MF 815

Midterm Project

Haoyang Shang

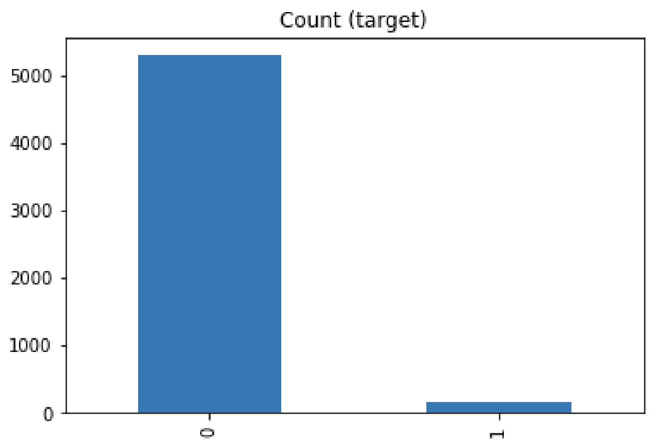
Prof. Hao Xing

Company Bankruptcy Prediction

In this project, we were investigating the best machine learning classifier to perform company bankruptcy predictions based on the provided training data, then performed predictions using the trained model on the test data. Upon receiving the assignment, we immediately wanted to see what kind of data we are dealing with here. Using commands such as describe, info, and columns, we could see that there is a response variable named “Bankrupt?”. Value 1 on this variable indicated bankruptcy and a 0 indicated otherwise. All other columns on the data frame were explanatory variables that we needed to determine a possible relationship to the response variable.

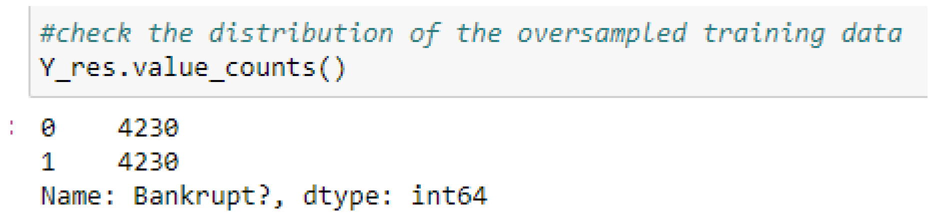
Naturally, we wanted to check the response variable first. We used a graph to visualize the distribution of bankrupt companies and surviving companies. From the graph below, we saw that we are dealing with highly imbalanced data here. This a problem we would later address by using the oversampling method on our training data set.

```
Class 0: 5285
Class 1: 170
Proportion: 31.09 : 1
```



After investigating the response variable, we wanted to research our explanatory variables. Using a correlation matrix, we found the pair of variables with high correlation with each other. A high correlation coefficient could mean that the two variables were repetitive, and we could eliminate one redundant variable. Here we choose to delete the variable pairs with a coefficient larger than 0.9.

After cleaning our data, we could split them into the training set and testing set, using a test size of 0.2. Then, we oversampled the training set using SMOTE. Here we can see the oversampled training data has 4230 entries of both classes 0 and 1.



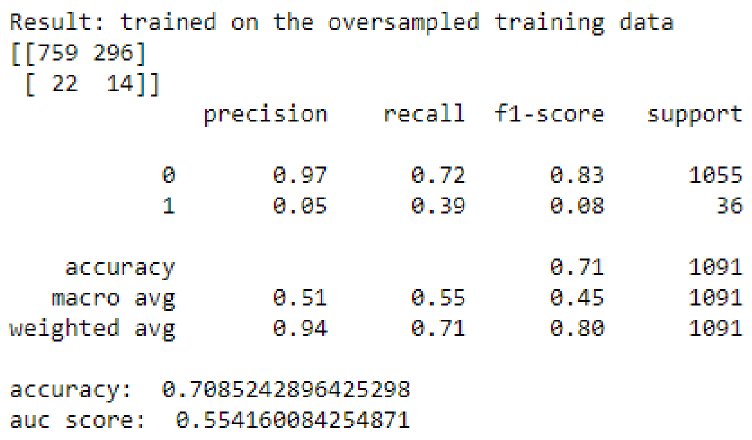
```
#check the distribution of the oversampled training data
Y_res.value_counts()
```

```
: 0    4230
  1    4230
  Name: Bankrupt?, dtype: int64
```

.

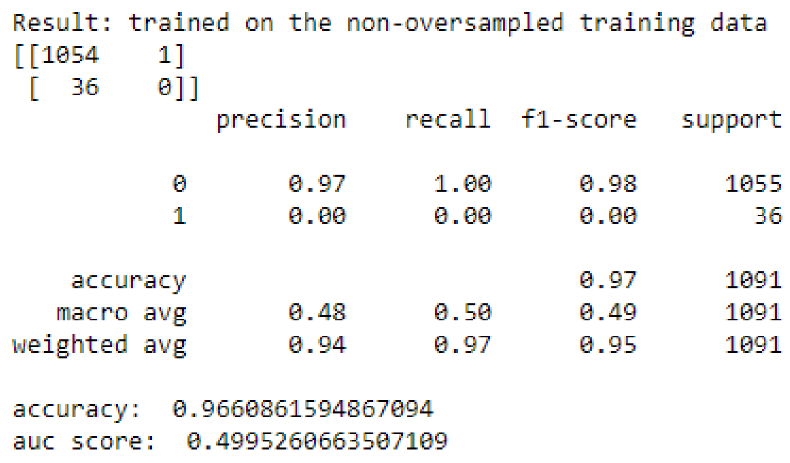
Logistic Regression:

After setting up the data, it was time to train our models. The first method we were testing is logistic regression. The simple classical method was a great starting point for the project. The graph below summarized the result of the logistic regression method, trained on the oversampled training set, and implemented on the testing set. From top to bottom are the confusion matrix, f1-score report, accuracy score, and auc score.



```
Result: trained on the oversampled training data
[[759 296]
 [ 22  14]]
              precision    recall  f1-score   support

           0       0.97      0.72      0.83      1055
           1       0.05      0.39      0.08        36

    accuracy                           0.71      1091
   macro avg       0.51      0.55      0.45      1091
weighted avg       0.94      0.71      0.80      1091

accuracy:  0.7085242896425298
auc score:  0.554160084254871
```

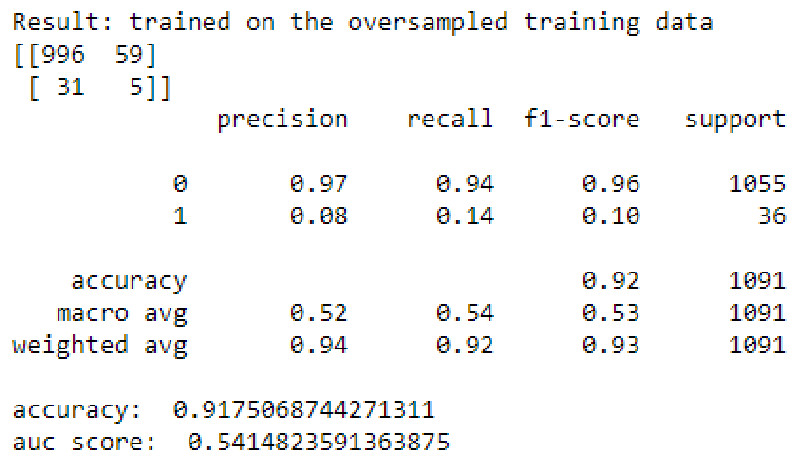
Naturally, we wanted to see if our oversampling worked. The graph below summarized the result of the logistic regression method implemented on the same testing set, while trained on a non-oversampled training set. Comparing the two reports, we saw that the model trained on non-oversampled had an f1-score of 0 on the class 1 set, and is incapable of predicting true negatives as we saw on the confusion matrix. Therefore, despite it having better accuracy, it had a lower auc score. The comparison proved the effectiveness of oversampling the imbalanced training data set.



```
Result: trained on the non-oversampled training data
[[1054    1]
 [  36    0]]
              precision    recall  f1-score   support

           0       0.97      1.00      0.98      1055
           1       0.00      0.00      0.00        36

    accuracy                           0.97      1091
   macro avg       0.48      0.50      0.49      1091
weighted avg       0.94      0.97      0.95      1091

accuracy:  0.9660861594867094
auc score:  0.4995260663507109
```

KNN Classifier:

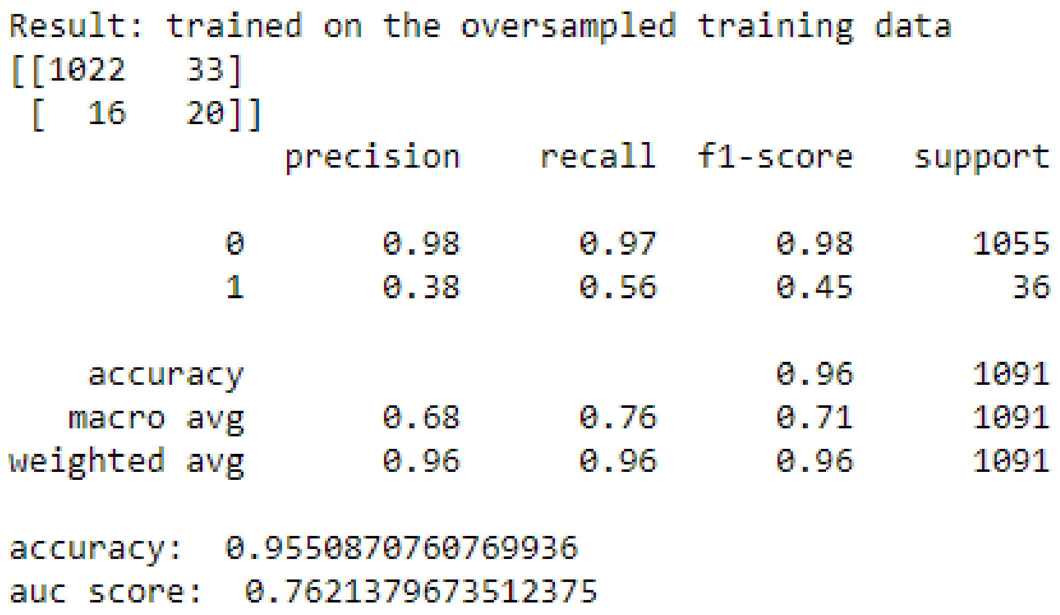
The second method we were testing is the KNN classifier. First, we scale the oversampled training data, as the KNN classifier requires. Then we ran a loop to find the optimal n-neighbors number that has the best auc score for the model. Finally, we generated the report of the KNN classifier using the best n-number we found. In the graph below, we saw that KNN Classifier had a similar performance as the logistic regression method.



```
Result: trained on the oversampled training data
[[996  59]
 [ 31   5]]
              precision    recall  f1-score   support

           0       0.97      0.94      0.96      1055
           1       0.08      0.14      0.10        36

    accuracy                           0.92      1091
   macro avg       0.52      0.54      0.53      1091
weighted avg       0.94      0.92      0.93      1091

accuracy:  0.9175068744271311
auc score:  0.5414823591363875
```

Random Forest:

The third model we were testing here is the random forest. First, we used GridSearchCV to find the best model parameter, then we generate the report using the arbitrary parameter. Here we saw that the Random Forest Classifier works best, with an f1-score of 0.98 and 0.45 for the two classes respectively, and an auc score of 0.76.



```
Result: trained on the oversampled training data
[[1022   33]
 [  16   20]]
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      1055
           1       0.38      0.56      0.45        36

    accuracy                           0.96      1091
   macro avg       0.68      0.76      0.71      1091
weighted avg       0.96      0.96      0.96      1091

accuracy:  0.9550870760769936
auc score:  0.7621379673512375
```

In summary, Random Forest appears to be the best classifier, as it had the best f1-score on both classes, the best accuracy score, and the best auc score. The confusion matrix also showed that it has the most success in predicting true negatives, which could be a challenge since we were given highly imbalanced data to begin with.

Reference

1. Code from “imbalanced data” In [219]
2. Code from “decision tree” In [250]