Machine Learning Engineer Nanodegree

Capstone Project

Mustafa Adel
May 2nd, 2019

I. Definition

Project Overview

Context

Credit card plays a very important rule in today's economy. It becomes an unavoidable part of household, business and global activities. Although using credit cards provides enormous benefits when used carefully and responsibly, significant credit and financial damages may be caused by fraudulent activities. Many techniques have been proposed to confront the growth in credit card fraud. However, all of these techniques have the same goal of avoiding the credit card fraud; each one has its own drawbacks, advantages and characteristics. In this project, I'm going to work on a real dataset has been collected in September 2013 to build *classification model (using supervised learning) to predict the fraudulent transaction.*

Acknowledgements

The dataset has been collected and analyzed during a research collaboration of Worldline and the Machine Learning Group (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles) on big data mining and fraud detection. More details on current and past projects on related topics are available on https://www.researchgate.net/project/Fraud-detection-5 and the page of the DefeatFraud project.

- *Check more about the dataset here.*
- *Related research paper here.*

Datasets and Inputs

The datasets contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. 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, we cannot provide the original features and more background information about the data. 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.

Problem Statement

In this project. I'll deal with anonymized and imbalanced dataset. My goal in some steps

1. *Understanding the dataset.*
2. *Dealing with imbalanced dataset.*
3. *Preprocessing.*
4. *Evaluating model.*
5. *Using Evaluation Metrics.*

Solution Statement

In this project, I'll use Correlation matrix to understand the correlation coefficients between features (especially V1 to V28) and target label. And a resampling method to deal with imbalanced dataset to create a 50-50 ratio (fraud-real) transactions. Using SVM classifier to fit the resampled data and predict the labels.

Project Design

1) Understanding our data
   a) Data Exploration
   b) Data Visualization
2) Preprocessing
   a) Data Cleaning
   b) Data normalization
   c) Shuffle and split the data
   d) Evaluating model on imbalanced data
   e) Resampling data
3) Evaluating model
4) Test our model using evaluation metrics

II. Analysis

Data Exploration

Overview

The datasets contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. 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, we cannot provide the original features and more background information about the data. 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.

Loading Data

```
#load the dataset
data = pd.read_csv('creditcard.csv')

#display first 5 transactions
data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...

5 rows × 31 columns

...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

Check Missing Data

```
#Check if there's a missing data at each column
data.isnull().sum()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
```

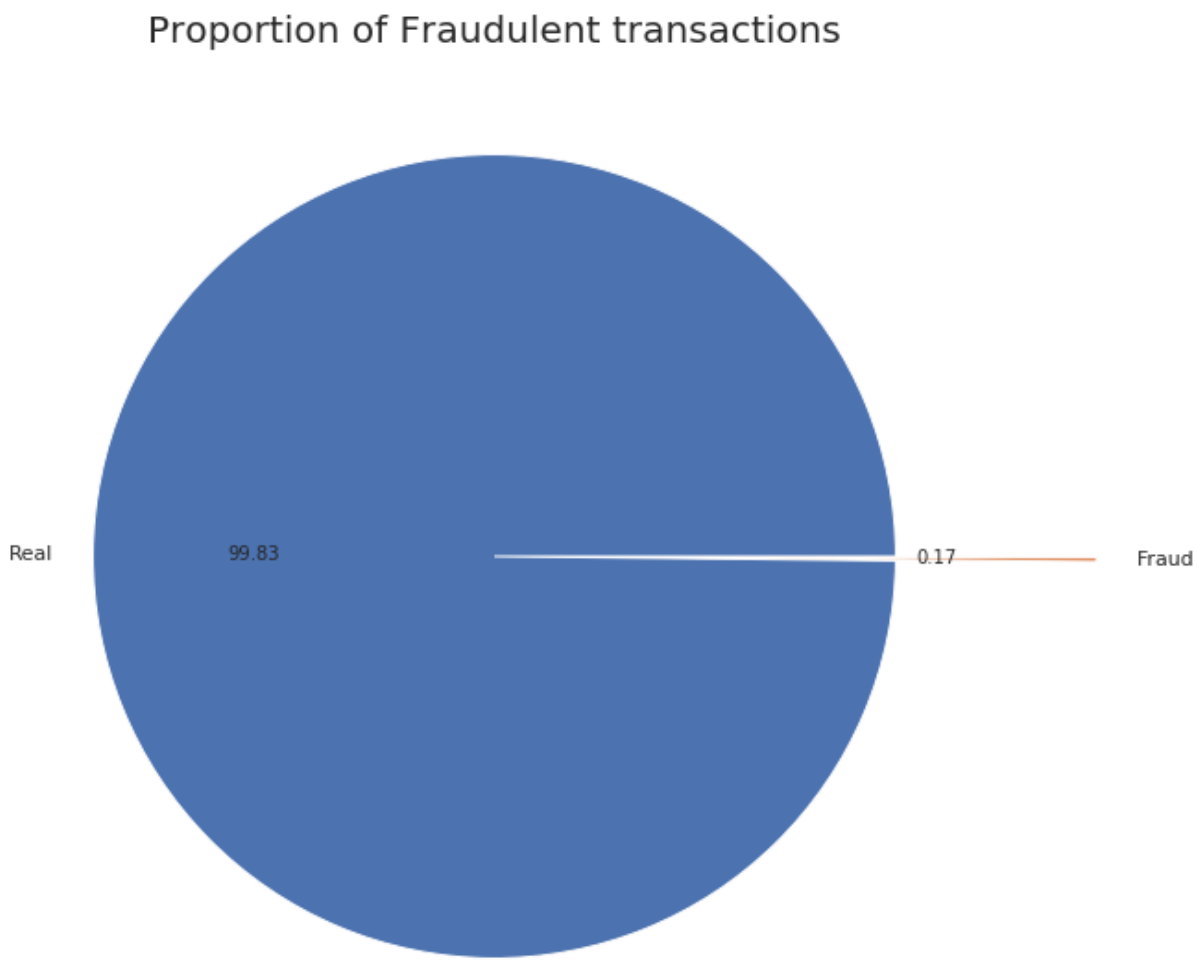
Data formatting

```
#check data formating
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
Time      284807 non-null float64
V1        284807 non-null float64
V2        284807 non-null float64
V3        284807 non-null float64
V4        284807 non-null float64
V5        284807 non-null float64
V6        284807 non-null float64
V7        284807 non-null float64
V8        284807 non-null float64
V9        284807 non-null float64
V10       284807 non-null float64
V11       284807 non-null float64
V12       284807 non-null float64
V13       284807 non-null float64
V14       284807 non-null float64
V15       284807 non-null float64
V16       284807 non-null float64
V17       284807 non-null float64
V18       284807 non-null float64
V19       284807 non-null float64
V20       284807 non-null float64
V21       284807 non-null float64
V22       284807 non-null float64
V23       284807 non-null float64
V24       284807 non-null float64
V25       284807 non-null float64
V26       284807 non-null float64
V27       284807 non-null float64
V28       284807 non-null float64
Amount    284807 non-null float64
Class     284807 non-null int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

Exploratory Visualization

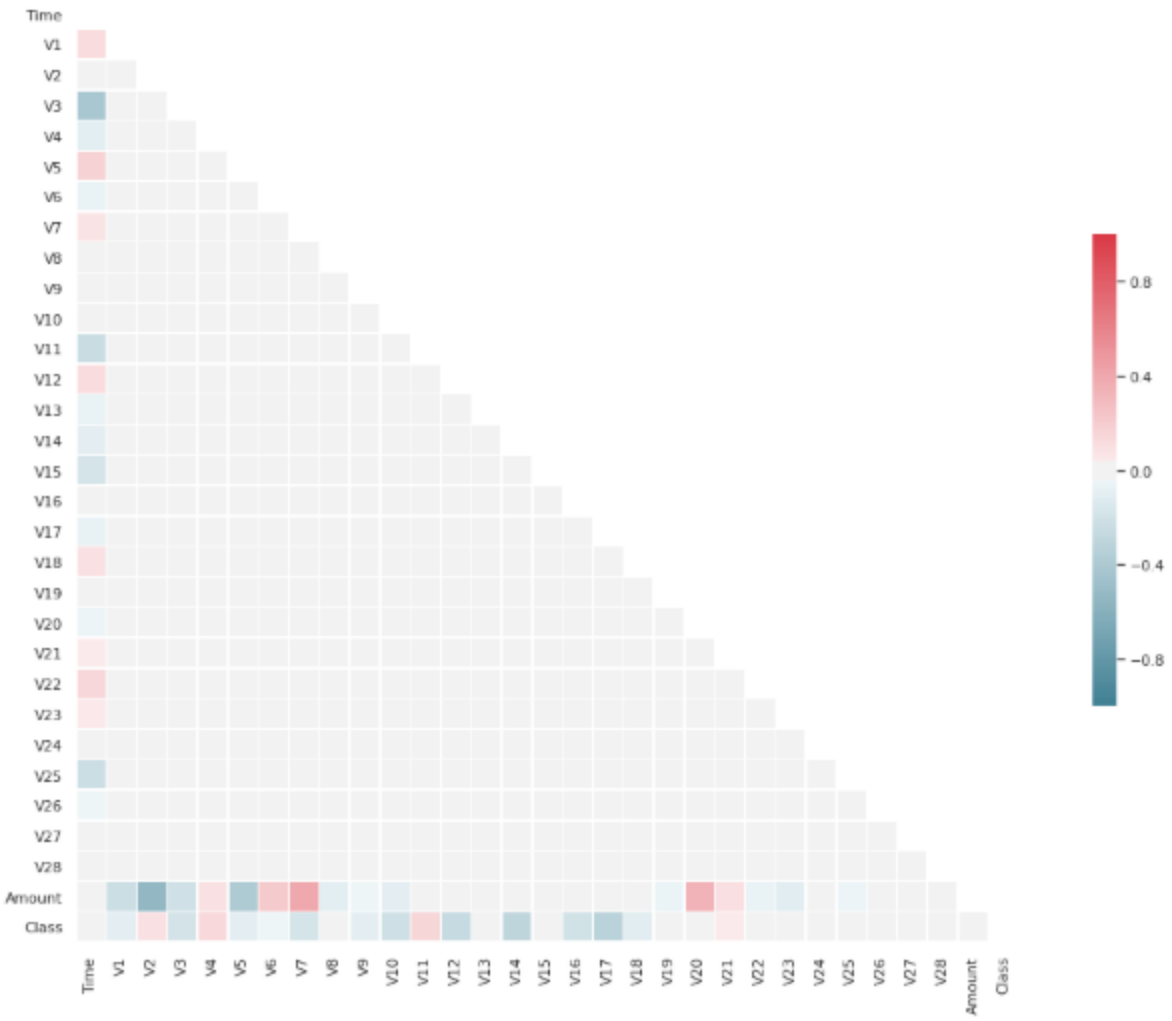
1. Proportion of Fraudulent transaction



Above figure shows that the dataset is totally imbalanced and I'll discuss how to deal with it in preprocessing section.

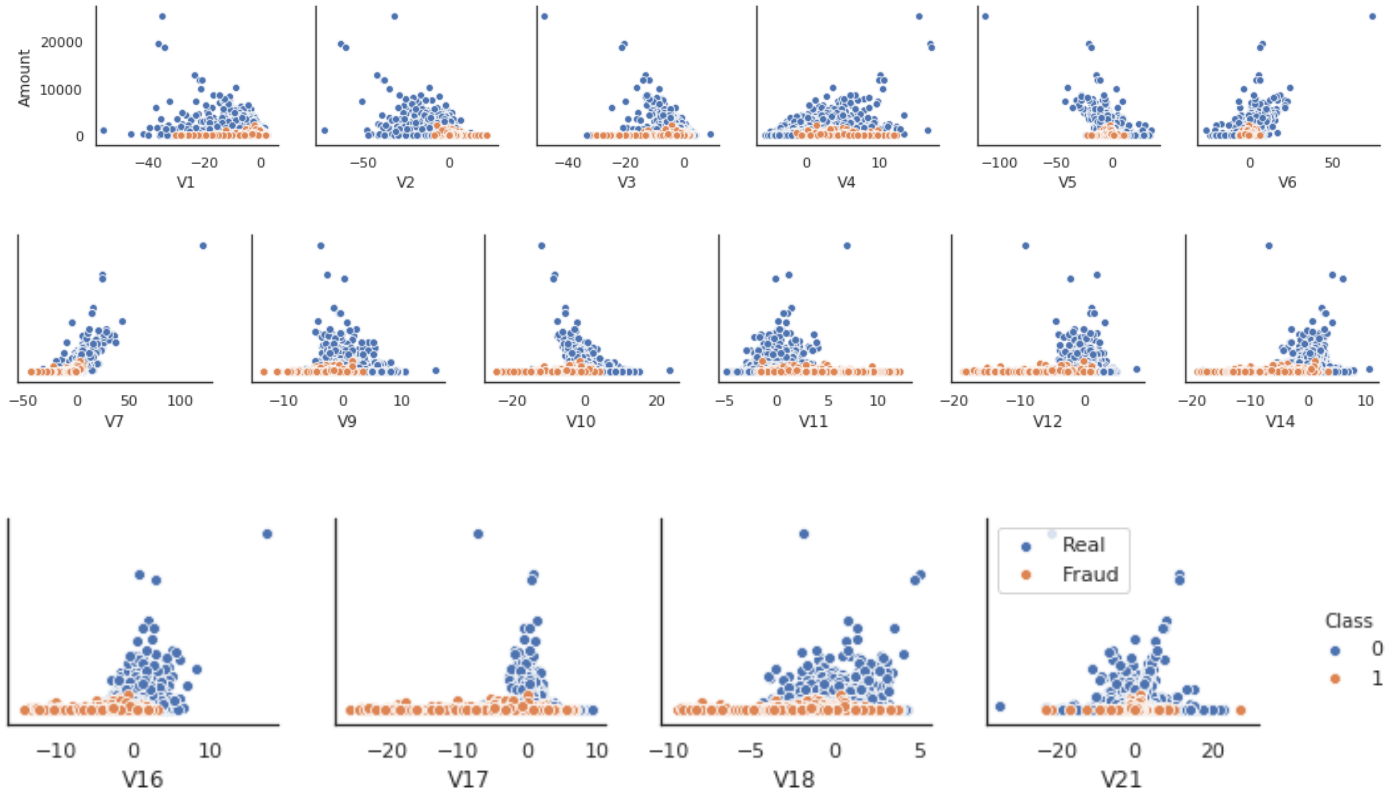
2. Correlation between the features

First let me define the correlation matrix. A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. A correlation matrix is used as a way to summarize data, as an input into a more advanced analysis, and as a diagnostic for advanced analyses. Correlation coefficient = 1 means there's a large positive correlation, -1 means a large negative correlation and 0 means there's no correlation between the 2 features or variables.



And here's a very useful figure that shows the correlation between our target "Class" and features (especially the V1 … V28) and it helps me to clean the data even when It's anonymized data. Time from description of dataset we knew it's not an important feature and the correlation matrix proved it now. And V8, V13, V15, V19, V20, V22 … V28 and amount almost there's no correlation between any one of them and Class but I'll keep "Amount" feature because the correlation between it and other dependent features but it won't affect the result if we dropped it.

3. Plot the relation between the dependent values and amount with hue using Class.



It shows that the fraudulent transaction is only when Amount under the 10000 or a very small Amount but I think it doesn't help but I'll leave the Amount feature.

Algorithms and Techniques

After understanding dataset, we can implement any of supervised learning algorithms such as KNN, SVM, Decision Trees, Naive Bayes, Logistic Regression, etc. Our data is numerical and quite small dataset. So most of these algorithms will work fine with it but I'll choose SVM to implement my model for no reason but I like this algorithm.

- *SVM is easy to implement like the others algorithms too.*
- *It produces a very good results in many fields because of the kernel it uses I think.*

Also we can use neural network in this project but I see we don't have to use it, our project is simple and supervised learning algorithms will produce a very good results. If I used Neural Network in this project, it'll be like I'm eating a chicken breast with a sword and garden fork.

SVM:

*A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (*supervised learning*), the*

algorithm outputs an optimal hyperplane which categorizes new examples. In two dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

Resampling Data Techniques:

Resampling techniques are a set of methods to either repeat sampling from a given sample or population, or a way to estimate the precision of a statistic. Although the method sounds daunting, the math involved is relatively simple and only requires a high school level understanding of algebra.

And check more about resampling techniques here.

Recall and precision:

Precision-Recall is a useful measure of success of prediction when the classes are very imbalanced. In information retrieval, precision is a measure of result relevancy, while recall is a measure of how many truly relevant results are returned.

A system with high recall but low precision returns many results, but most of its predicted labels are incorrect when compared to the training labels. A system with high precision but low recall is just the opposite, returning very few results, but most of its predicted labels are correct when compared to the training labels. An ideal system with high precision and high recall will return many results, with all results labeled correctly.

And here a very good article about Recall and precision score.

Benchmark

My Benchmark model will be a SVM Classifier model without resampling the dataset, work on imbalanced data and here's the results of recall and precision score.

```
from sklearn.metrics import recall_score, precision_score

recall_train = recall_score(y_train, prediction_train)
recall_test = recall_score(y_test, prediction_test)

precision_train = precision_score(y_train, prediction_train)
precision_test = precision_score(y_test, prediction_test)

print("Recall score on training set: {:.4f}".format(recall_train))
print("Recall score on testing set: {:.4f}".format(recall_test))
print("\nprecision score on training set: {:.4f}".format(precision_train))
print("precision score on testing set: {:.4f}".format(precision_test))
```

```
Recall score on training set: 0.8166
Recall score on testing set: 0.6489

precision score on training set: 0.9848
precision score on testing set: 0.9839
```

III. Methodology

Data Preprocessing

Data Cleaning

Cleaning or drop unnecessary features we found in data exploration section

```
data.drop(['Time', 'V8', 'V13', 'V15', 'V19', 'V20',
           'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28'], axis = 1, inplace = True)
data.head()
```

	V1	V2	V3	V4	V5	V6	V7	V9	V10	V11	V12
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.363787	0.090794	-0.551600	-0.617801
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	-0.255425	-0.166974	1.612727	1.065235
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	-1.514654	0.207643	0.624501	0.066084
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	-1.387024	-0.054952	-0.226487	0.178228
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	0.817739	0.753074	-0.822843	0.538196

V14	V16	V17	V18	V21	Amount	Class
-0.311169	-0.470401	0.207971	0.025791	-0.018307	149.62	0
-0.143772	0.463917	-0.114805	-0.183361	-0.225775	2.69	0
-0.165946	-2.890083	1.109969	-0.121359	0.247998	378.66	0
-0.287924	-1.059647	-0.684093	1.965775	-0.108300	123.50	0
-1.119670	-0.451449	-0.237033	-0.038195	-0.009431	69.99	0

Data Normalization

In this section we will normalize the numerical data, Keep in mind that in order to implement a PCA transformation features need to be previously scaled. (In this case, all the V features have been scaled or at least that is what we are assuming the people that develop the dataset did.) So we only using normalization techniques on Amount feature only. We can use any of normalization techniques such as Standard scaling, MinMaxScaler or Simple scaling (which I'll use)

```
#using simple scaling to rescale amount, range (0,1)
data['Amount'] = data['Amount'] / data['Amount'].max()
data.head()
```

V7	V9	V10	V11	V12	V14	V16	V17	V18	V21	Amount	Class
0.239599	0.363787	0.090794	-0.551600	-0.617801	-0.311169	-0.470401	0.207971	0.025791	-0.018307	0.005824	0
-0.078803	-0.255425	-0.166974	1.612727	1.065235	-0.143772	0.463917	-0.114805	-0.183361	-0.225775	0.000105	0
0.791461	-1.514654	0.207643	0.624501	0.066084	-0.165946	-2.890083	1.109969	-0.121359	0.247998	0.014739	0
0.237609	-1.387024	-0.054952	-0.226487	0.178228	-0.287924	-1.059647	-0.684093	1.965775	-0.108300	0.004807	0
0.592941	0.817739	0.753074	-0.822843	0.538196	-1.119670	-0.451449	-0.237033	-0.038195	-0.009431	0.002724	0

Resampling Data

Oversampling and **under-sampling** 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). These terms are used both in statistical sampling, survey design methodology and in machine learning.

Oversampling and under-sampling are opposite and roughly equivalent techniques. There are also more complex oversampling techniques, including the creation of artificial data points. For more details check this or check this article on medium here.

```
# store No. of fraud and indices
fraud_records = data['Class'].sum()
fraud_indices = np.array(data[data.Class == 1].index)

# Picking the indices of the normal classes
normal_indices = data[data.Class == 0].index

# Out of the indices we picked, randomly select number of normal records = number of fraud records
random_normal_indices = np.random.choice(normal_indices, fraud_records, replace = False)
random_normal_indices = np.array(random_normal_indices)

# Merge the 2 indices
under_sample_indices = np.concatenate([fraud_indices,random_normal_indices])

# Copy under sample dataset
under_sample_data = data.iloc[under_sample_indices,:]

# Split data into features and target labels
features_undersample = under_sample_data.drop(['Class'], axis = 1)
target_undersample = under_sample_data['Class']

# Show ratio
print("Percentage of normal transactions: ", under_sample_data.Class[under_sample_data['Class'] == 0].count())
print("Percentage of fraud transactions: ", under_sample_data.Class[under_sample_data['Class'] == 1].count())
print("Total number of transactions in resampled data: ", under_sample_data['Class'].count())
```

```
Percentage of normal transactions:  492
Percentage of fraud transactions:  492
Total number of transactions in resampled data:  984
```

Shuffle and split data

Split the resampled data into features and target labels

```
# Split the 'features_undersample' and 'target_anderSample' data into training and testing sets
X_train_sampled, X_test_sampled, y_train_sampled, y_test_sampled = train_test_split(features_undersample,
                                                                                    target_undersample,
                                                                                    test_size = 0.15,
                                                                                    random_state = 25)

# Show the results of the split
print("Training set has {} samples.".format(X_train_sampled.shape[0]))
print("Testing set has {} samples.".format(X_test_sampled.shape[0]))
```

```
Training set has 836 samples.
Testing set has 148 samples.
```

Implementation

Fit the model on the new data or resampled data and predict the resampled data

```
# Fit the SVC classifier to the under_sample_data
clf.fit(X_train_sampled, y_train_sampled)

# Predict
prediction_train_sampled = clf.predict(X_train_sampled)
prediction_test_sampled = clf.predict(X_test_sampled)

# Calculate the Recall score on training and testing set for the sampled data
recall_train_sampled = recall_score(y_train_sampled, prediction_train_sampled)
recall_test_sampled = recall_score(y_test_sampled, prediction_test_sampled)

# Print Recall score on training and testing set for the sampled data
print('Recall score on training set of sampled data: {:.4f}'.format(recall_train_sampled))
print('Recall score on testing set of sampled data: {:.4f}'.format(recall_test_sampled))
```

```
Recall score on training set of sampled data: 0.9277
Recall score on testing set of sampled data: 0.9221
```

Use the classifier model to predict the whole dataset

```
# Use this model to predict the training and testing set of whole dataset

# Create prediction
prediction_train_after_sampling = clf.predict(X_train)
prediction_test_after_sampling = clf.predict(X_test)

# Calculate the Recall Score
recall_train_after_sampling = recall_score(y_train, prediction_train_after_sampling)
recall_test_after_sampling = recall_score(y_test, prediction_test_after_sampling)

# Print Recall score on training and test set of the whole data
print('Recall score on training set: {:.4f}'.format(recall_train_after_sampling))
print('Recall score on testing set: {:.4f}'.format(recall_test_after_sampling))
```

```
Recall score on training set: 0.9221
Recall score on testing set: 0.9468
```

Have a look on other metrics

```
# Calculate Accuracy score
acc_train_after_sampling = accuracy_score(y_train, prediction_train_after_sampling)
acc_test_after_sampling = accuracy_score(y_test, prediction_test_after_sampling)

# Calculate F-beta score
f_train_after_sampling = fbeta_score(y_train, prediction_train_after_sampling, beta = 0.5)
f_test_after_sampling = fbeta_score(y_test, prediction_test_after_sampling, beta = 0.5)

# Calculate Precision score
precision_train_after_sampling = precision_score(y_train, prediction_train_after_sampling)
precision_test_after_sampling = precision_score(y_test, prediction_test_after_sampling)

#print Scores
print("Accuracy score on training set: {:.2f}%".format(acc_train_after_sampling*100))
print("Accuracy score on testing set: {:.2f}%".format(acc_test_after_sampling*100))
print("\nF-beta score on trainin set: {:.4f}".format(f_train_after_sampling))
print("F-beta score on testing set: {:.4f}".format(f_test_after_sampling))
print("\nPrecision score on trainin set: {:.4f}".format(precision_train_after_sampling))
print("Precision score on testing set: {:.4f}".format(precision_test_after_sampling))
```

```
Accuracy score on training set: 94.97%
Accuracy score on testing set: 95.06%

F-beta score on trainin set: 0.0386
F-beta score on testing set: 0.0381

Precision score on trainin set: 0.0311
Precision score on testing set: 0.0307
```

Refinement

As I describe above working on imbalanced data is a huge mistake and the results of evaluating metrics of benchmark shows that, after resampling the data using undersampling the results improved as I show above Recall score is above 94%.

There're other techniques to improve the result like using K-fold and Grid-Search to pick the best hyper-parameters.

IV. Results

Model Evaluation and Validation

As I showed above the final model is a SVM classifier which fit a resampled data and predict the whole data. The Results of final model shows that the model doesn't suffer from any of problem (Underfitting or Overfitting) and the final model is well generalized to unseen data.

Justification

Benchmark Results:

```
Accuracy score on Training set: 99.97%
Accuracy score on Testing set: 99.94%

F-beta score on Training set: 0.9459
F-beta score on Testing set: 0.8918

Recall score on training set: 0.8166
Recall score on testing set: 0.6489

precision score on training set: 0.9848
precision score on testing set: 0.9839
```

Final Model:

```
Accuracy score on training set: 94.97%
Accuracy score on testing set: 95.06%

F-beta score on trainin set: 0.0386
F-beta score on testing set: 0.0381

Precision score on trainin set: 0.0311
Precision score on testing set: 0.0307

Recall score on training set: 0.9221
Recall score on testing set: 0.9468
```

V. Conclusion

Free-Form Visualization

All needed visualization is attached in above sections.

Reflection

1. *Downloading the dataset from Kaggle.*
2. *Understanding our data*
   a. *Data Exploration*
   b. *Data Visualization*
3. *Preprocessing*
   a. *Data Cleaning*
   b. *Data normalization*
   c. *Shuffle and split the data*
   d. *Evaluating model on imbalanced data*
   e. *Resampling data*

4. *Evaluating model*
5. *Test our model using evaluation metrics*

Improvement

I'm trying to avoid repeating my words in section Refinement I discussed the improvement may be provide to my model. Such as using K-fold and Grid-Search to pick the best hyper parameters.