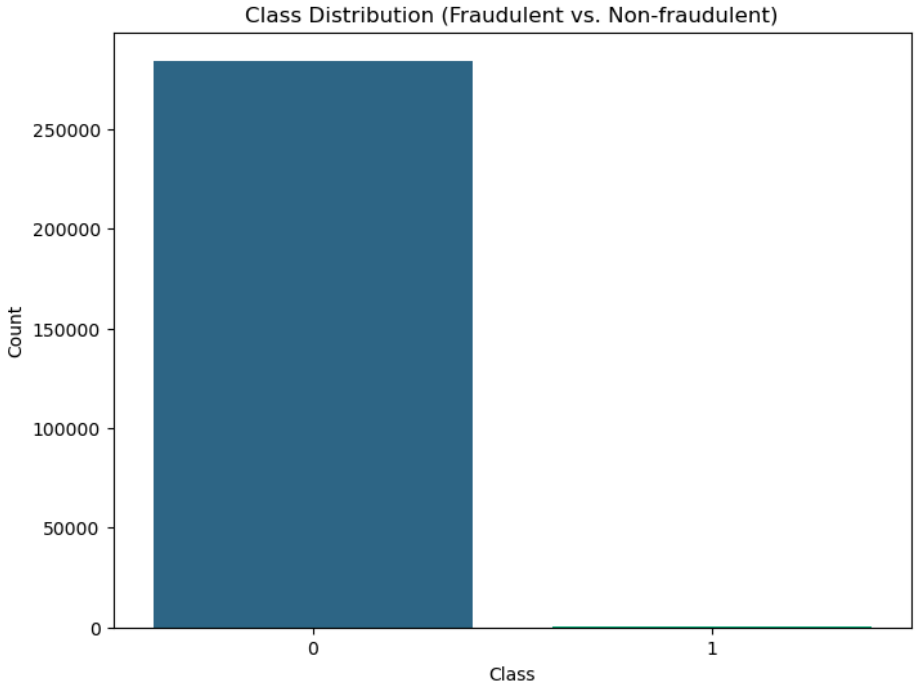
Project Report : Credit Card Fraud Prediction

Ashwin Rajh

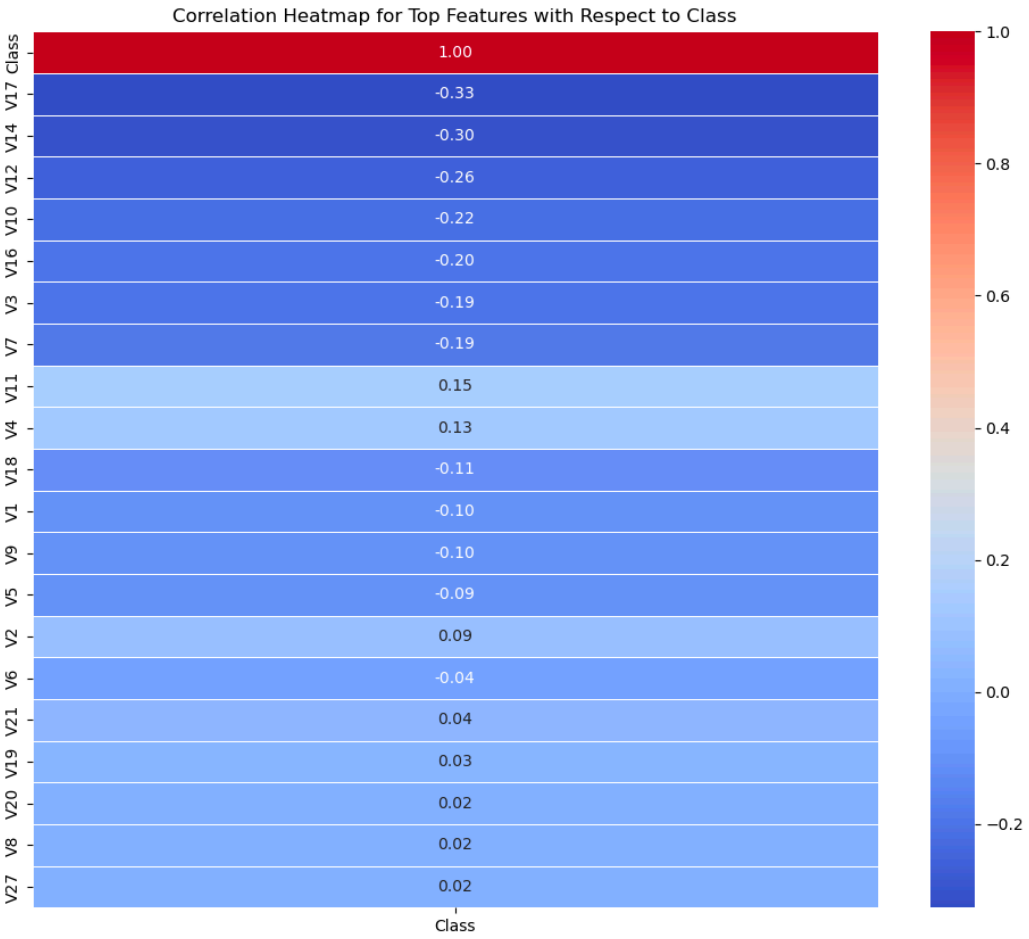
Table Of Contents

Table Of Charts:

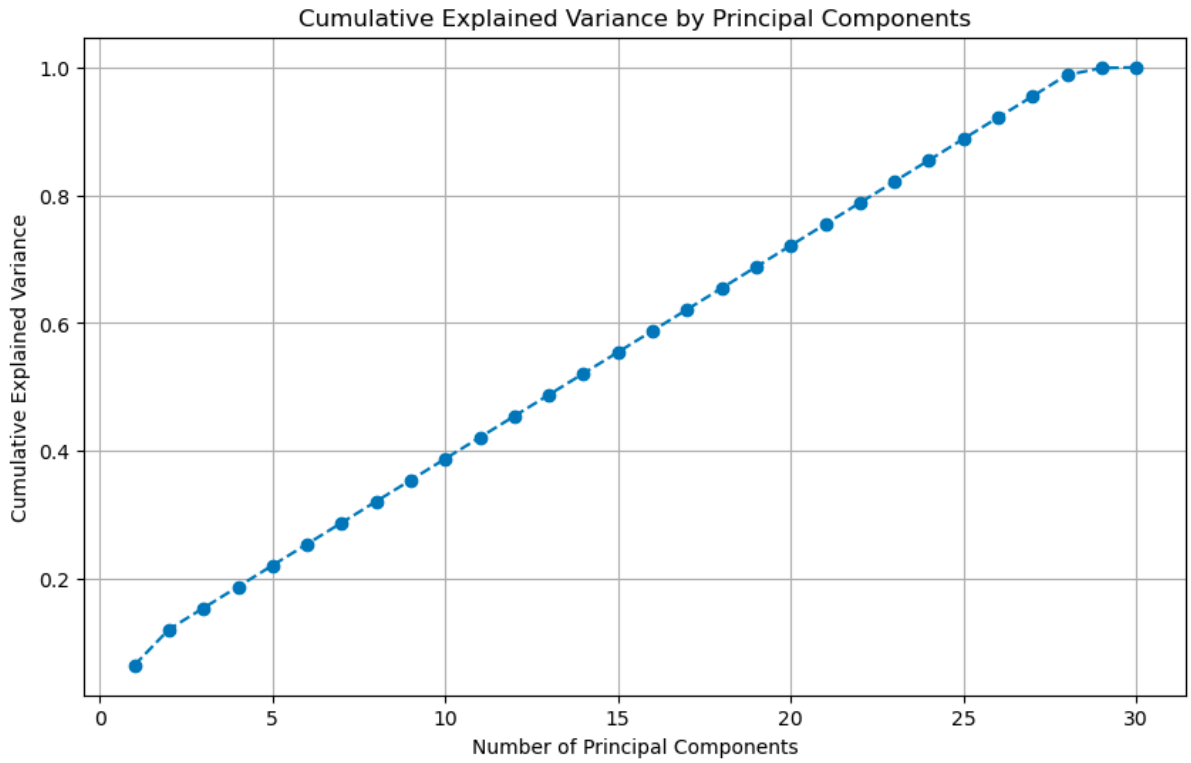
Class Distribution (Fraudulent vs. Non-fraudulent)



Correlation Heatmap for Top Features with Respect to Class:



Cumulative Explained Variance by Principal Components



Abstract:

Robust detection mechanisms are of the need of the hour for Credit card frauds to safeguard custom

In this project, we examine a dataset containing credit card transactions from European cardholders in September 2013. The dataset covers two days with a total of 284,807 transactions, where fraudulent transactions make up only 0.172%.

One of our goals is to figure out which factors among the 30 independent features are most crucial for predicting fraudulent transactions. To deal with the challenge of imbalanced data, we explore Principal Component Analysis (PCA) as a tool to simplify the dataset.

We implement three classification models—Logistic Regression, Support Vector Classifier (SVC), and Decision Tree Classifier—and evaluate how well they predict the 'fraudulent' label. Each model is explained in simple terms, highlighting its strengths and suitability for the task.

This project aims to contribute to creating effective credit card fraud detection systems, ensuring the financial safety of cardholders and enhancing the ability of credit card companies to combat fraud.

Introduction:

Credit card fraud poses a major threat to our financial systems. Addressing this issue requires a nuanced approach that identifies the key variables influencing fraudulence and attempting to develop models that minimise false negatives i.e recall as high recall is essential in credit card fraud detection because it represents the ability of the model to correctly identify most of the actual fraud cases. False negatives in this context are cases where the model fails to detect fraudulent transactions. If a fraudulent transaction goes undetected (false negative), it can result in financial losses and damage to the trustworthiness of the credit card system.

Software-libraries used:

Pandas, Numpy, Matplotlib are used for data manipulations and data visualisations..
Scikit-Learn is utilized for implementing and evaluating the performance of classification models such as Logistic Regression, Support Vector Classifier (SVC), and Decision Tree Classifier.

EDA:

As the dataset was clean, no major preprocessing was required. To find the top features , correlation wrt “Class” was done and no. of features was chosen to be 21.

```
In [126]: df.corr()['Class'].abs().sort_values(ascending=False)
Out[126]: Class     1.000000
          V17       0.326481
          V14       0.302544
          V12       0.260593
          V10       0.216883
          V16       0.196539
          V3        0.192961
          V7        0.187257
          V11       0.154876
          V4        0.133447
          V18       0.111485
          V1        0.101347
          V9        0.097733
          V5        0.094974
          V2        0.091289
          V6        0.043643
          V21       0.040413
          V19       0.034783
          V20       0.020090
          V8        0.019875
          V27       0.017580
          Time      0.012323
          V28       0.009536
          V24       0.007221
          Amount    0.005632
          V13       0.004570
          V26       0.004455
          V15       0.004223
          V25       0.003308
          V23       0.002685
          V22       0.000805
          Name: Class, dtype: float64
```

Feature Engineering:

Dimensionality reduction was done through the PCA framework such that the no. of components selected would result in 95% variance. Based on which, we got 27 as the no. of components. Therefore we have reduced the number of features without losing crucial information

Model Selection and Training:

With StratifiedKFold, where folds = 5, all the data points were test data and train data irrespective of the split.In order to correct the unbalanced dataset, class weights were opted and used as hyper parameter. Further with the help of three models: Logistic regression, SVC, Decision Tree Classifier, the data was trained. Other hyperparameters were chosen based on the model.

Algorithm:

Create a StratifiedKFold object (skf) with 5 splits for cross-validation.Specify the hyperparameters and their potential values for tuning logistic regression such as

- penalty: Regularization term ('l1', 'l2', 'elasticnet', ‘none').
- C: Inverse of the regularization strength, with values from 1 to 10.
- l1_ratio: Applicable for elasticnet, with values from 0.001 to 0.9999.
- class_weight: Different class weight configurations for handling imbalanced data.

Set Up the Pipeline:

Create a pipeline (m_pipe) consisting of two steps:

Step 1: Standard Scaling (StandardScaler).

Step 2: Logistic Regression (LogisticRegression), configured with specific parameters (random seed, solver, max iterations, and tolerance).

Set up a RandomizedSearchCV object (rcv) to search the hyperparameter space using randomized search. Fit the model on the data (rcv.fit(X_selected_pca, Y)). The randomized search explores various hyperparameter combinations based on the specified distributions. The model is trained and evaluated using the defined cross-validation strategy (cv=skf). The scoring metric used is recall (scoring=‘recall').

Similarly for LinearSVC, the model name changes to LinearSVC and the hyper parameters:

```
hyper_params = {'svc__penalty':['l1','l2'],
                'svc__loss':['hinge', 'squared_hinge'],
                'svc__C':np.linspace(1,100, 20),
                'svc__class_weight':[{0:1,1:90},{0:1,1:100},{0:1,1:110}]
               }
```

For Decision Tree Model, standardisation is not required, the model name changes, however its hyper parameters are:

```
model = DecisionTreeClassifier(random_state = 1)

hyper_params = {'criterion':['gini', 'entropy'], # 'log_loss'
                'splitter':['best','random'],
                'max_depth':range(2,7)
                }
```

Evaluation Metrics:

Hypertuning was done to ensure to get optimal results. Post which fit was completed. We calculated the "recall" scores of all the three models as shown below:

```
rcv.best_score_ #LinearSVC
```

```
0.7864357864357864
```

```
gcv.best_score_#Decision Tree
```

```
0.745578231292517
```

```
rcv.best_score_#Logistic Regression
```

```
0.8535765821480107
```

Results:

Logistic Regression gave us the highest recall score and the corresponding hyperparameters are as below. Based on the recall score, of 0.8536 means that, on average, the Logistic Regression model correctly identified approximately 85.36% of the actual positive instances during cross-validation.A higher recall score indicates that the model is effective at capturing a significant portion of the positive instances(i.e fraud is happening), which is crucial in scenarios where the cost of missing positive cases (False Negatives) is high.

```
rcv.best_params_#Logistic Regression
```

```
{'LR__penalty': 'l2',
 'LR__l1_ratio': 0.250725,
 'LR__class_weight': {0: 1, 1: 90},
 'LR__C': 7.0}
```

```
rcv.best_score_#Logistic Regression
```

```
0.8535765821480107
```

Conclusion:

Despite a meager 0.172% fraud rate, we successfully identified key variables influencing fraud prediction. By identifying key features and implementing machine learning techniques, we aim to learn to enhance the accuracy and efficiency of credit card fraud detection.