Credit Card Fraud Detection Using Anomaly Detection Techniques

Oyelakin Oluwadamilare, ID: 20115037 [* 1]
Word Count: 1556

Abstract

Credit cards are increasingly been used for completing transactions both online and offline in everyday life. With this increased usage of credit cards, credit card fraud is also growing. The detection of fraudulent transactions is an important application in anomaly detection. This paper compared different unsupervised techniques in credit card fraud detection; Local Outlier Factor, one-class SVM, and Isolation Forest was identified as the best technique with the highest accuracy.

1. Introduction

Billions of dollars are lost annually due to credit card fraud, (Chan et al., 1999; Chen et al., 2006). These losses due to frauds affect both the financial companies and the customers. According to the United States Federal Trade Commission report till the mid-2000s the theft rate of identity was stable, but an increase of 21% was noticed in 2008 (John & Naaz, 2019).

According to the Nilson Report, worldwide losses from card fraud rose to US$27.85 billion in 2019, up from about US$8 billion in 2010. That number is expected to reach US$35.67 billion by 2023 (Nilsonreport.com, 2019).

To reduce the losses which occur due to these credit card frauds an efficient fraud detecting systems to identify fraudulent transactions needs to be developed. Anomaly detection is one of the techniques that can be used to detect fraudulent transactions.

This research focuses on identifying whether new transactions are fraudulent or not by using anomaly detection techniques such as Isolation Forest, Local Outlier Factor and One-class Support Vector Machine.

The results demonstrate that Isolation Forest outperformed the other techniques by minimizing the number of false positives and detecting the highest number of fraud in the credit card transactions dataset used in this research.

This paper is structured as follows. Section 2 presents a detailed description of the problem domain; Section 3 describes the dataset; Section 4 describes the methodology which is used to tackle the problem; Section 5 discusses the experiments, and section 6 demonstrates the results obtained; Section 7 presents the conclusion of the report.

2. Background: Problem Domain

An anomaly refers to when something substantially deviates from the normal behaviour and detecting such anomaly in data is called anomaly detection (Mehrotra et al., 2017)

Several supervised and unsupervised anomaly detection methods have been used to detect credit card fraud. Supervised approaches like Support Vector Machine (SVM), Decision trees, k-nearest neighbours, logistic regression and others provide good results in credit card fraud detection but these methods require labelled data to form the classifier whereas in the unsupervised approaches the data does not have to be labelled (Flarence et al., 2018).

In real-world situations, it is quite difficult to access labelled datasets and this process is costly when the labelling is performed by humans. Most accessible datasets generally have a small percentage of fraudulent transactions leading to imbalanced class distribution, this can affect the efficiency of supervised algorithms negatively (Sabinasz, 2017). This research would focus on the unsupervised anomaly detection approach.

3. Dataset Description

The datasets contain 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 as seen in Figure 1. The dataset is highly unbalanced, the fraud cases account for 0.172% of all transactions.

The dataset is available on Kaggle: `http://www.kaggle.com/mlg-ulb/creditcardfraud`

[1]M.Sc. Big Data Analytics, School of Computing and Digital Technology, Birmingham City University, UK. Correspondence to: Oyelakin Oluwadamilare <oluwadamilare.oyelakin.bcu.ac.uk>.

Data Mining Report, CMP 7206 - Data Mining.

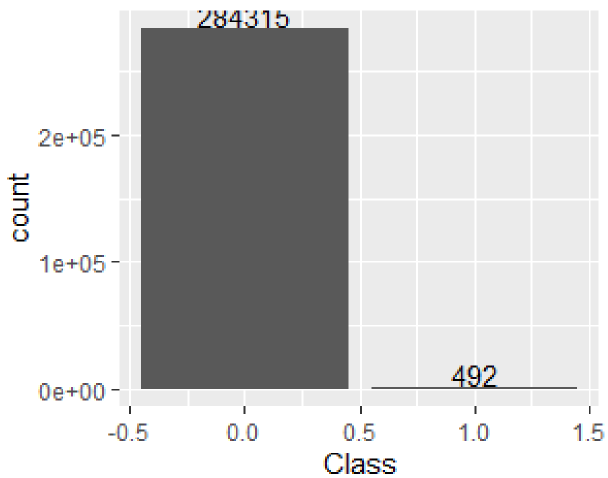


Figure 1. Dataset Distribution

Table 1. Dataset attributes description

#	ATTRIBUTE	VARIABLE TYPE
1	TIME	NUM.
2	V1	FLOAT
3	V2	FLOAT
	....	...
29	V28	FLOAT
30	AMOUNT	FLOAT
31	CLASS	NUM. (0 OR 1)

3.1. Data pre-processing and Transformation

Original features and more context about the dataset are not provided in the dataset due to confidentiality issues. Only numerical input variables are provided which are the results of Principal Component Analysis (PCA) Transformation. Features V1, V2 ... V28 are the principal components obtained with PCA (John & Naaz, 2019).

Table 2. Training and testing samples

DATASET	TRAINING	TEST	SAMPLE SIZE
TRANSACTIONS	199364.90	85442.10	284,807

Table 2 shows the dataset was randomly divided into two sets: 70% as the training set and 30% as the test set to guarantee valid predictions.

4. Methodology

Three separate algorithms were used in this paper for fraud detection. These algorithms are explained below:

4.1. Isolation Forest

Isolation forest is a tree-base model that is developed to detect outliers [13]. This algorithm is based upon the fact that anomalies are the data points which are few and different (John & Naaz, 2019). These properties result in susceptible mechanism to anomalies.

Isolation Forest introduces the use of isolation as an efficient and more effective way to detect anomalies rather than the basic distance and density measures.

The proposed Isolation Forest in Liu et al. 'isolates' observations by selecting an attribute and then selecting a split value between the maximum and minimum values of the selected attribute arbitrarily. The number of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node (Liu et al., 2009).

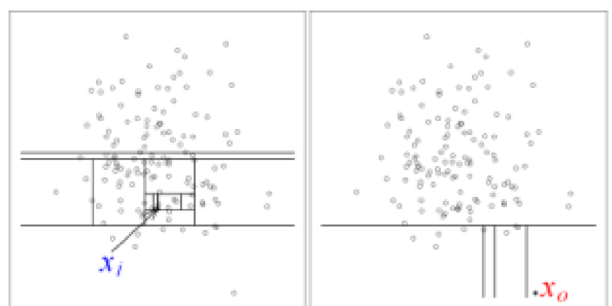


Figure 2. Identifying Normal vs. Abnormal

Figure 2 shows how Isolation Forest identifies a normal versus an abnormal data point. It shows that a normal point of xi would need more divisions for it to be isolated. While an anomaly $x0$ needs fewer divisions to be isolated. An anomaly score can be calculated as the number of conditions required to isolate a given observation (Liu et al., 2008).

This algorithm has low linear time complexity and space complexity.

4.2. Local Outlier Factor (LOF)

Local Outlier Factor was proposed by Breunig et al (Breunig et al., 2000). The algorithm is an unsupervised anomaly detection method which computes the local density deviation of a given data point to its neighbours. It considers a point as an outlier if the density of that point is much smaller than the densities of its neighbours.

LOF shares similar concepts with OPTICS and DBSCAN which are used for local density estimation such as the concepts of **reachability distance**—the distance between two points and **core distance**—the minimum search distance needed to make a distinct point a core point (Breunig et al., 2000)

4.3. One-class SVM (OCSVM)

It was proposed by Scholkopf as an unsupervised learning algorithm.

"OCSVM an algorithm that computes a binary function that is supposed to capture regions in input space where the probability density lives (its support), that is, a function such that most of the data will live in the region where the function is nonzero" (Schölkopf et al., 2000)

It is an application of support vector machine algorithms to one class problems. This algorithm uses a hyperplane to separate all the data points from the origin. It calculates the density level sets and gives an estimate of underlying density(Schölkopf et al., 2000).

5. Experiment Set up

Data exploratory analysis of the data has been conducted in RStudio, while the model building and anomaly detection carried out using python on google collab. Using python allowed me to take advantage of the machine learning package-sklearn. The model building processes were carried out on google collab GPU with 20GB RAM.

5.1. Model Evaluation Metrics

Due to the skewed nature of the dataset used, multiple measures of accuracy should be used. The metrics used for the performance assessment of these model must be resistant to errors.

5.2. Area under the curve (AUC)

The major model accuracy measures used is AUC. *AUC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance* (Fawcett, 2006).

This practically means that if the model randomly selects the "fraud transaction" and the "normal transaction" from our test set, the "fraud transaction" will have a higher score with a probability equal to the AUC.

This is essential because the objective is to ensure that fraudulent cases are highlighted with an anomaly score greater than that of a normal transaction.

The AUC of 0.7–0.9 is considered to be acceptable; an AUC of 0.5 is just as good as random guessing; a score of less than 0.5 is unacceptable.

5.3. Accuracy

The accuracy: the model positive hit rate. The formula is given below:

$$Accuracy = \frac{(TP + TN)}{(TN + FP + FN + FP)} \times 100\% \quad (1)$$

Where:

· TP is True positive and TN is True Negative; these are positive or negative cases that are classified correctly by the model.

· FP refers to False Positive; these are the negative cases that were classified incorrectly as positive.

· FN refers to False Negative, these are positive cases that were classified incorrectly as a negative.

6. Results and Discussions

Table 3. Performance comparison of various algorithms

Algorithm	Accuracy	Outlier Accuracy	AUC %
Iso. Forest	0.9392	**0.8675**	**94.6**
LOF	**0.9581**	0.3662	68.4
OCSVM	0.8005	0.3117	49.9

By comparing the results of Local Outlier Factor, Isolation Forest algorithm and One-class SVM, from table 3; it is clear that Local Outlier factor performs better at detecting non fraudulent transactions slightly better than Isolation Forest with an accuracy of 96%.

However, Isolation Forest performs better at detecting outliers or fraudulent transactions with an accuracy of 0.87.

One-class SVM performed fair at detecting non-fraudulent transactions but performed the worst of the three algorithms in detecting both fraudulent and non-fraudulent transactions.

Isolation Forest algorithm performs well in the case of credit card. Isolation forest has a 94.6% AUC score then LOF 68.4% and OCSVM 49.9%. The R

Table 4. Time Complexity of various algorithms

ALGORITHM	EXECUTION TIME
ISO. FOREST	1.56 S
LOF	1MIN 51S
OCSVM	6MIN 57S

Table 4 shows the time it took each of the algorithms to train on the data. Isolation Forest took the least time to complete training; OCSVM took the longest time to complete the training process. Isolation Forest is a time and cost-effective algorithm to use in detecting anomalies.

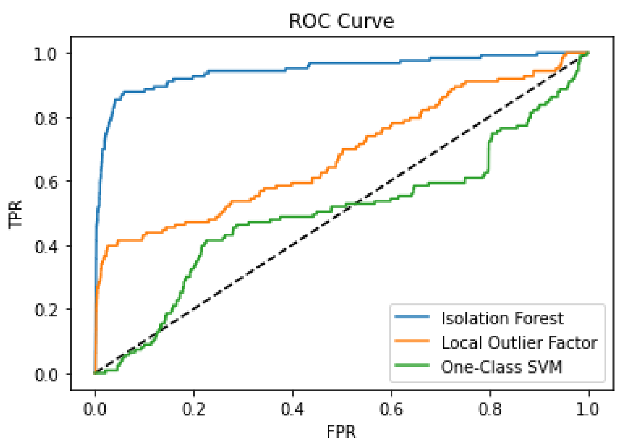


Figure 3. Receiver operating characteristic curve

Figure 3 shows the Receiver operating characteristic curve of the three algorithms, showing the relationship between sensitivity and specificity. The curve has the ”True Positive Rate” on the y-axis and the False Positive Rate on the x-axis. Based on this plot, Isolation Forest outperformed the other two algorithms.

7. Conclusion

This research assessed various methods for detecting fraudulent transactions such as OCSVM, LOF and Isolation Forest, in terms of accuracy, AUC score and false-positive system rate. Isolation Forest was proposed for detecting fraudulent credit card transactions. Experimentation has shown that the isolated forest is very effective in detecting anomalies in credit card transactions.

In the future, isolated forest could be used to detect fraudulent transactions in real-time.

References

Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pp. 93–104, 2000.

Chan, P. K., Fan, W., Prodromidis, A. L., and Stolfo, S. J. Distributed data mining in credit card fraud detection. *IEEE Intelligent Systems and Their Applications*, 14(6): 67–74, 1999.

Chen, R.-C., Chen, T.-S., and Lin, C.-C. A new binary support vector system for increasing detection rate of credit card fraud. *International Journal of Pattern Recognition and Artificial Intelligence*, 20(02):227–239, 2006.

Fawcett, T. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

Flarence, A. R., Bethu, S., Sowmya, V., Anusha, K., and Babu, B. S. Importance of supervised learning in prediction analysis. *Periodicals of Engineering and Natural Sciences*, 6(1):201–214, 2018.

John, H. and Naaz, S. Credit card fraud detection using local outlier factor and isolation forest. *Int. J. Comput. Sci. Eng.*, 7:1060–1064, 2019.

Liu, F. T., Ting, K. M., and Zhou, Z.-H. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pp. 413–422. IEEE, 2008.

Liu, F. T., Ting, K. M., and Zhou, Z.-H. Isolationforest: Isolation forest. *R package version 0.0-26/r4. URL: https://R-Forge. R-project. org/projects/iforest*, 2009.

Mehrotra, K. G., Mohan, C. K., and Huang, H. *Anomaly detection principles and algorithms*. Springer, 2017.

Nilsonreport.com. Nilson report. `https://nilsonreport.com/mention/407/1link/`, 2019. [Online; accessed 10-January-2021].

Sabinasz, D. Dealing with unbalanced classes in machine learning. *Deep Ideas*, 2017.

Schölkopf, B., Williamson, R. C., Smola, A. J., Shawe-Taylor, J., and Platt, J. C. Support vector method for novelty detection. In *Advances in neural information processing systems*, pp. 582–588, 2000.

Appendix A R Codes

```r

install.packages("readr")
install.packages("dplyr")
install.packages("ggplot2")
library(readr)
library(dplyr)
library(ggplot2)

#import data
creditData <- read_csv("creditcard.csv")

#preview the data
head(creditData)
#The data has 30 variables/features.
#The Class variable indicates if a transaction
#is fraud (positive case = 1) or genuine (negative case = 0).


#check for null values in data
anyNA(creditData)

#Data Summary
str(creditData)


#Calculate the percentage of fraud transactions
skew <- sum(as.numeric(creditData$Class))/nrow(creditData)
sprintf('Percentage of fraudulent transactions in the data set %f',
     skew*100)

#plot number of fraud cases against non fraud cases
ggplot(data=creditData, aes(x=Class)) +
  geom_bar() +
  geom_text(stat='count', aes(label=..count..), vjust=-.1)

#It is evident that the data is highly skewed
#with less than 0.2% of the transactions being fraud
#and the rest being legitimate.

#Analyze Fraud and Real Transaction summary
list <- split(creditData, creditData$Class)

#Analyze Normal Transactions
NormalTransactions <- list$`0`
summary(NormalTransactions$Amount)

#Analyze Fraud Transactions
FraudTransactions <- list$`1`
summary(FraudTransactions$Amount)
```

Appendix B Python Codes

In [156]:
```
!pip install ipython-autotime
import numpy as np
import pandas as pd
import sklearn
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, roc_auc_score, accuracy_score, average_
precision_score
%load_ext autotime
```

```
Requirement already satisfied: ipython-autotime in /usr/local/lib/python3.6/d
ist-packages (0.3.0)
Requirement already satisfied: ipython in /usr/local/lib/python3.6/dist-packa
ges (from ipython-autotime) (5.5.0)
Requirement already satisfied: pygments in /usr/local/lib/python3.6/dist-pack
ages (from ipython->ipython-autotime) (2.6.1)
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.4 in /usr/local/li
b/python3.6/dist-packages (from ipython->ipython-autotime) (1.0.18)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.6/dis
t-packages (from ipython->ipython-autotime) (4.3.3)
Requirement already satisfied: decorator in /usr/local/lib/python3.6/dist-pac
kages (from ipython->ipython-autotime) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.6/dist-p
ackages (from ipython->ipython-autotime) (0.7.5)
Requirement already satisfied: simplegeneric>0.8 in /usr/local/lib/python3.6/
dist-packages (from ipython->ipython-autotime) (0.8.1)
Requirement already satisfied: pexpect; sys_platform != "win32" in /usr/loca
l/lib/python3.6/dist-packages (from ipython->ipython-autotime) (4.8.0)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.6/d
ist-packages (from ipython->ipython-autotime) (51.1.1)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.6/dist-packa
ges (from prompt-toolkit<2.0.0,>=1.0.4->ipython->ipython-autotime) (0.2.5)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.6/dist-pa
ckages (from prompt-toolkit<2.0.0,>=1.0.4->ipython->ipython-autotime) (1.15.
0)
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.6/d
ist-packages (from traitlets>=4.2->ipython->ipython-autotime) (0.2.0)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.6/di
st-packages (from pexpect; sys_platform != "win32"->ipython->ipython-autotim
e) (0.6.0)
The autotime extension is already loaded. To reload it, use:
  %reload_ext autotime
time: 2.23 s (started: 2021-01-09 20:16:02 +00:00)
```

Import Data

In [157]:
```
data = pd.read_csv('creditcard.csv')
data = data.drop(['Time'] , axis=1)
```

```
time: 1.16 s (started: 2021-01-09 20:16:04 +00:00)
```

In [158]:
```
data.head()
```

Out[158]:

	V1	V2	V3	V4	V5	V6	V7	V8	V
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.36378
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.25542
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.51465
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.38702
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.81773

time: 53.5 ms (started: 2021-01-09 20:16:05 +00:00)

In [159]:
```
Fraud = data[data['Class']==1]
Nofraud = data[data['Class']==0]
outlier_fraction = len(Fraud)/float(len(Nofraud))
```

time: 21.8 ms (started: 2021-01-09 20:16:05 +00:00)

In [160]:
```
outlier_fraction
```

Out[160]: 0.0019275057574847303

time: 2.95 ms (started: 2021-01-09 20:16:05 +00:00)

In [161]:
```
data.isna().values.any()
```

Out[161]: True

time: 11.3 ms (started: 2021-01-09 20:16:05 +00:00)

Data Pre-processing

In [162]:
```
X = data.drop('Class',axis=1)
y = data['Class']
y = pd.DataFrame(y)
y.head(5)
```

Out[162]:

	Class
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0

time: 28.7 ms (started: 2021-01-09 20:16:05 +00:00)

In [163]:
```
#outlier dataframe to test model on Unsupervised
X_outliers = Fraud.drop(['Class'], axis=1)
len(X_outliers)
```

Out[163]:
```
385
```

```
time: 3.51 ms (started: 2021-01-09 20:16:05 +00:00)
```

In [164]:
```
#Split data into test and train set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .3, rand
om_state = 123)
```

```
time: 73.8 ms (started: 2021-01-09 20:16:05 +00:00)
```

In [165]:
```
#replace inifinty values in dataframe
X_test= np.nan_to_num(X_test)
y_test = np.nan_to_num(y_test)
```

```
time: 19 ms (started: 2021-01-09 20:16:06 +00:00)
```

Isolation forest

In [166]:
```
clf = IsolationForest(max_samples=100)

clf
```

Out[166]:
```
IsolationForest(behaviour='deprecated', bootstrap=False, contamination='aut
o',
                max_features=1.0, max_samples=100, n_estimators=100,
                n_jobs=None, random_state=None, verbose=0, warm_start=False)
```

```
time: 4.55 ms (started: 2021-01-09 20:16:06 +00:00)
```

Fit model on data

In [167]:
```
Iso_outliers = clf.fit(X_train)
```

```
time: 1.56 s (started: 2021-01-09 20:16:06 +00:00)
```

predictions

In [168]:
```
#Predict on train data
Iso_outliers_train = Iso_outliers.predict(X_train)
Iso_outliers_train
```

Out[168]:
```
array([1, 1, 1, ..., 1, 1, 1])
```

```
time: 4.59 s (started: 2021-01-09 20:16:07 +00:00)
```

In [169]:
```
#Predict on test data
Iso_outliers_test = Iso_outliers.predict(X_test)
Iso_outliers_test
```

Out[169]:
```
array([1, 1, 1, ..., 1, 1, 1])

time: 1.92 s (started: 2021-01-09 20:16:12 +00:00)
```

In [170]:
```
#Calcuate the decision function on the test data
y_score = - Iso_outliers.decision_function(X_test)
```

```
time: 1.91 s (started: 2021-01-09 20:16:14 +00:00)
```

In [171]:
```
#Predict on outlier data
Iso_outliers_pred = Iso_outliers.predict(X_outliers)
```

```
time: 61.8 ms (started: 2021-01-09 20:16:16 +00:00)
```

Isolation Forest Model Evaulation

In [172]:
```
#isolation Forest
print("Accuracy test :", list(Iso_outliers_test).count(1)/Iso_outliers_test.sh
ape[0])
print("Accuracy outliners:", list(Iso_outliers_pred).count(-1)/Iso_outliers_pr
ed.shape[0])
```

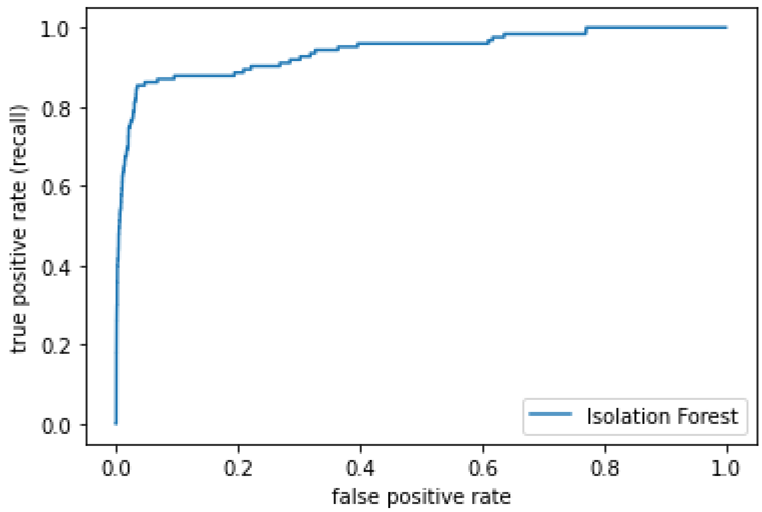
```
Accuracy test : 0.9391552017055864
Accuracy outliners: 0.8675324675324675
time: 26.4 ms (started: 2021-01-09 20:16:16 +00:00)
```

In [173]:
```
print("ROC AUC: %0.1f%%" % (roc_auc_score(y_test, y_score) * 100.))
```

```
ROC AUC: 94.6%
time: 28.7 ms (started: 2021-01-09 20:16:16 +00:00)
```

In [174]:
```
fp1, tp1, thres1 = roc_curve(y_test, y_score)
plt.plot(fp, tp, label="Isolation Forest")
plt.xlabel("false positive rate")
plt.ylabel("true positive rate (recall)")
plt.legend()
```

Out[174]:
```
<matplotlib.legend.Legend at 0x7f9a1a284438>
```




```
time: 189 ms (started: 2021-01-09 20:16:16 +00:00)
```

Local Outlier Factor

In [175]:
```
clf2= LocalOutlierFactor(n_neighbors=20, novelty=True)
clf2
```

Out[175]:
```
LocalOutlierFactor(algorithm='auto', contamination='auto', leaf_size=30,
                   metric='minkowski', metric_params=None, n_jobs=None,
                   n_neighbors=20, novelty=True, p=2)
```

```
time: 4.57 ms (started: 2021-01-09 20:16:16 +00:00)
```

Fit Model on data

In [176]:
```
lof_outliers = clf2.fit(X_train)
```

```
time: 1min 51s (started: 2021-01-09 20:16:16 +00:00)
```

In [177]:
```
#Predict on train data
lof_outliers_train = lof_outliers.predict(X_train)
lof_outliers_train
```

Out[177]:
```
array([1, 1, 1, ..., 1, 1, 1])
```

```
time: 1min 47s (started: 2021-01-09 20:18:08 +00:00)
```

In [178]:
```
#Predict on test data
lof_outliers_test = lof_outliers.predict(X_test)
lof_outliers_test
```

Out[178]:
```
array([1, 1, 1, ..., 1, 1, 1])
```

```
time: 46.2 s (started: 2021-01-09 20:19:55 +00:00)
```

In [179]:
```
#Calcuate the decision function on the test data
lof_y_score = - lof_outliers.decision_function(X_test)
```

```
time: 46.5 s (started: 2021-01-09 20:20:42 +00:00)
```

In [180]:
```
#Predict on outlier data
lof_outliers_pred = lof_outliers.predict(X_outliers)
```

```
time: 314 ms (started: 2021-01-09 20:21:28 +00:00)
```

Local Outlier Factor Model Evaulation

In [181]:
```
print("ROC AUC: %0.1f%%" % (roc_auc_score(y_test, lof_y_score) * 100.))
```

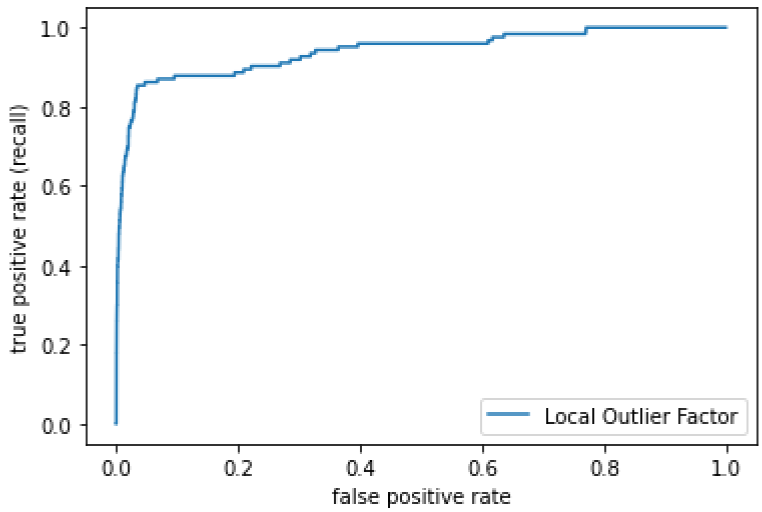
```
ROC AUC: 68.4%
time: 24.5 ms (started: 2021-01-09 20:21:28 +00:00)
```

In [182]:
```
#LOF
print("Accuracy test :", list(lof_outliers_test).count(1)/lof_outliers_test.sh
ape[0])
print("Accuracy outliners:", list(lof_outliers_pred).count(-1)/lof_outliers_pr
ed.shape[0])
```

```
Accuracy test : 0.9580932076351644
Accuracy outliners: 0.36623376623376624
time: 17.4 ms (started: 2021-01-09 20:21:28 +00:00)
```

In [183]:
```
fp2, tp2, thres2 = roc_curve(y_test, lof_y_score)
plt.plot(fp, tp, label="Local Outlier Factor")
plt.xlabel("false positive rate")
plt.ylabel("true positive rate (recall)")
plt.legend()
```

Out[183]:
```
<matplotlib.legend.Legend at 0x7f9a1cab2668>
```




```
time: 197 ms (started: 2021-01-09 20:21:28 +00:00)
```

One Class SVM

In [184]:
```
clf3 = OneClassSVM(nu=.2,kernel='linear',gamma=.001)
clf3
```

Out[184]:
```
OneClassSVM(cache_size=200, coef0=0.0, degree=3, gamma=0.001, kernel='linea
r',
            max_iter=-1, nu=0.2, shrinking=True, tol=0.001, verbose=False)
```

```
time: 4.58 ms (started: 2021-01-09 20:21:29 +00:00)
```

In [185]:
```
#Fit Model on train data
OneClassSVM = clf3.fit(X_train)
```

```
time: 6min 57s (started: 2021-01-09 20:21:29 +00:00)
```

In [186]:
```
#Predict on train data
OneClassSVM_train = OneClassSVM.predict(X_train)
OneClassSVM_train
```

Out[186]:
```
array([1, 1, 1, ..., 1, 1, 1])
```

```
time: 1min 55s (started: 2021-01-09 20:28:26 +00:00)
```

```
In [187]: #Predict on test data
          OneClassSVM_test = OneClassSVM.predict(X_test)
          OneClassSVM_test

Out[187]: array([-1,  1,  1, ...,  1,  1,  1])

          time: 49.1 s (started: 2021-01-09 20:30:21 +00:00)
```

```
In [188]: #Calcuate the decision function on the test data
          OneClassSVM_y_score = - OneClassSVM.decision_function(X_test)

          time: 49.1 s (started: 2021-01-09 20:31:10 +00:00)
```

```
In [189]: #Predict on outlier data
          OneClassSVM_pred = OneClassSVM.predict(X_outliers)

          time: 332 ms (started: 2021-01-09 20:31:59 +00:00)
```

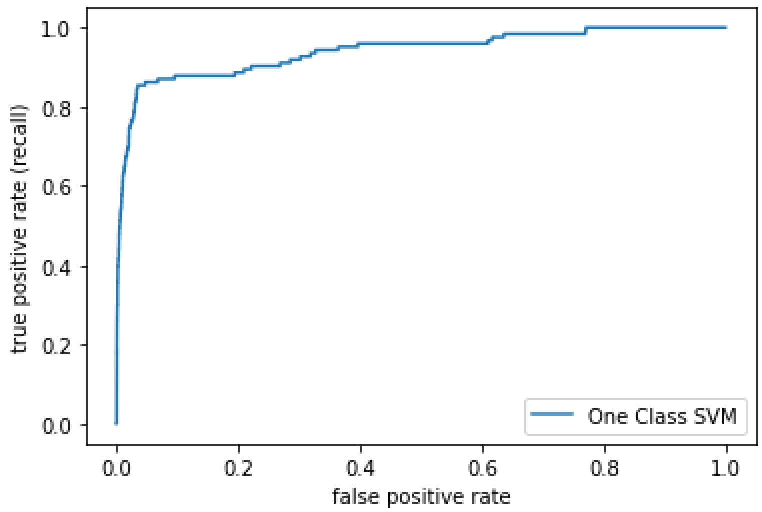
One Class SVM Evaluation

```
In [190]: print("ROC AUC: %0.1f%%" % (roc_auc_score(y_test, OneClassSVM_y_score) * 100.
          ))

          ROC AUC: 49.9%
          time: 24.9 ms (started: 2021-01-09 20:32:00 +00:00)
```

```
In [191]: print("Accuracy test :", list(OneClassSVM_test).count(1)/OneClassSVM_test.shap
          e[0])
          print("Accuracy outliners:", list(OneClassSVM_pred).count(-1)/OneClassSVM_pred
          .shape[0])

          Accuracy test : 0.8005429894400213
          Accuracy outliners: 0.3116883116883117
          time: 21.2 ms (started: 2021-01-09 20:32:00 +00:00)
```

In [192]:
```
fp3, tp3, thres3 = roc_curve(y_test, OneClassSVM_y_score)
plt.plot(fp, tp, label="One Class SVM")
plt.xlabel("false positive rate")
plt.ylabel("true positive rate (recall)")
plt.legend()
```

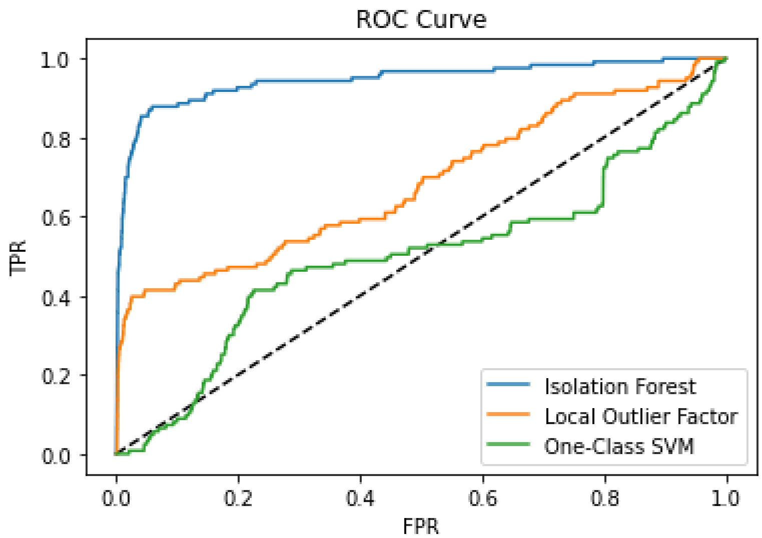
Out[192]: <matplotlib.legend.Legend at 0x7f9a1c8a3860>



time: 188 ms (started: 2021-01-09 20:32:00 +00:00)

Plot 3 ROC Curves together

In [193]:
```
plt.plot([0,1],[0,1], 'k--')
plt.plot(fp1, tp1, label= "Isolation Forest")
plt.plot(fp2, tp2, label= "Local Outlier Factor")
plt.plot(fp3, tp3, label= "One-Class SVM")
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title('ROC Curve')
plt.show()
```




```
time: 179 ms (started: 2021-01-09 20:32:00 +00:00)
```