Name: Yazan Hannun
Track: Data Science

Predicting Credit Cards Approvals

Date: September 12, 2022

Table of Contents

LIST OF FIGURES ..... **III**

1 ACKNOWLEDGEMENTS ..... **IV**

2 SUMMARY ..... **V**

3 INTRODUCTION **6**

4 METHODOLOGY **7**

4.1 CREDIT CARD APPLICATIONS 7
4.2 INSPECTING THE APPLICATIONS 8
4.3 FEATURES SELECTION 9
4.4 HANDLING MISSING VALUES 9
4.5 PREPROCESSING THE DATA 10
4.5.1 Label Encoding *10*
4.5.2 MinMaxScaler *10*
4.6 FITTING A MACHINE LEARNING MODEL 11
4.6.1 Fitting train data to models *12*
4.6.2 Enhancing Model Efficiency with Hyper parameter Tuning *15*
4.6.3 Summary *16*
4.7 APPLICATION PROGRAM INTERFACE (API) 18
4.7.1 API 1 ..... *18*
4.7.2 API 2 ..... *19*

5 CONCLUSION **19**

6 REFERENCES **20**

List of Figures

FIGURE 1: TYPES OF ML MODELS 6
FIGURE 2: SUPERVISED LEARNING 7
FIGURE 3: DATA TYPES IN UCI DATASET 8
FIGURE 4: COLUMNS CONTAIN MISSING VALUES 9
FIGURE 5: COUNTS OF APPROVED OR DECLINED APPLICATIONS UCI DATASET 11
FIGURE 6: LOGISTIC REGRESSION REPORT 12
FIGURE 7: ACCURACY OF KNN MODEL OVER VARYING NUMBER OF NEIGHBORS 13
FIGURE 8: KNN REPORT 13
FIGURE 9: PERFORMANCE FOR LOGISTIC REGRESSION AND KNN (N_NEIGBOURS = 8) 14
FIGURE 10: REPORT OF LOGISTIC REGRESSION AFTER HYPER TUNING 15
FIGURE 11: REPORT OF KNN AFTER HYPER TUNING 16
FIGURE 12: KNN AND LOGISTIC REGRESSION MODELS COMPARISON 17

List of Tables

TABLE 1: HEAD OF THE UCI DATASET 7
TABLE 2: APPLICATIONS' DESCRIPTIVE STATISTICS 8
TABLE 3: KNN AND LOGISTIC REGRESSION MODELS COMPARISON 16

1 Acknowledgements

I would like to offer my special gratitude to my instructors, Mr. Yazan Kheiri (Technical Track Instructor) and Ms. Safad Al-Safadi (English Language Instructor), who's contribution in suggestions and encouragement helped me to coordinate my project, especially in building the machine learning model and writing this report. Also, I would like to give a special thank for HTU for giving me the opportunity to take part in this upskilling training. Moreover, I would like to thank the GIZ Organization and the Netherland Embassy in Jordan for their support to such programs.

2 Summary

In this project a supervised machine learning model was built to help commercial banks to predict whether customers are eligible or not eligible for obtaining a credit card. The machine learning model was trained on hundreds of credit card applications and their corresponding outcomes. I implemented two different machine learning models and optimized the hyperparameters in order to better evaluate their performance. Then I evaluated the performance using the accuracy score, which demonstrated that the KNN model used, worked better than the Logistic Regression one. I have used python's machine learning libraries such as sicket-learn to implement machine learning algorithms. Moreover, two APIs were created to help me retrieve a file that needs to be processed and predict the outcome, and then save it to a sqlite database, which also connected to a NOSQL cloud database, firebase. The second API generates a report for prediction based on a client request.

3 Introduction

Commercial banks receive a lot of applications for credit cards, but many of those applications get rejected. Reasons for rejection might include high loan balances, low-income levels, or too many inquiries on an individual's credit report. Since analyzing these applications would be tedious, time-consuming, and error-prone, banks nowadays use machine learning to automate the process. Thankfully, this task can be automated with the power of machine learning and pretty much every commercial bank does so nowadays. In this project, I will build an automatic credit card approval predictor using machine learning techniques. I 'll use the Credit Card Approval dataset from the UCI Machine Learning Repository [1].

In this project a machine learning model based on supervised learning algorithms will be built to predict the approval status of customers applications based on features such as: Gender, Age, Debit, Married, Bank Customer, Education Level, Ethnicity, Years Employed, Prior Default, Employed, Credit Score, Driver's License, and Income. The following are the ideas and concepts that help us understand the project and are essential for the development process (Fig.1). As such, understanding them will go a long way in facilitating the development process.

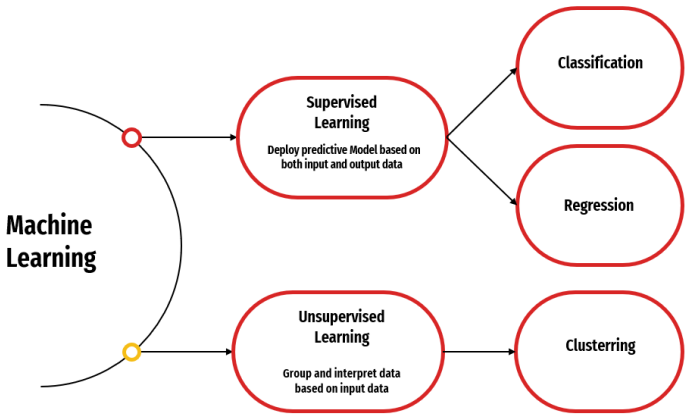

Figure 1: Types of ML Models

Supervised machine learning models are algorithms that use labeled data to predict outcomes. You need to feed the algorithm with input data with their labels such as illustrated in Fig.2, which is then used as a training set. This training set will teach the algorithm what to expect in order to obtain a prediction.

In this report, we will be analyzing the accuracy of the algorithm on new applications. I will be using a test set to evaluate the algorithm to see how accurate it is.

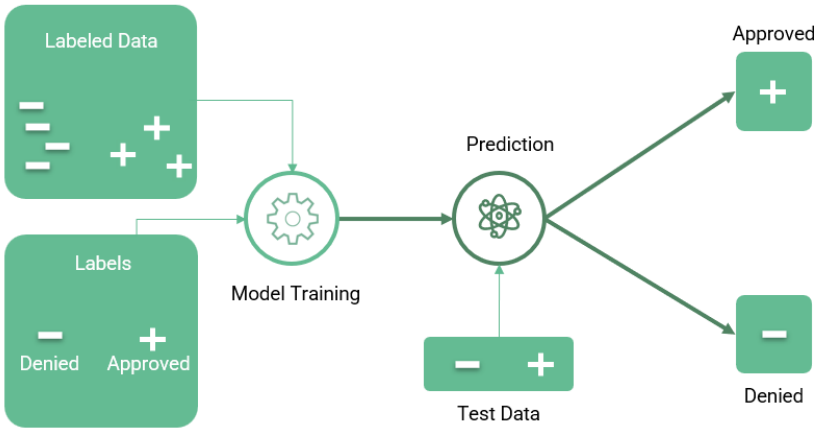

Figure 2: Supervised Learning

4 Methodology

4.1 Credit Card Applications

We will use the Credit Card Approval dataset from the UCI Machine Learning Repository [1]. First, we will start off by loading and viewing the dataset.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1	f	g	00202	0	+
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	00043	560	+
2	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	00280	824	+
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	00100	3	+

Table 1: Head of the UCI Dataset

After exploring this data set (Table 1), we have determined that there is a mixture of both numerical and non-numerical features, since these data are confidential, the contributor of the dataset has anonymized the feature names, but a blog [2] gives us a pretty good overview of the probable features. These features are:

[0:'Gender', 1:'Age', 2:'Debt', 3:'Married', 4:'BankCustomer', 5:'EducationLevel', 6:'Ethnicity', 7:'YearsEmployed', 8:'PriorDefault', 9:'Employed', 10:'CreditScore', 11:'DriversLicense', 12:'Citizen',13:'ZipCode', 14:'Income', 15:'ApprovalStatus'].

Looking at the last column in (Table 1), we can see that the status of credit card application is whether approved '+' or declined '-'. This gives us a pretty good starting point, and we can map these features with respect to the columns in the output.

4.2 Inspecting the applications

We can see from Fig.3, that there are a mix of numerical and non-numerical features in the dataset, but this can be fixed with some preprocessing. We should learn more about the dataset before proceeding to do any processing. Our dataset contains both numeric and non-numeric data (specifically data that are of float64, int64 and object types). 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-numerical values.

```
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Gender          690 non-null    object
 1   Age             690 non-null    object
 2   Debt            690 non-null    float64
 3   Married         690 non-null    object
 4   BankCustomer    690 non-null    object
 5   EducationLevel  690 non-null    object
 6   Ethnicity       690 non-null    object
 7   YearsEmployed   690 non-null    float64
 8   PriorDefault    690 non-null    object
 9   Employed        690 non-null    object
 10  CreditScore     690 non-null    int64
 11  DriversLicense  690 non-null    object
 12  Citizen         690 non-null    object
 13  ZipCode         690 non-null    object
 14  Income          690 non-null    int64
 15  ApprovalStatus  690 non-null    object
dtypes: float64(2), int64(2), object(12)
```

Figure 3: Data Types in UCI Dataset

To start, we will use the describe function to see the descriptive statistics of the numeric values such as min, max, mean, and median.

	2	7	10	14
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

Table 2: Applications' Descriptive Statistics

As we can see in Table 2, 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. Apart from these, we can get useful statistical information (like mean, max, and min) about the features that have numerical values.

4.3 Features Selection

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.

4.4 Handling missing values

In this stage our dataset contains missing values, the missing values are labeled by '?'. So, we must replace the missing values question mark by NaN. But before doing this let us explore which columns contain missing values.

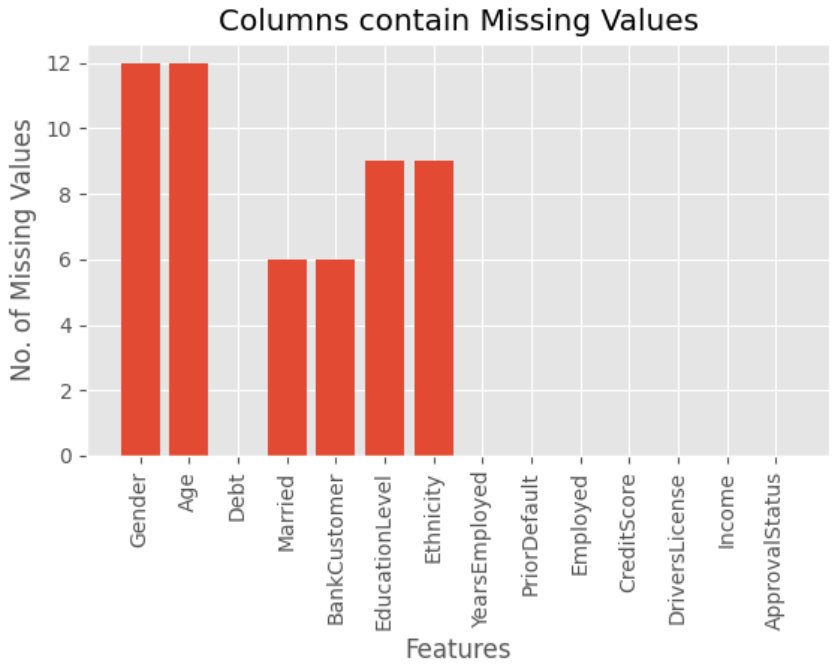

Figure 4: Columns contain missing values

As we can see in Fig. 4, many features contain missing values, and these features are numerical and non-numerical values. Missing or incomplete data can impact the accuracy of a machine learning model. When you ignore missing data, your model might not be able to learn information that is useful for its training. So, to avoid this problem, we are going to impute the missing values with a strategy called mean imputation for numerical values and with the most frequent values for non-numerical ones.

4.5 Preprocessing the data

The missing values are now successfully handled. There are still some minor but essential data preprocessing needed before we proceed towards building our machine learning model. We are going to divide these remaining preprocessing steps into two main tasks:

- Convert the non-numeric data into numeric.
- Scale the feature values to a uniform range.

First, we will be converting all the non-numeric values into numerical ones, this can be done using Label Encoding. In addition, we are going to use MinMaxScaler to scale our data to a uniform range. But before using MinMaxScaler we will split our dataset into train and test datasets.

4.5.1 Label Encoding

It is a technique that refers to the process of converting the labels to numeric form so that they can be converted into machine-readable form. This will help the machine learning algorithms to decide in an efficient way as to how those labels should be operated. It is a crucial step in the pre-processing stage for the dataset in supervised learning since all data should be strictly numerical.

4.5.2 MinMaxScaler

This assessor scales and deciphers each feature separately to such an extent that it is in the given range on the training set, for example somewhere in the range of zero and one. Trying to understand what these scaled values mean in the real world. Let's use CreditScore as an example. The credit score of a person is their creditworthiness based on their credit history. The higher this number, the more financially trustworthy a person is. So, a CreditScore of 1 is the highest since we're rescaling all the values to the range of (0 – 1).

`

4.6 Fitting a machine learning model

Now our data are ready to be fitted for a classification model. Essentially, predicting if a credit card application will be approved or not is a classification task. According to UCI [1], 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 as stated in Fig.5. 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.

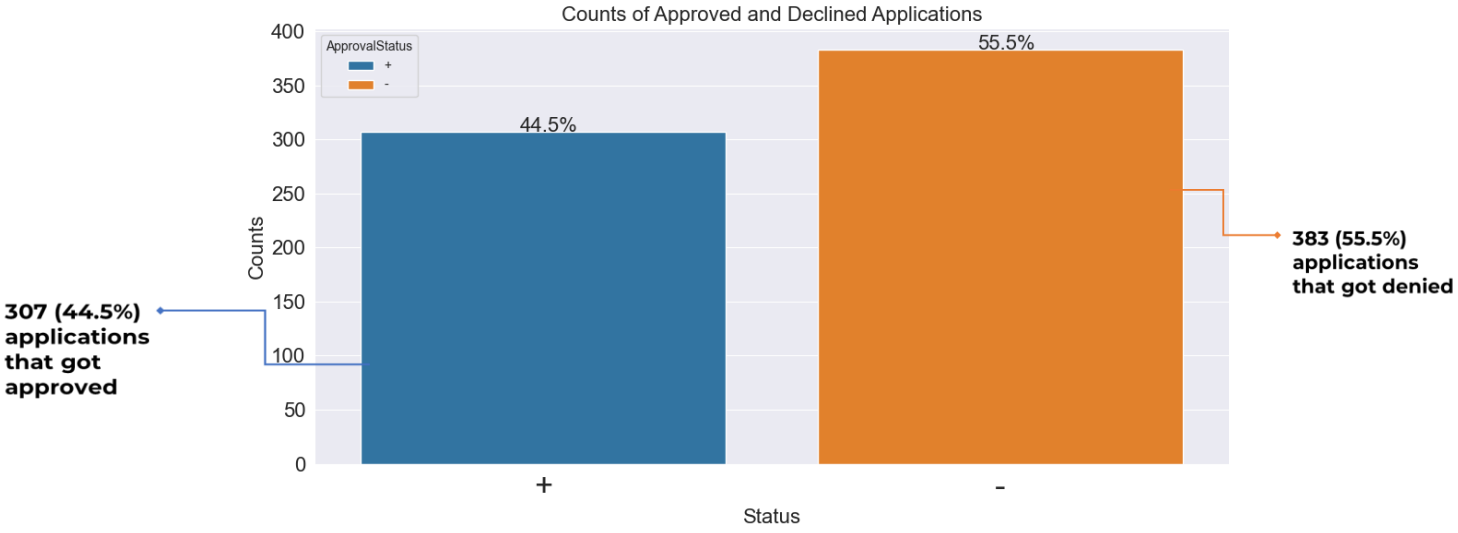

Figure 5: Counts of Approved or Declined Applications UCI Dataset

There are several classification models that can be used for this task. In this analysis, we will build two different types of classification models namely Logistic Regression, and K-Nearest Neighbors (KNN). These are the most popular models used for solving classification problems. All these models can be conveniently implemented using python's scikit-learn.

4.6.1 Fitting train data to models

4.6.1.1 Logistic Regression

Let's start our machine learning modeling with a Logistic Regression model. We will train the Logistic Regression model with standard parameters using the training dataset.

We evaluate the performance of our model using test dataset. We use the metric classification accuracy defined as the fraction of times model prediction matches the value of the target variable. For a detailed evaluation of our model, we look at the confusion matrix. The values in the diagonal of the confusion matrix denote the count of correct rejection (first-row first entry) or correct approval (second-row second entry) predictions by our classification model.

Results:

- Accuracy of LogisticRegression for train data: 0.8744588744588745
- Accuracy of LogisticRegression for test data: 0.8377192982456141

```
[[99 26]
 [11 92]]
              precision    recall  f1-score   support

           0       0.90      0.79      0.84       125
           1       0.78      0.89      0.83       103

    accuracy                           0.84       228
   macro avg       0.84      0.84      0.84       228
weighted avg       0.85      0.84      0.84       228
```

Figure 6: Logistic Regression Report

Statistics:

$$\textbf{Approved} = \frac{True\ Positive}{True\ Negative + True\ Positive} = \frac{92}{92+99} = 0.48167 = 48.167\%$$

$$\textbf{Denied} = \frac{True\ Negative}{True\ Negative + True\ Positive} = \frac{99}{92+99} = 0.51832 = 51.832\%$$

4.6.1.2 K-Nearest Neighbor

The second model we try for our classification task is the K-Nearest Neighbors (KNN). I have built the model using sklearn's KNeighborsClassifier algorithm. I have optimized the hyperparameter n_neighbors by iterating through a range of values from n=1 to n=25 and comparing the accuracy scores. I selected the value n_neighbors =8 as it avoids both overfitting and underfitting.

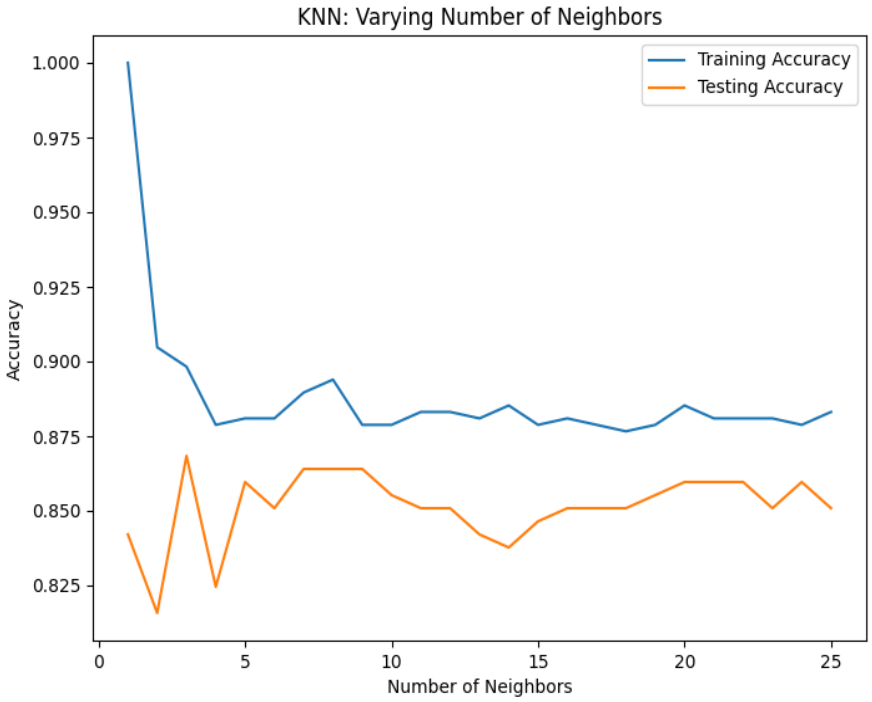

Figure 7: Accuracy of KNN model over varying number of neighbors

Results after choosing n_ neighbors = 8:

- Accuracy of KNeighborsClassifier for train data: 0.8939393939393939
- Accuracy of KNeighborsClassifier for test data: 0.8640350877192983

```
[[111  14]
 [ 17  86]]
              precision    recall  f1-score   support

           0       0.87      0.89      0.88       125
           1       0.86      0.83      0.85       103

    accuracy                           0.86       228
   macro avg       0.86      0.86      0.86       228
weighted avg       0.86      0.86      0.86       228
```

Figure 8: KNN Report

Statistics:

$$\textbf{Approved} = \frac{True\ Positive}{True\ Negative + True\ Positive} = \frac{86}{86+111} = 0.43216 = 43.216\%$$

$$\textbf{Denied} = \frac{True\ Negative}{True\ Negative + True\ Positive} = \frac{111}{86+111} = 0.55778 = 55.778\%$$

4.6.1.3 Evaluating classification models performance:

For the confusion matrix, the first element of the first row of the confusion matrix denotes the true negatives which indicates 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 indicates the number of positive instances (approved applications) predicted by the model correctly. As we can see in Fig.7, the KNN model is performing better compared to Logistic Regression, after looking at the median accuracy for KNN compared to the Logistic Regression.

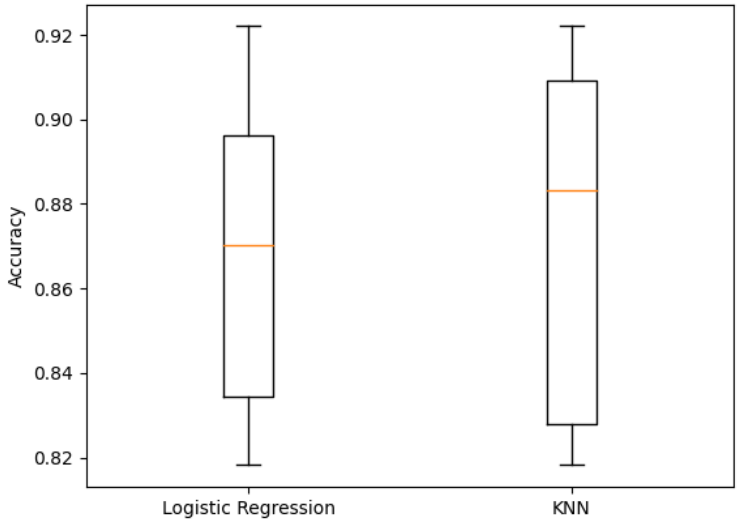

Figure 9: Performance for logistic regression and KNN (n_neigbours = 8)

4.6.2 Enhancing Model Efficiency with Hyper parameter Tuning

In this part we will try to improve the accuracy of our models by choosing the best parameters so that each model will perform better at those parameters. To do so we will use GridSearchCV.

4.6.2.1 Logistic Regression

In Logistic Regression we will try to enhance our model by choosing the best parameters that our model will perform better using them, these parameters are (tol and max_iter).

```
Best: 0.867965 using {'max_iter': 100, 'tol': 0.01}
[[99 26]
 [11 92]]
              precision    recall  f1-score   support

           0       0.90      0.79      0.84       125
           1       0.78      0.89      0.83       103

    accuracy                           0.84       228
   macro avg       0.84      0.84      0.84       228
weighted avg       0.85      0.84      0.84       228
```

Figure 10: Report of Logistic Regression after hyper tuning

Statistics:

$$\textbf{Approved} = \frac{True\ Positive}{True\ Negative + True\ Positive} = \frac{92}{99+92} = 0.481675 = 48.1675\%$$

$$\textbf{Denied} = \frac{True\ Negative}{True\ Negative + True\ Positive} = \frac{99}{92+99} = 0.51832 = 51.832\%$$

4.6.2.2 KNeighborsClassifier

In KNN we will try to enhance our model by choosing the best parameter that our model will perform better using it, this parameter is (n_neighbors).

```
Fitting 6 folds for each of 26 candidates, totalling 156 fits
{'n_neighbors': 22} 0.8852813852813853
Best: 0.885281 using {'n_neighbors': 22}
[[105  20]
 [ 12  91]]
              precision    recall  f1-score   support

           0       0.90      0.84      0.87       125
           1       0.82      0.88      0.85       103

    accuracy                           0.86       228
   macro avg       0.86      0.86      0.86       228
weighted avg       0.86      0.86      0.86       228
```

Figure 11: Report of KNN after hyper tuning

Statistics:

$$\textbf{Approved} = \frac{\textit{True Positive}}{\textit{True Negative+True Positive}} = \frac{91}{91+105} = 0.46428 = 43.216\%$$

$$\textbf{Denied} = \frac{\textit{True Negative}}{\textit{True Negative+True Positive}} = \frac{105}{91+105} = 0.53571 = 53.571\%$$

4.6.3 Summary

Model	Data	Before Hyper tuning (n = 8)	After Hyper tuning (n = 22) (tol=0.01, max_itr = 100)
KNN	Train Accuracy	0.893939	0.88528
	Test Accuracy	0.864035	0.86
Logistic Regression	Train Accuracy	0.874458	0.86796
	Test Accuracy	0.837719	0.84

Table 3: KNN and Logistic Regression models comparison

As we see in Table.3, the KNN model is performing better over our datasets at (neighbors = 8) in the test and train sets compared to Logistic Regression model at (tol=0.01, max_itr = 100). This can lead us to conclude that the KNN model is performing better. Moreover, the statistics for approved and denied applications in predicting test data are closer to statistics stated in the beginning of section 2.6. After creating our model, a file will be created called model.py, which we will be using to predict the new instances in the future.

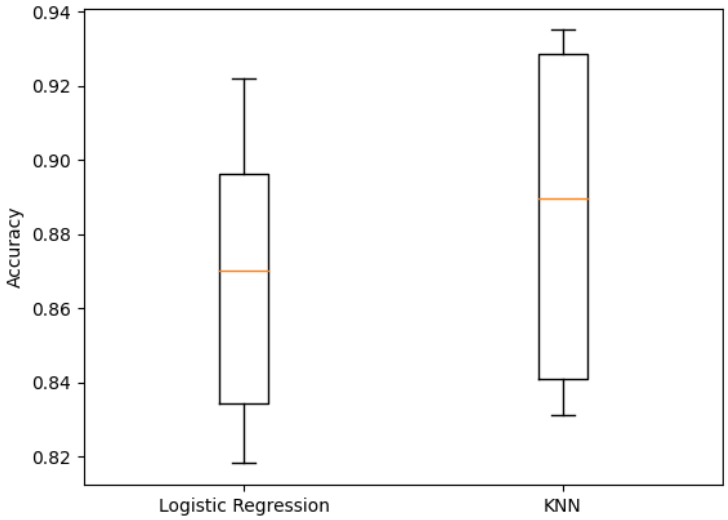

Figure 12: KNN and Logistic Regression models comparison

Date: September 12, 2022

4.7 Application Program Interface (API)

After creating the model and choosing the best performing one, two APIs will be created the first one will retrieve applications from the client for example (a bank) and then send back a .xlsx report to the client contains the predictions of the status of their customers credit card applications. Moreover, the sqlite database should also be auto synchronized to a cloud database (Firebase). Below is a description of how to create such APIs:

4.7.1 API 1

4.7.1.1 Retrieve the data from the client

In this API the client must send us a file containing the application data that their customers will fill during the application process for a credit card. This file will in .csv form, which contains all the features that will help us to predict the status of their customers (approved or denied). The client will convert the data in the file into a Json object, package and send us the file using get method, then we will catch the response as a Json object. After this we will convert the Json object into a DataFrame to make it easier for us to load and viewing the data in an easier way.

4.7.1.2 Handle and preprocess for prediction

After converting the file into a DataFrame, we will handle and preprocess the data using a file have been already created called preroccessing.py. This file contains a function that will handle the missing data by mean imputation and most frequent values as mentioned in section 2.4. Then the data will be ready to pass them for the model that we have created before to predict the application status for the customers using the file called model.py.

4.7.1.3 Create SQLite database

Moreover, my new predictions should be saved in a database using Sqlite, we will establish a connection inside the API to a database, create a table contains three columns (CustomerID, Date, and Status) and then insert the new predictions as approved or denied applications. After this an excel report should be created and sent back to the client in order to inform his customers of their applications' status.

4.7.1.4 Synchronize the data into a cloud database

At the end of the API a connection should be established to a cloud database to auto synchronize the data in the local database (SQLite) to a cloud database (NoSQL), firebase. After running our API, the data will be synchronized automatically to firebase cloud, each customer ID will show the date of the prediction and the application status.

4.7.2 API 2

This API was built in case the client wants a report about the predictions of the file, which has already been sent in API 1. After receiving a request to generate a report, this API will generate a report as an excel file contains the CustomerID, date of prediction and status of this customer application.

5 Conclusion

We have tried two different classification models for our credit card approval prediction task. The training and test accuracy of the models is summarized in the Table.3 below. We have obtained the best test data accuracy (89.39%) from the KNN classifier. The small difference in train and test accuracy scores indicates the absence of overfitting and underfitting.

6 References

[1] "UCI Machine Learning Repository," [Online]. Available: http://archive.ics.uci.edu/ml/datasets/credit+approval.

[2] A. o. C. A. Data. [Online]. Available: http://rstudio-pubs-static.s3.amazonaws.com/73039_9946de135c0a49daa7a0a9eda4a67a72.html.