Credit_card_adjudication_using_python

April 24, 2019

0.1 Project: Credit Card Application Approvals using Python

This notebook contains a credit card approval predictor for commercial banks using machine learning techniques.

0.1.1 Dataset

For this project, the dataset was extracted from the UCI Machine Learning Repository Credit Card Approval dataset

The dataset contains data for 690 customers that applied for credit with a retail bank. There are 16 attributes captured for each customer; including a decision flag which allows you to identify those customers which were approved and denied for credit.

0.1.2 Summary

The analysis of this project consist on the creation of a model to evaluate the decision to approve or deny credit card applications. The final model created is a logarithmic regression model. This model was able to predict the outcome of a credit applications with 84% accuracy which was significantly better performance than the baseline model.

As a conclusion, there are four drivers that possitively affect the approval decision, as these factors increase, so does the probability that a credit card will be issued.

Applications can get rejected for many reasons, including, like high loan balances, low income levels, or too many inquiries on an individual's credit report, among others. The four influencing factors are:

Prior default, Years employed, Credit score, and Income level. Other variables such as age, sex, or ethnicity did not have an influence on whether the application was denied. A Chi Squared test for independence validated our conclusion Ethnicity and Approval status are independent.

0.1.3 Notebook' structure

The structure of this notebook is as follows:

First, loading and viewing the dataset.

Second, preprocessing the dataset to ensure the machine learning model we choose can make good predictions.

Third, doing some exploratory data analysis to build our intuitions.

Finally, we will build a machine learning model using Logistic Regression that can predict if an individual's application for a credit card will be accepted.

0.2 1. Loading and viewing the dataset

We find that since this data is confidential, the dataset has alterations on the original data.

1 2. Inspecting the applications

Inspecting the structure, numerical summary, and specific rows of the dataset. - the dataset has a mixture of numerical and non-numerical features. This can be fixed with some preprocessing. - Specifically, the features 2, 7, 10 and 14 contain numeric values (of types float64, float64, int64 and int64 respectively) and all the other features contain non-numeric values. - The dataset also contains values from several ranges. Some features have a value range of 0 - 28, some have a range of 2 - 67, and some have a range of 1017 - 100000. - We can get useful statistical information (like mean, max, and min) about the features that have numerical values.

```
In [16]: # Print summary statistics
         credit_df_description = credit_df.describe()
         print(credit_df_description)

         print("\n")

         # Print DataFrame information
         credit_df_info = credit_df.info()
         print(credit_df_info)

         print("\n")

         # Inspect missing values in the dataset
         print(credit_df.tail(17))
```

```
             Debt  YearsEmployed  CreditScore         Income
count  690.000000     690.000000    690.00000     690.000000
mean     4.758725       2.223406      2.40000    1017.385507
std      4.978163       3.346513      4.86294    5210.102598
min      0.000000       0.000000      0.00000       0.000000
25%      1.000000       0.165000      0.00000       0.000000
50%      2.750000       1.000000      0.00000       5.000000
75%      7.207500       2.625000      3.00000     395.500000
max     28.000000      28.500000     67.00000  100000.000000


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
Gender            690 non-null object
Age               690 non-null object
Debt              690 non-null float64
Married           690 non-null object
BankCustomer      690 non-null object
```

```
EducationLevel    690 non-null object
Ethnicity         690 non-null object
YearsEmployed     690 non-null float64
PriorDefault      690 non-null object
Employed          690 non-null object
CreditScore       690 non-null int64
DriversLicense    690 non-null object
Citizen           690 non-null object
ZipCode           690 non-null object
Income            690 non-null int64
ApprovalStatus    690 non-null object
dtypes: float64(2), int64(2), object(12)
memory usage: 86.3+ KB
None
```

```
    Gender    Age    Debt Married BankCustomer EducationLevel Ethnicity  \
673      ?   29.5   2.000       y            p              e         h
674      a  37.33   2.500       u            g              i         h
675      a  41.58   1.040       u            g             aa         v
676      a  30.58  10.665       u            g              q         h
677      b  19.42   7.250       u            g              m         v
678      a  17.92  10.210       u            g             ff        ff
679      a  20.08   1.250       u            g              c         v
680      b   19.5   0.290       u            g              k         v
681      b  27.83   1.000       y            p              d         h
682      b  17.08   3.290       u            g              i         v
683      b  36.42   0.750       y            p              d         v
684      b  40.58   3.290       u            g              m         v
685      b  21.08  10.085       y            p              e         h
686      a  22.67   0.750       u            g              c         v
687      a  25.25  13.500       y            p             ff        ff
688      b  17.92   0.205       u            g             aa         v
689      b     35   3.375       u            g              c         h

     YearsEmployed PriorDefault Employed  CreditScore DriversLicense Citizen  \
673          2.000            f        f            0              f       g
674          0.210            f        f            0              f       g
675          0.665            f        f            0              f       g
676          0.085            f        t           12              t       g
677          0.040            f        t            1              f       g
678          0.000            f        f            0              f       g
679          0.000            f        f            0              f       g
680          0.290            f        f            0              f       g
681          3.000            f        f            0              f       g
682          0.335            f        f            0              t       g
683          0.585            f        f            0              f       g
684          3.500            f        f            0              t       s
```

```
685           1.250            f        f            0             f       g
686           2.000            f        t            2             t       g
687           2.000            f        t            1             t       g
688           0.040            f        f            0             f       g
689           8.290            f        f            0             t       g

    ZipCode  Income ApprovalStatus
673     256      17              -
674     260     246              -
675     240     237              -
676     129       3              -
677     100       1              -
678       0      50              -
679       0       0              -
680     280     364              -
681     176     537              -
682     140       2              -
683     240       3              -
684     400       0              -
685     260       0              -
686     200     394              -
687     200       1              -
688     280     750              -
689       0       0              -
```

1.1 3. Handling missing values (Marking missing values as NaN)

Marking Missing Values or corrupted data as NaN. Then, we can count the number of true values in each column.

- The dataset has missing values. The missing values in the dataset are labeled with '?'.
- Let's temporarily replace these missing value question marks with NaN.
- A total of 67 missing values were identified

```
In [17]: # Import numpy
         import numpy as np

         # Inspect missing values in the dataset
         print(credit_df.tail(17))

         # Count the number of NaNs in each column
         print(credit_df.isnull().sum())

         # Replace the '?'s with NaN
         credit_df = credit_df.replace('?', np.nan)

         # Inspect the missing values again
```

```
print(credit_df.tail(17))

# Count the number of NaNs in each column
print(credit_df.isnull().sum())
```

```
    Gender    Age    Debt Married BankCustomer EducationLevel Ethnicity  \
673      ?   29.5   2.000       y            p              e         h
674      a  37.33   2.500       u            g              i         h
675      a  41.58   1.040       u            g             aa         v
676      a  30.58  10.665       u            g              q         h
677      b  19.42   7.250       u            g              m         v
678      a  17.92  10.210       u            g             ff        ff
679      a  20.08   1.250       u            g              c         v
680      b   19.5   0.290       u            g              k         v
681      b  27.83   1.000       y            p              d         h
682      b  17.08   3.290       u            g              i         v
683      b  36.42   0.750       y            p              d         v
684      b  40.58   3.290       u            g              m         v
685      b  21.08  10.085       y            p              e         h
686      a  22.67   0.750       u            g              c         v
687      a  25.25  13.500       y            p             ff        ff
688      b  17.92   0.205       u            g             aa         v
689      b     35   3.375       u            g              c         h

     YearsEmployed PriorDefault Employed  CreditScore DriversLicense Citizen  \
673          2.000            f        f            0              f       g
674          0.210            f        f            0              f       g
675          0.665            f        f            0              f       g
676          0.085            f        t           12              t       g
677          0.040            f        t            1              f       g
678          0.000            f        f            0              f       g
679          0.000            f        f            0              f       g
680          0.290            f        f            0              f       g
681          3.000            f        f            0              f       g
682          0.335            f        f            0              t       g
683          0.585            f        f            0              f       g
684          3.500            f        f            0              t       s
685          1.250            f        f            0              f       g
686          2.000            f        t            2              t       g
687          2.000            f        t            1              t       g
688          0.040            f        f            0              f       g
689          8.290            f        f            0              t       g

    ZipCode  Income ApprovalStatus
673     256      17              -
674     260     246              -
675     240     237              -
676     129       3              -
```

```
677      100      1            -
678        0     50            -
679        0      0            -
680      280    364            -
681      176    537            -
682      140      2            -
683      240      3            -
684      400      0            -
685      260      0            -
686      200    394            -
687      200      1            -
688      280    750            -
689        0      0            -
Gender            0
Age               0
Debt              0
Married           0
BankCustomer      0
EducationLevel    0
Ethnicity         0
YearsEmployed     0
PriorDefault      0
Employed          0
CreditScore       0
DriversLicense    0
Citizen           0
ZipCode           0
Income            0
ApprovalStatus    0
dtype: int64
    Gender    Age    Debt Married BankCustomer EducationLevel Ethnicity  \
673    NaN   29.5   2.000       y            p              e         h
674      a  37.33   2.500       u            g              i         h
675      a  41.58   1.040       u            g             aa         v
676      a  30.58  10.665       u            g              q         h
677      b  19.42   7.250       u            g              m         v
678      a  17.92  10.210       u            g             ff        ff
679      a  20.08   1.250       u            g              c         v
680      b   19.5   0.290       u            g              k         v
681      b  27.83   1.000       y            p              d         h
682      b  17.08   3.290       u            g              i         v
683      b  36.42   0.750       y            p              d         v
684      b  40.58   3.290       u            g              m         v
685      b  21.08  10.085       y            p              e         h
686      a  22.67   0.750       u            g              c         v
687      a  25.25  13.500       y            p             ff        ff
688      b  17.92   0.205       u            g             aa         v
689      b     35   3.375       u            g              c         h
```

```
     YearsEmployed PriorDefault Employed  CreditScore DriversLicense Citizen  \
673          2.000            f        f            0              f       g
674          0.210            f        f            0              f       g
675          0.665            f        f            0              f       g
676          0.085            f        t           12              t       g
677          0.040            f        t            1              f       g
678          0.000            f        f            0              f       g
679          0.000            f        f            0              f       g
680          0.290            f        f            0              f       g
681          3.000            f        f            0              f       g
682          0.335            f        f            0              t       g
683          0.585            f        f            0              f       g
684          3.500            f        f            0              t       s
685          1.250            f        f            0              f       g
686          2.000            f        t            2              t       g
687          2.000            f        t            1              t       g
688          0.040            f        f            0              f       g
689          8.290            f        f            0              t       g

    ZipCode  Income ApprovalStatus
673     256      17              -
674     260     246              -
675     240     237              -
676     129       3              -
677     100       1              -
678       0      50              -
679       0       0              -
680     280     364              -
681     176     537              -
682     140       2              -
683     240       3              -
684     400       0              -
685     260       0              -
686     200     394              -
687     200       1              -
688     280     750              -
689       0       0              -
Gender            12
Age               12
Debt               0
Married            6
BankCustomer       6
EducationLevel     9
Ethnicity          9
YearsEmployed      0
PriorDefault       0
Employed           0
```

```
CreditScore        0
DriversLicense     0
Citizen            0
ZipCode           13
Income             0
ApprovalStatus     0
dtype: int64
```

1.2 5. Handling the missing values (Data Imputation)

Median Imputation for numerical data and Frequent value for categorical data.

- There are not missing values for numerical variables.
- There are still some missing values to be imputed for columns Gender, Age, Married, BankCustomer, EducationLevel, Ethnicity and ZipCode. All of these columns contain non-numeric data and we are going to impute these missing values with the most frequent values as present in the respective columns.

```
In [18]: # Iterate over each column of credit_df
         for col in credit_df.columns:
             # Check if the column is of object type
             if credit_df[col].dtypes == 'object':
                 # Impute with the most frequent value
                 credit_df = credit_df.fillna(credit_df[col].value_counts().index[0])

         # Count the number of NaNs in the dataset and print the counts to verify
         print(credit_df.isnull().sum())
```

```
Gender            0
Age               0
Debt              0
Married           0
BankCustomer      0
EducationLevel    0
Ethnicity         0
YearsEmployed     0
PriorDefault      0
Employed          0
CreditScore       0
DriversLicense    0
Citizen           0
ZipCode           0
Income            0
ApprovalStatus    0
dtype: int64
```

1.3 6. Preprocessing the data (Convert the non-numeric values to numeric)

we will be converting all the non-numeric values into numeric ones using label encoding.

We do this because not only it results in a faster computation but also many machine learning models (like XGBoost and especially the ones developed using scikit-learn) require the data to be in numeric format.

```
In [19]: # Import LabelEncoder
         from sklearn import preprocessing

         # Instantiate LabelEncoder
         le = preprocessing.LabelEncoder()

         # Iterate over all the values of each column and extract their dtypes
         for col in credit_df.columns:
             # Compare if the dtype is object
             if credit_df[col].dtypes =='object':
             # Use LabelEncoder to do the numeric transformation
                 credit_df[col]=le.fit_transform(credit_df[col])
             print(le.classes_)
```

```
['a' 'b']
['13.75' '15.17' '15.75' '15.83' '15.92' '16' '16.08' '16.17' '16.25'
 '16.33' '16.5' '16.92' '17.08' '17.25' '17.33' '17.42' '17.5' '17.58'
 '17.67' '17.83' '17.92' '18' '18.08' '18.17' '18.25' '18.33' '18.42'
 '18.5' '18.58' '18.67' '18.75' '18.83' '18.92' '19' '19.17' '19.33'
 '19.42' '19.5' '19.58' '19.67' '19.75' '20' '20.08' '20.17' '20.25'
 '20.33' '20.42' '20.5' '20.67' '20.75' '20.83' '21' '21.08' '21.17'
 '21.25' '21.33' '21.42' '21.5' '21.58' '21.67' '21.75' '21.83' '21.92'
 '22' '22.08' '22.17' '22.25' '22.33' '22.42' '22.5' '22.58' '22.67'
 '22.75' '22.83' '22.92' '23' '23.08' '23.17' '23.25' '23.33' '23.42'
 '23.5' '23.58' '23.75' '23.92' '24.08' '24.17' '24.33' '24.42' '24.5'
 '24.58' '24.75' '24.83' '24.92' '25' '25.08' '25.17' '25.25' '25.33'
 '25.42' '25.5' '25.58' '25.67' '25.75' '25.83' '25.92' '26' '26.08'
 '26.17' '26.25' '26.33' '26.5' '26.58' '26.67' '26.75' '26.83' '26.92'
 '27' '27.17' '27.25' '27.33' '27.42' '27.58' '27.67' '27.75' '27.83' '28'
 '28.08' '28.17' '28.25' '28.33' '28.42' '28.5' '28.58' '28.67' '28.75'
 '28.92' '29.17' '29.25' '29.42' '29.5' '29.58' '29.67' '29.75' '29.83'
 '29.92' '30' '30.08' '30.17' '30.25' '30.33' '30.42' '30.5' '30.58'
 '30.67' '30.75' '30.83' '31' '31.08' '31.25' '31.33' '31.42' '31.58'
 '31.67' '31.75' '31.83' '31.92' '32' '32.08' '32.17' '32.25' '32.33'
 '32.42' '32.67' '32.75' '32.83' '32.92' '33' '33.08' '33.17' '33.25'
 '33.5' '33.58' '33.67' '33.75' '33.92' '34' '34.08' '34.17' '34.25'
 '34.42' '34.5' '34.58' '34.67' '34.75' '34.83' '34.92' '35' '35.17'
 '35.25' '35.42' '35.58' '35.75' '36' '36.08' '36.17' '36.25' '36.33'
 '36.42' '36.5' '36.58' '36.67' '36.75' '37.17' '37.33' '37.42' '37.5'
 '37.58' '37.75' '38.17' '38.25' '38.33' '38.42' '38.58' '38.67' '38.75'
 '38.92' '39' '39.08' '39.17' '39.25' '39.33' '39.42' '39.5' '39.58'
 '39.83' '39.92' '40' '40.25' '40.33' '40.58' '40.83' '40.92' '41' '41.17'
 '41.33' '41.42' '41.5' '41.58' '41.75' '41.92' '42' '42.08' '42.17'
 '42.25' '42.5' '42.75' '42.83' '43' '43.08' '43.17' '43.25' '44' '44.17'
 '44.25' '44.33' '44.83' '45' '45.17' '45.33' '45.83' '46' '46.08' '46.67'
 '47' '47.17' '47.25' '47.33' '47.42' '47.67' '47.75' '47.83' '48.08'
 '48.17' '48.25' '48.33' '48.5' '48.58' '48.75' '49' '49.17' '49.5'
 '49.58' '49.83' '50.08' '50.25' '50.75' '51.33' '51.42' '51.58' '51.83'
 '51.92' '52.17' '52.33' '52.42' '52.5' '52.83' '53.33' '53.92' '54.33'
 '54.42' '54.58' '54.83' '55.75' '55.92' '56' '56.42' '56.5' '56.58'
 '56.75' '56.83' '57.08' '57.42' '57.58' '57.83' '58.33' '58.42' '58.58'
 '58.67' '59.5' '59.67' '60.08' '60.58' '60.92' '62.5' '62.75' '63.33'
 '64.08' '65.17' '65.42' '67.75' '68.67' '69.17' '69.5' '71.58' '73.42'
 '74.83' '76.75' '80.25' 'b']
['13.75' '15.17' '15.75' '15.83' '15.92' '16' '16.08' '16.17' '16.25'
 '16.33' '16.5' '16.92' '17.08' '17.25' '17.33' '17.42' '17.5' '17.58'
 '17.67' '17.83' '17.92' '18' '18.08' '18.17' '18.25' '18.33' '18.42'
 '18.5' '18.58' '18.67' '18.75' '18.83' '18.92' '19' '19.17' '19.33'
 '19.42' '19.5' '19.58' '19.67' '19.75' '20' '20.08' '20.17' '20.25'
 '20.33' '20.42' '20.5' '20.67' '20.75' '20.83' '21' '21.08' '21.17'
 '21.25' '21.33' '21.42' '21.5' '21.58' '21.67' '21.75' '21.83' '21.92'
 '22' '22.08' '22.17' '22.25' '22.33' '22.42' '22.5' '22.58' '22.67'
 '22.75' '22.83' '22.92' '23' '23.08' '23.17' '23.25' '23.33' '23.42'
 '23.5' '23.58' '23.75' '23.92' '24.08' '24.17' '24.33' '24.42' '24.5'
 '24.58' '24.75' '24.83' '24.92' '25' '25.08' '25.17' '25.25' '25.33'
 '25.42' '25.5' '25.58' '25.67' '25.75' '25.83' '25.92' '26' '26.08'
 '26.17' '26.25' '26.33' '26.5' '26.58' '26.67' '26.75' '26.83' '26.92'
 '27' '27.17' '27.25' '27.33' '27.42' '27.58' '27.67' '27.75' '27.83' '28'
 '28.08' '28.17' '28.25' '28.33' '28.42' '28.5' '28.58' '28.67' '28.75'
 '28.92' '29.17' '29.25' '29.42' '29.5' '29.58' '29.67' '29.75' '29.83'
 '29.92' '30' '30.08' '30.17' '30.25' '30.33' '30.42' '30.5' '30.58'
 '30.67' '30.75' '30.83' '31' '31.08' '31.25' '31.33' '31.42' '31.58'
 '31.67' '31.75' '31.83' '31.92' '32' '32.08' '32.17' '32.25' '32.33'
 '32.42' '32.67' '32.75' '32.83' '32.92' '33' '33.08' '33.17' '33.25'
 '33.5' '33.58' '33.67' '33.75' '33.92' '34' '34.08' '34.17' '34.25'
 '34.42' '34.5' '34.58' '34.67' '34.75' '34.83' '34.92' '35' '35.17'
 '35.25' '35.42' '35.58' '35.75' '36' '36.08' '36.17' '36.25' '36.33'
 '36.42' '36.5' '36.58' '36.67' '36.75' '37.17' '37.33' '37.42' '37.5'
 '37.58' '37.75' '38.17' '38.25' '38.33' '38.42' '38.58' '38.67' '38.75'
 '38.92' '39' '39.08' '39.17' '39.25' '39.33' '39.42' '39.5' '39.58'
 '39.83' '39.92' '40' '40.25' '40.33' '40.58' '40.83' '40.92' '41' '41.17'
 '41.33' '41.42' '41.5' '41.58' '41.75' '41.92' '42' '42.08' '42.17'
 '42.25' '42.5' '42.75' '42.83' '43' '43.08' '43.17' '43.25' '44' '44.17'
 '44.25' '44.33' '44.83' '45' '45.17' '45.33' '45.83' '46' '46.08' '46.67'
 '47' '47.17' '47.25' '47.33' '47.42' '47.67' '47.75' '47.83' '48.08'
 '48.17' '48.25' '48.33' '48.5' '48.58' '48.75' '49' '49.17' '49.5'
 '49.58' '49.83' '50.08' '50.25' '50.75' '51.33' '51.42' '51.58' '51.83'
 '51.92' '52.17' '52.33' '52.42' '52.5' '52.83' '53.33' '53.92' '54.33'
 '54.42' '54.58' '54.83' '55.75' '55.92' '56' '56.42' '56.5' '56.58'
 '56.75' '56.83' '57.08' '57.42' '57.58' '57.83' '58.33' '58.42' '58.58'
 '58.67' '59.5' '59.67' '60.08' '60.58' '60.92' '62.5' '62.75' '63.33'
 '64.08' '65.17' '65.42' '67.75' '68.67' '69.17' '69.5' '71.58' '73.42'
 '74.83' '76.75' '80.25' 'b']
['b' 'l' 'u' 'y']
['b' 'g' 'gg' 'p']
['aa' 'b' 'c' 'cc' 'd' 'e' 'ff' 'i' 'j' 'k' 'm' 'q' 'r' 'w' 'x']
['b' 'bb' 'dd' 'ff' 'h' 'j' 'n' 'o' 'v' 'z']
['b' 'bb' 'dd' 'ff' 'h' 'j' 'n' 'o' 'v' 'z']
['f' 't']
['f' 't']
['f' 't']
['f' 't']
['g' 'p' 's']
['0' '100' '102' '108' '110' '112' '1160' '117' '120' '121' '128' '129'
 '130' '132' '136' '140' '141' '144' '145' '150' '152' '154' '156' '160'
 '163' '164' '167' '168' '17' '170' '171' '174' '176' '178' '180' '181'
 '186' '188' '195' '20' '200' '2000' '202' '204' '208' '21' '210' '211'
 '212' '216' '22' '220' '221' '224' '225' '228' '230' '231' '232' '239'
 '24' '240' '250' '252' '253' '254' '256' '260' '263' '268' '272' '274'
 '276' '28' '280' '288' '29' '290' '292' '30' '300' '303' '309' '311'
 '312' '32' '320' '329' '330' '333' '340' '348' '349' '350' '352' '356'
 '360' '368' '369' '370' '371' '372' '375' '380' '381' '383' '393' '395'
 '396' '399' '40' '400' '408' '410' '411' '416' '420' '422' '43' '431'
 '432' '434' '440' '443' '45' '450' '454' '455' '460' '465' '470' '480'
 '487' '49' '491' '50' '500' '510' '515' '519' '52' '520' '523' '550' '56'
 '560' '583' '60' '600' '62' '640' '680' '70' '711' '720' '73' '75' '76'
 '760' '80' '840' '86' '88' '92' '928' '93' '94' '96' '980' '99' 'b']
['0' '100' '102' '108' '110' '112' '1160' '117' '120' '121' '128' '129'
 '130' '132' '136' '140' '141' '144' '145' '150' '152' '154' '156' '160'
 '163' '164' '167' '168' '17' '170' '171' '174' '176' '178' '180' '181'
 '186' '188' '195' '20' '200' '2000' '202' '204' '208' '21' '210' '211'
 '212' '216' '22' '220' '221' '224' '225' '228' '230' '231' '232' '239'
 '24' '240' '250' '252' '253' '254' '256' '260' '263' '268' '272' '274'
 '276' '28' '280' '288' '29' '290' '292' '30' '300' '303' '309' '311'
 '312' '32' '320' '329' '330' '333' '340' '348' '349' '350' '352' '356'
 '360' '368' '369' '370' '371' '372' '375' '380' '381' '383' '393' '395'
 '396' '399' '40' '400' '408' '410' '411' '416' '420' '422' '43' '431'
 '432' '434' '440' '443' '45' '450' '454' '455' '460' '465' '470' '480'
 '487' '49' '491' '50' '500' '510' '515' '519' '52' '520' '523' '550' '56'
 '560' '583' '60' '600' '62' '640' '680' '70' '711' '720' '73' '75' '76'
 '760' '80' '840' '86' '88' '92' '928' '93' '94' '96' '980' '99' 'b']
['+' '-']
```

1.4 7. Splitting the dataset into train and test sets and Feature selection

Now, we will split our data into train and test sets. Ideally, no information from the test data should be used to scale the training data or should be used to direct the training process of a machine learning model. Hence, we first split the data and then apply the scaling.

Also, features like DriversLicense and ZipCode are not as important as the other features in the dataset for predicting credit card approvals. We should drop them to design our machine learning model with the best set of features.

```
In [25]: # Import train_test_split
         from sklearn.model_selection import train_test_split

         # Drop the features 11 and 13
         credit_df = credit_df.drop(['DriversLicense', 'ZipCode'], axis=1)

         # Convert the DataFrame to a NumPy array
         credit_df = credit_df.values

         # Segregate features and labels into separate variables
         X,y = credit_df[:,0:12] , credit_df[:,13]

         # Split into train and test sets
         X_train, X_test, y_train, y_test = train_test_split(X,
                                         y,
                                         test_size=0.25,
                                         random_state=42)
         print(X_train.shape)
         print(X_test.shape)
```

```
(462, 12)
(228, 12)
```

1.5 8. Preprocessing the data (Rescaling Data to an uniform range)

We are only left with one final preprocessing step of scaling data between 0-1 before we can fit a machine learning model to the data.

For example, the credit score, CreditScore, of a person is their creditworthiness based on their credit history. The higher this number, the more financially trustworthy a person is considered to be. So, a CreditScore of 1 is the highest since we're rescaling all the values to the range of 0-1.

```
In [29]: # Import MinMaxScaler
         from sklearn.preprocessing import MinMaxScaler

         # Instantiate MinMaxScaler and use it to rescale X_train and X_test
         scaler = MinMaxScaler(feature_range=(0,1))
         rescaledX_train = scaler.fit_transform(X_train)
         rescaledX_test = scaler.fit_transform(X_test)1

         rescaledX_train[:1]
```

```
Out[29]: array([[0.        , 0.24928367, 0.0949307 , 1.        , 1.        ,
                 0.5       , 0.11111111, 0.225     , 0.        , 0.        ,
                 0.        , 0.        ]])
```

1.6 9. Fitting a logistic regression model to the train set

Essentially, predicting if a credit card application will be approved or not is a classification task. According to UCI, our dataset contains more instances that correspond to "Denied" status than instances corresponding to "Approved" status. Specifically, out of 690 instances, there are 383 (55.5%) applications that got denied and 307 (44.5%) applications that got approved.

This gives us a benchmark. A good machine learning model should be able to accurately predict the status of the applications with respect to these statistics.

Which model should we pick? A question to ask is: are the features that affect the credit card approval decision process correlated with each other? they indeed are correlated. Because of this correlation, we'll take advantage of the fact that generalized linear models perform well in these cases. Let's start our machine learning modeling with a Logistic Regression model (a generalized linear model).

```
In [37]: # Import LogisticRegression
         from sklearn.linear_model import LogisticRegression

         # Instantiate a LogisticRegression classifier with default parameter values
         logreg = LogisticRegression(solver='lbfgs')

         # Fit logreg to the train set
         logreg.fit(rescaledX_train,y_train )
```

```
Out[37]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='warn',
                   n_jobs=None, penalty='l2', random_state=None, solver='lbfgs',
                   tol=0.0001, verbose=0, warm_start=False)
```

1.7 10. Making predictions and evaluating performance

But how well does our model perform?

We will now evaluate our model on the test set with respect to classification accuracy. But we will also take a look the model's confusion matrix. In the case of predicting credit card applications, it is equally important to see if our machine learning model is able to predict the approval status of the applications as denied that originally got denied. If our model is not performing well in this aspect, then it might end up approving the application that should have been approved. The confusion matrix helps us to view our model's performance from these aspects.

- Our model was pretty good! It was able to yield an accuracy score of almost 84%.
- For the confusion matrix, the first element of the of the first row of the confusion matrix denotes the true negatives meaning the number of negative instances (denied applications) predicted by the model correctly. And the last element of the second row of the confusion matrix denotes the true positives meaning the number of positive instances (approved applications) predicted by the model correctly.

```
In [38]: # Import confusion_matrix
         from sklearn.metrics import confusion_matrix

         # Use logreg to predict instances from the test set and store it
         y_pred = logreg.predict(rescaledX_test)

         # Get the accuracy score of logreg model and print it
         print("Accuracy of logistic regression classifier: ", logreg.score(rescaledX_test, y_

         # Print the confusion matrix of the logreg model
         confusion_matrix(y_test, y_pred) # y_true Vs y_pred
```

```
Accuracy of logistic regression classifier:  0.8377192982456141
```

```
Out[38]: array([[93, 10],
                [27, 98]])
```

1.8 11. Grid searching and making the model perform better

Let's see if we can do better. We can perform a grid search of the model parameters to improve the model's ability to predict credit card approvals.

scikit-learn's implementation of logistic regression consists of different hyperparameters but we will grid search over the following two:

tol

max_iter

```
In [39]: # Import GridSearchCV
         from sklearn.model_selection import GridSearchCV

         # Define the grid of values for tol and max_iter
         tol = [0.01, 0.001, 0.0001]
         max_iter = [100, 150, 200]

         # Create a dictionary where tol and max_iter are keys and the lists of their values a
         param_grid = dict(tol=tol, max_iter=max_iter)
```

1.9 12. Finding the best performing model

We have defined the grid of hyperparameter values and converted them into a single dictionary format which GridSearchCV() expects as one of its parameters. Now, we will begin the grid search to see which values perform best.

We will instantiate GridSearchCV() with our earlier logreg model with all the data we have. Instead of passing train and test sets separately, we will supply X (scaled version) and y. We will also instruct GridSearchCV() to perform a cross-validation of five folds.

We'll end the notebook by storing the best-achieved score and the respective best parameters.

While building this credit card predictor, we tackled some of the most widely-known preprocessing steps such as scaling, label encoding, and missing value imputation. We finished with some machine learning to predict if a person's application for a credit card would get approved or not given some information about that person.

```
In [40]: # Grid searching is a process of finding an optimal set of values for the parameters

         # Instantiate GridSearchCV with the required parameters
         grid_model = GridSearchCV(estimator=logreg, param_grid=param_grid, cv=5)

         # Use scaler to rescale X and assign it to rescaledX
         rescaledX = scaler.fit_transform(X)

         # Fit data to grid_model
         grid_model_result = grid_model.fit( rescaledX, y)

         # Summarize results
         best_score, best_params = grid_model_result.best_score_, grid_model_result.best_param
         print("Best: %f using %s" % (best_score, best_params))
```

```
Best: 0.850725 using {'max_iter': 100, 'tol': 0.01}
```

```
In [ ]:
```