HOMEWORK 2

NAME : ISHANA SHINDE
[redacted]
ACCURACY SCORE: 0.68(with One hot encoding)/0.69(with label encoder)
RANK: 11(At time of report)

Introduction:
The goal of this assignment is to determine the credit risk for a particular individual by applying different classification models on the data and selecting the one with the best f1_score.

Approach:

For this assignment as well I divided it into 2 parts -
1. Data Preprocessing
2. Model Selection

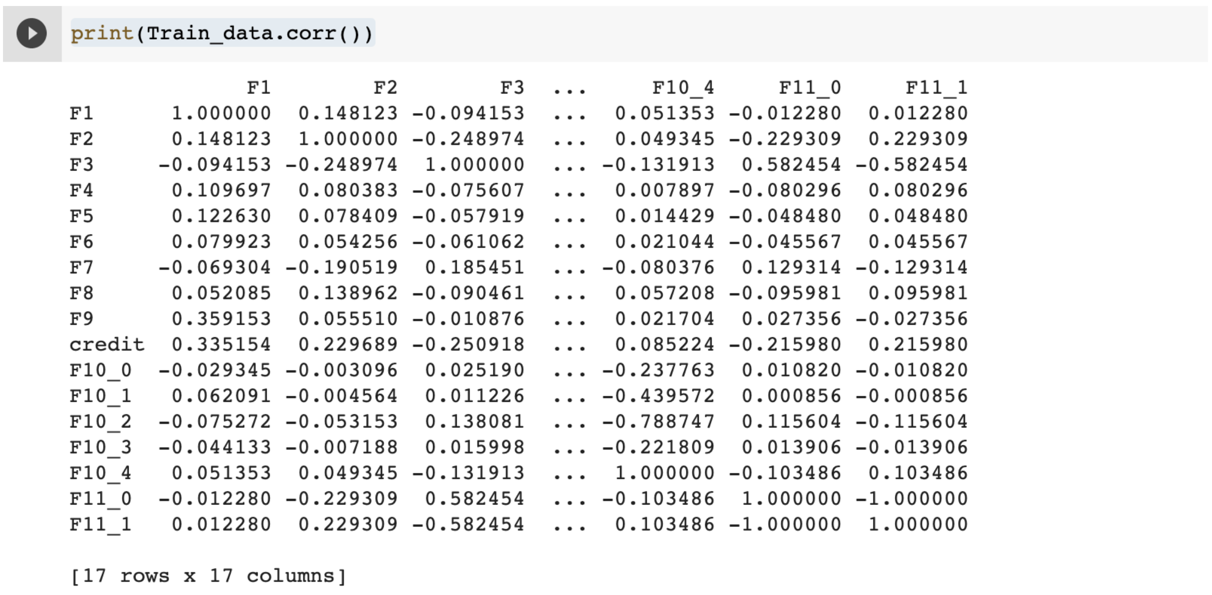
Data Preprocessing:

In the Data Preprocessing stage, First drop all the rows that contain missing values. The training data file consists of 13 columns that contain the attributes related to credit fraud prediction. Out of the 13 columns, two columns are of Nominal Categorical type - F10 (Categorical value denoting different race)and F11 (Categorical value denoting gender). So we need to convert this data to numeric form and we do that by using encoders.

How did you deal with different features?
There are 2 types of encoders that I used for this assignment. I first started with LabelEncoder from the sklearn.preprocessing library. LabelEncoder encodes labels with a value between 0 and n_classes-1 where n is the number of distinct labels present in that column. If a label repeats it assigns the same value to that as assigned earlier. The problem with this encoder is that since there are different numbers in the same column, the model will misunderstand the data to be in some kind of order like 4>3>1.
Because of this, I switched to One Hot Encoding where it resolves the issue created by the label encoder. This encoder takes a column which has categorical data that has been label encoded and then splits the columns into multiple columns. The numbers are replaced by 1’s and 0’s depending on which column has what value.

Did you exclude any features?

I used the pandas corr() function on the Train Dataframe to find out the correlation between the different columns in the dataframe. This corr() function by default returns the Pearson correlation and it returns a dataframe with pairwise correlation of columns excluding null values.



```
print(Train_data.corr())
```

```
              F1        F2        F3  ...     F10_4     F11_0     F11_1
F1      1.000000  0.148123 -0.094153  ...  0.051353 -0.012280  0.012280
F2      0.148123  1.000000 -0.248974  ...  0.049345 -0.229309  0.229309
F3     -0.094153 -0.248974  1.000000  ... -0.131913  0.582454 -0.582454
F4      0.109697  0.080383 -0.075607  ...  0.007897 -0.080296  0.080296
F5      0.122630  0.078409 -0.057919  ...  0.014429 -0.048480  0.048480
F6      0.079923  0.054256 -0.061062  ...  0.021044 -0.045567  0.045567
F7     -0.069304 -0.190519  0.185451  ... -0.080376  0.129314 -0.129314
F8      0.052085  0.138962 -0.090461  ...  0.057208 -0.095981  0.095981
F9      0.359153  0.055510 -0.010876  ...  0.021704  0.027356 -0.027356
credit  0.335154  0.229689 -0.250918  ...  0.085224 -0.215980  0.215980
F10_0  -0.029345 -0.003096  0.025190  ... -0.237763  0.010820 -0.010820
F10_1   0.062091 -0.004564  0.011226  ... -0.439572  0.000856 -0.000856
F10_2  -0.075272 -0.053153  0.138081  ... -0.788747  0.115604 -0.115604
F10_3  -0.044133 -0.007188  0.015998  ... -0.221809  0.013906 -0.013906
F10_4   0.051353  0.049345 -0.131913  ...  1.000000 -0.103486  0.103486
F11_0  -0.012280 -0.229309  0.582454  ... -0.103486  1.000000 -1.000000
F11_1   0.012280  0.229309 -0.582454  ...  0.103486 -1.000000  1.000000

[17 rows x 17 columns]
```

Depending on the value of Pearson correlation, I have done feature selection by dropping "id, F8, F9" columns that have very less correlation value with the credit column.

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

As the training data provided to us is imbalanced with a large number of 0's than 1's. Some ways to deal with imbalanced data is to Oversample or Undersample your data. Oversampling is a technique used to adjust the class distribution to adjust the class distribution of a data set. There are different types of oversampling - Random, SOMTE : Synthetic Minority Oversampling Technique, ADASYN. I have tried SMOTE and ADASYN. SMOTE works by picking a point from the minority class and then computes the K-nearest point for the selected point. The synthetic points are then added between the chosen point and its neighbors.

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

For Model Selection I tried a huge set set of classifiers like: Decision Tree, Random Forest, SVM, KNN, Logistic Regression, Gradient Boost, Perceptron, Neural Networks, XGBoost.

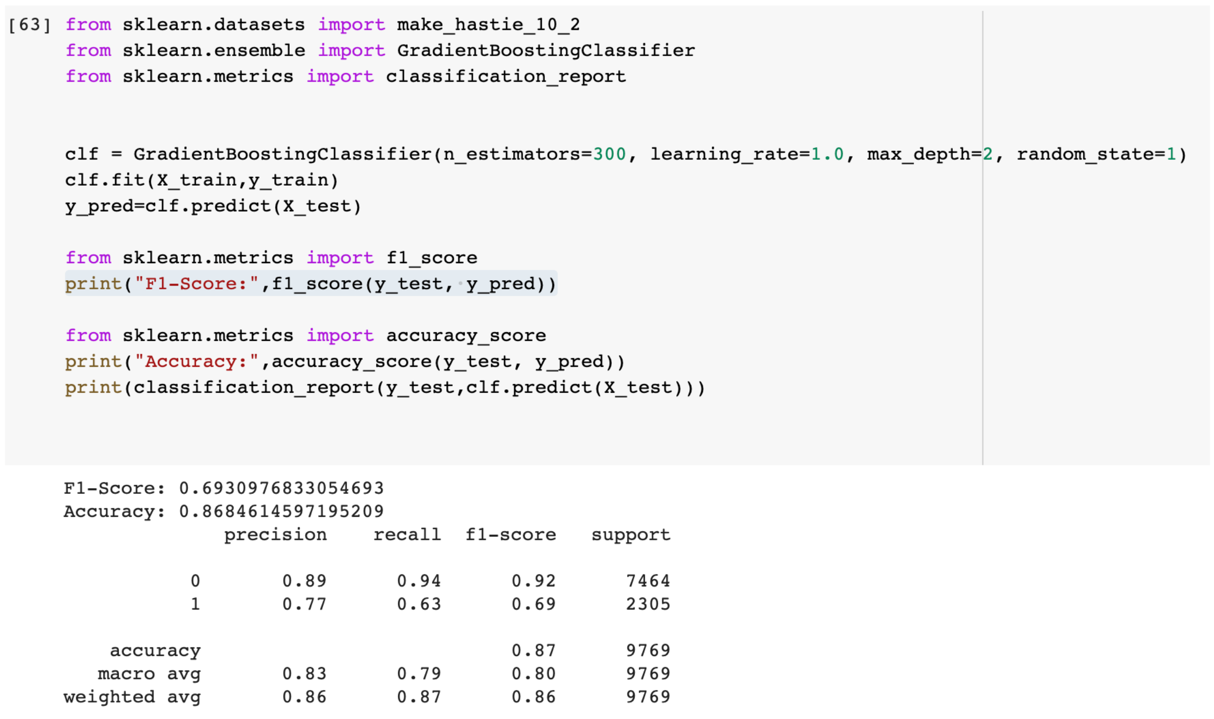
Of all these models the best f1-score was by using ***Gradient Boost***

Classifier. Gradient boosting is a type of Ensemble learning method, more specifically a boosting technique that works by combining several weak learners(features with less accuracy) into strong learners. Here each predictor tries to improve on its predecessor by reducing the errors. If we get a large biased data it will predict some rows as wrong in the first iteration and then from the next iteration it will work on the wrong predictions. and The learning rate hyper parameter (learning_rate=1.0) is used to scale each tree's contribution, sacrificing bias for better variance.

Conclusion:

With all the models used Gradient Boost Classifier gave a better f1-score of 0.69 than the rest. This suggests that we could use the Gradient Boost Classifier on imbalanced datasets. Also we can use different methods like oversampling on imbalanced datasets.

Output:



```
[63] from sklearn.datasets import make_hastie_10_2
     from sklearn.ensemble import GradientBoostingClassifier
     from sklearn.metrics import classification_report


     clf = GradientBoostingClassifier(n_estimators=300, learning_rate=1.0, max_depth=2, random_state=1)
     clf.fit(X_train,y_train)
     y_pred=clf.predict(X_test)

     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))
     print(classification_report(y_test,clf.predict(X_test)))
```

```
F1-Score: 0.6930976833054693
Accuracy: 0.8684614597195209
              precision    recall  f1-score   support

           0       0.89      0.94      0.92      7464
           1       0.77      0.63      0.69      2305

    accuracy                           0.87      9769
   macro avg       0.83      0.79      0.80      9769
weighted avg       0.86      0.87      0.86      9769
```