



CREDIT CARD FRAUD DETECTION

INFO 7390: Advances in Data Science

Under the guidance of
Prof. Srikanth Krishnamurthy

Team:

Sayali Borse

Rishi Rajani

Komal Ambekar

Table of Contents

Contents:

1. Overview
2. Goal
3. Dataset
4. Approach to the problem
5. Architecture
6. Process outline:
 - i. Exploratory Data Analysis: Summarizations and Analysis
 - ii. Under sampling and oversampling the data
 - iii. Study the models like Logistic Regression, Random Forest, SVM and Naïve Bayes
 - iv. Choosing the Best Model
 - v. Uploading the pickled model to AWS S3
 - vi. Dockerize the projects
7. Deployment Architecture
 - i. Get pickled models from AWS S3 bucket
 - ii. User selects the Day and Quarter
 - iii. Dockerize entire application
 - iv. Deploy the application on AWS EC2
8. Implementation details
9. Deployment details
10. Application

CREDIT CARD FRAUD DETECTION

Overview:

Credit card fraud detection relies on the analysis of recorded transactions. Transaction data are mainly composed of several attributes (e.g. credit card identifier, transaction date, recipient, amount of the transaction etc.). Automatic systems are essential since it is not always possible or easy for a human analyst to detect fraudulent patterns in transaction datasets, often characterized by many samples, many dimensions and online updates. Also, the cardholder is not reliable in reporting the theft, loss or fraudulent use of a card which may become a problem later for both parties. We propose to use different machine learning classification algorithms to decipher a fraudulent transaction from a genuine one.

Goals:

To classify the transactions as fraudulent and non-fraudulent transactions from a history of records.

Use various classification models and deploy the model to AWS S3. Build a classification model to classify the problem taking the day and quarter as input.

Get day wise and quarter wise scores.

Dataset:

We are using the Credit Card Fraud Detection dataset from <https://data.world/vlad/credit-card-fraud-detection/discuss/credit-card-fraud-detection/mm4wiyjv>

The data set contains 32 columns:

- Column_a: This column is a serial number for all the transactions in the dataset
- Time: Time depicts the number of seconds from the first transaction in this dataset
- V1-V28: Upon doing some research the owners of the data set did not want to make its content public. Hence, they performed PCA (Principal Component Analysis) on the features and then published the results.
- Amount: Amount of the transaction
- Class: Variable indicating if the transaction is fraudulent or genuine

Approach:

Performed analysis on the data set and noticed that only 492 out 284,807 are true. This is a clear case of uneven distribution of data which will not help us solve our problem. Therefore, we will be performing under sampling and over sampling over our data to ensure we can perform proper classification on both our resulting datasets.

- Perform feature engineering on datasets to select best features for our classification problem
- Apply various classification models (Logistic Regression, Random Forest, Support Vector Machine, Naïve base Classifier)

- Decide the best hyper parameters for the above algorithms
- Perform cross validation and evaluate confusion matrix and dice matrix to determine accuracy of the model
- Present final pipeline

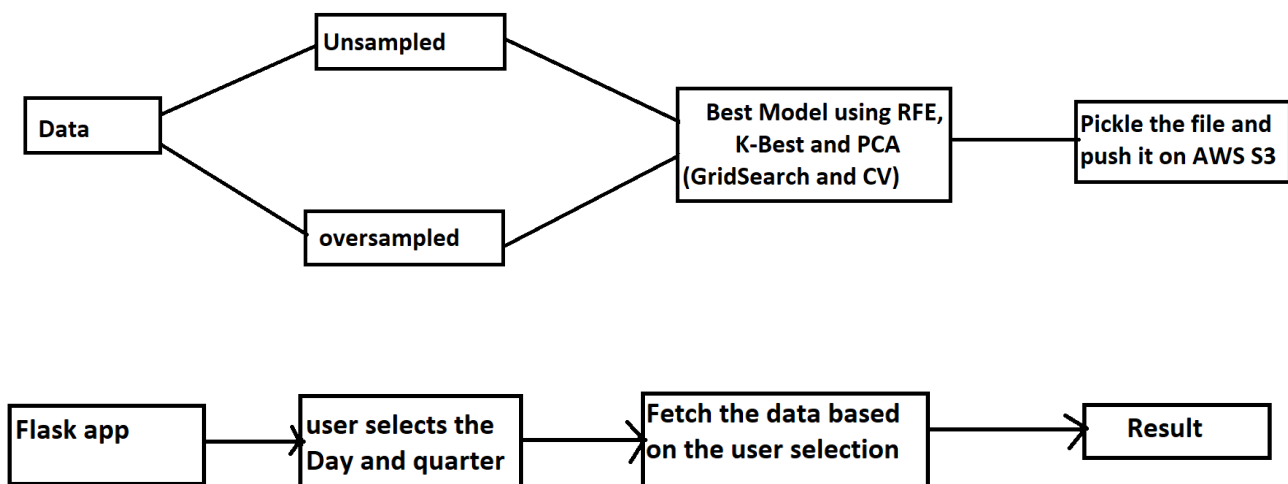
Process outline:

- i. Exploratory Data Analysis: Summarizations and Analysis
- ii. Under sampling and oversampling the data
- iii. Study the models like Logistic Regression, Random Forest, SVM and Naïve Bayes
- iv. Training the model and summarizing
- v. Choosing the Best Model
- vi. Uploading the pickled model to AWS S3
- vii. Dockerize the projects

Deployment Architecture

- i. Get pickled models from AWS S3 bucket
- ii. User selects the Day and Quarter
- iii. Dockerize entire application
- iv. Deploy the application on AWS EC2

Architecture:



Implementation Details:

i. Exploratory Data Analysis: Summarizations and Analysis

The dataset contains only numerical data. Features V1, V2 to 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.

```
In [8]: credit_card_data = pd.read_csv("./CC.csv")
credit_card_data.head()
```

[8]:

	Unnamed: 0	Time	V1	V2	V3	V4	V5	V6	V7	V8	...	V21	V22	V23	V24
0	1	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	...	-0.018307	0.277838	-0.110474	0.066928
1	2	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	...	-0.225775	-0.638672	0.101288	-0.339846
2	3	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	...	0.247998	0.771679	0.909412	-0.689281
3	4	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	...	-0.108300	0.005274	-0.190321	-1.175575
4	5	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	...	-0.009431	0.798278	-0.137458	0.141267

5 rows × 32 columns

There are no null values present in the dataset

- **5. Checking for null values in the dataset**

```
In [13]: credit_card_data.isnull().any().sum()
Out[13]: 0
```

The below graph represents the Class 0 and Class 1 transactions.

It can be observed that the Class 0(Non-fraudulent transactions) are higher in number and Class 1(Fraudulent transactions) are very minor in number.

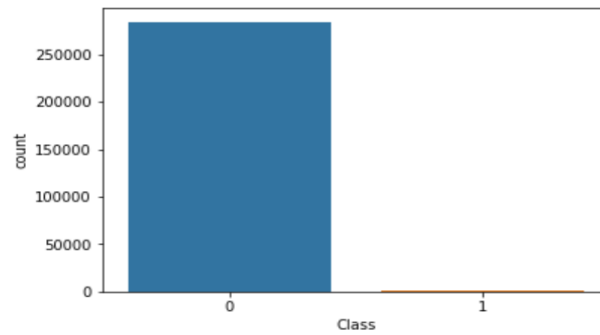
Hence, the dataset is imbalanced and needs to be balanced for correct accuracies.

The two techniques for handling imbalanced data are:

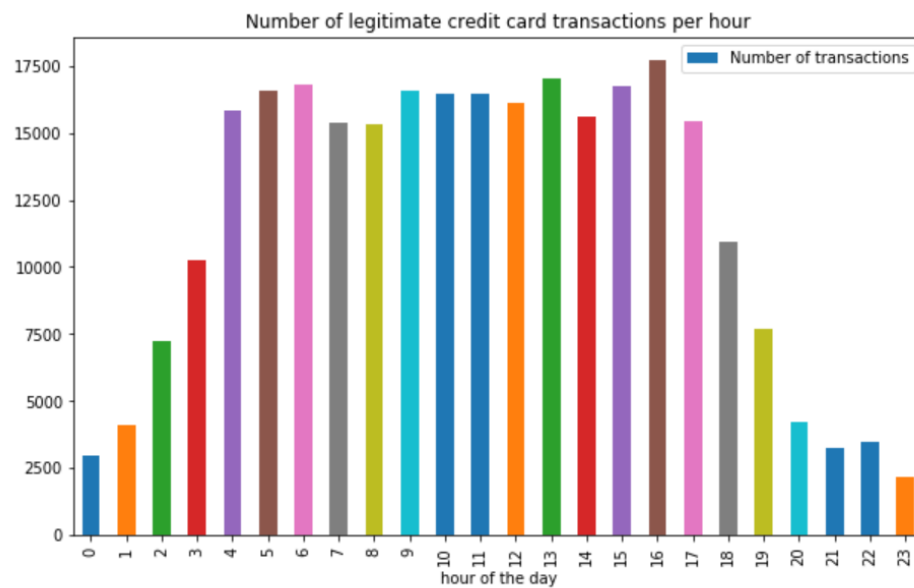
- a) Undersampling and
- b) Oversampling

```
In [28]: sns.countplot("Class",data=credit_card_data)
```

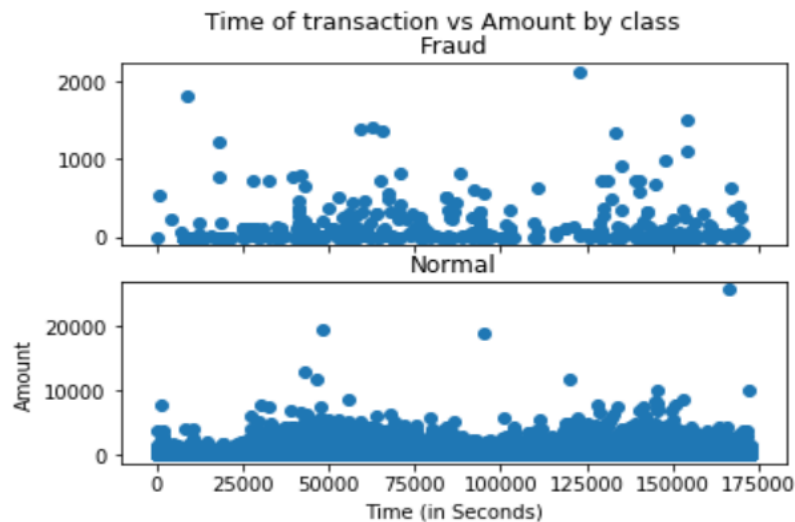
```
Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x195e50012e8>
```



The below histogram shows the number of true credit card transactions for every hour for 24 hours. The transactions are the highest at the 16th hour of the day.



The below scatter plot shows the Time of transaction vs Amount by class for fraud and non-fraud transactions.



ii. Undersampling and Oversampling the data:

The dataset is highly imbalanced with fraud transactions as 1 and non-fraudulent as 0. Out of 284807 records, only 492 of the records are fraud transactions which would give the incorrect model accuracy and hence we need to undersample and oversample the data.

Oversampling and **undersampling** in data analysis are techniques used to adjust the class distribution of a data set (i.e. the ratio between the different classes/categories represented).

Oversampling and undersampling are opposite and roughly equivalent techniques. They both involve using a bias to select more samples from one class than from another.

The usual reason for oversampling is to correct for a bias in the original dataset. One scenario where it is useful is when training a classifier using labelled training data from a biased source, since labelled training data is valuable but often comes from un-representative sources.

UNDERSAMPLING THE DATA

```
In [145]: no_frauds = len(df[df['Class'] == 1])
non_fraud_indices = df[df.Class == 0].index
non_fraud_indices = df[df.Class == 0].index
random_indices = np.random.choice(non_fraud_indices, no_frauds, replace=False)
fraud_indices = df[df.Class == 1].index
under_sample_indices = np.concatenate([fraud_indices, random_indices])
under_sample = df.loc[under_sample_indices]
```

Undersampling the data has some data loss associated with it. For this dataset the data has 984 records which comprises of fraud and non-fraudulent data.

Oversampled data:

```
➤ X = data[['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount']]
Y = data['Class']
```

Oversampling the data

```
➤ from imblearn.over_sampling import SMOTE

os = SMOTE(random_state=0)
columns = X.columns

os_data_X, os_data_y = os.fit_sample(X, Y)
os_data_X = pd.DataFrame(data=os_data_X, columns=columns)
os_data_y = pd.DataFrame(data=os_data_y, columns=["Class"])
# we can Check the numbers of our data
print("length of oversampled data is ", len(os_data_X))
print("Number of normal transaction in oversampled data", len(os_data_y[os_data_y["Class"]==0]))
print("No. of fraud transaction", len(os_data_y[os_data_y["Class"]==1]))
print("Proportion of Normal data in oversampled data is ", len(os_data_y[os_data_y["Class"]==0])/len(os_data_X))
print("Proportion of fraud data in oversampled data is ", len(os_data_y[os_data_y["Class"]==1])/len(os_data_X))
```

```
length of oversampled data is 568630
Number of normal transaction in oversampled data 284315
No. of fraud transaction 284315
Proportion of Normal data in oversampled data is 0.5
```

Oversampling the data using SMOTE:
SMOTE(Synthetic Minority Over-Sampling Technique):

SMOTE synthesizes new minority instances between existing (real) minority instances. Imagine that SMOTE draws lines between existing minority instances.

SMOTE() thinks from the perspective of existing minority instances and synthesizes new instances at some distance from them towards one of their neighbors.

After applying SMOTE to the given data, the number of records increase to 568630.

iii. Studied the models like Logistic Regression, Random Forest, SVM and Naïve Bayes:

- Performed RFE, K-Best and PCA for Feature selection on Logistic regression, Random Forest, SVM and Naïve Bayes.
- This is done on both under-sampled and over-sampled data.
- Performed Grid Search and Cross Validation on all the models.

iv. Choosing the Best Model:

The above step resulted into Random Forest as the best model using K-Best features with an accuracy of 95.3%

v. Uploading the pickled model to AWS S3

After getting the best model as Random Forest we pickle it and upload it to AWS S3 which is then fetched by the Flask app in order to run the model for Classification.

vi. Dockerize the flask

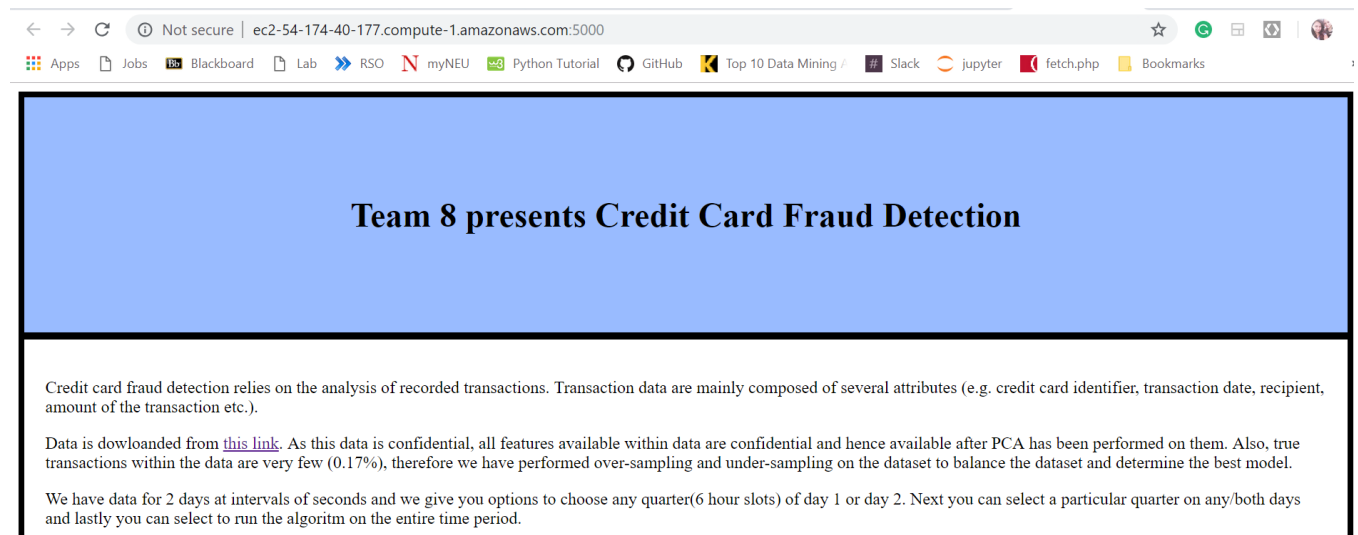
Deployment Architecture

- i. Get pickled models from AWS S3 bucket
Once the pickled model is uploaded to S3 it is then fetched by the flask app for further classification
- ii. User selects the Day and Quarter
On the app for Credit card fraud detection the user selects Day and the quarter of the day. One day has four quarters.
- iii. Dockerize entire application
- iv. Deploy the application on AWS EC2
The final step is to deploy the application on AWS EC2 instance.

Deployment details:

1. Language: Python, HTML, CSS
2. Container: Docker
3. Cloud technologies: AWS S3 and EC2
4. Tools: Jupyter

Application:



← → ↻ Not secure | ec2-54-174-40-177.compute-1.amazonaws.com:5000 ☆ 📄 🔄 🌐

📱 Apps 📁 Jobs 🖨️ Blackboard 📄 Lab ➡️ RSO 📄 myNEU 📄 Python Tutorial 📄 GitHub 📄 Top 10 Data Mining / 📄 Slack 📄 jupyter 📄 fetch.php 📄 Bookmarks

Team 8 presents Credit Card Fraud Detection

Credit card fraud detection relies on the analysis of recorded transactions. Transaction data are mainly composed of several attributes (e.g. credit card identifier, transaction date, recipient, amount of the transaction etc.).

Data is downloaded from [this link](#). As this data is confidential, all features available within data are confidential and hence available after PCA has been performed on them. Also, true transactions within the data are very few (0.17%), therefore we have performed over-sampling and under-sampling on the dataset to balance the dataset and determine the best model.

We have data for 2 days at intervals of seconds and we give you options to choose any quarter(6 hour slots) of day 1 or day 2. Next you can select a particular quarter on any/both days and lastly you can select to run the algorithm on the entire time period.

User selects the Day and quarter:

and lastly you can select to run the algorithm on the entire time period.

Kindly Select any day and any period from the below drop downs.

Day 1 ▾

12:00 - 17:59 ▾

Result

This is the result table with F-1 score, accuracy, actual fradulent transactions, predicted genuine transaction, actual genuine transaction, recall score.

Results							
---------	--	--	--	--	--	--	--

Predicted Fradulent Transactions	Actual Fradulent Fransactions	Predicted Genuine Transation	Actual Genuine Transactions	F1-Score	Recall Score	Accuracy	P-Score
58	71	84	71	0.8992248062015504	0.8169014084507042	0.9084507042253521	1.0