ASSIGNMENT 02 – CREDIT RISK PREDICTION

Username on miner: shirdisai

Name: Varshaa Shree Bhuvanendar

Rank: 15(At the time of making report)

FI-Score: 0.68

AIM: To perform credit risk prediction using different prediction algorithms

Introduction:

I have used Gradient Boosting algorithm to perform credit risk prediction as the data set has imbalanced data.

Steps Followed:

1. Performed one hot encoding on data
2. covariance of every column with the destination column "credit"
3. Columns with values close to 0 covariance I dropped
4. Since data is imbalanced performed SMOTE algorithm over the dataset
5. I then tried Random Forest, Linear Support Vector Machine, Perceptron and Gradient Boosting
6. Found F1-Score

How did you deal with different features?

Since the dataset is a mix of textual and number-based data. To handle textual data, I have tried both label encoding and one hot encoding.

I found one hot encoding gives better results.

Label encoding: If there are n unique groups in a column of dataset label encoding, assigns values from 0 to n-1 based on alphabetical order. By doing this we are indirectly assigning rank to certain groups.

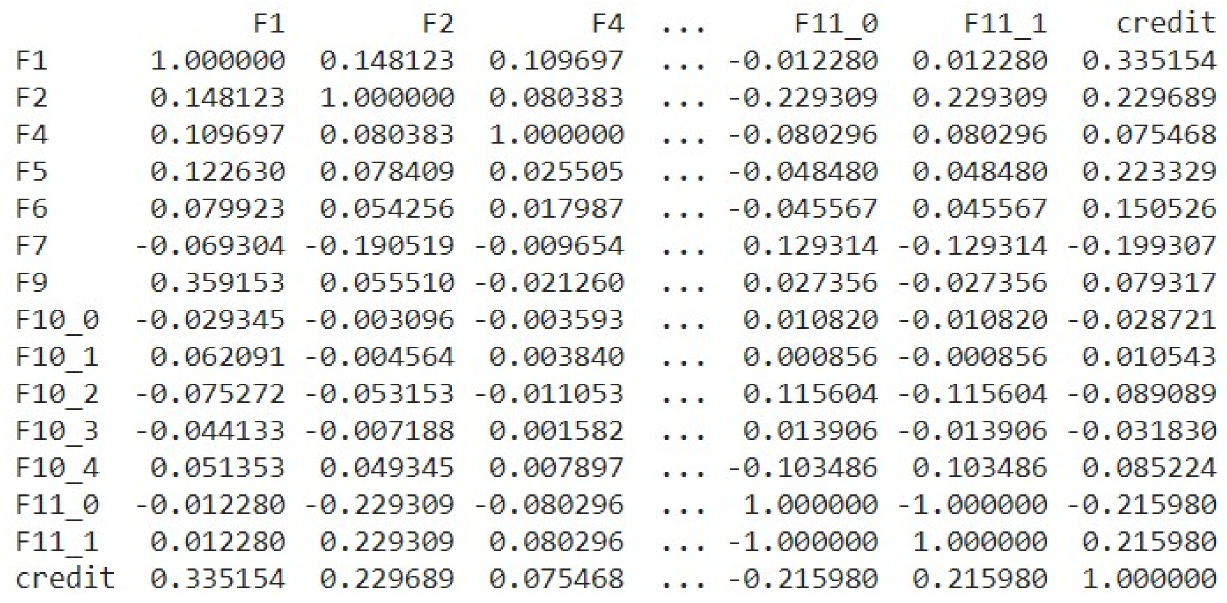
To avoid this, we use one hot encoding. It simply creates additional features based on the number of unique values in the categorical feature. Each category is represented as a one-hot vector.

In the current dataset I applied it to columns F10 and F11.

Did you exclude any specific features?

To find out which features I can remove I found correlation between credit determinant column and all other columns.

Anything with variance close to 0 like id,F8 and F9 are removed



```
              F1        F2        F4  ...     F11_0     F11_1    credit
F1      1.000000  0.148123  0.109697  ... -0.012280  0.012280  0.335154
F2      0.148123  1.000000  0.080383  ... -0.229309  0.229309  0.229689
F4      0.109697  0.080383  1.000000  ... -0.080296  0.080296  0.075468
F5      0.122630  0.078409  0.025505  ... -0.048480  0.048480  0.223329
F6      0.079923  0.054256  0.017987  ... -0.045567  0.045567  0.150526
F7     -0.069304 -0.190519 -0.009654  ...  0.129314 -0.129314 -0.199307
F9      0.359153  0.055510 -0.021260  ...  0.027356 -0.027356  0.079317
F10_0  -0.029345 -0.003096 -0.003593  ...  0.010820 -0.010820 -0.028721
F10_1   0.062091 -0.004564  0.003840  ...  0.000856 -0.000856  0.010543
F10_2  -0.075272 -0.053153 -0.011053  ...  0.115604 -0.115604 -0.089089
F10_3  -0.044133 -0.007188  0.001582  ...  0.013906 -0.013906 -0.031830
F10_4   0.051353  0.049345  0.007897  ... -0.103486  0.103486  0.085224
F11_0  -0.012280 -0.229309 -0.080296  ...  1.000000 -1.000000 -0.215980
F11_1   0.012280  0.229309  0.080296  ... -1.000000  1.000000  0.215980
credit  0.335154  0.229689  0.075468  ... -0.215980  0.215980  1.000000
```

How did you perform model selection and which classifier stood out? Any theoretical reasoning why?

I implemented KNN, Random Forest, Perceptron, Naïve Bayes, Gradient Boosting.

Gradient Boosting stood out.

Reason:

Gradient Boosting works on the wrong prediction. So, if we have large set of biased data. So, in first iteration when the algorithm predicts some of the rows wrong then in next iteration tries to find why it predicted wrong and works on the wrong prediction.

Was there a certain way you dealt with imbalance in the class distributions?

Tried using oversampling.

Oversampling can be defined as adding more copies to the minority class. Oversampling can be a good choice when you don’t have a ton of data to work with.

I tried the SMOTE Synthetic Minority Oversampling Technique.

SMOTE (Synthetic Minority Oversampling Technique) works by randomly picking a point from the minority class and computing the k-nearest neighbors for this point. The synthetic points are added between the chosen point and its neighbors.

`Used following library imblearn.over_sampling`

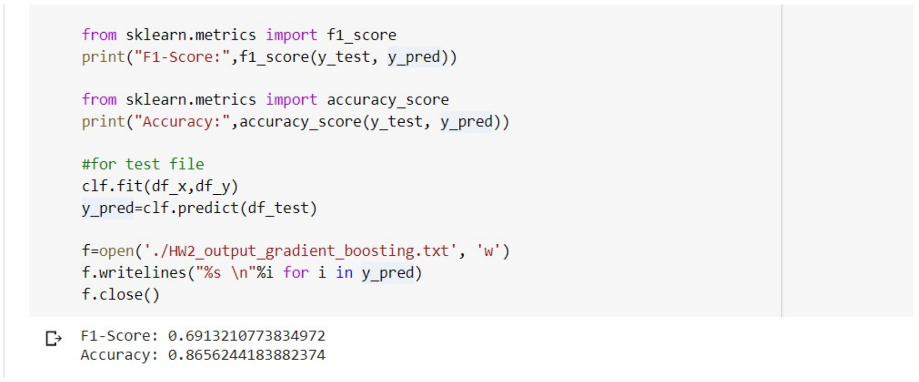
Conclusion:

Gradient Boosting technique can be used as a prediction model for an imbalanced dataset.

For feature selection we use correlation and determine which features to remove.

Also oversampling and undersampling can be applied on the dataset to avoid biasing.

OUTPUT (f1 score):



```
from sklearn.metrics import f1_score
print("F1-Score:",f1_score(y_test, y_pred))

from sklearn.metrics import accuracy_score
print("Accuracy:",accuracy_score(y_test, y_pred))

#for test file
clf.fit(df_x,df_y)
y_pred=clf.predict(df_test)

f=open('./HW2_output_gradient_boosting.txt', 'w')
f.writelines("%s \n"%i for i in y_pred)
f.close()
```

```
F1-Score: 0.6913210773834972
Accuracy: 0.8656244183882374
```

Output Explanation:

- By running the first cell it uses gradient boosting and one hot encoding output from first cell is final output.
- By running second cell the label encoder based output with gradient boosting is obtained.
- Third cell shows correlation over the dataset
- The further cells show accuracy and model fitting on the train_data for SVM,Perceptron, Random Forest, KNN, Logistic Regression and more.