Machine Learning Engineer Nanodegree

Capstone Project

Bharath Kumar Loganathan
August 23rd, 2017

I. Definition

Project Overview

Credit Card Fraud is defined as when an individual uses another individual's credit card for personal reasons. Credit card fraud has been divided into two types: Offline fraud and On-line fraud. Offline fraud is committed by using a stolen physical card at call center or any other place. On-line fraud is committed via internet, phone, shopping, web, or in absence of card holder. In 1999, out of 12 billion transactions made annually, approximately 10 million—or one out of every 1200 transactions—turned out to be fraudulent. Also, 0.04% (4 out of every 10,000) of all monthly active accounts was fraudulent. As fraudsters are increasing day by day, it has become important for banks to solve this problem.

I chose this problem from Kaggle. The dataset for this problem 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, original features and more background information about the data were not provided. There are totally 28 PCA transformed features available, apart from this, there are 3 features that are not PCA transformed and one of them is target variable. This data set is available in Kaggle in below URL

URL: https://www.kaggle.com/dalpozz/creditcardfraud

Note: You have to register in Kaggle to download the dataset.

Problem Statement

In credit card fraud detection, the number of fraudulent transactions is usually a very low fraction of the total transactions. Hence the task of detecting fraud transactions in an accurate and efficient manner is fairly difficult and challengeable. Therefore, development of efficient methods which can distinguish rare fraud activities from billions of legitimate transaction seems essential. The problem here is to predict whether a transaction can be categorized into normal or fraud which can be called as Binary Classification problem in Machine Learning terms. The data set that I am going to use is the labeled data set which makes the problem as Supervised Binary Classification.

The data set is imbalanced, so we have to develop a model which can accommodate this imbalanced dataset. I am going to try the following 3 methods as a solution. 1] Resampling the data which includes oversampling the minority class and undersampling the majority class. 2] Using Cost-sensitive algorithms 3] Use Tree Algorithms. These models will be analyzed based upon the performance metrics such as fbeta_score, time, recall and precision scores. Then the best model will be chosen and AUC-PR curve will be analyzed finally in this model.

Metrics

Given the class imbalance ratio, we will measure the accuracy using the Area under the Precision-Recall Curve (AUPRC). The precision-recall plot is a model-wide evaluation measure that is based on two basic evaluation measures – recall and precision. The fbeta-score can be interpreted as a weighted average of the precision and recall, so fbeta-score will be a very useful evaluation of the model for this data set. Fbeta-score reaches its best value at 1 and worst at 0. The beta parameter determines the weight of precision in the combined score. beta < 1 lends more weight to precision, while beta > 1 favors recall. I will be using beta value of 2 to give emphasis on recall as we want to catch the more number of fraudulent cases. The formula of fbeta_score is

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

The reason for AUC-PR curve over ROC is that, you can say that if your model needs to perform equally well on the positive class as the negative class (for example, for classifying images between cats and dogs, you would like the model to perform well on the cats as well as on the dogs. For this you would use the ROC AUC.

On the other hand, if you're not really interested in how the model performs on the negative class, but just want to make sure every positive prediction is correct (precision), and that you get as many of the positives predicted as positives as possible (recall), then you should choose PR AUC. Therefore, for detecting fraud, you don't care how many of the negative predictions are correct, you want to make sure all the positive predictions are correct, and that you don't miss any. (In fact, in this case missing a fraud would be worse than a false positive so you'd want to put more weight towards recall.) .

II. Analysis

Data Exploration

- Data set contains only numerical input variables which are the result of a PCA transformation. Due to confidentiality issues, features have named V1, V2, ... V28. These features 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, we can drop this feature as time difference between the first and the subsequent transactions does not have any relation to customer being fraud.

- The feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

- The feature 'Amount' is the transaction Amount. We can use this feature only when we do the cost-effective training. This feature is positively skewed. So, we need to do some preprocessing before it is used with the SVM algorithm.

- **Sample Data:**

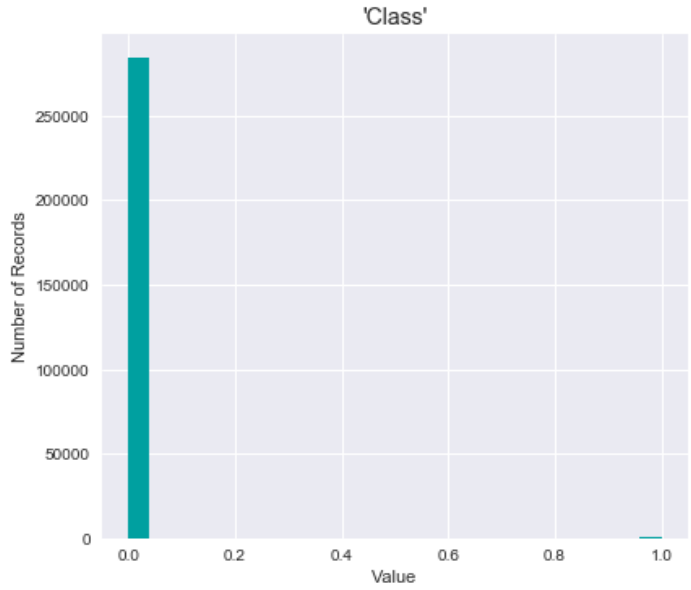
  The data set contains 31 features of which 28 features are PCA transformed named other features are Class, Time and Amount.

Columns	Values
Time	0
V1	-1.35980713
V2	-0.07278117
V3	2.536346738
V4	1.378155224
V5	-0.33832077
V6	0.462387778
V7	0.239598554
V8	0.098697901
V9	0.36378697
V10	0.090794172
V11	-0.55159953
V12	-0.61780086
V13	-0.99138985
V14	-0.31116935
V15	1.468176972
V16	-0.47040053
V17	0.207971242
V18	0.02579058
V19	0.40399296
V20	0.251412098
V21	-0.01830678
V22	0.277837576
V23	-0.11047391
V24	0.066928075
V25	0.128539358
V26	-0.18911484
V27	0.133558377
V28	-0.02105305
Amount	149.62
Class	0

- As the data is already PCA transformed, we don't have to see its statistics information as we are not going to modify the PCA components again and compare the statistics with original PCA components.

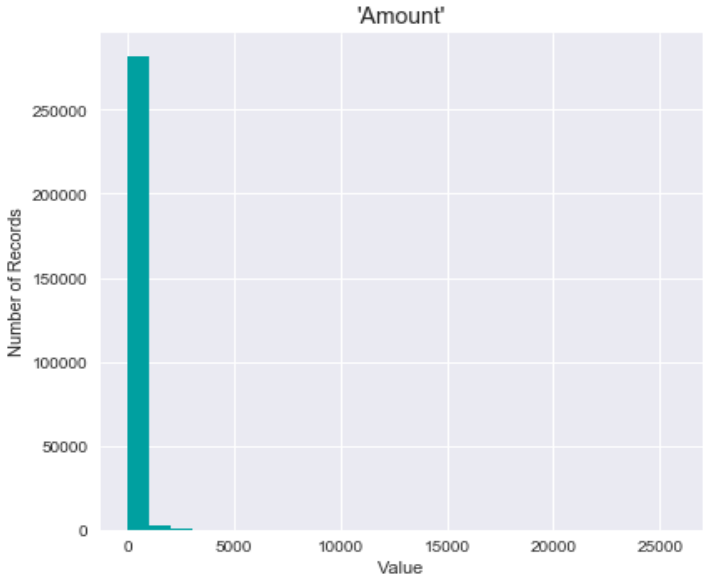
Exploratory Visualization

First, we will see how the feature ‘Class’ which is the target variable distributed.



From the picture, we can clearly see that feature ‘Class’ is totally imbalance with a lot of 0’s and very less number of 1’s

Now, we will see how the feature ‘Amount’ is distributed.



From the picture, we can see that feature is positively skewed and we need to do some pre-processing before using it in the SVM.

Algorithms and Techniques

As I said earlier, this is supervised binary classifier problem. So, I am going to use the following algorithms for implementation:

- Logistic Regression Classifier:

  Logistic Regression Classifier model build by resampling the data i.e., data is undersampled and oversampled and fed into it.

- Decision Tree Classifier:

  This algorithm will demonstrate how the tree based algorithms perform when the data set is imbalanced. Original data will be fed into it without resampling.

- Random Forest Classifier:

  Similar to decision tree algorithm, this model will demonstrate how the ensemble tree based algorithm performs with imbalanced data. Original data will be fed into it without resampling.

- Support Vector Machine:

  SVM is implemented as cost-sensitive training here for the handling imbalanced data set. Original data including the feature 'Amount' will be fed into it.

- Gaussian Naïve Bayes:

  This was recommended to me the reviewer of project proposal.

Benchmark

Since I chose this problem from Kaggle, I will be comparing my models with that of other models in the Kaggle Kernels, but people there have used different metrics like Recall Score, Accuracy score etc. Since, I am going to use Fbeta-score which is different

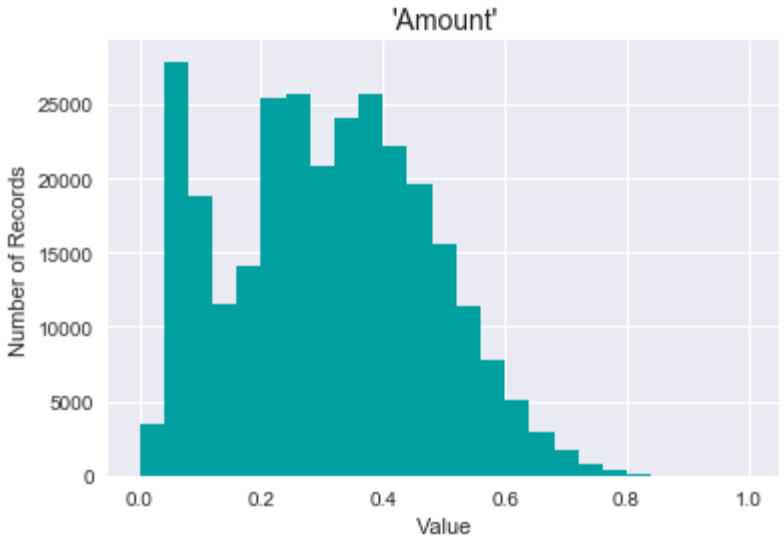
from Kernels that are already submitted, I going to use the median of the Kernels score which will be more 90%. So, 90% of fbeta_score (0.9) will be my benchmark.

III. Methodology

Data Preprocessing

As discussed above we are going to preprocess only the 'Amount' feature. For highly-skewed feature distributions such as 'Amount' it is common practice to apply a logarithmic transformation on the data so that the very large and very small values do not negatively affect the performance of a learning algorithm. Using a logarithmic transformation significantly reduces the range of values caused by outliers. We will apply logarithmic transformation on the 'Amount' feature and visualize again how the data distribution is changed.

We can see that now the data distribution is improved. Now we need to normalize within a range similar to other features in order to feed it into SVM classifier. I will use preprocessing.MinMaxScaler() from Sklearn to normalize the values. Visualizing again after normalization to see how the data distribution is changed:



We can see that data is normalized within range 0 to 1.

Implementation

1. Resampling the Data:

This is the first solution that mentioned earlier in the document. In resampling we will Undersample the majority class and Oversample the minority class.

Resampling process is done using two different libraries to resample the data and compare the results.

- **Using sklearn.utils.resample:**
  - ❖ Over-sampling was done by randomly duplicating observations from the minority class in order to reinforce its signal. There are several heuristics for doing so, but the most common way is to simply resample with replacement.
  - ❖ Under-sampling involves randomly removing observations from the majority class to prevent its signal from dominating the learning algorithm. The most common heuristic for doing so is resampling without replacement.
  - ❖ **Disadvantages of sklearn.utils.resample:**

- The disadvantage of Random over Sampling is that adding the same minority samples might result in over fitting, thereby reducing the generalization ability of the classifier.
- The disadvantage of Random under Sampling approach is that some useful information might be lost from the majority class due to the undersampling.
- The oversampling with replacement was not working Windows-7 32 bit OS.

- **Using imbalanced-learn:**
  - Over-sampling will be done using SMOTE (Synthetic Minority Over-sampling Technique), it is a approach in which the minority class is over-sampled by creating "synthetic" examples rather than by over-sampling with replacement. The minority class is over-sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the k minority class nearest neighbors. Depending upon the amount of over-sampling required, neighbors from the k nearest neighbors are randomly chosen.
  - Undersampling will be done using NearMiss. It is a family of methods performs undersampling of points in the majority class based on their distance to other points in the same class. There are 3 variants of this technique available, they are 1] NearMiss-1 2] NearMiss-2 and 3] NearMiss-3. I used NearMiss-1 in implementation.

2. Using Cost-sensitive algorithms:

The next tactic is to use Cost-sensitive (penalize algorithms) that increase the cost of classification mistakes on the minority class. We can use the argument class_weight='balanced' to penalize mistakes on the minority class by an amount proportional to how under-represented it is. Support Vector Machine was implemented with linear kernel and setting the parameter class_wieght equal to 'balanced'.

3. Tree Algorithms:

The next tactic is to implement tree-based algorithms. Decision trees often perform well on imbalanced datasets because their hierarchical structure allows them to learn signals from both classes. Original data will be fed into it without resampling.

Now, we know that imbalanced data set can be handled by the Ensemble algorithms, Penalized Algorithms and Tree Algorithms separately. Now, we will combine all these techniques in 1 one algorithm and check the performance. All these 3 techniques can be combined using Random Forest Classifier.

We know Random Forest is an ensemble algorithm with decision tree as the base learner. Random Forest has a parameter called 'class_weight' parameter, by setting this parameter to 'balanced', weights inversely proportional to the class sizes are used to multiply the loss function.

4. Gaussian Naïve Bayes:

As mentioned earlier in the document, this was recommended to me the reviewer of project proposal. So, we will how this performs.

Refinement

To improve results, model was fine tuned using grid search (GridSearchCV) on the best chosen model. The best chosen model was logistic regression classifier with SMOTE (details explained in the Results section). Its output values are

Fbeta-score: 0.9255
Recall-score: 0.9144
Precision-score: 0.9727

This model was used as the classifier in the grid search and the parameter 'C' was tried with different values. Finally, it was found that the best optimal value for parameter 'C' is 5. There was not much variation in the results, it was same as before tuning the model.

F-score after optimizing: 0.9255
Recall-score after optimizing: 0.9144
Precision-score after optimizing: 0.9727

IV. Results

Model Evaluation and Validation

All the models are evaluated by the time taken to train and predict as well as metrics such as Fbeta-Score, Recall-Score and Precision-Score and these values was used for the comparing the models and choosing the best one.

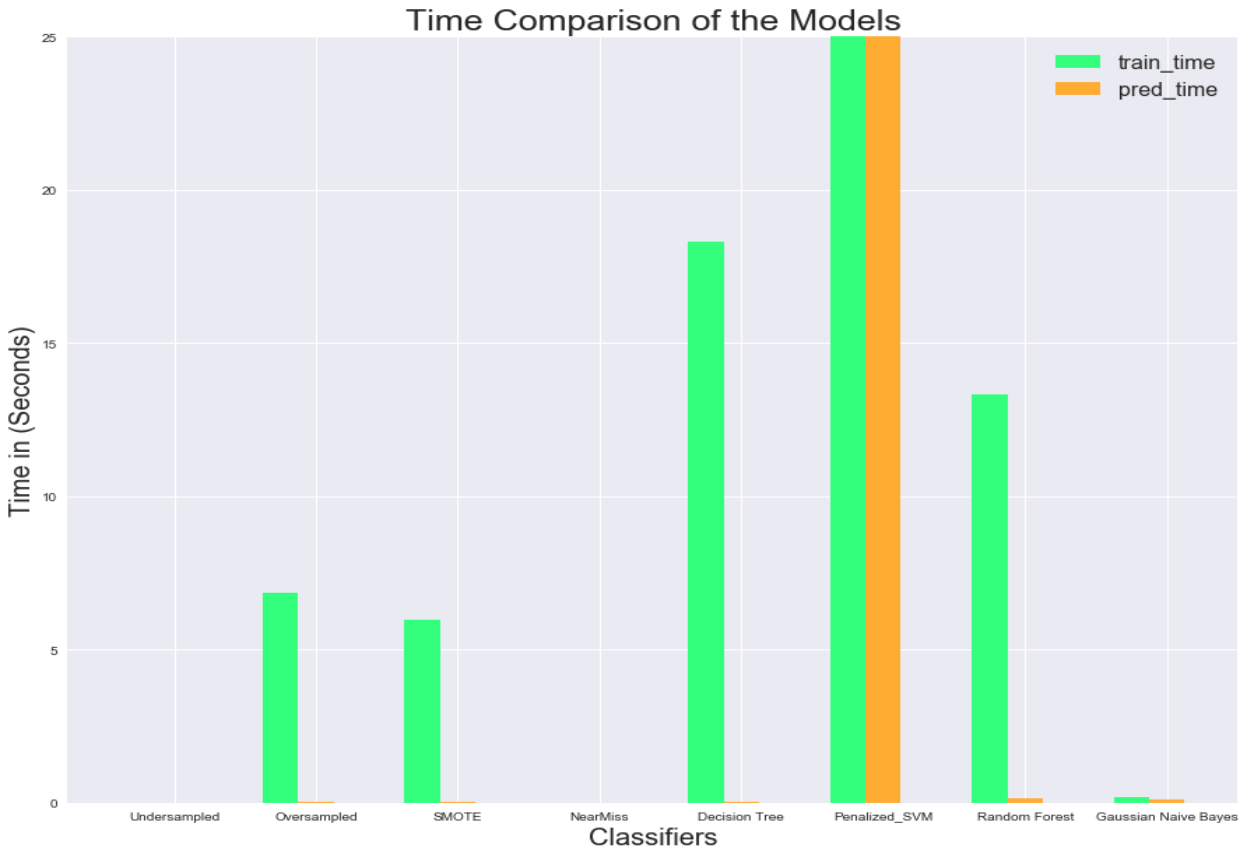
1. Performance Summary

Summary of all the values obtained from the models are given in the table below:

	Decision_Tree	GNB	NearMiss	Random_forest	SMOTE	SVC_CLF	oversampled	undersampled
fbeta	0.773481	0.078214	0.918508	0.783476	0.925513	0.293722	0.926651	0.904762
precision	0.823529	0.063764	0.977941	0.964912	0.972714	0.079781	0.975387	0.956835
pred_time	0.024000	0.091000	0.000000	0.128000	0.013000	92.585000	0.021000	0.000000
recall	0.761905	0.836735	0.904762	0.748299	0.914420	0.891156	0.915218	0.892617
resample_time	NaN	NaN	12.031000	NaN	6.099000	NaN	NaN	NaN
train_time	18.293000	0.179000	0.009000	13.323000	5.976000	4551.336000	6.871000	0.008000

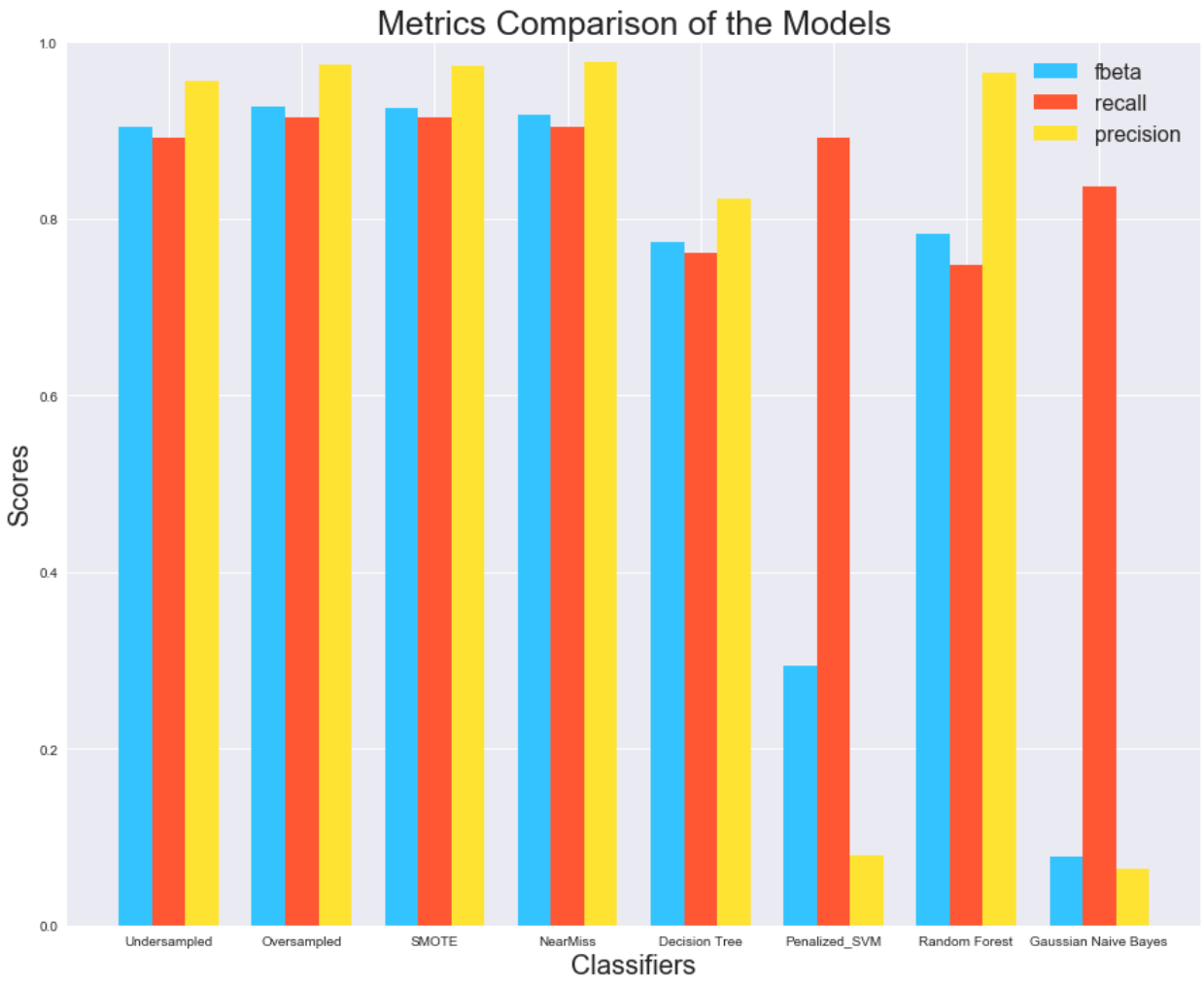
2. Time Comparison of the models:

Time taken by all the models is represented visually in the below diagram:



3. Metrics Comparison of the models:

Metric values of all the models are represented visually in the below diagram:



4. Choosing the Best Classifier

We will now choose the best classifier using the figures above. We will choose the top 4 classifiers by ranking them from 1 to 4 with rank 1 being best and 4 being the worst.

1] Time: We will first rank the classifiers based on their training time and prediction time.

Rank 1: Undersampled

Rank 2: Gaussian Naive Bayes

Rank 3: NearMiss

Rank 4: SMOTE

2] Metrics: We will now rank the classifiers based on their fbeta_score, precision_score and recall score with recall score given the highest priority

Rank 1: SMOTE

Rank 2: Oversampled

Rank 3: NearMiss

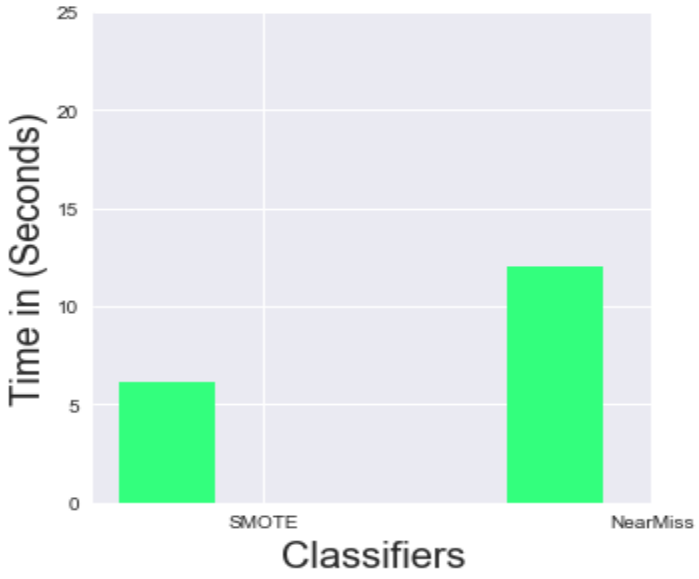
Rank 4: Undesampled

Now, of the two comparisons made above, we have to give more importance to the metrics than time because at the end, classifier with best accuracy is needed and a little trade-off in the time is acceptable.

So, with that we choose the classifiers with SMOTE and NearMiss as the best models they have best metrics with not much delay in training and testing time.

Now, we have to choose the best classifier among these 2. For, that we compare the resampling time taken by them, i.e., the time taken by SMOTE to oversample the data and the time taken by the NearMiss to undersample the data.

The comparison chart is given below



From the above chart we can conclude that classifier that uses SMOTE has a best performance compared to the rest of the classifiers

This model was then refined using grid search and the 'C' hyperparameter was fined and the model tested more thoroughly with various values of C and at the got the best optimal parameters for the model which was mentioned in the refinement section above. The model was robust and results did not change that much and in fact results are either improved or declined which also showed that model is reliable.

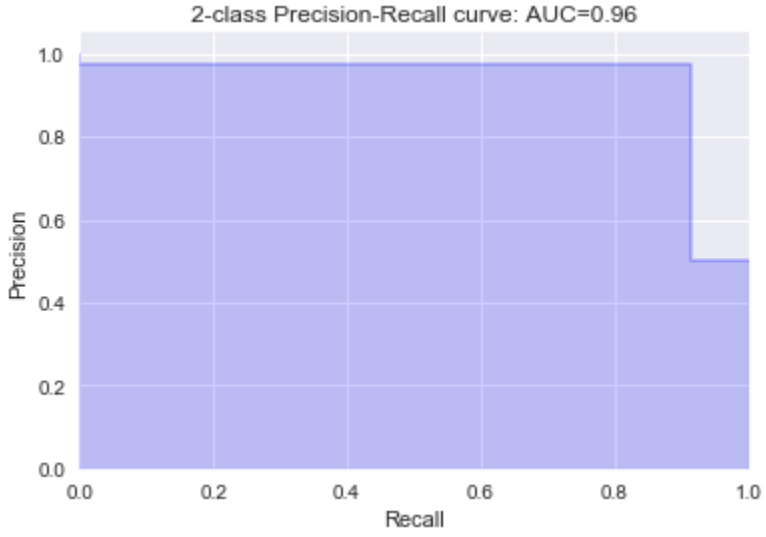
Justification

The benchmark was set 90% of fbeta-score and the SMOTE model got the fbeta-score of 92.55% which is better than the benchmark. Also, the recall score is 91.44 % and precision score is 97.27%. High scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall). An ideal system with high precision and high recall will return many results, with all results labeled correctly. So, the model will predict high number fraudulent transaction with very high precision.

V. Conclusion

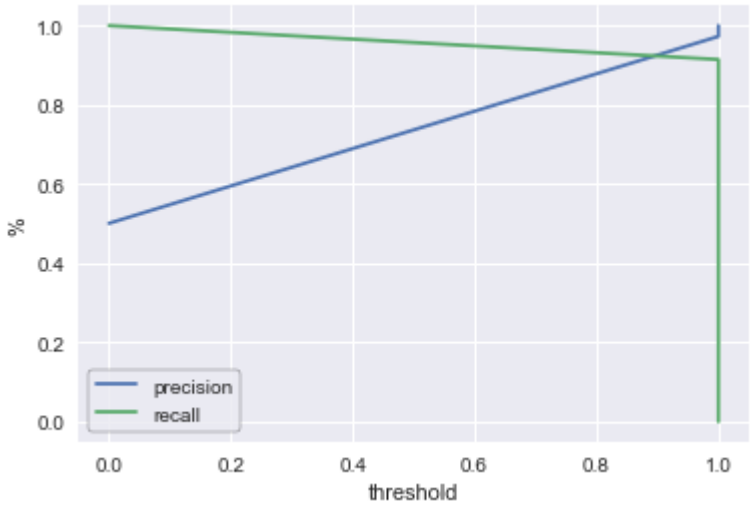
Free-Form Visualization

The precision-recall curve shows the tradeoff between precision and recall for different threshold. The figure below represents PR curve



We got 96% of area (Area Under the Curve) for PR curve, a high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate.

We can also find the best threshold for the classifier. The figure below shows the different values of precision and recall against the threshold.



From above figure, we can conclude the best threshold for the classifier is around 0.85.

Reflection

The process used for this project can be summarized using the following steps:

1. An initial problem and relevant, public datasets were found. This was a challenge hosted in Kaggle website.
2. The data was downloaded and analyzed and the data set was found to be imbalanced.
3. The data was preprocessed by applying logarithmic transformation on the 'Amount' feature and then normalizing it.
4. Different models were tried out.
5. Finally, the model that uses SMOTE for oversampling the data was found to be best model with highest fbeta-score, recall-score and precision-score.
6. This model was optimized and test thoroughly using grid search with parameter 'C'. It was found that model was robust and reliable as the results did not vary.
7. Then, AUC-PR curve was plotted and 96% area was achieved and threshold of the classifier was found out to be around 0.85.

Obviously like in many machine learning problems, the challenge here was to get the model that gives high accuracy. Therefore, I found the steps 4, 5 and 6 as the most difficult process while implementing the project.

Improvement

Although, the results found here were promising, we can try to improve the results using a neural network for fraud detecting. Other techniques like clustering and Hidden Markov Models can also be tried for credit card fraud detection project with different data sets.

References:

https://en.wikipedia.org/wiki/Credit_card_fraud

http://ieeexplore.ieee.org/document/7100189/

http://usir.salford.ac.uk/2595/1/BBS.pdf

http://dl.acm.org/citation.cfm?id=2695990

http://www.mydigitalshield.com/credit-card-fraud-detection-techniques/

https://www.kaggle.com/dalpozz/creditcardfraud

http://dl.acm.org/citation.cfm?id=2695990

http://contrib.scikit-learn.org/imbalanced-learn/stable/