Capstone Project

Machine Learning Engineer Nanodegree

Domain Background

In this project, I will be training a model to automatically recognize fraud for several credit card transactions. This is a real-time classic example of anomaly detection. The model built is a supervised classification based on the data has a lot of fraudulent records and a very few fraudulent Our primary dataset is from Kaggle and available in reference section.

I chose this project because it gives me a chance to for an anomaly detection issue. This is a classic example of that category and credit cards, who does not use them? So, that interested me to know even a little more about the insights of credit card fraud analysis.

Project Overview

The dataset contains transactions made by credit cards in September 2013 by European cardholders.

The dataset has been collected and analyzed during a research collaboration of World line and the Machine Learning Group (http://mlg.ulb.ac.be) of ULB on big data mining and fraud detection.

Problem Statement

The problem is to identify and classify the fraudulent of the credit card transactions using the given data set. I have used different supervised learning models to learn from the data set and made predictions.

This project deals with anomaly detection. It helps build a model that can detect any future fraudulent transactions based on the data set. The machine learning models will learn from the given data and can predict any future transactions that can be fraudulent. This helps protect the customer from his money and the credit card company that time and effort after the effect.

Proposed strategy

The solution to the problem will be a classification model that will output a prediction of transaction as fraudulent or non-fraudulent. The possible algorithms that I might use are support vector machines, logistic regression or random forests as these are good models that work for classification.

I propose to identify the better model based on the metrics decided below.

Metrics

Given the class imbalance ratio, the accuracy using the Area Under the Precision-Recall Curve (AUPRC) is used as a metric. AUC and ROC curve are also interesting to check if the model is also making right predictions and not making many errors.

Accuracy score, Confusion matrix and Classification report are not meaningful for unbalanced classification. But, I am interested in the recall score for the entire test set (with skewed data), because that is the metric that will help us try to capture the most fraudulent transactions. (by attempting to not miss a fraud transaction)

- Accuracy = (TP+TN)/total
- Precision = TP/(TP+FP)
- Recall = TP/(TP+FN)

Data Exploration

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. T

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.

Resampling

As we can that data is highly skewed and only 492 of the 284,807 transactions are fraud.

Below, we can see visualize that data, no fraudulent cases are displayed. The fraudulent records are shown when we normalize them.

This is a good example of anomaly detection and we have several sampling techniques to tackle problems of this kind.

Exploratory Visualization

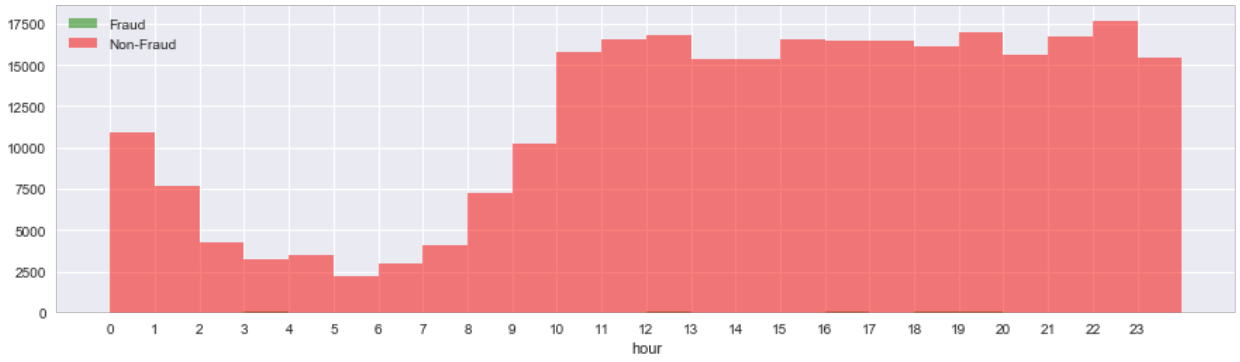


Fig.1

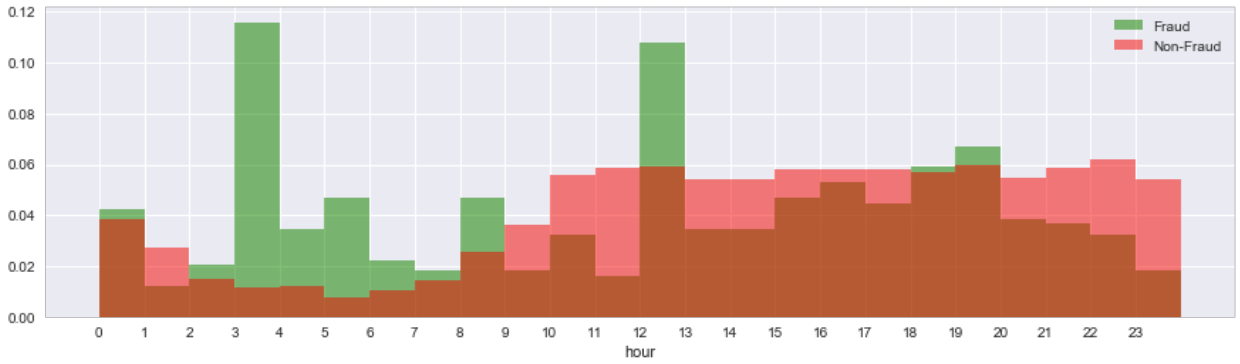
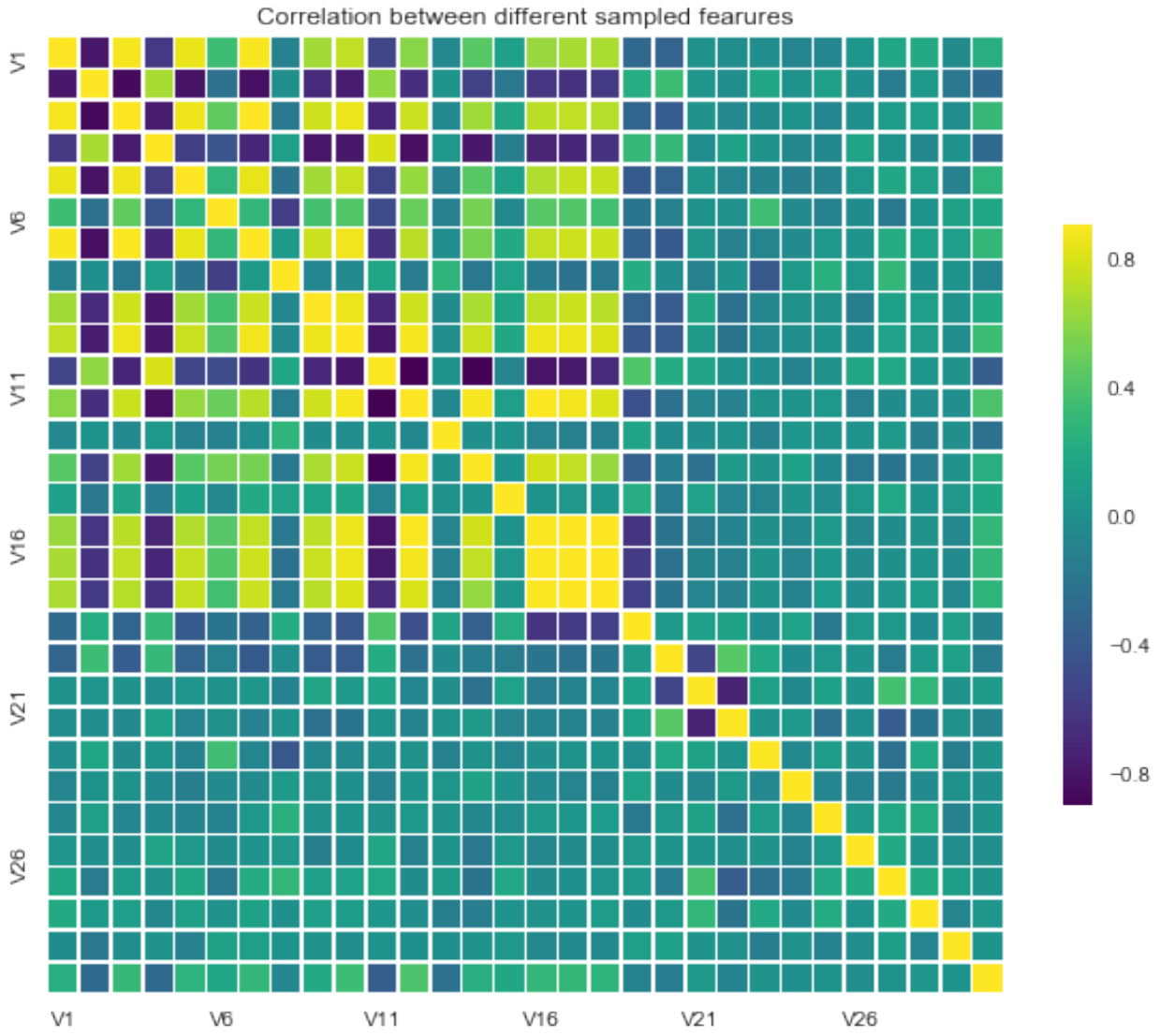


Fig.2

I used traditional UNDER-sampling. I sampled the dataset by creating a 50-50 records, there by representing both the classes equally.

This is achieved by randomly selecting "x" amount of sample from the majority class, being "x" the total number of records with the minority class.

I also tried to look at the correlation matrix for the sampled features:



Algorithms and Techniques

Due to the labeled data problem, this is, I picked below three models as they work well for classification problem.

1. Logistic Regression
2. SVMs

3. Random Forests

More details about Random forest:

1. Random forests is an ensemble algorithm that uses a group of decision trees
2. Each decision tree makes a prediction using a sample of the given training dataset.
3. Now, Random forests model picks the winner using the majority voting principle to identify the most voted class by all the decision trees.

Pros about Random forests:

1. Random forests does not over fit and it is fast.
2. You can run as many trees as you want.
3. For large data sets the major memory requirement is the storage of the data itself, and three integer arrays with the same dimensions as the data.

I will work on building these models and evaluate the accuracy score, recall score and AUC-ROC to validate the model, tune it and pick the best one.

Benchmark

Given the class imbalance ratio, the primary benchmarks here would be the accuracy using the Area Under the Precision-Recall Curve (AUPRC) and accuracy score.

AUC and ROC curve are also interesting to check if the model is also making right predictions and not making many errors.

During our data exploration experiments and other competitors from kaggle leaderboard, a good classifier had an accuracy score was around 90% and recall of about 0.90. That's my benchmark.

Data Preprocessing

For data preprocessing, first I worked on the Amount field.

This is has a high range of values and so I used standard scaling method to rescale it.

Now, the range of Amount field is changed from (25691.16, 0.0) to (102.36224270928423, -0.35322939296682354).

Implementation

My implementation is covered in Jupyter notebook file.

I briefly plan to cover the following in the ipynb file:

1. Pull download and extract the train and test datasets.
2. Explore, analyze the dataset and generate visualizations such as a histogram of original train data.
3. We train our model and log important information like accuracy of the validation set and the loss.
4. Save the trained model to disk so it can be loaded again or exported.
5. During the training, try to minimize loss and increase the accuracy and performance and tune the model.
6. Once the model is trained, evaluate it using the test dataset.

Refinement

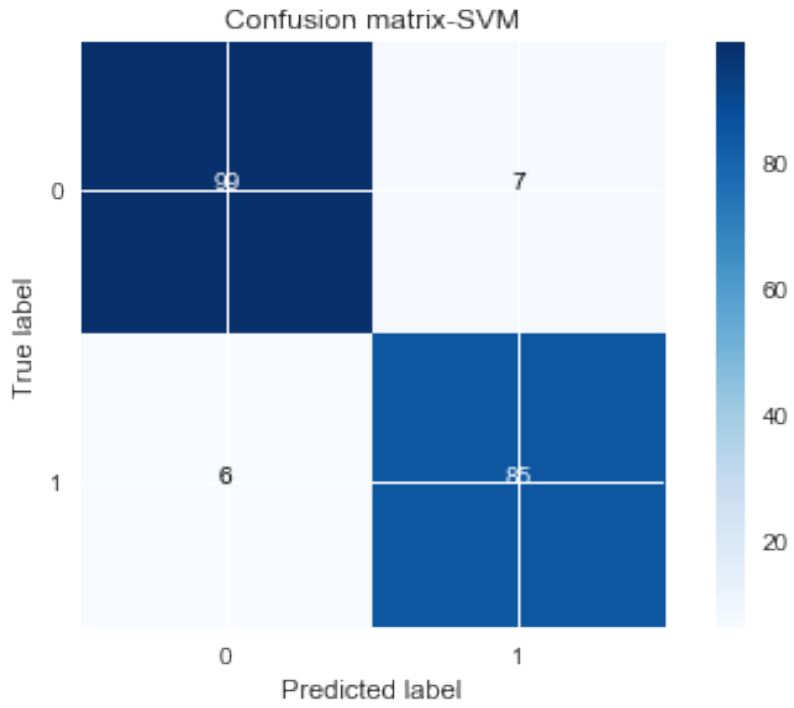
With the SVM, I initially got a score less than Logistic Regression and Random forests models. So, I decided to fine tune it and test the

performance with the optimal C and gamma values. I used the GridSearchCV method to run the model with different combinations of C and gamma and identify the best parameter combination for my model.

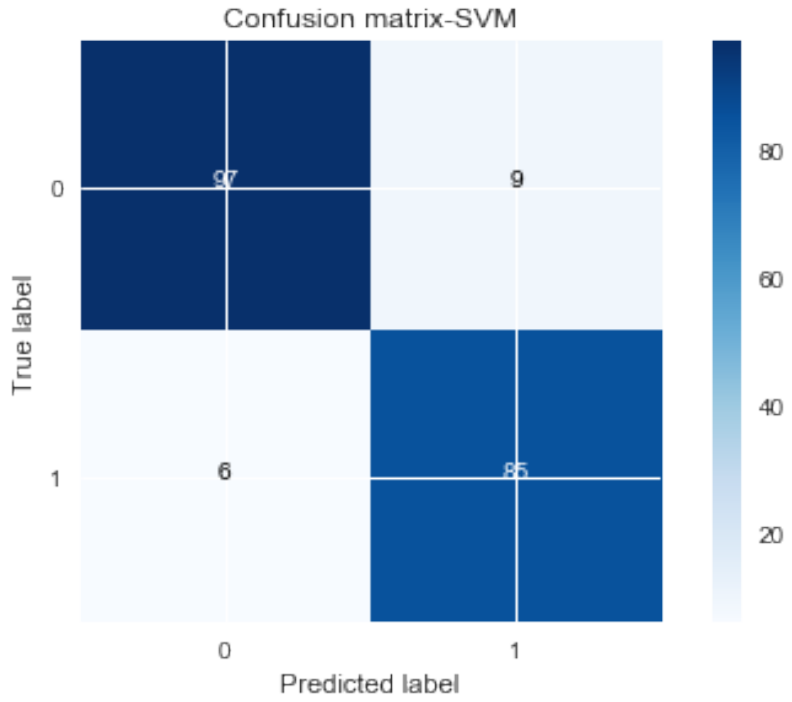
- param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001]}

After this implementation, I still got a score of 0.93 which is closer to the one original one.

Confusion matrix before:



Confusion Matrix with grid:

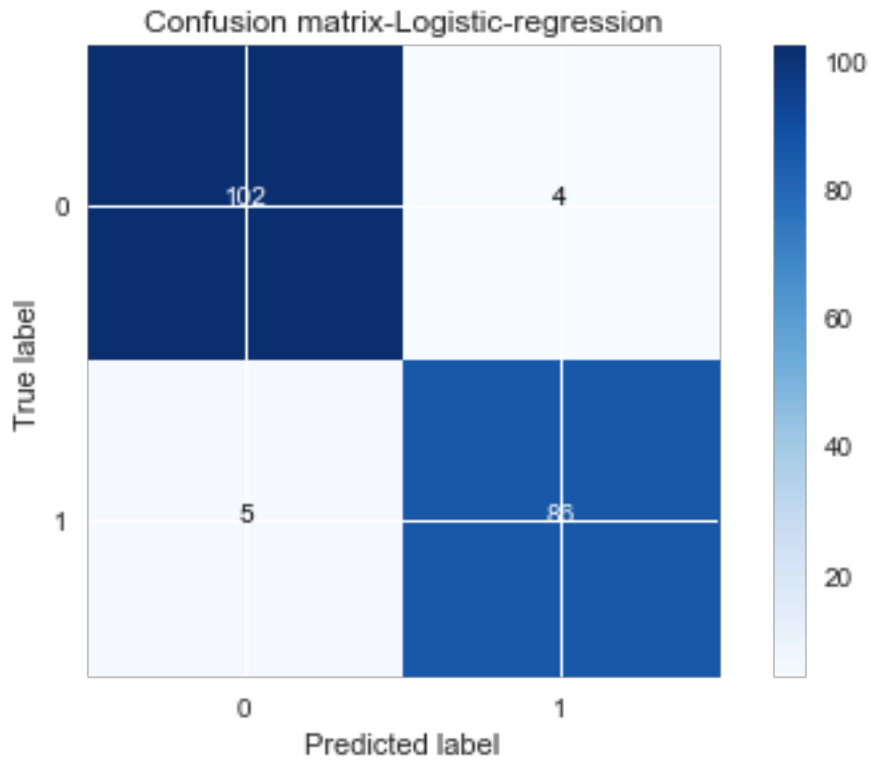


So, I am not sure if this final solution is better than the initial one. But because the other model Random Forests has better performance metrics and so I chose that over SVM.

Model Evaluation and Validation

1. Logistic Regression:

Confusion Matrix for Logistic Regression:

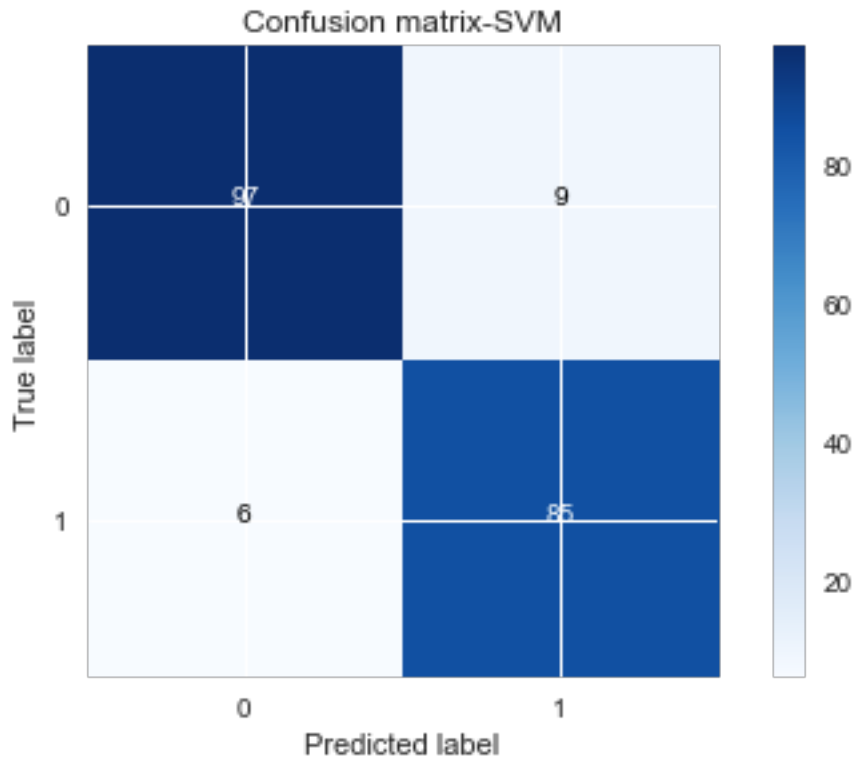


	precision	recall	f1-score	support
0	1.00	0.97	0.98	85296
1	0.04	0.93	0.08	147
avg / total	1.00	0.97	0.98	85443

The accuracy score was around 0.95.

The recall metric also showed 0.95 when tested with the whole test data.

2. SVM:

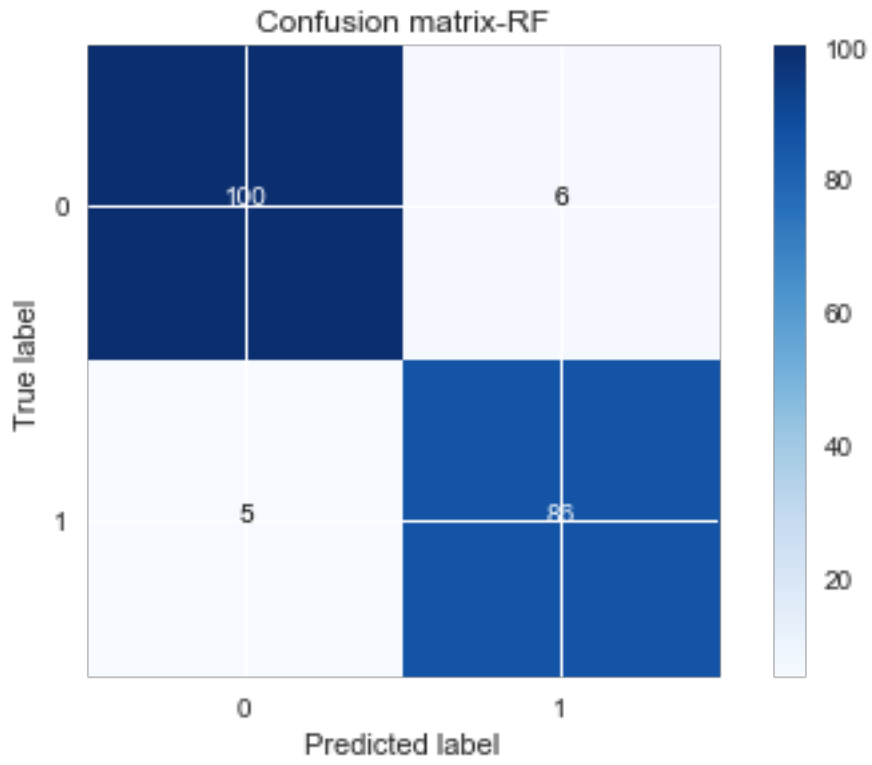


	precision	recall	f1-score	support
0	1.00	0.93	0.96	85296
1	0.02	0.95	0.05	147
avg / total	1.00	0.93	0.96	85443

The accuracy score was around 0.94.

The recall metric also showed 0.93 when tested with the whole test data.

3. RANDOM FORESTS:



	precision	recall	f1-score	support
0	1.00	0.96	0.98	85296
1	0.04	0.97	0.08	147
avg / total	1.00	0.96	0.98	85443

The accuracy score was around 0.96.

The recall metric also showed 0.96 when tested with the whole test data.

Justification

The model I chose was Random forests and the metrics' scores are:

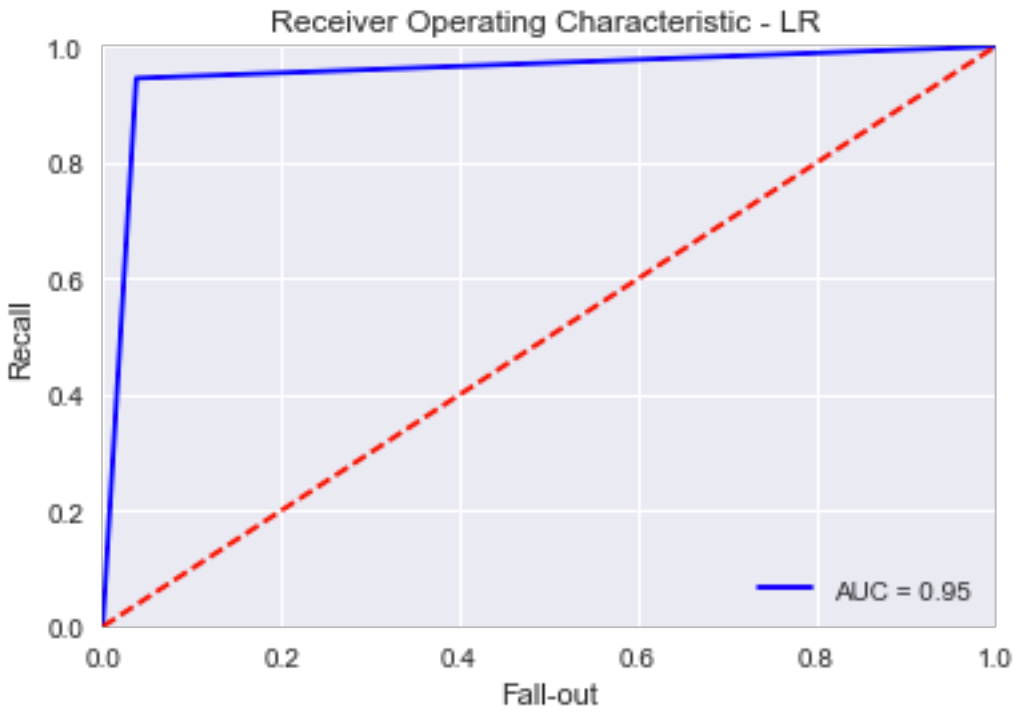
- Accuracy score - 0.96
- Recall metric – 0.96

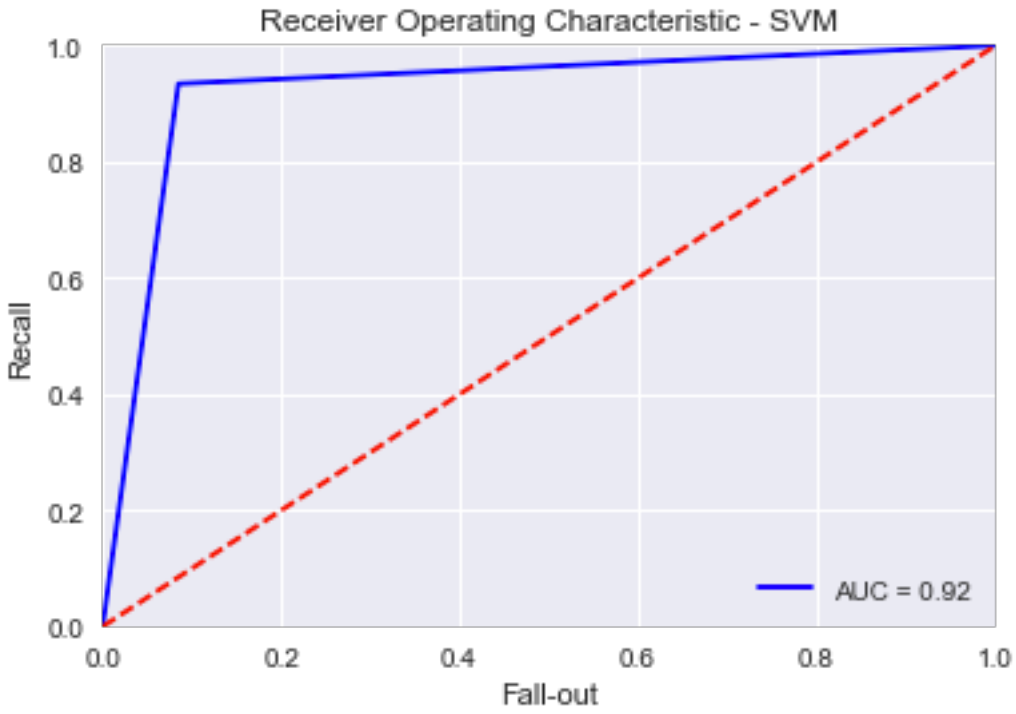
The scores are compared to benchmark and found to be stronger. The visualization is described above for the same.

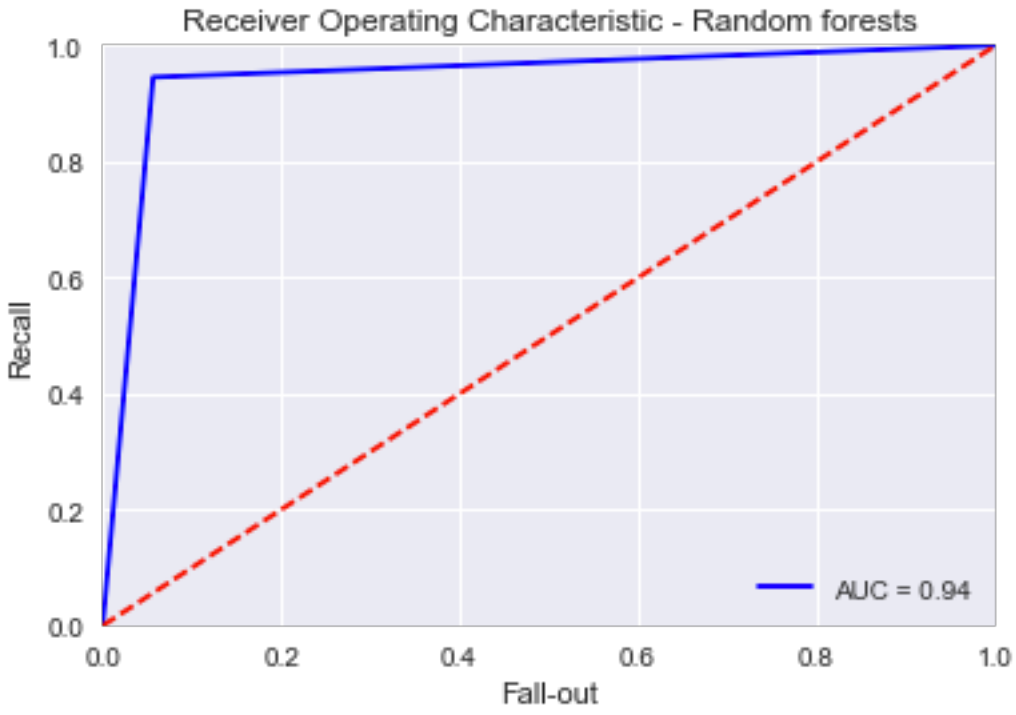
Free-Form Visualization

Plotting ROC curve and Precision-Recall.

- As this is an anomaly detection and predicting positives is more impactful than negative class, I have chosen Precision-recall scores for the positive rate calculation.
- AUC and ROC curve is a good metric to check if the model is also predicting correctly as a whole correctly.




Receiver Operating Characteristic - SVM
Recall
Fall-out
AUC = 0.92
0.0
0.2
0.4
0.6
0.8
1.0


Receiver Operating Characteristic - Random forests
Recall
Fall-out
AUC = 0.94
0.0
0.2
0.4
0.6
0.8
1.0

From the above confusion matrix and classification report, recall score is better for random forests than the other models – SVM and Logistic regression.

Also, based on the accuracy and AUC score, I picked Random forests as my model.

Reflection

To summarize my work, I tried to work on the credit card data set which was clearly skewed labeled data.

I briefly plan to cover the following steps in the ipynb file:

1. Pull download and extract the train and test datasets.
2. Explore, analyze the dataset and generate visualizations such as a histogram of original train data.
3. We train our model and log important information like accuracy of the validation set and the loss.
4. Save the trained model to disk so it can be loaded again or exported.
5. During the training, try to minimize loss and increase the accuracy and performance and tune the model.
6. Once the model is trained, evaluate it using the test dataset.

Improvement

As an improvement, I think this model can further be validated against with other resampling techniques other than the used method – under-sampling. I propose to use SMOTE technique for this.

Also, I would like to check this data set using other ensemble methods like XGBoost and see if that improves my model performance.

Reference:

Kaggle site - https://www.kaggle.com/dalpozz/creditcardfraud