train - validation split is in the ratio of 67:33.
2. The following is the distribution of the classes in the training set

```
## What is the distribution of the
## classes in the training set

print(y_train.value_counts())
print('--------------------------------------')
print(y_train.value_counts(normalize = True))
```

```
0    4952
1     162
Name: Bankrupt?, dtype: int64
--------------------------------------
0    0.968322
1    0.031678
Name: Bankrupt?, dtype: float64
```

Even distribution of the classes

The even distribution of the classes is one way of ensuring that the model at hand is not biased towards one class. It [ensuring even distribution] could be done in two ways.

1. Oversampling
2. Undersampling

We'll use oversampling for this project

Why not undersampling?

Because in the training set, only a meagre no. of records (169) belong to class 1. Even in the whole dataset, only 220 records pertain to this class. So even if we do undersampling on the larger dataset, we'll be left only with 220 records which will be very less considering the high dimensionality in the dataset. Now, considering that there are almost 6600 records corresponding to the negative class (0), undersampling will be equivalent to losing a large chunk of data. So we should oversample the dataset instead. For now the training dataset.

a. SMOTE on the Training and the Validation Set

1. Oversampling was done first on the training set and then on the validation set.

2. The reason for oversampling the validation set was to see whether it makes a uniformly distributed validation set makes a difference to its accuracy or not.

3. So all the models were fitted a couple of times - once on the non uniformly distributed validation set and then on the evenly distributed one.

4. The validation accuracy were different for both and have been illustrated in the next slide.

When the validation set was not oversampled

```
Training score is: 0.8894386106623586
Validation score is: 0.8692082111436951
-----------------------------------------------

Training score is: 0.9086227786752827
Validation score is: 0.8744868035190616
-----------------------------------------------

Training score is: 0.9727382875605816
Validation score is: 0.9313782991202346
-----------------------------------------------

Training score is: 1.0
Validation score is: 0.9536656891495601
-----------------------------------------------

Training score is: 0.9500201938610663
Validation score is: 0.9249266862170088
-----------------------------------------------

Training score is: 1.0
Validation score is: 0.9624633431085043
-----------------------------------------------
```

When the validation set was oversampled

```
Training score is: 0.8894386106623586
Validation score is: 0.8916211293260473
-----------------------------------------------

Training score is: 0.9086227786752827
Validation score is: 0.908621736490589
-----------------------------------------------

Training score is: 0.9727382875605816
Validation score is: 0.9043715846994536
-----------------------------------------------

Training score is: 1.0
Validation score is: 0.8682452944748027
-----------------------------------------------

Training score is: 0.9500201938610663
Validation score is: 0.8776563448694596
-----------------------------------------------

Training score is: 1.0
Validation score is: 0.899210686095932
-----------------------------------------------
```

The models are in the following order:

```
1
LogisticRegression()


2
SVC()


3
GradientBoostingClassifier()


4
RandomForestClassifier()


5
AdaBoostClassifier()


6
LGBMClassifier()
```

b. Fitting the Model and Making Predictions

Using the average of CV scores to evaluate the models

The LGBM classifier works best followed by the Gradient Boosting classifier. So, we'll use it to make predictions.

Note that the CV scores correspond to the entire dataset. That is, the entire dataset was used for cross validation. For that, SMOTE was employed on the whole dataset such that each class occurs exactly 6599 no. of times.

```
The CV scores are: [0.8594697  0.81363636 0.91098485 0.89655172 0.91739295]
The avg CV score is: 0.8796071170209101
-----------------------------------------------

The CV scores are: [0.86628788 0.84318182 0.92424242 0.91322471 0.93330807]
The avg CV score is: 0.8960489797558763
-----------------------------------------------

The CV scores are: [0.91666667 0.92386364 0.96515152 0.95035998 0.96513831]
The avg CV score is: 0.9442360225980917
-----------------------------------------------

The CV scores are: [0.95643939 0.95871212 0.9844697  0.9806745  0.98408488]
The avg CV score is: 0.9728761181347387
-----------------------------------------------

The CV scores are: [0.89583333 0.88977273 0.93787879 0.92724517 0.9480864 ]
The avg CV score is: 0.9197632826943172
-----------------------------------------------

The CV scores are: [0.95340909 0.95227273 0.99015152 0.98749526 0.98825313]
The avg CV score is: 0.9743163445749653
-----------------------------------------------
```

PRINCIPAL COMPONENT ANALYSIS

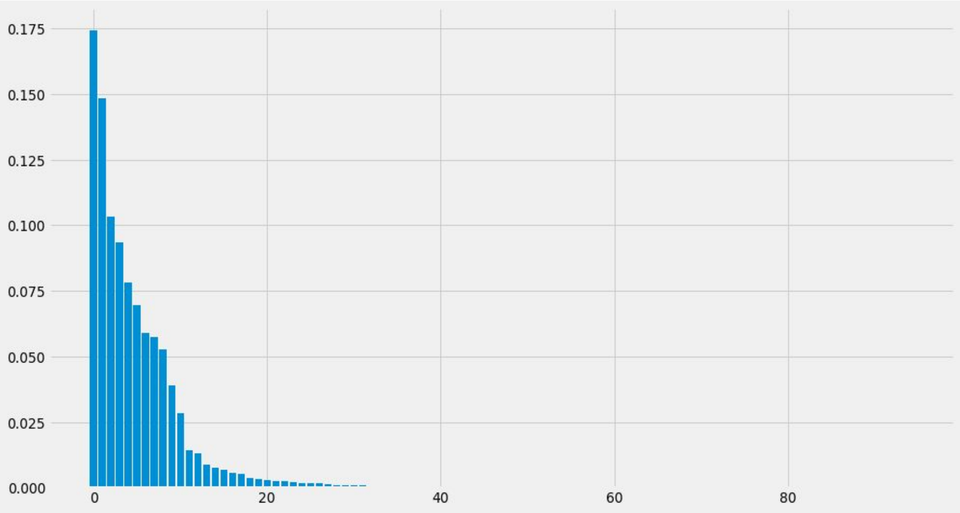
a. Finding the no. of Principal Components

A 3 step process have been used for finding the no. of PCs heuristically

1. Use n (no. of features) components initially.

2. Estimate the variance explained by each components and reject the redundant components.

3. Fit the model again



Apparently, almost 30 components were useful. The variance explained by the first two PCs were 17% and 15% respectively.

Why is such a low amount of variance explained by the first couple of PCs unlike most other situations?

It is a rare situation but it occurs sometimes when the dimensions are very large as is the case in this dataset.

Cross Validation scores on the Newly Generated Dataset

```
The CV scores are: [0.86136364 0.8125     0.90719697 0.89655172 0.91360364]
The avg CV score is: 0.8782431935880212
-------------------------------------------------

The CV scores are: [0.925      0.90416667 0.9719697  0.96816976 0.9746116 ]
The avg CV score is: 0.9487835440421646
-------------------------------------------------

The CV scores are: [0.91325758 0.89621212 0.95643939 0.95604396 0.96097006]
The avg CV score is: 0.9365846222742775
-------------------------------------------------

The CV scores are: [0.96325758 0.96325758 0.98901515 0.9867374  0.98825313]
The avg CV score is: 0.9781041659489935
-------------------------------------------------

The CV scores are: [0.85871212 0.87537879 0.91439394 0.90905646 0.91587723]
The avg CV score is: 0.8946837070975002
-------------------------------------------------

The CV scores are: [0.95909091 0.94090909 0.98181818 0.98332702 0.98863206]
The avg CV score is: 0.9707554514451067
-------------------------------------------------
```

On the new dataset - using the PCs - Random Forest apparently performs the best.

b. Hyperparameter Tuning

1. Since the Random Forest Classifier outperformed its counterparts on the newly created dataset, the hyperparameters would be tuned for this model only.
2. One hyperparameter - "n_estimators" which is the no. of decision trees being used in the random forest was tuned.
3. Sklearn's "GridSearchCV" library was used for this purpose.
4. "Accuracy" was used as a scoring metric.
5. The best value for the hyperparameter with the highest accuracy turned out to be 400.
6. It slightly increased the accuracy.

```
[64] decision_trees = [110, 150, 180, 200, 250, 300, 350, 400]

     parameters = {'n_estimators': decision_trees}

     gscv = GridSearchCV(final_model, parameters, scoring = 'accuracy', cv = 5)
     gscv.fit(X_pca, y1)

     print("The best fit values are :" ,gscv.best_params_)
     print("\nUsing ",gscv.best_params_, " accuracy is: ", gscv.best_score_)
```

```
The best fit values are : {'n_estimators': 400}

Using  {'n_estimators': 400}  accuracy is:  0.9793922743922744
```

CONCLUSION

The task of prediction was realized in two broad ways:

1. Oversampling using SMOTE, and
2. Dimensionality Reduction using Principal Component Analysis.

It should be noted that the latter didn't improve the situation much than what it was before. Even a tuned Random Forest - that performed the best on the PC dataset - didn't help improve the accuracy. So the author proposes a LightGBM on the usual data. Of course PCA could be used on the test set and if so, the Random Forest would be advisable, but why take the pain of dimensionality reduction if little or less gains are being made from that.

The author used SMOTE as it was at his discretion. While the idea of undersampling was rejected, cost sensitive learning, though in the sight of the author, was not employed for no particular reason. Moreover, had the dataset been significantly large, the author might have advised using the dataset as it is.

THANK YOU!