Machine Learning Engineer Nanodegree Capstone Project

Sharath Rathinakumar

July 9, 2017

I. Definition

Project Overview

In the domain of finance, a leading problem is fraud. Compromised credit cards and data breaches have dominated the headlines in the past couple of years. Technology such as EMV promises to make some payments safer, but experts predict fraud will remain a growing problem for years to come. Data breaches totaled 1,540 worldwide in 2014 -- up 46 percent from the year before -- and led to the compromise of more than one billion data records.

The United States accounted for 72% of these breaches[1]. Twelve percent of breaches occurred in the financial services sector; 11 percent happened in the retail sector. Malicious outsiders were the culprits in 55 percent of data breaches, while malicious insiders accounted for 15 percent. I have personally been affected 3 times in the past two years and had 3 of my cards replaced and the unauthorized transactions reimbursed. Flagging fraudulent transactions is critical for everyone involved to come out with least damage.

Problem Statement

The problem is to identify the fraudulent transactions and label as possible red flags when provided with a transaction's details. The details are the features. Once we are able to flag these transactions, this information can then be used to process the next steps of verification by the financial institution. This is a classification problem with 2 possible labels – fraudulent or legitimate. It is also a special case of classification "Anomaly Detection" due to the nature of the data – a large number of legitimate labels and a very small number of fraudulent transactions. The data will be presented with more details in the next section.

Metrics

The input as will be the case with transactions encountered by a large financial institution has a large number of legitimate transactions while having a small number of fraudulent transactions. In terms of a good way to measure the efficiency of the model – accuracy score is not very useful as we have a very small number of true positives(fraud) compared to true negatives. F-score is a good option here due to the fact that both precision and recall are taken into account to get the f-score which is more suitable for binary classification problems. The mathematical formula[6] for F Score is:

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Precision and Recall are defined[7] below:

$$\text{Precision} = \frac{tp}{tp + fp} \qquad \text{Recall} = \frac{tp}{tp + fn}$$

Where tp, fp and fn are numbers of true positives, false positives, and false negatives, respectively.

II. Analysis

Data Exploration

The dataset used is one of the highest voted datasets in Kaggle and the dataset contains transactions made by credit cards in September 2013 by european cardholders[2]. The dataset contains time, amount, label, and 28 anonymized features for all the transactions in csv format. 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 detection3. Hence the input is a set of anonymized features which reflect details of the financial transaction. I’m working under the assumption that these features correspond to useful information such as billing zip code, transaction zip code, transaction amount, average of last 10 transactions, and such details. Below is an example row from the dataset which shows the Class is the label which reflects 0 is a legitimate transaction and 1 denotes a fraudulent transaction:

Time	**V1**	**V2**	**V3**	**V9**	**...**	**V21**	**V22**	**V23**	**V24**	**V25**	**V26**	**V27**	**V28**	**Amount**	**Class**
0.0	-1.359807	-0.072781	2.536347	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0

1 rows × 31 columns

I used to below code to load the dataset and have a basic understanding of the data:

```
# Import libraries necessary for this project
import numpy as np
import pandas as pd
from time import time
from IPython.display import display # Allows using display() for DataFrames

# Import supplementary visualization code visuals.py
import visuals as vs

# Pretty display for notebooks
%matplotlib inline

# Load the Census dataset
data = pd.read_csv("creditcard.csv")

# Success - Display the first record
display(data.head(n=1))
```

Exploratory Visualization

This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, below are some statistics from the dataset:

Total number of transactions	284807
Total number of fraudulent transactions	492
Total number of legitimate transactions	284315
Percentage of fraudulent transactions	0.17%

The below code was used to explore the dataset further and gain some insight on the different labels and to get an understanding of how many true and false negatives and positives exist in the data.

```
#  Total number of records
n_records = len(data.index)

#  Number of records with fraudulent transactions
n_fraud = len(data[data['Class'] != 0])

#  Number of records with legitimate transactions
n_legitimate = len(data[data['Class'] == 0])

#  Percentage of fraudulent transactions
fraud_percent = float(n_fraud)/n_records*100

# Print the results
print "Total number of transactions: {}".format(n_records)
print "Total number of fraudulent transactions: {}".format(n_greater_50k)
print "Total number of legitimate transactions: {}".format(n_at_most_50k)
print "Percentage of fraudulent transactions: {:.2f}%".format(greater_percent)
```

Algorithms and Techniques

I studied the nature of the information in the data set and determine that scaling is not needed as the data already was already reasonably scaled. The next step was to research and identify which algorithms perform best for the problem in hand. Given that this is an anomaly detection problem, the algorithms I considered were Ensemble Methods (AdaBoost, Random Forest), one class Support Vector Machines (SVM), density-based techniques (k-nearest neighbor) and Logistic Regression as the benchmark model. I had made my decisions based on a variety of factors such as F Score, complexity due to features/training time. Below I have listed some of the test runs I did.

Benchmark

The first option to consider is the naïve solution. The naïve approach would be to label all transactions as legitimate and we will still be right 99.8% of the time. But we will not be able to flag any fraudulent transactions, thus not achieving the fundamental objective we set out to do. A more reasonable benchmark I think would be to use Logistic regression. I will compare the performance of my solution against the performance using Logistic regression which is a widely accepted algorithm for binary classification4,5. Hence comparing against logistic regression provides a good idea of how good the performance is against a generic benchmark solution.

III. Methodology

Data Preprocessing

The dataset contains 28 features. The data did not need scaling and there were no text data, so one hot encoding was not necessary either. Firstly, I extracted the labels(Class) and used a simple train_test_split to get the below split:

The code used for the split is below:

```
# Import train_test_split
from sklearn.cross_validation import train_test_split

# Split the 'features' and 'income' data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, income, test_size = 0.2,
random_state = 0)

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

Training set has 227845 samples

Testing set has 56962 samples.

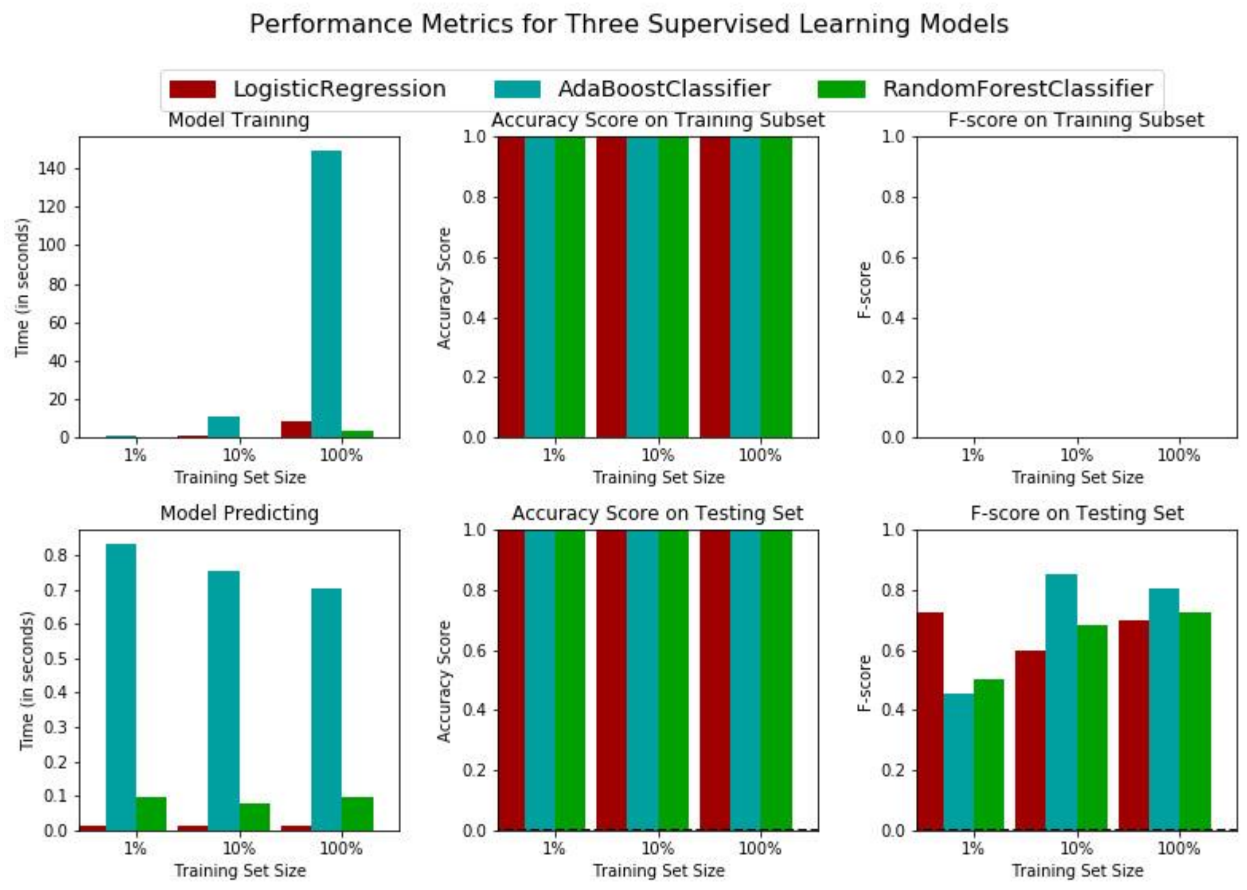
Then I trained and tested with classification algorithms detailed in the algorithms section to test the performance with all 28 features. The below was the table for time taken for 1% (2278), 10% (22784) and 100% (227845) of the training set size:

Algorithm	Time for 1%	Time for 10%	Time for 100%
LogisticRegression	0.047000169754	0.52999997139	5.03800010681
AdaBoostClassifier	0.957999944687	6.07400012016	76.118999958
OneClassSVM	10.236000061	309.464999914	-

Since some algorithms are taking several minutes (OneClassSVM took more than hour with all the features before I cancelled the run). In the next sections I will implement PCA and also feature selection. This will help with faster training and test runs and I also believe the reduced complexity the algorithms will perform better F-Score wise.

Implementation

After replacing OneClassSVM with Random Forest Classifier for the initial run with full features, the below diagram summarizes the statistics of the run. (I will compare this against the final model I choose finally). This run is before applying PCA and feature selection.

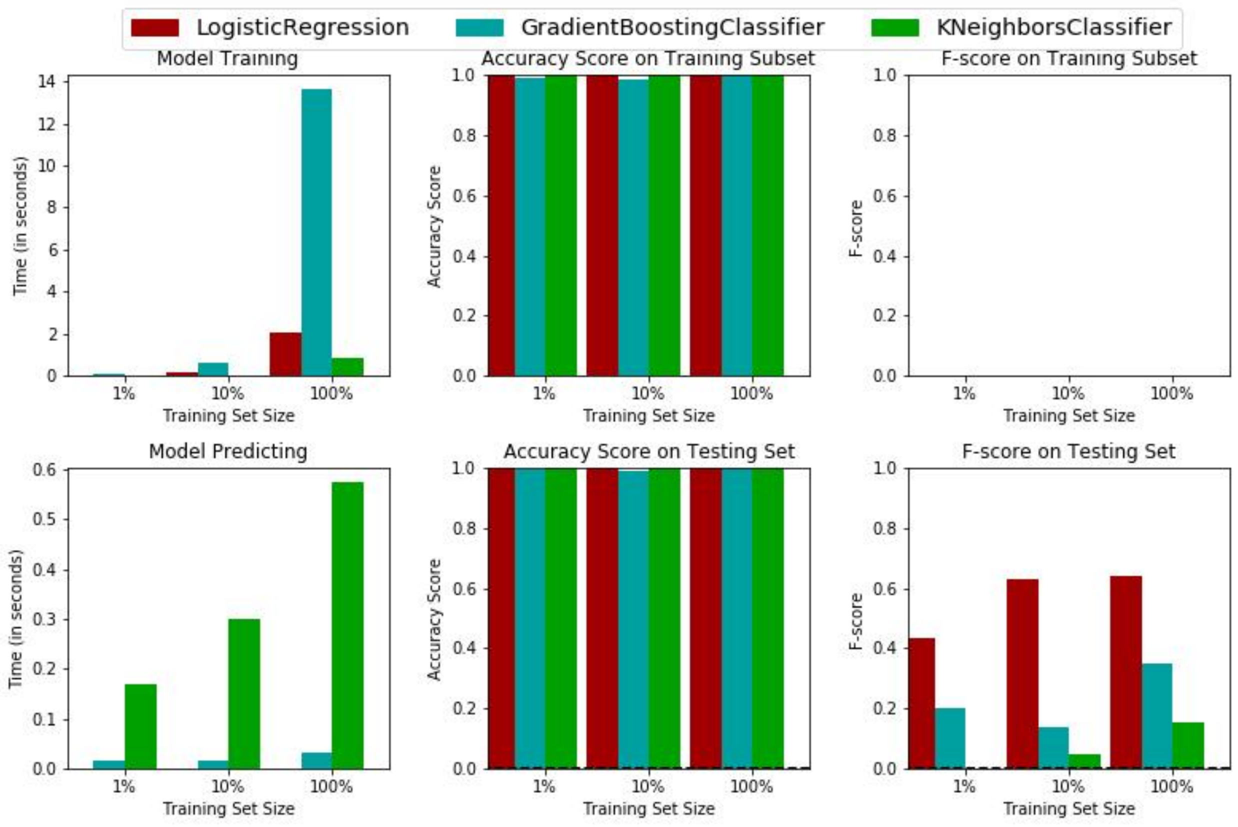


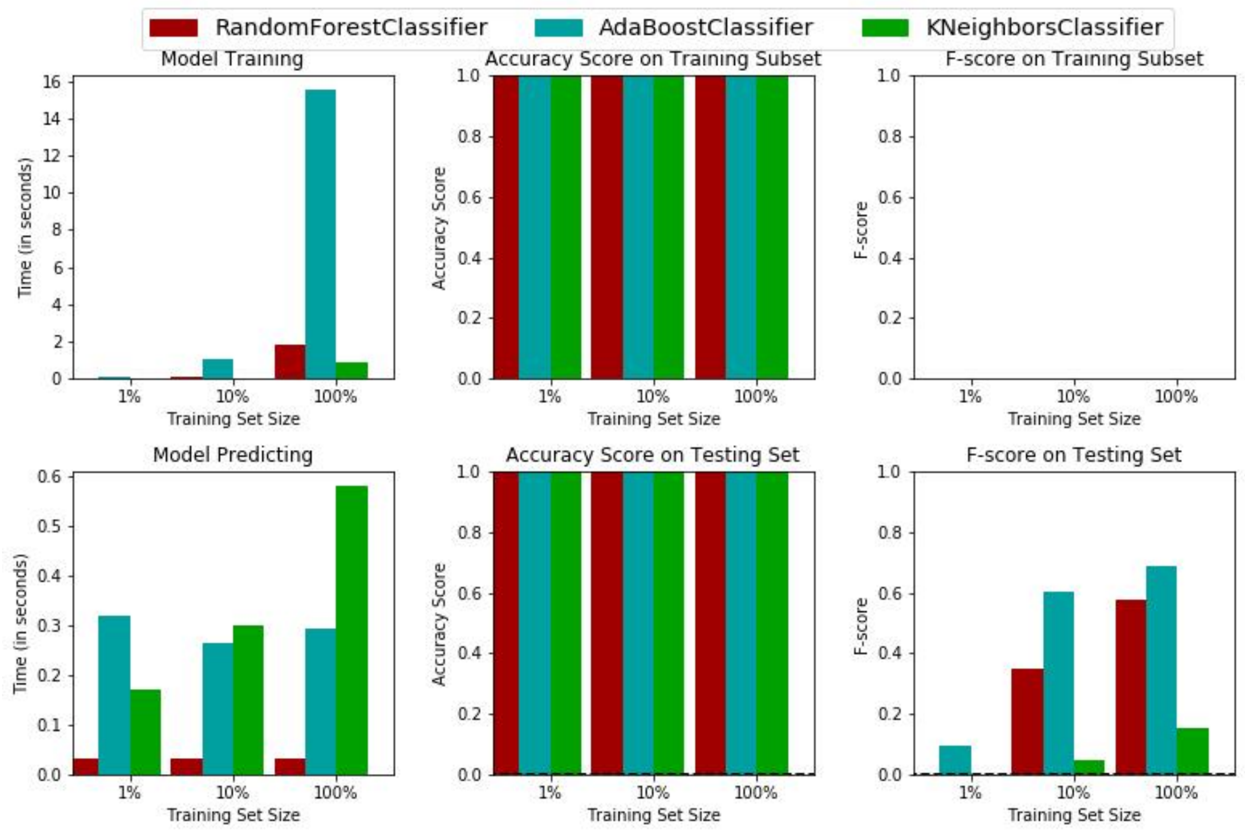
The methodology used was as follows, I did two iterations with 5 of the most suited algo rithms for the problem as discussed in the algorithms section:

```
Iteration 1:
clf_B = KNeighborsClassifier(n_neighbors=3)
clf_C = GradientBoostingClassifier(n_estimators=100, max_depth=1, random_state=0)
clf_D = LogisticRegression(random_state=0)
Iteration 2:
clf_B = KNeighborsClassifier(n_neighbors=3)
clf_C = RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1)
clf_D = AdaBoostClassifier()
```

Using PCA I transformed the features and looked into which transformed features captur ed the data most accurately and then used those features to train the algorithms. The co de I used for this is below:

```
# Running PCA to get transformed # from sklearn.decomposition import PCA
pca = PCA(n_components=6)
pca.fit(X_train)
print(pca.explained_variance_ratio_)
[9.999727e-01 2.726349e-05 1.640776e-09 1.050605e-09 8.846861e-10 8.351285e-10]
```





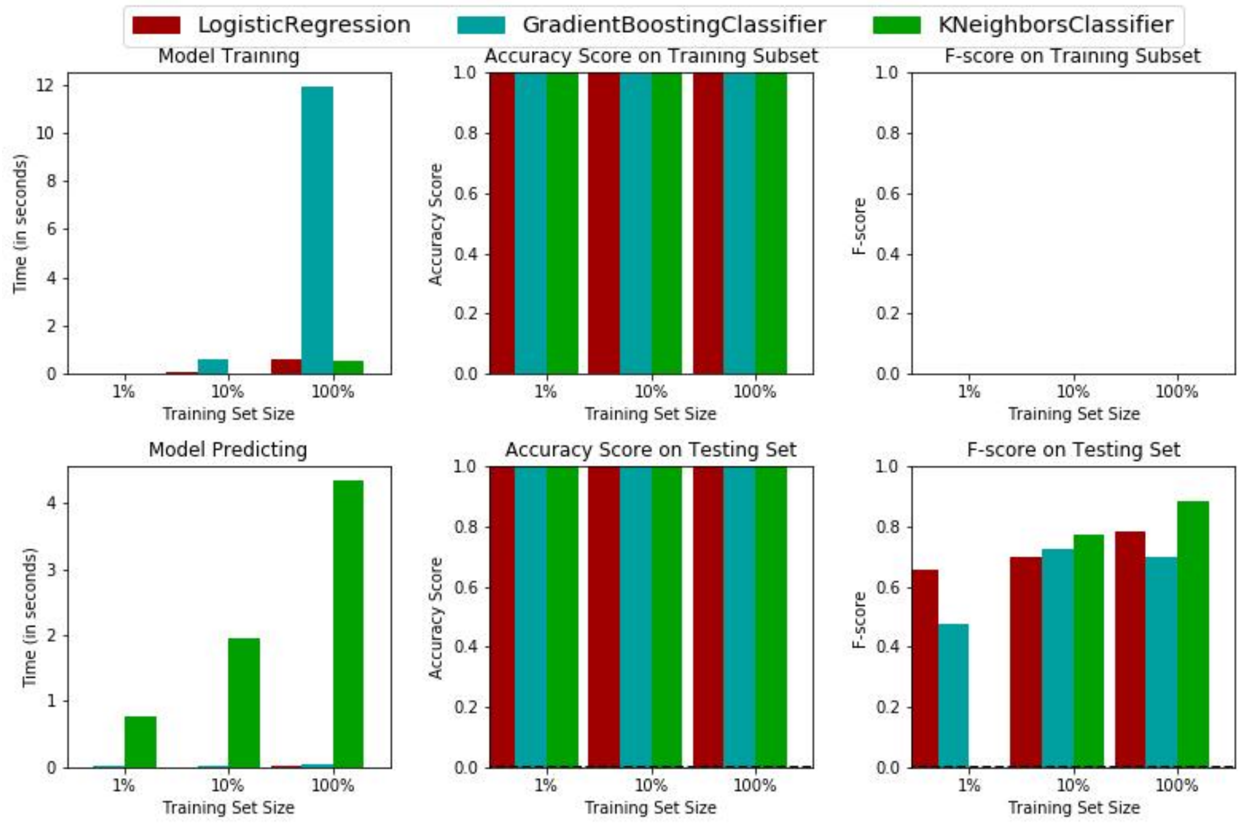
The performance went down significantly with the transformed features. I believe this is due to the fact that the nature and range of the features ends up under representing the important features that actually indicate fraud. In the next section, I will detail employing feature reduction.

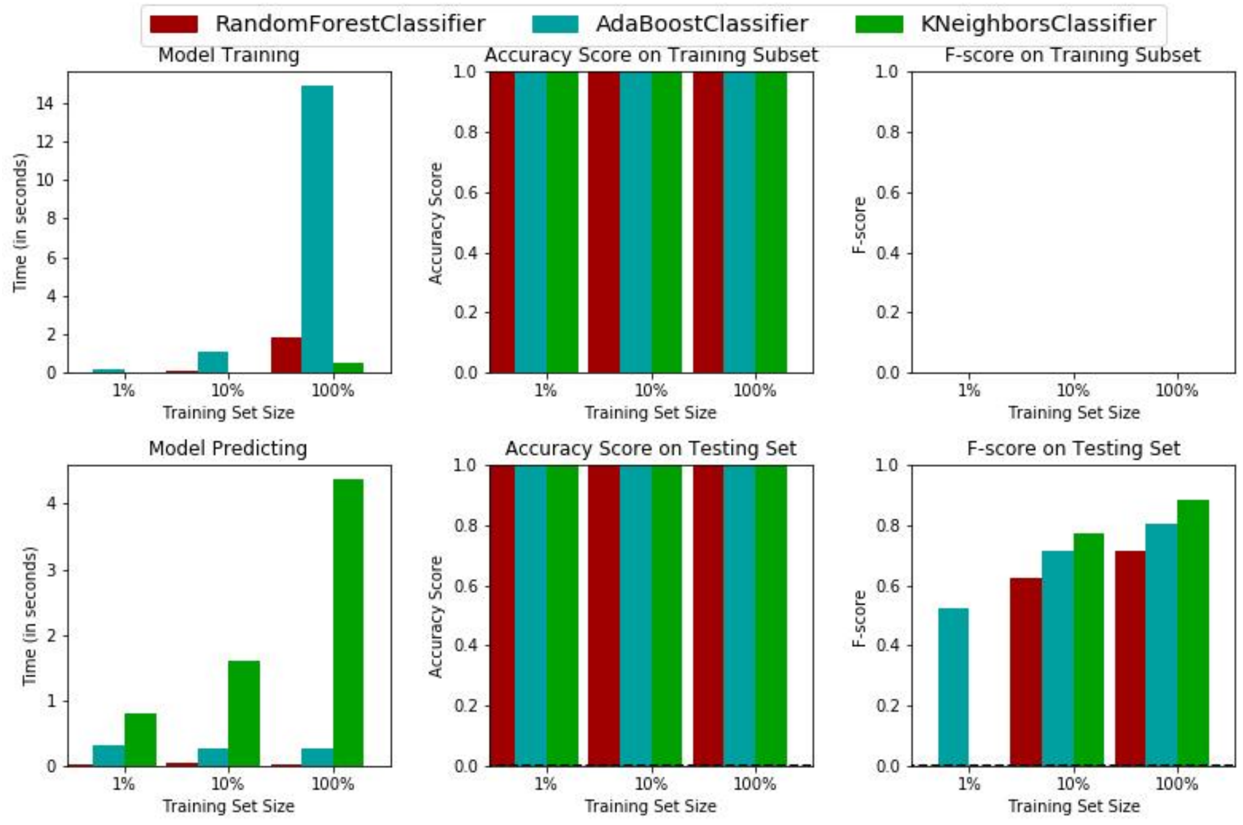
Refinement

From the preliminary run AdaBoostClassifier performed well with a F Score of 0.79. Hence, I used AdaBoostClassifier and the feature_importances_ method to pick the most useful features. Used the below code to get the top 6 features: [V14 V24 V18 V19 V3 V4]

```
clf_D = AdaBoostClassifier()
preliminary_model = clf_D.fit(X_train, y_train.values.ravel())
importances_1 = preliminary_model.feature_importances_
print X_train.columns.values[(np.argsort(importances_1)[::-1])[:6]]
```

The below statistics were generated using the same iteration of algorithms as detailed in the previous section training the models using the top 6 useful features.


LogisticRegression
GradientBoostingClassifier
KNeighborsClassifier
Model Training
Accuracy Score on Training Subset
F-score on Training Subset
Model Predicting
Accuracy Score on Testing Set
F-score on Testing Set
Time (in seconds)
Accuracy Score
F-score
Training Set Size
1%
10%
100%

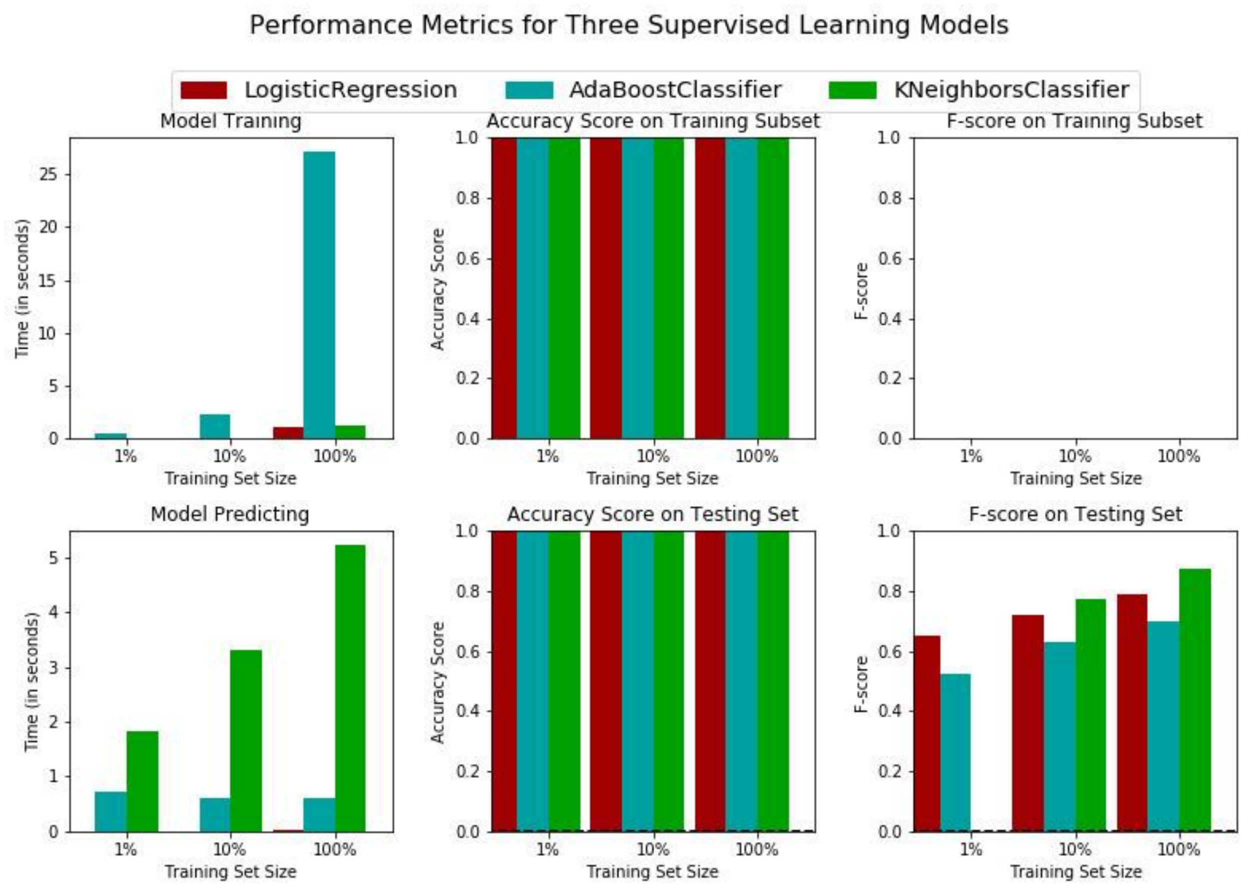

RandomForestClassifier
AdaBoostClassifier
KNeighborsClassifier
Model Training
Accuracy Score on Training Subset
F-score on Training Subset
Model Predicting
Accuracy Score on Testing Set
F-score on Testing Set
Time (in seconds)
Accuracy Score
F-score
Training Set Size
1%
10%
100%

As seen from the data KNeighborsClassifier has performed the best compared to all other algorithms with a F-Score close to 0.8824. This is a significant improvement over the intermediate performance score of 0.8 with the AdaBoost Classifier trained with all features.

IV. Results

Model Evaluation and Validation

Be reducing the input space to 6 features, the performance has improved significantly with a F-Score edging on 0.8824 with K Neighbors Classifier and the training time has also reduced. To test the robustness of the model I also reduced the number of features from 6 to 5 and the performance did not drop much and was still close to 0.86. The high F-Score shows how reliable the model is. The below summarizes the run with 5 features:



After verifyin the reliability of the model in order to further optimize the model I performed GridSearchCV and tried different number of neighbors and leaf size.

The below code was used to obtain the optimized model:

```
parameters = { 'n_neighbors':[3,6,9], 'leaf_size': [10,30,60]}
scorer = make_scorer(fbeta_score, beta=0.5)
#  Perform grid search on the classifier using 'scorer' as the scoring method
grid_obj = GridSearchCV(clf,parameters,scoring=scorer)

#  Fit the grid search object to the training data and find the optimal parameters
grid_fit = grid_obj.fit(X_train_reduced, y_train.values.ravel())

# Get the estimator
best_clf = grid_fit.best_estimator_

# Make predictions using the unoptimized and model
predictions = (clf.fit(X_train_reduced, y_train.values.ravel())).predict(X_test_reduced)
best_predictions = best_clf.predict(X_test_reduced)
```

Unoptimized model
Accuracy score on testing data: 0.9994
F-score on testing data: 0.8824
Optimized Model
Final accuracy score on the testing data: 0.9994
Final F-score on the testing data: 0.8802

Justification

The below table summarizes the comparison of the performance of the final model against the benchmark model of logistic regression trained against all features. As seen below the final model is a statistically significant improvement over the benchmark model and also has a comparatively lower training/run time.

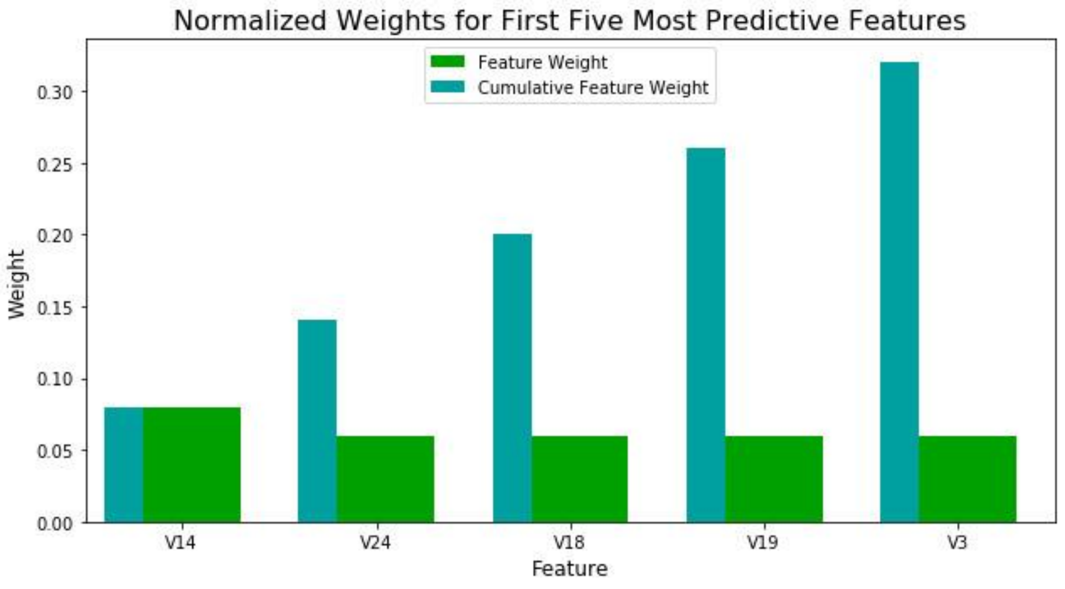
Metric	Model Algorithm	Accuracy Score	F-score
Benchmark Predictor	LogisticRegression	0.9994	0.88
Unoptimized Model	KNeighborsClassifier	0.9994	0.88
Optimized Model	KNeighborsClassifier	0.9994	0.68

The goal is to identify fraudulent transactions which are an anomaly among the large number of legitimate transactions. So, we can discard accuracy score. With an F-Score of 0.88 the final model is able to predict and identify fraudulent transactions quite well.

V. Conclusion

Free-Form Visualization

Below I have extracted the normalized weights of the 5 most important features. The top feature is more useful and the next 4 features. The nature of transactions having a lot of common features and only few of them actually indicate fraud. Thus, by reducing all the noise we are able to significantly improve the performance.



Reflection

To summarize, I started with training AdaBoost and few other algorithms with all the features, extracted the most important 6 features. Thus decreasing the problem space to 6 features in the training and testing data. I ran a second round of analysis of anomaly detection algorithms on the reduced data. Among which KNeighborsClassifier performed the best, achieving an F-Score of 0.88 and much faster training/run times. This methodology can be used in solving other anomaly detection problems as well.

Improvement

I do think it is possible to make further improvements. I think using a boost classifier with a very low learning rate and also experimenting with a mix of important features the F-Score can be further improved. But it will take a really long time to train. But if training time is not a limit, then this approach can be employed.