Credit Card Fraud Detection

J COMPONENT PROJECT REPORT

Winter 2020-21

Submitted by

ANUSHKA DIXIT (19BCE0577)

in partial fulfillment for the award of the degree of

B. Tech

in

Computer Science and Engineering

Vellore-632014, Tamil Nadu, India

School of Computer Science and Engineering

May, 2021

Contents

Abstract
The main purpose of this project is to understand and implement the distinct approach of Random forest algorithm to identify fraudulent transactions in a database instead of SVM and Logical Regression Model. The model will be able to identify the transactions with greater accuracy and by comparing the approaches, a more optimal solution can be found.

The Credit Card Fraud Detection Problem includes modelling past credit card transactions with the knowledge of the ones that turned out to be fraud. This model is then used to identify whether a new transaction is fraudulent or not. The aim of the project here is to detect 100% of the fraudulent transactions while minimizing the incorrect fraud classifications.

Introduction

A robust system needs to be made to detect fraudulent transactions. The following challenges have to be overcome to make sure that the final product is resilient.

The challenge is to recognize fraudulent credit card transactions so that the customers of credit card companies are not charged for items that they did not purchase. Main challenges involved in credit card fraud detection are:

- Enormous Data is processed every day and the model build must be fast enough to respond to the scam in time.
- Imbalanced Data i.e., most of the transactions (99.8%) are not fraudulent which makes it really hard for detecting the fraudulent transactions.
- Data availability, as the data is mostly private.
- Misclassified Data can be another major issue, as not every fraudulent transaction is caught and reported.
- Adaptive techniques used against the model by the scammers.

The Matthew's correlation coefficient (MCC) is used to compare the performance of different models.

Dataset:

The dataset contains transactions made by credit cards in September 2013 by European cardholders, over a span of two days. 492 frauds out of 284,807 transactions are present. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation.

Unfortunately, due to confidentiality issues, the original features and more background information about the data cannot be provided.

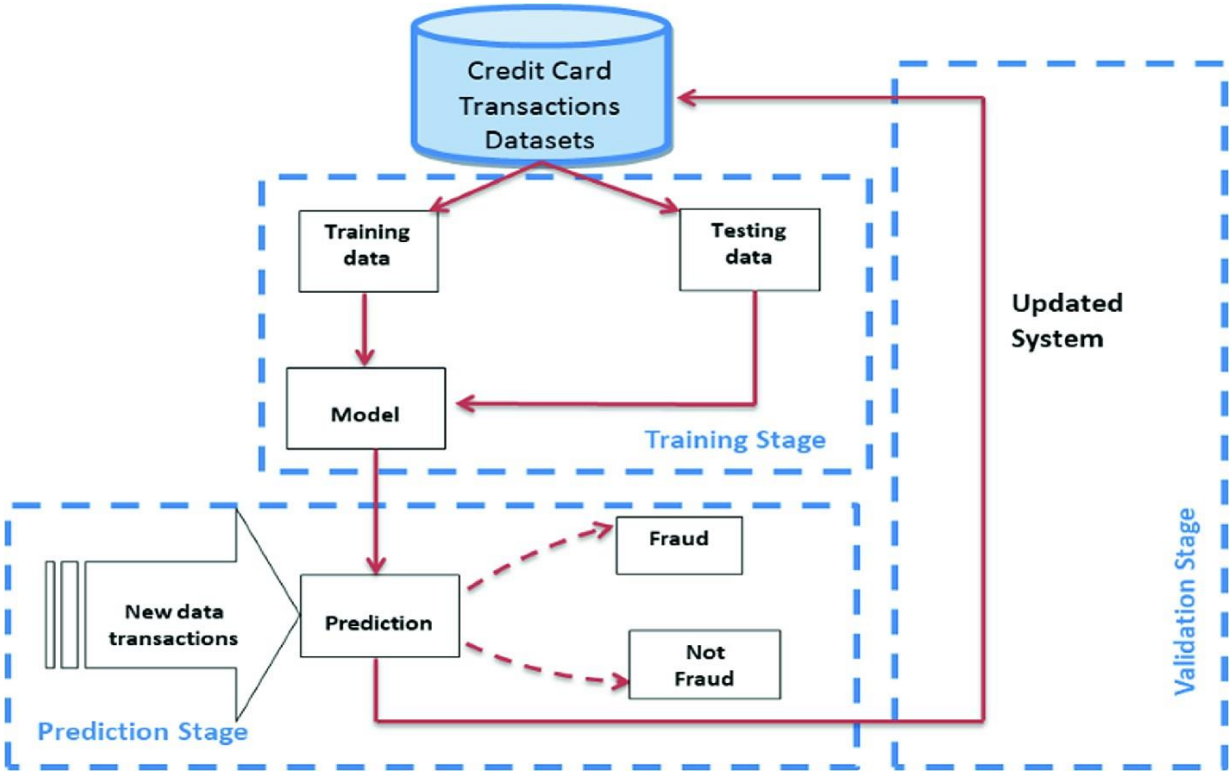
Random Forest Algorithm:

The method used here is Random Forest Classifier, which is basically a collection of various decision trees. Random forest adds randomness by searching for best feature among a random subset of features. A subset of the training set is sampled randomly to train each tree and then a decision tree is built. Each node then splits on a feature selected from a random subset of the full feature set. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees and leads to a higher rate of accuracy even for large datasets.

The dataset, as mentioned above, consists of a large volume of data and the method of analysis used by Random forest allows fast analysis and prediction, thus making the model suitable for real-world scenarios. Two basic assumptions of Random Forest:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

Architecture Design



Background study

Credit card fraud detection problem has been explored widely through Neural Networks, Bayesian networks and classification models, including decision trees, regression and random forest classification.

“Cost sensitive Modeling of Credit Card Fraud Using Neural Network strategy” [1] uses ANN to detect credit card fraud. Data normalization is applied before Cluster Analysis to increase the accuracy of the model. Accuracy of an algorithm is around 50%. This algorithm used to reduce the cost measure and result can be obtained 23% and this algorithm used find a Bayes minimum risk.

“Maes, S., Tuyls, K., Vanschoenwinkel, B., & Manderick, B. (2002, January). Credit card fraud detection using Bayesian and neural networks” [2] focuses on the machine learning methods of credit card fraud detection: artificial neural networks and Bayesian belief networks. It concentrates on the results and ways to improve the results of these algorithms.

“Shen, Aihua & Tong, Rencheng & Deng, Yaochen. (2007). Application of Classification Models on Credit Card Fraud Detection. International Conference on Service Systems and Service Management.” [3] investigates the efficiency of classification models, namely: decision tree, neural networks and logistic regression. A framework is provided that compares the applicability of these algorithms on the fraud detection problem.

M. R. Dileep, A. V. Navaneeth and M. Abhishek, "A Novel Approach for Credit Card Fraud Detection using Decision Tree and Random Forest Algorithms," 2021 explores the decision tree and random forest algorithm in fraud detection. Accuracy of the algorithms is tested using public data sample. The model proposed follows three steps: Clustering, Gaussian Mixture and Bayesian Network technique. Initially, clustering model is applied to categorize the data into fraud and genuine transactions by means of data clusterization of areas of factor values. The Gaussian method is used to model the credit card operator’s past performance which is then followed by the Bayesian Network Technique, which connects the dependencies of the model into a DAG and gives an approximate area where frauds may occur.

S. Xuan, G. Liu, Z. Li, L. Zheng, S. Wang and C. Jiang, "Random forest for credit card fraud detection," 2018 compares the performance of two kinds of random forests and makes efforts to improve the algorithm while dealing with highly imbalanced data. The accuracy of these models came at 91.96% and 97.77%, tested on a dataset with 0.27% fraudulent transactions.

Methodology

1. Data Pre-processing
    i. Importing necessary libraries:

Here, library is a collection of built-in core modules that provide function to system functionality. They are imported in the following manner:

import <library name>

Since typing names of libraries whenever they are required makes the code cumbersome, they are often represented using a short form as:

import <library name> as <short form>

ii. Reading data:

Using the method read.csv, data is read by the program. It is more efficient to store the data in the same directory as the code but it can also be stored in any other place. The path to the file must be mentioned in this case. Basic metadata can be seen by using the method:

<name of data frame>.info()

iii. Checking for missing values

The method used is: <name of data frame>.isnull().any().any()
This allows us to know if there are any null values present in the dataset. If any, these should be removed to ensure that the data we use is clean.

iv. Splitting Train/Test data

Before beginning preprocessing, a test data set is split off. First the date is split into features and response variable, then stratification is done on the response variable, which is very important to do because there are so few fraudulent transactions, that is, only 0.17%.

v. Exploratory data analysis

The exploratory data analysis includes only the training set, to leave the test unknown. Analysis is done to check the skewness of data and see the distribution of ‘Time’ and ‘Amount’ variables.

a. Standardization of data

The data is then normalized, to remove skewness of the data using Box-Cox transform. After normalization, data is again visualized to check for the skewness.

b. PCA analysis

The dimensionality of the dataset has been reduced by using Principal Component Analysis (PCA). Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time'

contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependent cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

For the features V1-V28, mean, standard deviation, variance, skewness, kurtosis, medians and inter-quartile ranges are plotted. This allows visualization of the given variables in an effective manner.

2. Feature Selection

Features are selected using Filter methods, using statistics to find the mutual dependance between two variables. Mutual information of 0 indicates no dependence, and higher values indicate higher dependence.

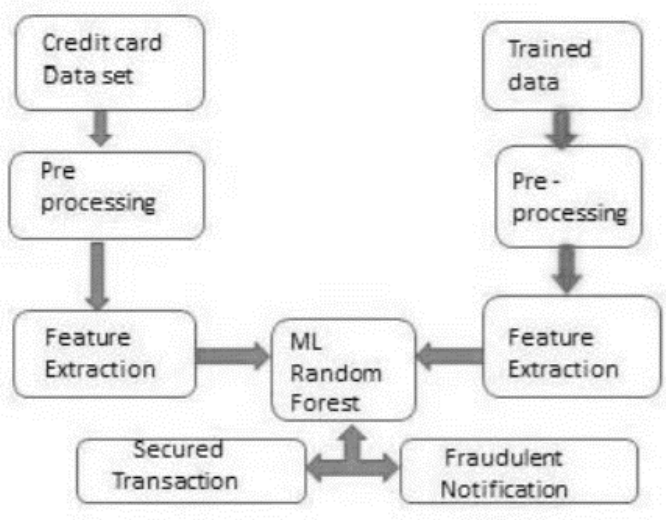
3. Training Models

Models are trained using the training data set and then compared using MCC as a metric. MCC produces a more informative and truthful score in evaluating binary classifications than accuracy and F_1 score. It is a more reliable statistical rate which produces a high score only if the prediction obtained good results in all of the four confusion matrix categories (true positives, false negatives, true negatives, and false positives), proportionally both to the size of positive elements and the size of negative elements in the dataset.

4. Graphic User Interface

Implementation of the system created, by normal people requires a user interface where they can easily input values and find out whether or not the transaction and fraudulent. A web-app serves this purpose, making it easy for people to make use of the model.

Proposed model



Results and Discussion

1. Data Preprocessing

i. Import Libraries

```
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

# Pandas options
pd.set_option('display.max_colwidth', 1000, 'display.max_rows', None, 'display.max_columns', None)

# Plotting options
%matplotlib inline
mpl.style.use('ggplot')
sns.set(style='whitegrid')
```

ii. Reading Data

```
transactions = pd.read_csv('creditcard.csv\creditcard.csv')
```

```
transactions.shape
```

```
(284807, 31)
```

iii. Checking for missing values

```
transactions.isnull().any().any()
```

```
False
```

```
transactions['Class'].value_counts()
```

```
0    284315
1       492
Name: Class, dtype: int64
```

```
transactions['Class'].value_counts(normalize=True)
```

```
0    0.998273
1    0.001727
Name: Class, dtype: float64
```

Only 0.17% (492 out of 284,807) transactions are fraudulent.

iv. Splitting Train/Testing data

```
X = transactions.drop(labels='Class', axis=1) # Features
y = transactions.loc[:,'Class']               # Response
del transactions                              # Delete the original data
```

We'll use a test size of 20%. We also stratify the split on the response variable, which is very important to do because there are so few fraudulent transactions.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1, stratify=y)
del X, y
```

```
X_train.shape
```

```
(1642, 30)
```

```
X_test.shape
```

```
(280607, 30)
```

```
# Prevent view warnings
X_train.is_copy = False
X_test.is_copy = False
```

v. Exploratory Data analysis

On exploration of Time variable, it was found that the dataset consists of transaction over a two-day time period, which can be seen in the histogram plotted below. (x axis: time of transaction, y axis: number of transactions)

Variable: Time ¶

```
X_train['Time'].describe()
```

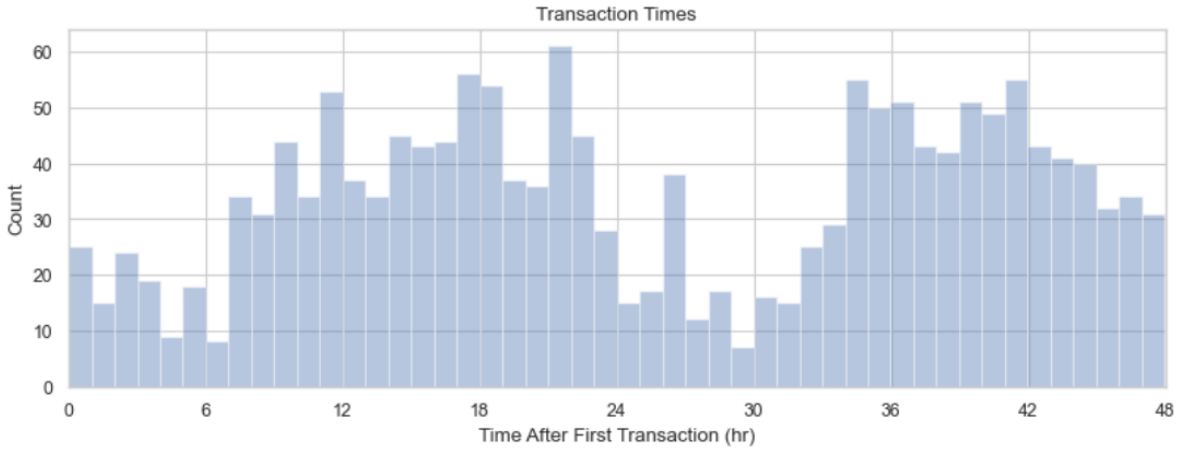
```
count      1642.000000
mean      92433.306943
std       48242.962200
min         130.000000
25%       52914.500000
50%       84956.000000
75%      137195.500000
max      172751.000000
Name: Time, dtype: float64
```

The time of last transaction in days:

```
X_train['Time'].max() / 24
```

```
1.9994328703703703
```

Conclusion: Transactions occur over an almost two day period.



Histogram shows two observable lulls, around the dawn hours of each day

On exploration of variable ‘Amount’, the data was found to be highly right skewed.

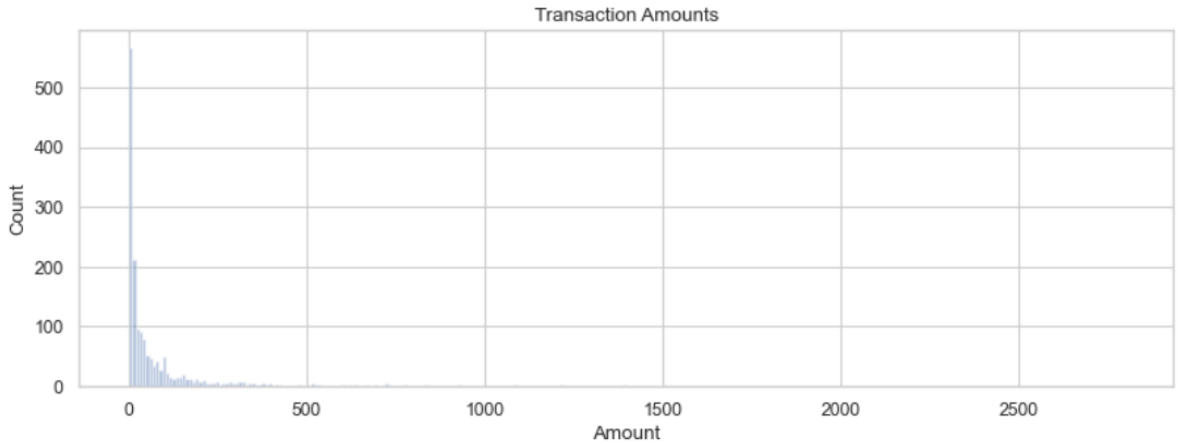
Variable: Amount

```
X_train['Amount'].describe()
```

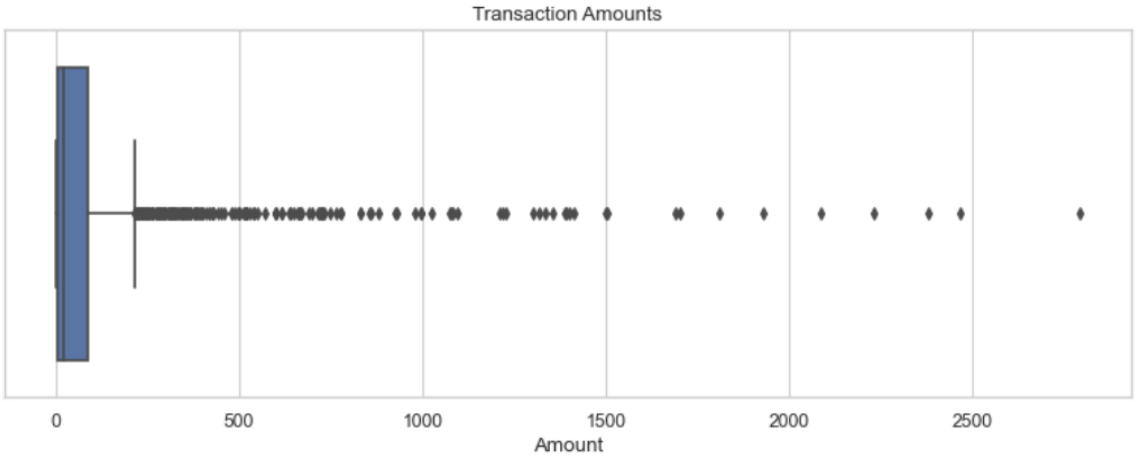
```
count    1642.000000
mean      101.257400
std       241.643296
min         0.000000
25%         3.790000
50%        22.010000
75%        88.900000
max      2793.600000
Name: Amount, dtype: float64
```

```
plt.figure(figsize=(12,4), dpi=80)
sns.distplot(X_train['Amount'], bins=300, kde=False)
plt.ylabel('Count')
plt.title('Transaction Amounts')
```

```
Text(0.5, 1.0, 'Transaction Amounts')
```



In the histogram, some data values are seen which differ greatly from the values. To visualize these better, a boxplot is plotted:



With the right skewness calculated to 5.364167630279151, the data was transformed to reduce the skewness and normalize the distribution.

a. Standardization

The variable amount underwent Box Cox transformation to reduce the skewness.

```
X_train.loc[:,'Amount'] = X_train['Amount'] + 1e-9 # Shift all amounts by 1e-9
```

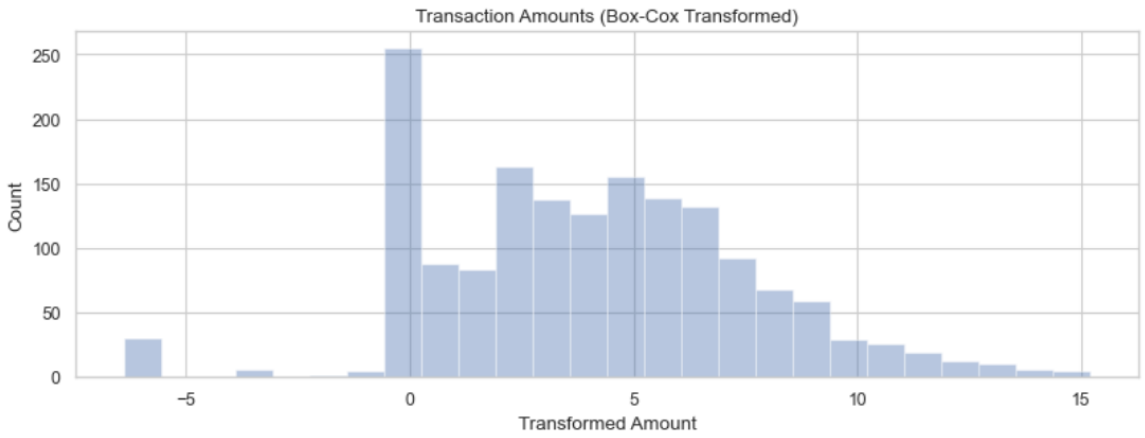
```
X_train.loc[:,'Amount'], maxlog, (min_ci, max_ci) = sp.stats.boxcox(X_train['Amount'], alpha=0.01)
maxlog
```

```
0.14949765833649528
```

```
(min_ci, max_ci)
```

```
(0.13516924575449346, 0.16475064326517136)
```

The histogram of the modified values when plotted can be seen as:



The skewness of the transformed data is 0.03461653644260046. While 0 skewness is Considered to be ideal, it can be agreed that the present value is very less.

b. PCA Analysis

```
pca_vars = ['V%i' % k for k in range(1,29)]
```

Full table of descriptive stats:

```
X_train[pca_vars].describe()
```

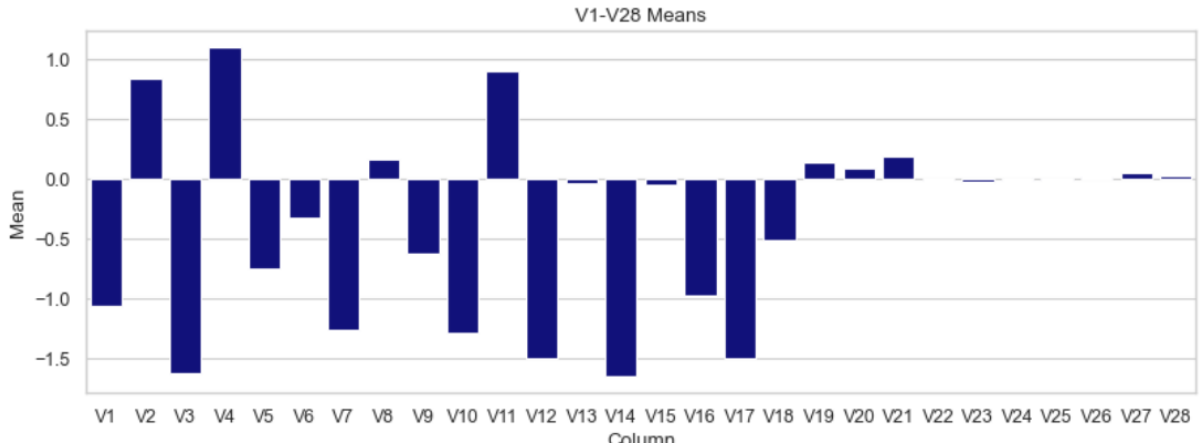
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V
count	1642.000000	1642.000000	1642.000000	1642.000000	1642.000000	1642.000000	1642.000000	1642.000000	1642.000000	1642.000000	1642.000000	1(
mean	-1.061141	0.835007	-1.621241	1.098603	-0.748050	-0.327042	-1.258980	0.161707	-0.616610	-1.280203	0.899395	-1
std	4.112756	2.842036	4.684822	2.674662	3.079729	1.570857	4.200748	3.540338	1.840953	3.454796	2.263710	3.
min	-30.552380	-12.040133	-31.103685	-3.566075	-22.105532	-6.406267	-43.557242	-41.044261	-13.434066	-24.588262	-2.697275	-1
25%	-1.374446	-0.431227	-1.853534	-0.593051	-1.071590	-1.020999	-1.043627	-0.197431	-1.235018	-1.122900	-0.529750	-1
50%	-0.342491	0.318434	-0.310425	0.412981	-0.227877	-0.421628	-0.131725	0.064905	-0.304473	-0.262418	0.340180	-0
75%	1.204791	1.288695	0.831732	2.074103	0.525445	0.234016	0.482510	0.495366	0.446712	0.284318	1.441860	0.
max	2.342858	22.057729	4.017561	11.927512	11.095089	6.933729	8.866539	20.007208	7.173635	11.304842	12.018913	4.

The rows seen in the above table, namely mean, standard deviation, variance and inter-quartile ranges were plotted to observe patterns in the data”

Mean:

```
plt.figure(figsize=(12,4), dpi=80)
sns.barplot(x=pca_vars, y=X_train[pca_vars].mean(), color='darkblue')
plt.xlabel('Column')
plt.ylabel('Mean')
plt.title('V1-V28 Means')
```

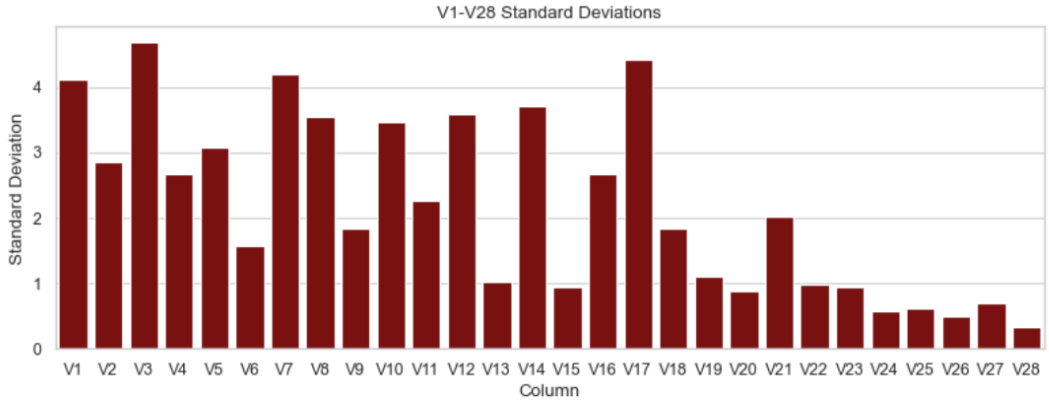
```
Text(0.5, 1.0, 'V1-V28 Means')
```



All of V1-V28 have approximately zero mean. Now plotting standard deviation:

```
plt.figure(figsize=(12,4), dpi=80)
sns.barplot(x=pca_vars, y=X_train[pca_vars].std(), color='darkred')
plt.xlabel('Column')
plt.ylabel('Standard Deviation')
plt.title('V1-V28 Standard Deviations')
```

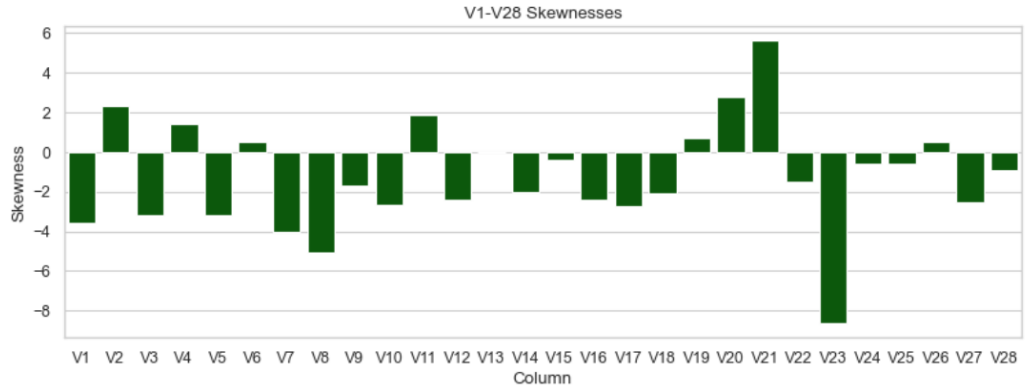
```
Text(0.5, 1.0, 'V1-V28 Standard Deviations')
```



Plotting the skewness next:

```
plt.figure(figsize=(12,4), dpi=80)
sns.barplot(x=pca_vars, y=X_train[pca_vars].skew(), color='darkgreen')
plt.xlabel('Column')
plt.ylabel('Skewness')
plt.title('V1-V28 Skewnesses')
```

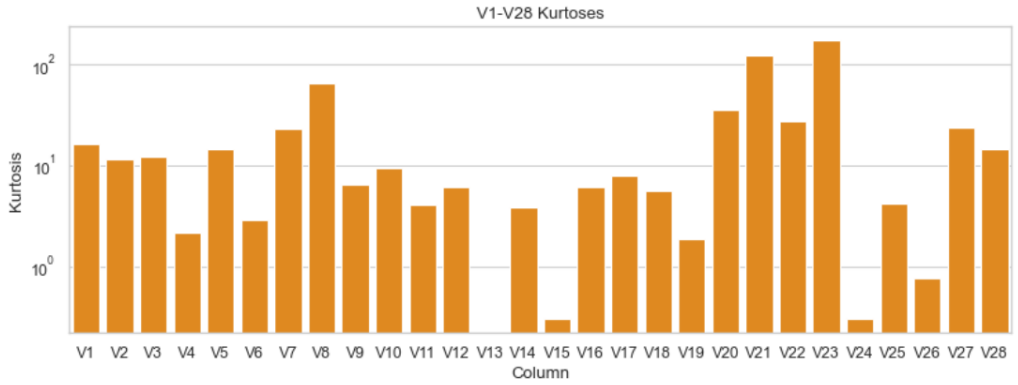
```
Text(0.5, 1.0, 'V1-V28 Skewnesses')
```



On further study of variables with high skewness, the readings were not clear in the boxplots, which means high kurtosis. This motivates us to plot the kurtoses of the PCA variables. The kurtosis method employed in pandas is Fisher's definition, for which the standard normal distribution has kurtosis 0.

```
plt.figure(figsize=(12,4), dpi=80)
plt.yscale('log')
sns.barplot(x=pca_vars, y=X_train[pca_vars].kurtosis(), color='darkorange')
plt.xlabel('Column')
plt.ylabel('Kurtosis')
plt.title('V1-V28 Kurtoses')
```

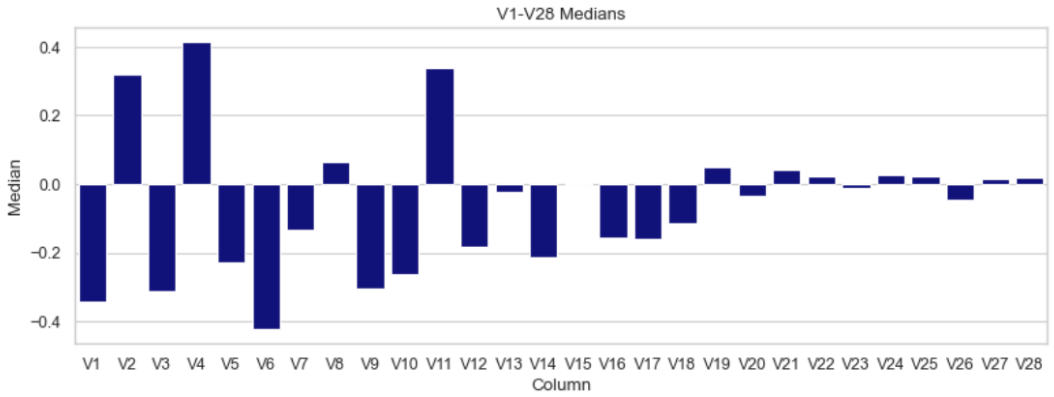
```
Text(0.5, 1.0, 'V1-V28 Kurtoses')
```



The large numbers of outliers in V1-V28 motivates us to consider robust descriptive statistics, namely the medians and the inter-quartile ranges:

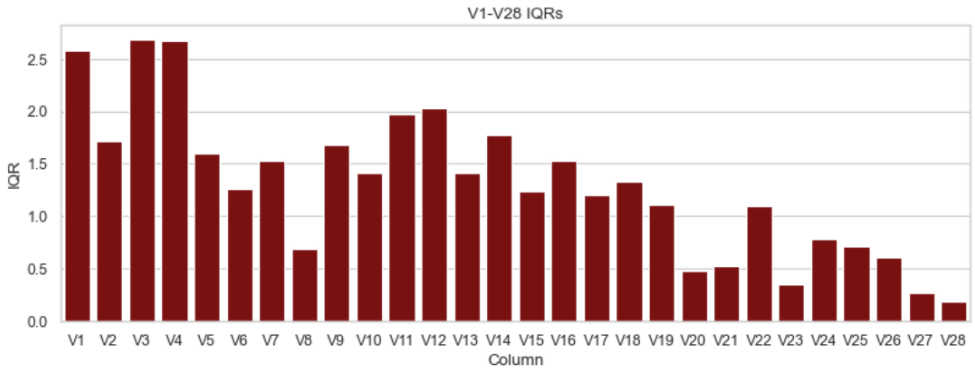
```
plt.figure(figsize=(12,4), dpi=80)
sns.barplot(x=pca_vars, y=X_train[pca_vars].median(), color='darkblue')
plt.xlabel('Column')
plt.ylabel('Median')
plt.title('V1-V28 Medians')
```

```
Text(0.5, 1.0, 'V1-V28 Medians')
```




```
plt.figure(figsize=(12,4), dpi=80)
sns.barplot(x=pca_vars, y=X_train[pca_vars].quantile(0.75) - X_train[pca_vars].quantile(0.25), color='darkred')
plt.xlabel('Column')
plt.ylabel('IQR')
plt.title('V1-V28 IQRs')
```

```
Text(0.5, 1.0, 'V1-V28 IQRs')
```



2. Feature Selection

```
mutual_infos.sort_values(ascending=False)
```

```
V14       0.394476
V17       0.370166
V10       0.361705
V12       0.334941
V11       0.321570
V4        0.301208
V3        0.282378
V16       0.269494
V7        0.244984
V2        0.216069
V9        0.199601
V27       0.186795
V21       0.184152
V18       0.175403
V1        0.157867
V6        0.154948
V28       0.140208
V5        0.129992
Amount    0.126033
Time      0.117537
V8        0.115669
V19       0.096085
V20       0.092927
V23       0.065111
V24       0.039981
V15       0.024805
V25       0.020511
V22       0.019040
V26       0.015325
V13       0.008094
dtype: float64
```

The five most correlated variables with `Class` are, in decreasing order, V17, V14, V10, V12, and V11.

3. Training Models

The Logistic Regression and Support Vector classifier:

The class SGDClassifier implements multiple linear classifiers with SGD training, which makes learning much faster on large datasets. The model is implemented as a machine learning pipeline that includes StandardScaler for data standardization (rescaling each variable to zero mean and unit variance).

A grid search was conducted over several hyperparameter choices. The grid search, implemented by GridSearchCV, uses StratifiedKFold with 5 folds for the train/validation

splits. Then, the MCC was used as a scoring parameter -- random guessing has a score of 0, and a perfect predictor has a score of 1.

Perform the grid search:

```
import warnings
with warnings.catch_warnings(): # Suppress warnings from the matthews_corrcoef function
    warnings.simplefilter("ignore")
    grid_sgd.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 80 candidates, totalling 400 fits
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   21.3s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 400 out of 400 | elapsed:  2.3min finished
```

Mean cross-validated MCC score of the best estimator found:

```
grid_sgd.best_score_
```

```
0.8075666039570759
```

For the random forest model, it takes much longer to train on this fairly large dataset, so a hyperparameter grid search is not actually done, only the number of estimators is specified. The grid search can be implemented in case different hyperparameter values are tried out in the future.

```
grid_rf.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of   5 | elapsed:    0.7s remaining:    1.1s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    0.7s finished
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('model',
                                        RandomForestClassifier(n_jobs=-1,
                                                               random_state=1))]),
             n_jobs=-1, param_grid={'model__n_estimators': [75]},
             scoring=make_scorer(matthews_corrcoef), verbose=1)
```

```
grid_rf.best_score_
```

```
0.8567203839327452
```

The random forest performed much better than the linear SVC.

The final performance report of random forest is:

```
classification_eval(grid_rf, X_test, y_test)
```

```
CONFUSION MATRIX
[[56854    10]
 [   15    83]]

CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0    0.99974   0.99982   0.99978     56864
           1    0.89247   0.84694   0.86911        98

avg / total    0.99955   0.99956   0.99956     56962

SCALAR METRICS
          MCC = 0.86919
        AUPRC = 0.85098
        AUROC = 0.95924
Cohen's kappa = 0.86889
     Accuracy = 0.99956
```

4. Graphic User Interface

The basic code for GUI is as follows:
home.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Credit Card Fraud Detection Web App</title>
    <!--
 <link rel="stylesheet" type="text/css" href="../static/css/styles.css"> -->
    <link
      href="https://fonts.googleapis.com/css2?family=Quicksand:wght@500&display
=swap"
      rel="stylesheet"
    />
    <link
      rel="stylesheet"
      type="text/css"
      href="{{ url_for('static', filename='styles.css') }}"
    />
  </head>
  <body>
    <header>
```

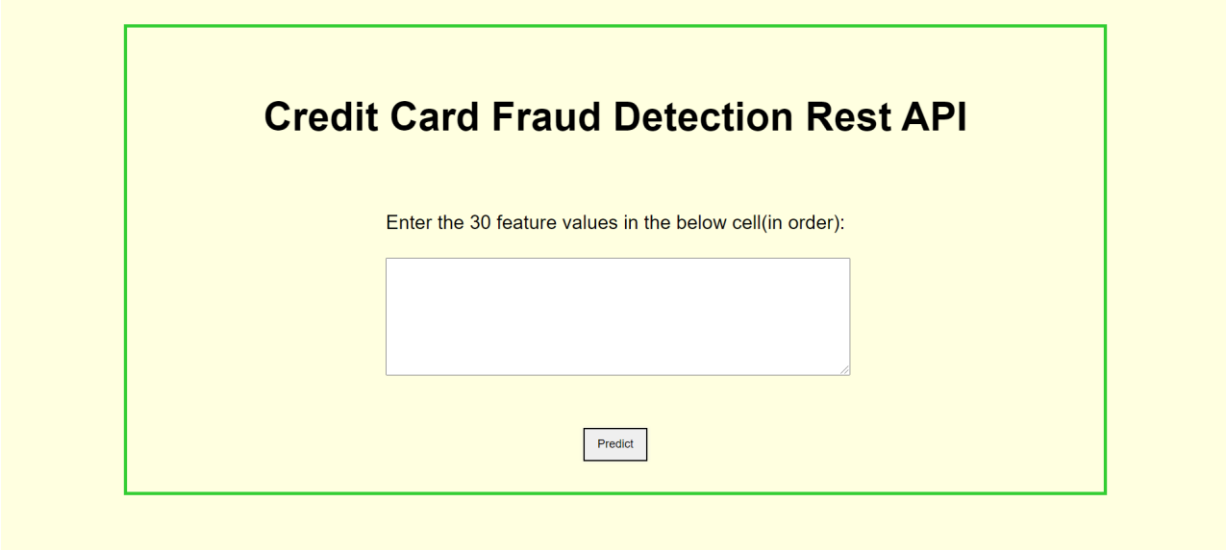
```
      <div class="container">
        <h1>
          <p style="text-align: center">Credit Card Fraud Detection</p>
        </h1>
      </div>
    </header>

    <div class="ml-container">
      <p style="text-align: center">
        Enter the 30 feature values in the below cell(in order):
      </p>

      <form action="{{ url_for('predict')}}" method="POST">
        <!-- <input type="text" name="comment"/> -->
        <div class="justify">
          <textarea name="message" rows="8" cols="50"></textarea>
        </div>
        <br/>

        <input type="submit" class="btn-info" value="Predict" />
      </form>
    </div>
  </body>
</html>
```

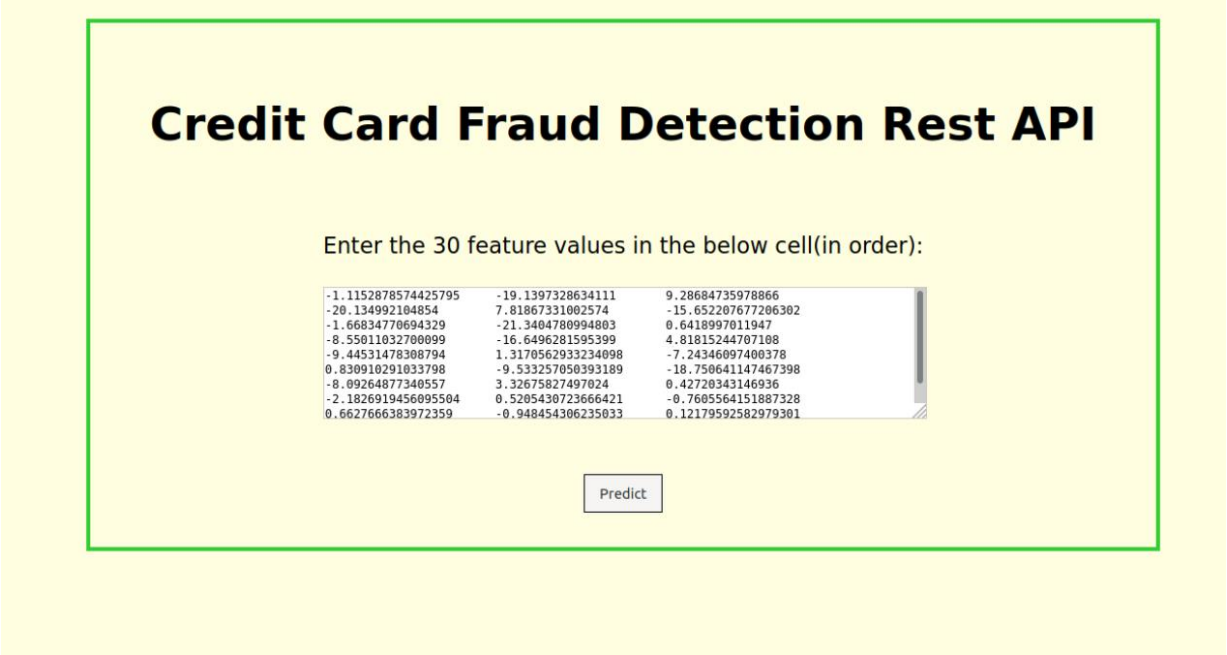
result.html

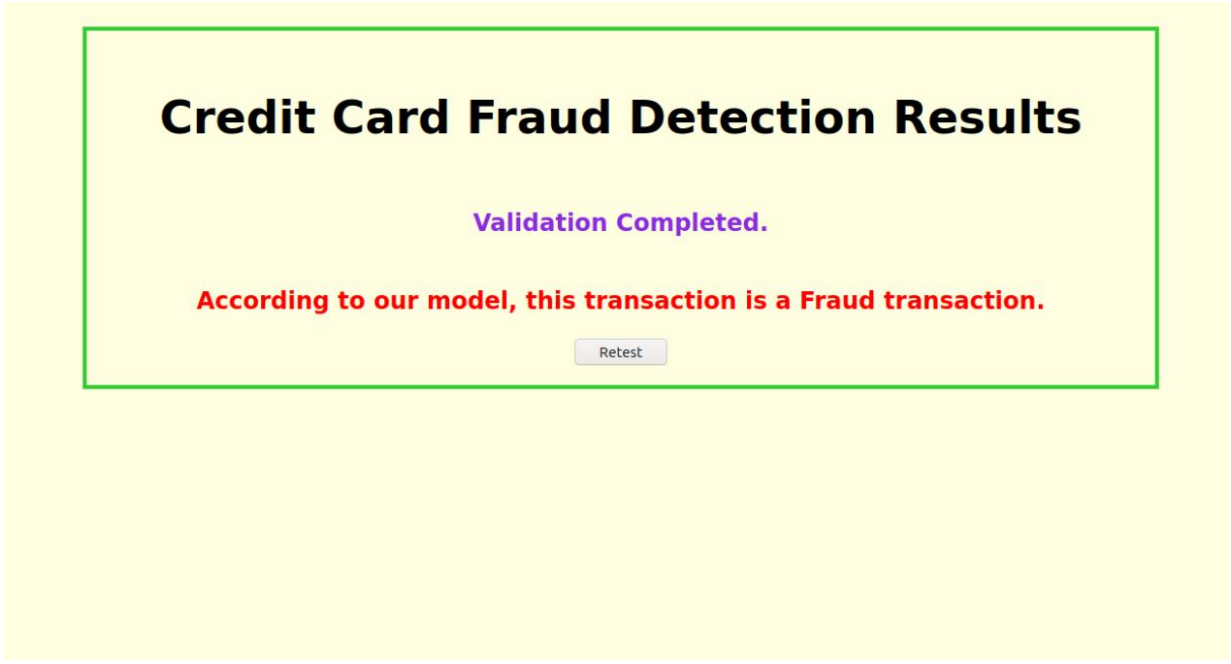
```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <link
      href="https://fonts.googleapis.com/css2?family=Quicksand:wght@500&display
=swap"
      rel="stylesheet"
    />
    <link
      rel="stylesheet"
      type="text/css"
      href="{{ url_for('static', filename='styles.css') }}"
    />
  </head>
  <body>
    <header>
```

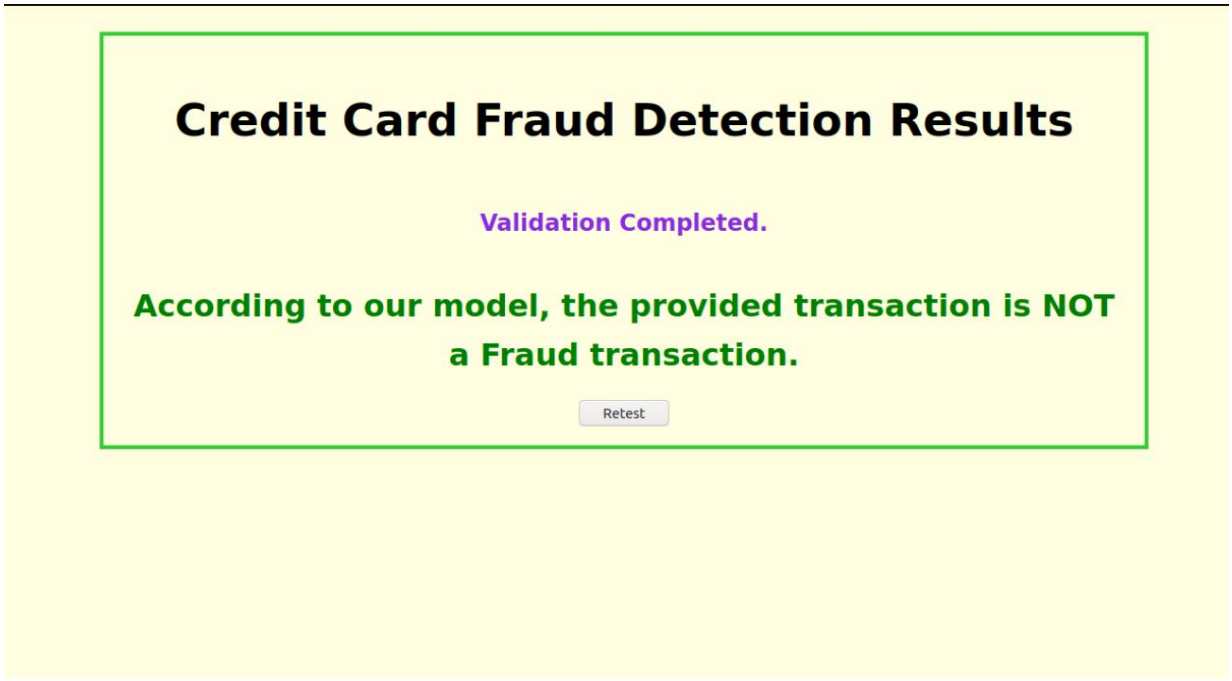
```
      <div class="container">
        <div id="brandname">
          <h1 style="color: black">
          Credit Card Fraud Detection Results
          </h1>
        </div>
        <!-- <h2><p style="text-align: center">Detection</p></h2> -->
      </div>
    </header>
    <h2 style="color: rgb(3, 9, 94);">
      <b>Validation Completed.</b>
    </h2>
    <div class="results">
      {% if prediction == 0 %}
      <h1 style="color: green">
          According to our model, the provided transaction is NOT a Fraud trans
action.
      </h1>
      {% elif prediction == 1 %}
      <h2 style="color: red">
          According to our model, this transaction is a Fraud transaction.
      </h2>
      {% endif %}
    </div>
    <a href="/">
      <button class="btn-info">Retest</button>

    </a>
  </body>
</html>
```

Screenshots:









The website can be visited at: https://ai-project-ccf.herokuapp.com/

Conclusion:

We were able to accurately identify fraudulent credit card transactions using a random forest model. We found that the five variables most correlated with fraud are, in decreasing order, V17, V14, V10, V12, and V11. Only a few pre-processing steps were necessary before constructing predictive models:

- Split the data using a random, stratified train/test split with a test size of 20%
- Box-Cox power transform of the transaction amounts to remove skewness in the data
- Mean and variance standardization of all features as part of a machine learning pipeline

We used the Matthew's correlation coefficient (MCC) to compare the performance of different models. In cross validation, the best linear model (logistic regression, linear SVC) achieved a cross-validated MCC score of 0.807, and a random forest achieved a cross-validated MCC score of 0.856. We therefore chose the random forest as the better model, which obtained an MCC of 0.869 on the test set.

To improve a chosen model, we searched over a grid of hyperparameters and compared performance with cross-validation. It may be possible to improve the random forest model by further tweaking the hyperparameters, given additional time and/or computational power.

References

[1] Chicco, D., Jurman, G. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics* 21, 6 (2020).

[2] F. Ghobadi and M. Rohani, "Cost sensitive modeling of credit card fraud using neural network strategy," 2016 2nd International Conference of Signal Processing and Intelligent Systems (ICSPIS), 2016

[3] M. R. Dileep, A. V. Navaneeth and M. Abhishek, "A Novel Approach for Credit Card Fraud Detection using Decision Tree and Random Forest Algorithms," 2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV), 2021

[4] Maes, Sam & Tuyls, Karl & Vanschoenwinkel, Bram & Manderick, Bernard. (2002). Credit Card Fraud Detection Using Bayesian and Neural Networks.

[5] S. Xuan, G. Liu, Z. Li, L. Zheng, S. Wang and C. Jiang, "Random forest for credit card fraud detection," 2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC), 2018

[6] Shen, Aihua & Tong, Rencheng & Deng, Yaochen. (2007). Application of Classification Models on Credit Card Fraud Detection. International Conference on Service Systems and Service Management.