ML First Project Documentation

Professor: Dr. Kiani

Ilya Shabanpour Fouladi

4003613041

GitHub Rep. link:

https://github.com/ilya-shabanpour/ML_Credit_Prediction

1402 – 1403 second semester

Table of Contents

Foreword

In this Documentation I want to elaborate on the implementation of my first Machine Learning Project.

Our task is to use a Regression Model to predict the credit limit of clients based on some features that were given to us in the form of a csv dataset.

This goal is achieved through a process that will be explained in two chapters, Data Preprocessing and Model Selection.

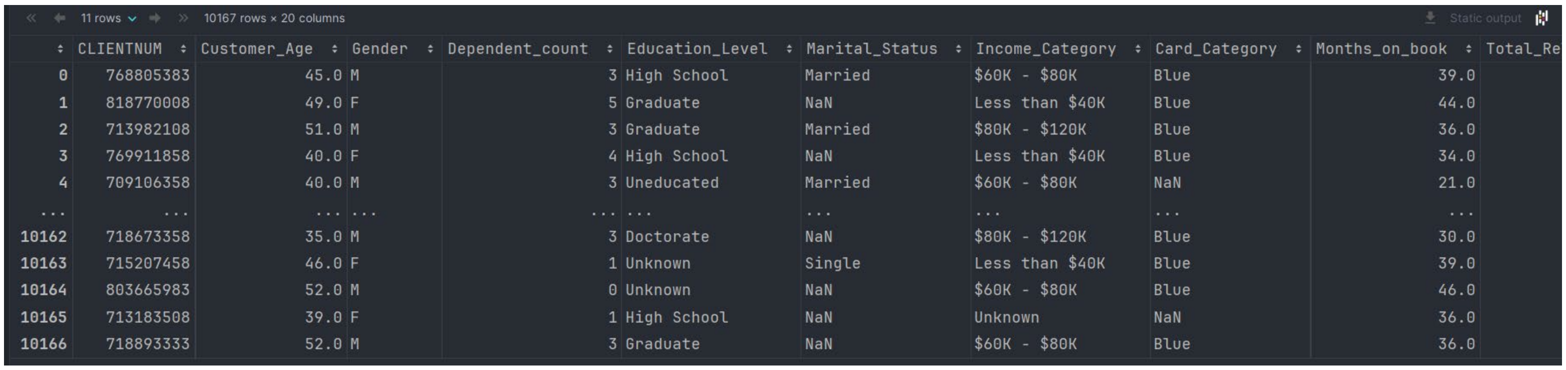
In my implementation I've used these Python Libraries:

- Numpy
- Pandas
- Sklearn
- MatPlotLib

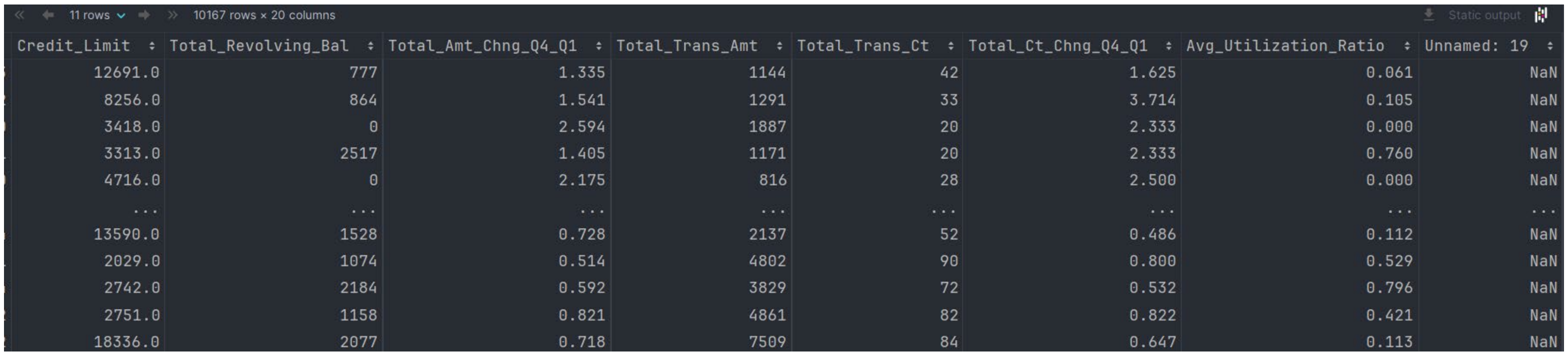
Chapter 1: Data Preprocessing

In this chapter I'll discuss the process of preprocessing our data so that it will be ready to be given to our models.

At first let's look at our dataset:



```
11 rows    10167 rows × 20 columns                                                                              Static output
       CLIENTNUM  Customer_Age  Gender  Dependent_count  Education_Level  Marital_Status  Income_Category  Card_Category  Months_on_book  Total_Re
    0  768805383          45.0  M                     3  High School      Married         $60K - $80K      Blue                     39.0
    1  818770008          49.0  F                     5  Graduate         NaN             Less than $40K   Blue                     44.0
    2  713982108          51.0  M                     3  Graduate         Married         $80K - $120K     Blue                     36.0
    3  769911858          40.0  F                     4  High School      NaN             Less than $40K   Blue                     34.0
    4  709106358          40.0  M                     3  Uneducated       Married         $60K - $80K      NaN                      21.0
  ...        ...           ...  ...                 ...  ...              ...             ...              ...                       ...
10162  718673358          35.0  M                     3  Doctorate        NaN             $80K - $120K     Blue                     30.0
10163  715207458          46.0  F                     1  Unknown          Single          Less than $40K   Blue                     39.0
10164  803665983          52.0  M                     0  Unknown          NaN             $60K - $80K      Blue                     46.0
10165  713183508          39.0  F                     1  High School      NaN             Unknown          NaN                      36.0
10166  718893333          52.0  M                     3  Graduate         NaN             $60K - $80K      Blue                     36.0
```



```
11 rows    10167 rows × 20 columns                                                                              Static output
Credit_Limit  Total_Revolving_Bal  Total_Amt_Chng_Q4_Q1  Total_Trans_Amt  Total_Trans_Ct  Total_Ct_Chng_Q4_Q1  Avg_Utilization_Ratio  Unnamed: 19
     12691.0                  777                 1.335             1144              42                1.625                  0.061          NaN
      8256.0                  864                 1.541             1291              33                3.714                  0.105          NaN
      3418.0                    0                 2.594             1887              20                2.333                  0.000          NaN
      3313.0                 2517                 1.405             1171              20                2.333                  0.760          NaN
      4716.0                    0                 2.175              816              28                2.500                  0.000          NaN
         ...                  ...                   ...              ...             ...                  ...                    ...          ...
     13590.0                 1528                 0.728             2137              52                0.486                  0.112          NaN
      2029.0                 1074                 0.514             4802              90                0.800                  0.529          NaN
      2742.0                 2184                 0.592             3829              72                0.532                  0.796          NaN
      2751.0                 1158                 0.821             4861              82                0.822                  0.421          NaN
     18336.0                 2077                 0.718             7509              84                0.647                  0.113          NaN
```

As you can see there are useless columns, NaN cells, Unknown cells, and both numerical and String type data.

1) Useless Columns

The first column of our dataset is the client number which has no meaning for our prediction of client's credit limit. The last column also contains NaN cell for each row, so the aforementioned columns should be dropped and not be considered as features:

```python
# drop last column because all NaN
df.drop("Unnamed: 19", axis=1, inplace=True)
df.drop("CLIENTNUM", axis=1, inplace=True)
```

2) Filling NaN and Unknown Cells

NaN and Unknown Cells must be Handled as they are one of the important parts of our data preprocessing. One way to handle this event is to drop the rows that contain such elements, but this is undesired as it causes the removal of nearly half of the dataset. So, we should take another approach.

One way is to calculate the mode or mean of that column and replace any Unknown or NaN values with mean or mode of that column.

For string type columns mode is used and for numerical type columns mean of that column.

Here is the code:

```python
# replace mod of Marital Status for NaN and Unknown
mod_marriage = df["Marital_Status"].mode()[0]
new_col = df["Marital_Status"].replace(np.nan, mod_marriage)
df["Marital_Status"] = new_col.values
new_col = df["Marital_Status"].replace("Unknown", mod_marriage)
df["Marital_Status"] = new_col.values

# replace mod of Gender for NaN
mod_gender = df["Gender"].mode()[0]
new_col = df["Gender"].replace(np.nan, mod_gender)
df["Gender"] = new_col.values

# replace mod of Education_Level for Unknown
mod_edu = df["Education_Level"].mode()[0]
new_col = df["Education_Level"].replace("Unknown", mod_edu)
df["Education_Level"] = new_col.values

# replace mod Income_Category for Unknown
mod_inc = df["Income_Category"].mode()[0]
new_col = df["Income_Category"].replace("Unknown", mod_inc)
df["Income_Category"] = new_col.values

# replace mod Card_Category for NaN
mod_card_cat = df["Card_Category"].mode()[0]
new_col = df["Card_Category"].replace(np.nan, mod_card_cat)
df["Card_Category"] = new_col.values

# replace mean Months_on_book for NaN
mean_month = df["Months_on_book"].mean().round()
new_col = df["Months_on_book"].replace(np.nan, mean_month)
df["Months_on_book"] = new_col.values

# replace mean Total_Relationship_count for NaN
mean_total = df["Total_Relationship_Count"].mean().round()
new_col = df["Total_Relationship_Count"].replace(np.nan, mean_total)
df["Total_Relationship_Count"] = new_col.values
```

3) Dropping The Duplicate Rows

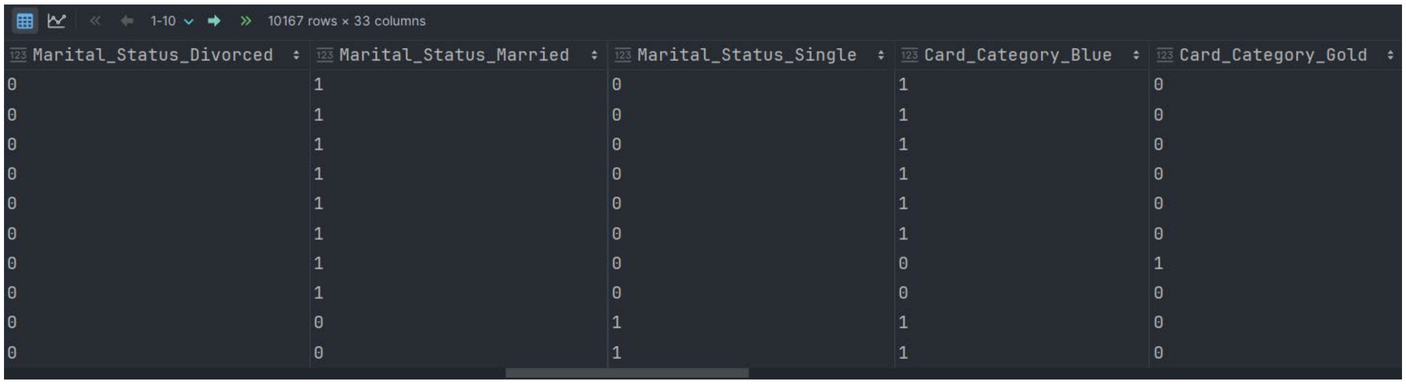
Now that NaN and Unknown cells are filled, some rows might become duplicated. In order to fix that we use the following code to drop those rows:

```
df = df.drop_duplicates()
```

4) One Hot Encoding

It's time to convert string type elements to numbers. In order to do that we can use a method called "One Hot Encoding".

In this method each unique element is considered to be a separate feature and column in our dataset. If an element is in a row, then the value in that column is 1 and 0 if not. Everything will be clearer with looking at an output of this method:



Marital_Status_Divorced	Marital_Status_Married	Marital_Status_Single	Card_Category_Blue	Card_Category_Gold
0	1	0	1	0
0	1	0	1	0
0	1	0	1	0
0	1	0	1	0
0	1	0	1	0
0	1	0	1	0
0	1	0	0	1
0	1	0	0	0
0	0	1	1	0
0	0	1	1	0

Here is the code for implementing this method:

```
# Now we convert the string type columns into numbers using One Hot Encoding
df_encoded = pd.get_dummies(df['Marital_Status'], prefix='Marital_Status')
df_encoded = df_encoded.astype(int)
df = pd.concat([df, df_encoded], axis=1)
df.drop('Marital_Status', axis=1, inplace=True)

df_encoded = pd.get_dummies(df['Card_Category'], prefix='Card_Category')
df_encoded = df_encoded.astype(int)
df = pd.concat([df, df_encoded], axis=1)
df.drop('Card_Category', axis=1, inplace=True)

df_encoded = pd.get_dummies(df['Gender'], prefix='Gender')
df_encoded = df_encoded.astype(int)
df = pd.concat([df, df_encoded], axis=1)
df.drop('Gender', axis=1, inplace=True)

df_encoded = pd.get_dummies(df["Education_Level"], prefix="Education_Level")
df_encoded = df_encoded.astype(int)
```

```python
df = pd.concat([df, df_encoded], axis=1)
df.drop("Education_Level", axis=1, inplace=True)

df_encoded = pd.get_dummies(df["Income_Category"], prefix="Income_Category")
df_encoded = df_encoded.astype(int)
df = pd.concat([df, df_encoded], axis=1)
df.drop("Income_Category", axis=1, inplace=True)
```

5) Outliers

This is a very important step in our process. It consists of two parts:

- Outlier Detection
- Outlier Handling

1) Outlier Detection

For detecting outliers, I have come across two methods. The first one requires assuming that our data has Normal distribution. Then we can use the interval of [μ - 3 * σ, μ + 3 * σ] to detect any cell in a column that is outside of this range (the mean and std. are calculated for each column).

The other method is for skewed distributions. We should first find the first and third Quartiles (Q1 and Q3) of each column and find the IQR (Interquartile range) that is:

IQR = Q3 – Q1

And then find the interval of [Q1 – 1.5 * IQR, Q3 + 1.5 * IQR]. Again, any cell in a column outside of this range is considered an outlier.

2) Outlier Handling

After detecting outliers it’s time to handle them. There are two approaches:

1 – dropping rows containing outliers:

This method is not desired because we may lose a large part of our data.

2 – replacing outliers with mean value of that column

I have used IQR and mean value replacement in my process.

Here is the code:

```
result = pd.DataFrame(index=data.index, columns=data.columns)

# Loop through each column in the DataFrame
for col in data.columns:
    Q1 = data[col].quantile(0.25)  # 25th percentile (Q1)
    Q3 = data[col].quantile(0.75)  # 75th percentile (Q3)
    IQR = Q3 - Q1  # Interquartile Range (IQR)
    lower_bound = Q1 - 1.5 * IQR  # Lower bound for outliers
    upper_bound = Q3 + 1.5 * IQR  # Upper bound for outliers

    # Replace outliers with mode for the current column
    mean_val = data.loc[(data[col] >= lower_bound) & (data[col] <=
upper_bound), col].mean()
    result[col] = data[col].apply(lambda x: mean_val if x < lower_bound or x
> upper_bound else x)
```

6) Train Test Split

Now I split the data into test and train.

Train is used to train the model to find the best Regressor.

Test is for checking the MSE of our Regressor.

I have set 10% of the data for test and the rest for train.

Shuffling the data ensures that you have difficult data in your train batch.

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1,
shuffle=True)
```

7) Scaling

Scaling our data makes it reasonable for our model. Columns have values in different ranges, scaling the data ensures that our data is in a specified range.

I've tested two scaling methods:

1 - Min Max Scaling

2- Standard Scaling

The outcome of both Scaling methods didn't differ much in the final test and I decided to use Min Max Scaling in the end:

```
scaler = MinMaxScaler()

x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

Chapter 2: Model Selection

1) Used Models

I have used these models:

1 – Linear Regression

2 – Polynomial Ridge Regression

3 – MLP (Multiple Layer Perceptron)

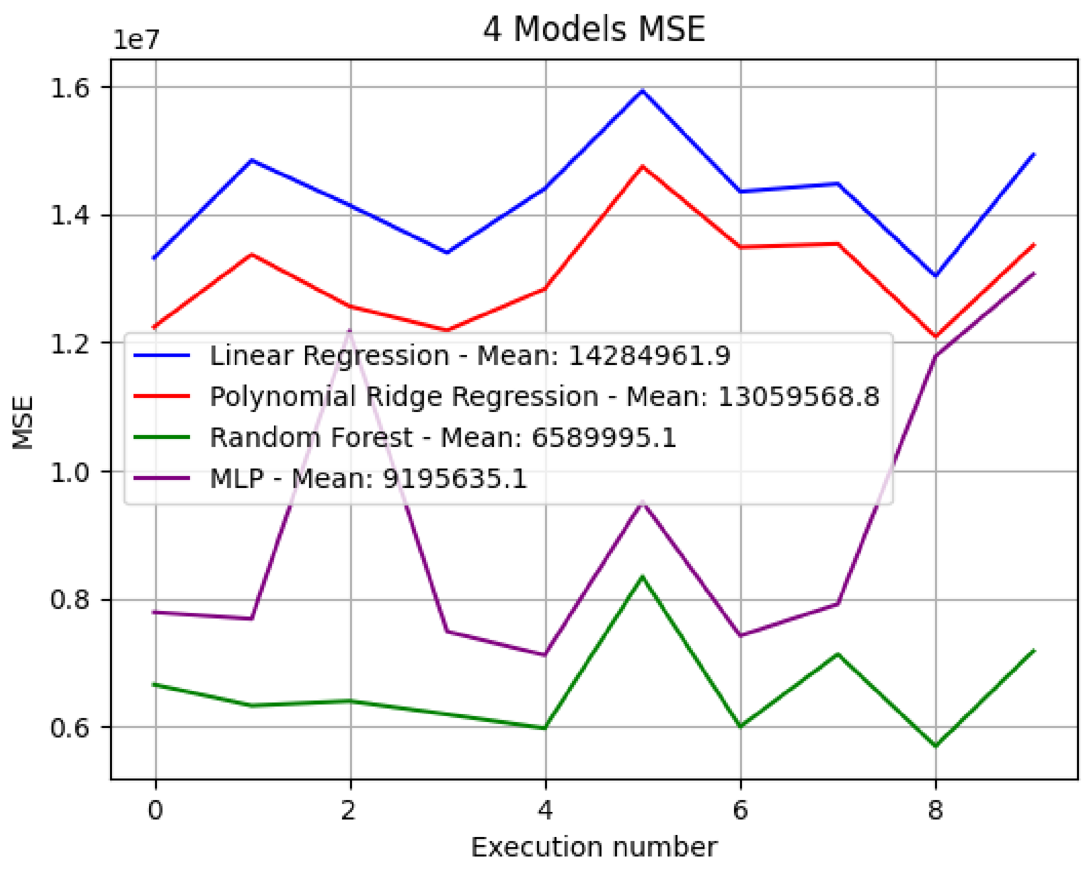
4 – Random Forest

```
lr = LinearRegression()
ridge = Ridge()
rf = RandomForestRegressor(n_estimators=100, max_depth=15, random_state=10)
mlp = MLPRegressor(max_iter=5000, random_state=0, learning_rate_init=0.005)

rf.fit(x_train, y_train)
y_pred_rf = rf.predict(x_test)

lr.fit(x_train, y_train)
y_pred_lr = lr.predict(x_test)

ridge.fit(x_train_poly, y_train)
y_pred_ridge = ridge.predict(x_test_poly)

mlp.fit(x_train, y_train)
y_pred_mlp = mlp.predict(x_test)

mse_rf = round(mean_squared_error(y_true=y_test, y_pred=y_pred_rf))
mse_lr = round(mean_squared_error(y_true=y_test, y_pred=y_pred_lr))
mse_ridge = round(mean_squared_error(y_true=y_test, y_pred=y_pred_ridge))
mse_mlp = round(mean_squared_error(y_true=y_test, y_pred=y_pred_mlp))
```

2) Models Comparison

After fitting with every model and calculating the MSE for 10 runs, I calculated the mean MSE of every model and plotted it using MatPlotLib library.

Here are my results:



As you can see Random Forest has the lowest mean MSE compared to the other models. The mean MSE of Random Forest is the range of 6 million and it is as low as I could get MSE to be in any of used models.

So, my final model is Random Forest and the hyperparameters were set to be:

n_estimators = 100 (determines the number of different decision trees used for averaging, and also determines the different subsamples used)

max_depth = 15 (the maximum depth of the decision trees in the forest)