Manmeet Singh Khanna
CS 584
Homework 2

HW 2 (584-Rangwala): Credit Risk Prediction

User Name: Deviant
Accuracy Score: 0.68

Approach:

The main parts of this problem include:

1. Processing the data
2. Using different classification models to get the best results

These two processes have their own significance and are equally important as they are dependent on each other.

Understanding the Data:

The number of categories of each type is mentioned below:

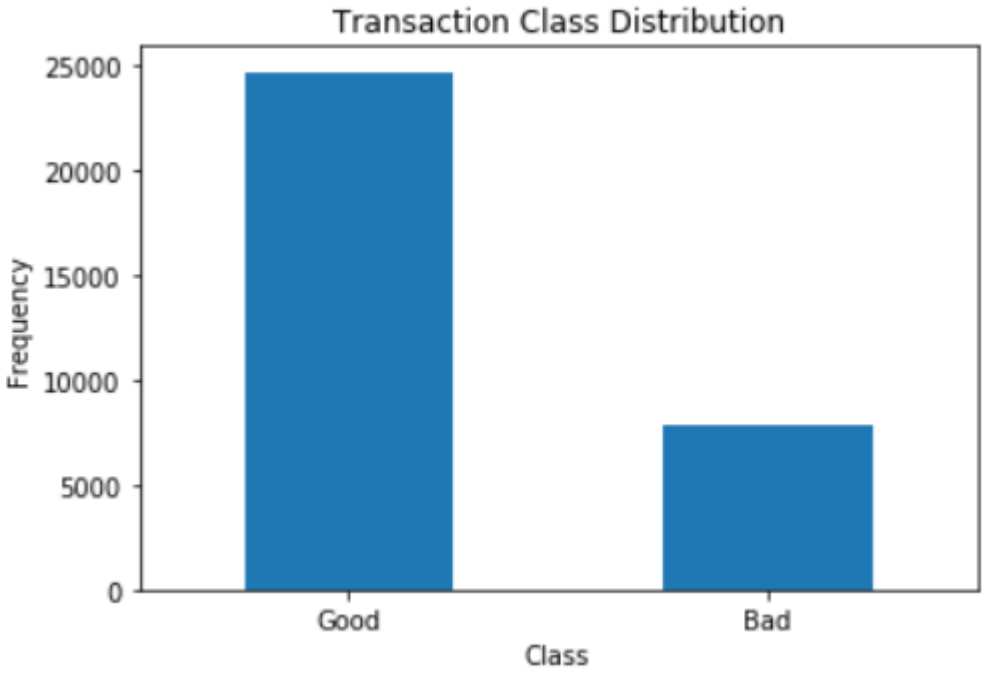
id - unique identifier
F1 - Continuous value describing the number of years since the last degree was completed
F2 - Continuous value indicating hours worked per week
F3 - Categorical Value: **5 types**
F4 - Categorical Value indicating the type of occupation: **14 types**
F5 - continuous value denoting gains: **Not that useful**
F6 - continuous value denoting loss: **Kinda decent as compared to F5**
F7 - Categorical value denoting the marital status: **7 types**
F8 - Categorical value denoting type of employment (e.g., Self) : **8 types**
F9 - Categorical Value denoting education type: **15 types**
F10 - Categorical Value denoting different races: **5 types**
F11 - Categorical - **Female/Male**
credit - 0: Bad, 1: Good

Here in this problem, the data is cleaned in a precise format in two categories: Categorical and Continuous. I dropped the ‘id’ column as that seemed of no use in the following data. The categorical values need to be encoded as they are in the form of integers. For Eg: There are the following types in the F10 i.e. different types of Races in the data.

```
df['Race'].value_counts() #number and types of races
```

```
 White                 27816
 Black                  3124
 Asian-Pac-Islander     1039
 Amer-Indian-Eskimo      311
 Other                   271
Name: Race, dtype: int64
```

The data contains the final binary classification labels in the credit section as credit - 0: Bad, 1: Good



We can clearly see that the data is imbalanced. There are 24,720 numbers of ‘0’ and 7,841 numbers of ‘1’.

```
df['credit'].value_counts()
```

```
0    24720
1     7841
Name: credit, dtype: int64
```

Processing the Data:

In this dataset, as we saw the data and it was already clean but needed encoding.
I have made use of two techniques:

- One Hot Encoding
- Dummy Values
- Scaling the Data

I have used sklearn's OneHotEncoder() on some categorical values which are ['Categorical','Occupation','Marital','Employment','Education'] and Panda's pd.get_dummies() on ['Race','Gender'].

OneHotEncoder() gives the categories a binary representation based on the number of features. The feature is represented as 01000 if it is 1st out of 6 features. Similarly, the 0th is represented as 100000 and 6th (last) is 000001.

```
ohe = OneHotEncoder(sparse = False)
```

```
t = ohe.fit_transform(df[['Categorical']])
```

```
t
```

```
array([[0., 1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0.],
       ...,
       [0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 1.]])
```

Thus the features are transformed into such an array.

The dummy values are like OneHotEncoding but they create an independent variable of all the different types of features and represent it as 1 if present and 0 if absent.
Here, I have converted Race and Gender into their dummy values.

	F10_ Amer-Indian-Eskimo	F10_ Asian-Pac-Islander	F10_ Black	F10_ Other	F10_ White	F11_ Female	F11_ Male
0	0	0	0	0	1	0	1
1	0	0	0	0	1	0	1
2	0	0	0	0	1	0	1
3	0	0	1	0	0	0	1
4	0	0	1	0	0	1	0

Here, I have used sklearn's StandardScaler() on the continuous values which are then scaled between 0 and 1. To make this all possible, I have made use of a column transformer, which does this work altogether.

```
ct = make_column_transformer(
    (OneHotEncoder(), ['Categorical','Occupation','Marital',
                       'Employment','Education']),
    (StandardScaler(),['Years since last degree (Continuous)',
                       'Working hours per week (Continuous)',
                       'Gains (Continuous)']),
    remainder = 'passthrough')
```

I tried to balance the data using three different techniques and applied Random Forest on all types of data, these were the results using the F1 score:

1. Over Sampling - 0.879
2. SMOTETomek - 0.872
3. Under Sampling - 0.755

Data after Over Sampling:

Thus, I selected Over Sampling for this following data as it gives more accurate results when the dataset is moderate in size like this dataset i.e. thousands.

Mention:
I tried to remove some continuous and categorical features like Loss, Gain, Categorical but they decreased the final test accuracy.

Classification Models:

Classification Model	Accuracy (Miner)
Logistic Regression	**0.63**
Decision Trees	**0.62**
Random Forest	**0.65**
XG Boost	**0.68**

The train data was split using train_test_split() in a 70/30 split where test size is 30.
The final model which gave the most precise results - 0.68 accuracy on Miner - is XG Boost. However, Random Forest gave an accuracy of 0.879 in the test train split. It gave an accuracy of 0.65 in the unseen test data. Meanwhile, XG Boost gave an accuracy of 0.69 in the test train split, it gave 0.68 in the unseen test data. As XG Boost uses Gradient Boosting Framework which provides a parallel tree boosting which gives a very great result.

References:

https://scikit-learn.org/
pandas - Python Data Analysis Library (pydata.org)
Natural Language Toolkit — NLTK 3.6.2 documentation
https://xgboost.readthedocs.io/en/latest/