Detecting Outliers in Datasets Using Machine Learning

Reem Omer Mohammed Elmahdi (reemomer@aims.ac.za)
ريم عمر محمد المهدي
African Institute for Mathematical Sciences (AIMS)

Supervised by: Dr. Michelle Lochner
African Institute for Mathematical Sciences - Research Centre, South Africa

18 May 2017
Submitted in partial fulfillment of a structured masters degree at AIMS South Africa

Abstract

Detecting outliers in datasets is crucial in many domains. While in many cases outliers are considered a nuisance in the dataset, other times they actually contain important information. There are various Machine Learning methods which have been developed to detect outliers, including Local Outlier Factor and One Class Support Vector Machine.

The purpose of this research is to compare Local Outlier Factor and One Class Support Vector Machines models in order to find the better performing algorithm in a highly unbalanced dataset. The dataset was taken from the Kaggle data science repository. Specifically, a fraudulent credit card dataset was used to evaluate the models.

Evaluation of these models is based on True Positive Rate and False Positive Rate. Comparing results of the algorithms shows that the One Class Support Vector Machine model performs better than the Local Outlier Factor model in the given dataset.

الملخص

تتأثر نتائج التحليل الإحصائي بشكل كبير بوجود قيم شاذة أو متطرفة في البيانات، و بالتالي تظهر أهمية تحديد هذه القيم. حيث يعتبر تعليم الآلة من أفضل النماذج لتحديد هذه القيم. يناقش هذا البحث نموذجين من نماذج تعليم الآلة: أولا نموذج العوامل المحلية الشاذة. ثانيا نموذج آلة الدعم الموجه ذات الفئة الواحدة. حيث تتم مقارنة نتائج النموذجين من خلال تحديد المعدل الإيجابي الصحيح و المعدل الإيجابي الخاطئ؛ لمعرفة أي النموذجين أفضل لتحديد القيم الشاذة.

Declaration

I, the undersigned, hereby declare that the work contained in this research project is my original work and that any work done by others or by myself previously has been acknowledged and referenced accordingly.

Reem Omer Mohammed Elmahdi, 18 May 2017

Contents

1. Introduction

Outlier detection is the process of finding patterns in data that do not adhere to normal behavior. These patterns are known as outliers or anomalies (Chandola et al., 2009). Detection of outliers has applicability in various domains such as fraud detection for credit card and insurance, intrusion detection for cyber security and industrial damage detection. It is important to detect the outliers in a set of data because they can be translated into actionable information in various applications. In fact, there are different outlier detection techniques that have been developed for specific domains. These techniques can operate in supervised learning, semi-supervised learning and unsupervised learning (Singh and Upadhyaya, 2012). According to the number of data dimensions, outlier detection methods are divided into univariate and multivariate methods (MR Tight and Clark, 1993; Knox and Ng, 1998). Of all the different methods available for outlier detection, machine learning methods have the highest success rates for detecting outliers.

This research attempts to test two machine learning outlier detection models (Silvia Cateni and Vannucci, 2008): Local Outlier Factor and One Class Support Vector Machine on a real dataset to automatically detect credit card fraud. Models will be discussed in Chapter 2 and Chapter 3 respectively. The models are compared using True Positive Rates and False Positive Rates they obtain. Chapter 4 contains the research methodology, results and evaluation of the models.

This research will help to understand the performance of outlier detection methods using credit card transactions dataset, where the outliers are fraudulent transaction. Conclusions and suggestions for future work are presented in Chapter 5.

The reminder of this chapter covers the basic terminology and concepts of outlier detection, outlier detection techniques and methods, and relevant machine learning concepts.

1.1 Defining Outliers

An outlier is an observation (or subset of observations) which appears to be inconsistent with the dataset (Barnett and Lewis, 1994). Figure 1.1 illustrates outliers in a 2-dimensional dataset. Clearly, o_1, o_2 and O_3 are the points that are far from other points N_1 and N_2 which contain most of the data.

Outliers are often treated as errors or noise in datasets. Researchers may either reject or delete outliers in the datasets. However, sometimes outliers indicate something scientifically interesting or significant and as such may be useful.

Outliers may appear in datasets because of manual errors in the transmission or transcription process, changes in the behavior of the system or through deviations in populations. Alternatively, outliers could be caused by a flaw in the theory. Outliers may cause a negative effect on data analyses, so it is very important to define them (Outlier).

Types of Outliers: Outliers can be classified into three categories:

- Point Outliers: This is the simplest type of outlier. An outlier is a point when an individual or a group of data patterns can be considered as unusual compared to the rest of the data (Outlier). Point o_1 and group O_3 in Figure 1.1 are examples of point outliers.

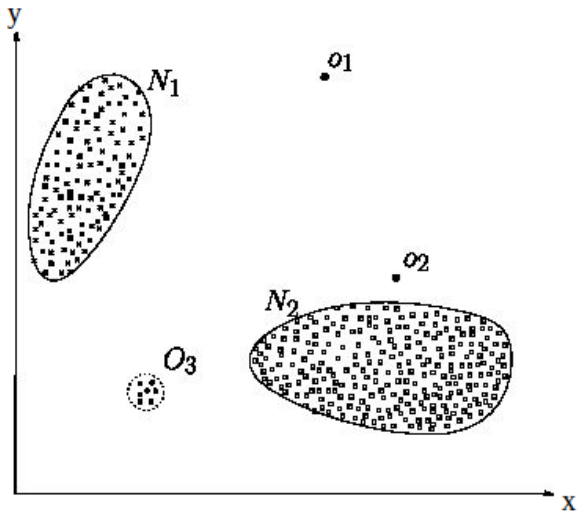

Figure 1.1: Observations in 2-dimensional space (Singh and Upadhyaya, 2012).

- Contextual Outliers: These are outliers found when observations are unusual in a specific context and are caused by the structure in the dataset. An observation might be an outlier in a certain context, but normal in a different context (Singh and Upadhyaya, 2012). There are two sets of attributes that define data patterns: contextual attributes and behavioral attributes. An example of contextual outlier is a sudden drop of temperature (assume 10F) in summer time is an outlier. But it considered as normal during winter.
- Collective Outliers: These are collected data patterns which are unusual in the dataset. Individual data point in the collective outlier may not be an outlier by itself. In fact, the occurrence of the data points together is unusual. Point outliers can appear in any dataset, whereas collective outliers appear only in datasets where the data patterns are related (Singh and Upadhyaya, 2012). The unusual regions on a human electrocardiogram is an example of collective outliers.

1.2 Outlier Detection

Most of the background in this section is a summery of (Singh and Upadhyaya, 2012). Where details about outliers applications and techniques may be found.

Outlier detection is the process of identifying the unusual observations in the datasets. However there are many reasons that make detecting outliers challenging. First of all, the unavailability of labeled data to apply in training and validation of the model. Secondly, the encompassing of all normal behavior in the region. In addition, the uncertainty of the boundary between normal and outlier behavior. At the top of that, the difficulty of applying a developed technique from one domain to another because the problem formulation of outlier detection is different.

The input of an outlier detection process is a collection of observations, where each one of them is a set of features of different types. These types can be binary, categorical or continuous. To distinguish between normal data and outliers in a dataset labels are used. Labels should be accurate and should represent all types of behaviors. It is easier to label normal behavior of the observation, than anomalous data, this is due to the dynamic nature of the anomalous data, for example arrival of new types of outliers which are not labeled (Chandola et al., 2009). Based on the extent to which labels are available, outlier detection techniques can operate in different modes.

Modes of Outlier detection techniques

- Supervised Outlier Detection technique assumes the availability of labeled training datasets. The labels are either normal or outlier; which helps the predictive model in classification process. This technique faces a major issue which is the unlimited number of unusual data in training sets, as well as the accuracy of the data labels. Some techniques inject artificial outliers in the normal data (Singh and Upadhyaya, 2012).
- Semi-Supervised Outlier Detection technique assumes that training data has labeled patterns for the normal data, and is commonly used. There are some limited techniques that use labeled outliers instead of labeled normal data (Singh and Upadhyaya, 2012).
- Unsupervised Outlier Detection is the most flexible technique due to non-essentiality of training data. The assumption made in this technique is that outliers are far more than normal patterns in the dataset (Singh and Upadhyaya, 2012).

Output of Outlier Detection

Outlier detection techniques produce the output in one of the following types:

- Scores: This technique assign scores for all of the observations in the test data based on the degree to which that observation is estimated to be an outlier. Therefore, the output will be a ranked list of all outliers which help the analyst to select the outliers (Singh and Upadhyaya, 2012).
- Labels: This technique assigns the label normal or outlier for each observation in the dataset, in this case, the analyst has no control on the output (Singh and Upadhyaya, 2012).

Outlier Detection Methods

- Univariate Methods: There are different statistical theory methods to detect outliers in univariate datasets such as Chauvenet's criterion, Dixon's Q test and Grubb's test for outliers. However, there are major weaknesses of these approaches: The dataset may not follow Gaussian distribution. There are some methods use the mean and the variance to define outliers. However, mean and variance are outlier dependant (Seo, 2006). These methods may include sample minimum or sample maximum which are not always outliers.
- Multivariate Methods: There are many methods to detect outliers in multivariate datasets, such as Robust Statistical-based Methods (e.g. Mahalanobis Distance method), Distance-based Methods (e.g. Distance to k^{th} nearest neighbour method), Density-based Methods (e.g. Local Outlier Factor) and Artificial Intelligence Methods (e.g. Artificial Neural Networks and Support Vector Machines).

1.3 Outlier Detection Applications

There are several applications of outlier detection in different domains among them being the following:

- Fraud Detection: These are the fraudulent applications for credit cards, mobile phones and insurance claims (Singh and Upadhyaya, 2012). The unauthorized usages may appear in different ways, such as a purchase from outside experienced geographical locations, credit card large amount transaction data and unauthorized and illegal insurance claims.

- Intrusion Detection: Is the detection of malicious activity (break-ins, penetrations, and other forms of computer abuse) in a computer related system which is interesting from a computer security perspective (Chandola et al., 2009). Detection of malicious activity in operating system calls, network traffic, or other unauthorized access activity in the system of host-based or networked computer systems.
- Industrial Damage Detection: This domain uses recorded sensors data which monitor the performance of industrial components (Singh and Upadhyaya, 2012), then these systems classify the defects that might occur due to wear and tear or any other reason (Chandola et al., 2009).

1.4 Machine Learning

Machine Learning (ML) is the field of study that gives computers the ability to learn without being explicitly programmed (Ng, 2017). machine learning is one of the fields of artificial intelligence that focuses on building smart systems such as autonomous robotics, pattern recognition systems, Natural Language Processing(NLP) systems, computer vision and so forth.

ML automatically builds a highly non-linear and general model of the data. **Types of ML**

- **Supervised Learning:** Algorithms of this type take the input data as well as the correct output, and it associates the datasets and predicts better answers. The problems of these algorithms could be **regression** if we want to predict continuous valued output, an example of this is an algorithm that is used to predict the house price based on the size of the house. There is also a **classification** problem which is focuses on predicting discrete values, for example, if we want to decide whether a tumor type is either malignant or benign using tumor size (Ng, 2017).
- **Unsupervised Learning:** This type of algorithm finds structure out of the dataset by partitioning/clustering data. Unsupervised learning algorithm is used in clustering genes data, where the algorithm groups the genes based on how they respond to different experiments (Ng, 2017).

ML in Outlier detection

There are various models in machine learning that are used in outlier detection. The supervised detection technique requires a labeled training data which contains the normal observations and the outliers. It uses the testing data to test the model and find the results. Semi-supervised detection technique requires normal labeled training data and it uses the testing data to test the model and find results. While unsupervised detection technique doesn't require neither training data nor testing data (Goldstein M, 2016).

This research will focus on the methods of semi-supervised outlier detection technique.

2. Local Outlier Factor

It is one of the density based outlier detection methods (Silvia Cateni and Vannucci, 2008). The idea of the Local Outlier Factor (LOF) method is to assign a probability for each object/observation of being an outlier (Breunig et al., 2000). The outlier factor is local because it takes into account only the neighbors of each observation, therefore, it doesn't require comparing observations with the global data distribution (LOF). Figure 2.1 shows the basic idea of LOF of a point with densities of its neighbors.

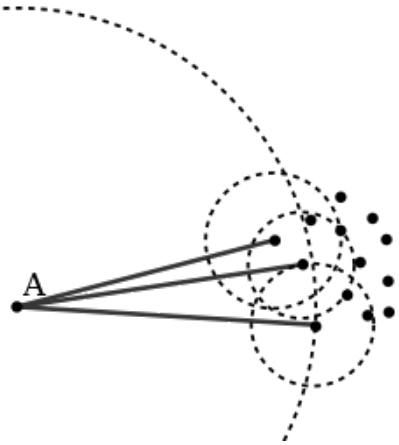

Figure 2.1: Point A has a much lower density than its neighbors (LOF).

The LOF of an observation is based on one parameter k, which is the number of nearest neighbors that were considered to the observation (Breunig et al., 2000). Developing the LOF method requires some notations and definitions which will be described in this section. Consider p, q, o as observations in the dataset; the distance between observation p and q is given by $d(p, q)$. Whereas C denote a cluster and D is a dataset, then $d(p, C) = \min\{d(p, q), q \in C\}$ (Knox and Ng, 1998).

2.0.1 Definition (Global Outlier in dataset D)**.** An observation p in a dataset D is a Distance-based(pct, k)-outlier if at least a percentage (pct) of the observations in D lies larger than distance k from p (Breunig et al., 2000).

Definition 2.0.1 captures only some of the outliers (global outliers) which are far from the clusters of the dataset. However, in some datasets the interest lies in the observations that are outliers relative to their local neighbourhoods with respect to neighbourhood densities (local outliers). LOF method use a previous concepts to assign degrees to observations of being outlying, unlike other algorithms that use a binary property, therefore, the observation is either normal point or an outlier.

2.0.2 Definition (k-Distance of an object p)**.** For any positive k, $k - distance(p)$ is defined as the distance $d(p, o)$ between p and an object $o \in D$ such that:

$$\begin{aligned} &\text{for at least } k \text{ objects } o' \in D \setminus \{p\} \text{ it holds that } d(p, o') \leq d(p, o), \\ &\text{for at least } k-1 \text{ objects } o' \in D \setminus \{p\} \text{ it holds that } d(p, o') < d(p, o). \end{aligned} \tag{2.0.1}$$

In this step the distance that is the farthest to k observations that near to p is defined (Breunig et al., 2000).

2.0.3 Definition (k-Distance Neighbourhood of an object p)**.** The k-distance neighbourhood is the k-distance of p neighbors (Breunig et al., 2000).

$$N_k(p) = \{q \in D \ \{p\} | d(p,q) \leq k - distance(p)\} \tag{2.0.2}$$

2.0.4 Definition (Reachability Distance of an Object p with respect to object o)**.** The reachability distance is the distance between p and o but at least the k-distance(o) (LOF).

$$reach - dist_k(p,o) = \max\{k - distance(o), d(p,o)\} \tag{2.0.3}$$

The reachability distance step 2.0.4 is used to smooth statistical fluctuations (Breunig et al., 2000); and it is a replacement for Euclidean distance. If the Euclidean distance between the two points is very small, the following steps will give a bias ratio of distance. Therefore, the LOF algorithm uses reachability distance instead (Mathew X. Ma and Liu, 2016).

2.0.5 Definition (Local Reachability Density of an object p)**.** It is an inverse of the average reachability distance of an object p from its neighbours of p (LOF):

$$lrd(p) = 1/\left(\frac{\sum_{o \in N_k(p)} reach - dist_k(p,o)}{|N_k(p)|}\right) \tag{2.0.4}$$

In Figure 2.2 the reachability distance of the observation of p_1 is different than the reachability distance of observation p_2, because p_2 is not one of the k-nearest neighbour of o.

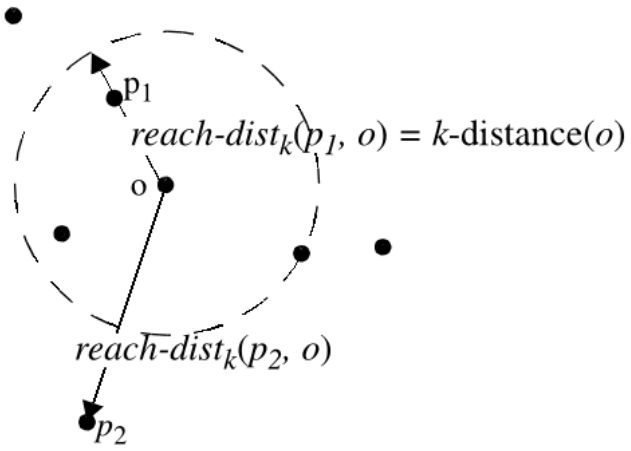

Figure 2.2: Reachability density (k=3) of two points p_1 and p_2 w.r.t. point o (LOF).

If the summation of reachability distances is 0, then the local density tends to infinity (∞), which can only happens if there are two or more observations that share the same spatial coordinates (Breunig et al., 2000).

2.0.6 Definition (Local Outlier Factor for an object p)**.** This step compares the local reachability density of an observation with the local reachability density of its neighbours, which captures the degree to which p is called an outlier (Cui, 2005). The higher the value of LOF, the more likely it is to be an outlier, in other words, if density around p is lower than density around p's neighbors, then p is an outlier (Breunig et al., 2000).

$$LOF_k(p) = \frac{\sum_{o \in N_k(p)} \frac{lrd_k(o)}{lrd_k(p)}}{|N_k(p)|} \tag{2.0.5}$$

2.1 Properties of LOF

2.1.1 LOF for Observations Deep in a Cluster: The LOF value for the observations that is in the cluster C is approximately equal to 1, indicating that they shouldn't be labeled as outliers.

2.1.2 Lemma. (Breunig et al., 2000) Let C be a collection of observations. Let the minimum reachability distance and the maximum reachability distance of observations in C be $\min\{reach-dist(p,q)|p,q \in C\}$ and $\max\{reach-dist(p,q)|p,q \in C\}$ respectively. Let ϵ be defined as $min-reach-dist/max-reach-dist-1$. Then for all observations $p \in C$ it holds:

$$1/(1+\epsilon) \leq LOF(p) \leq (1+\epsilon) \tag{2.1.1}$$

such that:

$$\forall N_k(p) : N_k(p) \in C \tag{2.1.2}$$
$$\forall N_k(q) : N_k(q) \in C \tag{2.1.3}$$

Consider a tight cluster C, the ϵ value in Lemma 2.1.2 can be quite small for deep observations p, thus it forces the LOF of p to be close to 1 (Breunig et al., 2000).

2.1.3 Upper and Lower Bound on LOF: Lemma 2.1.2 shows that the LOF value for the deep observations is approximately equal to 1. The observations outside and in the border of the cluster C; as well as upper and lower bound of the LOF value should be obtained by using Theorem 2.1.4 which requires the following terms (Breunig et al., 2000):

- Minimum reachability distance between p and k-nearest neighbour of p is

$$direct_{min}(p) = min\{reach-dist(p,q)|q \in N_k(p)\} \tag{2.1.4}$$

- Maximum reachability distance between p and k-nearest neighbour of p is

$$direct_{max}(p) = max\{reach-dist(p,q)|q \in N_k(p)\} \tag{2.1.5}$$

- Minimum reachability distance between q and k-nearest neighbour of q is

$$indirect_{min}(p) = min\{reach-dist(q,o)|q \in N_k(p) \text{ and } o \in N_k(q)\} \tag{2.1.6}$$

- Maximum reachability distance between q and k-nearest neighbour of q is

$$indirect_{max}(p) = max\{reach-dist(q,o)|q \in N_k(p) \text{ and } o \in N_k(q)\} \tag{2.1.7}$$

2.1.4 Theorem. *Let p be an observation from the dataset D, and $1 \leq k \leq |D|$ (Breunig et al., 2000):*

$$\frac{direct_{min}(p)}{indirect_{max}(p)} \leq LOF(p) \leq \frac{direct_{max}(p)}{indirect_{min}(p)} \tag{2.1.8}$$

Theorem 2.1.4 is a function of the reachability distances in p that is in direct neighbourhood relative to those in p that is in indirect neighbourhood. It could be applied to all the observations p in the dataset D and it can give tighter bounds for deep observations than Lemma 2.1.2 does, and this because Theorem 2.1.4 is based on k-nearest neighbours, unlike Lemma 2.1.2 which is based on minimum and maximum reachability distances (Breunig et al., 2000).

2.1.5 Bounds Tightness: Tightness of the bounds depends on the nature of the point under consideration. It defines the difference is between the LOF upper and lower bounds. If there is a fluctuation of the average reachability distance in the direct and indirect neighbourhoods that is small, then the bounds are tight. But if the observation p has neighbours in multiple clusters having different densities, then the bounds are not tight (Breunig et al., 2000).

2.1.6 impact of k on LOF: The value of LOF is affected by the parameter k, but k has unpredictable impact on LOF, and this results in problems. However, there is a heuristic method determines bounds for k, that helps to avoid the previous weakness. The method proposed by Markus M. Breunig and others (Breunig et al., 2000).

2.2 LOF Algorithm

The algorithm described below (Malak Alshawabkeh, 2010) computes the LOF(p) for all $p \in D$, Definition 2.0.6 with details to local reachability density.

Algorithm 1 Local Outlier Factor algorithm

Input:
k ▷ The number of neighbours
D ▷ A set of observations

Output:
$lofvalues$ ▷ This algorithm returns a vector with local density factors

ASSUME: ,
$k - dist(D, p)$ ▷ Return a matrix the k-dist neighbours 2.0.2 and their k-distances 2.0.3
$reach - dist_k(p)$ ▷ Return the local reachability density of each $p \in D$ 2.0.4

1: **procedure** LOF(k, D)
2: $lof \leftarrow NULL$
3: **for** each point p **do** ▷ Assign value to k-th nearest neighbours
4: $kNNeighbours \leftarrow k - dist(D, k)$
5: $lrd \leftarrow reach - dist_k(KNNeighbours, k)$
6: **for** each p in kNNeigbours **do**
7: $\max\{lof, templof\}$ ▷ Find the reachability distance between p and k-neighbours 2.0.4
8: $templof[i] \leftarrow \sum((lrd[o \in N(p)])/lrd[i])/|N(p)|$
▷ This step computes the local reachability density of point p 2.0.5
9: **end for**
10: **end for**
11: **return** $\top(lof)$ ▷ return the top values of the lof vector 2.0.6
12: **end procedure**

3. Support Vector Machines

Support Vector Machines (SVMs) method is one of the Artificial Intelligence Outlier Detection methods (Silvia Cateni and Vannucci, 2008) and it can be applied in classification problems. The main idea of SVMs is to find the optimal hyperplane in feature space that best separates classes (Oded Maimon, 2010). In this section classifiers concepts will be described, then it will extended to One Class Support Vector Machine (OCSVM).

3.1 Basics of SVMs

To describe the method consider the set $\Omega = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$; where $i \in \mathbb{R}^+$ is ith element in the set, $x_i \in \mathbb{R}^d$ is the input observation number i in d dimensional space and $y_i \in \{-1, 1\}$ is the output number i, which indicate the class membership (Vlasveld, Accessed April 2017). SVM uses a hyperplane to separate the observations into classes. SVM aims to maximize the margin $\mathcal{M}$ to get the optimal result. In Figure 3.1 observations are in 2-dimensional space which belongs to one of two classes. The observations can be separated by multiple hyperplanes, but the hyperplane that has the maximum distance between the observations and the optimal hyperplane is in Figure 3.2. Figures are reproduced from (SVM, Accessed May 2017).

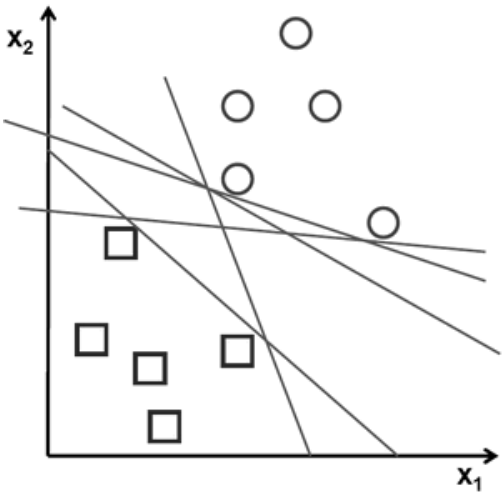

Figure 3.1: The possible hyperplanes with its different distances from the observations, some hyperplanes are close to one observations of the classes.

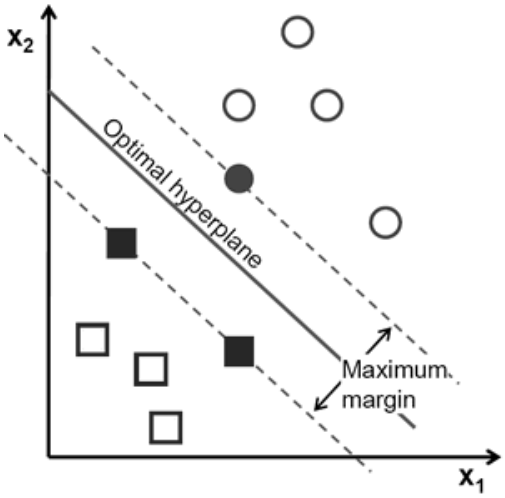

Figure 3.2: The optimal hyperplane the one that has maximum distances between the classes, it shouldn't also biased to one side.

SVM has a low computational cost of evaluating the decision function, which is the function that used to classify the observations. SVM doesn't take all of the training examples into account, but a subset of it which is called support vectors (Oded Maimon, 2010). The support vector is chosen to be the closest observations to the optimal hyperplane. In Figure 3.2 the filled shapes are referred to support vectors in that space.

Hyperplane Classifiers

The objective of classification in SVM is to finds a function $f : \mathbb{R}^N \rightarrow \{-1, +1\}$ by using the training observations according to the unknown probability distribution $P(x, y)$ of the observations $\{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{R}^N \times Y, Y = \{-1, +1\}$ such that f will classify the coordinates of a new test observation (x, y) correctly (Oded Maimon, 2010).

- **The Linear Classifier**

  Let Ω be a linearly separable observations, the hyperplane will be defined as:

  $$w.x + b = 0, \quad w \in \mathbb{R}^N, b \in \mathbb{R} \tag{3.1.1}$$

  Which corresponds to:

  $$f(x) = \text{sign}((w.x) + b) \tag{3.1.2}$$

  Where w is the weight vector and b is the bias and sign is either positive or negative.

  The margin $\mathcal{M}$ is the largest distance between two parallel hyperplanes that separate the data into two classes. $\mathcal{M}$ is given by:

  $$\mathcal{M} = \frac{2}{||w||} \tag{3.1.3}$$

  Note that the margin depends on the weight vector. Maximizing $\mathcal{M}$ is equivalent to minimizing $||w||$ value. A constraint to prevent observations from falling into the margin should be introduced. The problem becomes:

  $$\min_{w,b} \frac{1}{2}||w||^2 \tag{3.1.4}$$

  $$\text{s.t. } y_i.((w.x_i) + b) \geq 1, \quad i = 1, \ldots, n \tag{3.1.5}$$

  Since this problem is convex optimization, it can be solved using the Lagrange function (Oded Maimon, 2010) as follows:

  $$\mathcal{L}(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \left(\alpha_i \Big[y_i((w.x_i) + b) - 1 \Big] \right) \tag{3.1.6}$$

  Where α_i is the Lagrange multiplier. And this has to be minimized with respect to primal variables w, b:

  $$\frac{\partial \mathcal{L}}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{3.1.7}$$

  $$\frac{\partial \mathcal{L}}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{3.1.8}$$

  The value of $\alpha_i = 0$ for all the constraints in Equation 3.1.6, which means the values of α_i can be eliminating from the equation without affecting the solution except for support vector (by using Karush-Kuhn-Tucker (KKT) conditions of optimization), thus:

  $$\alpha_i(y_i.((w.x_i) + b) - 1) = 0, \quad i = 1, \ldots, n \tag{3.1.9}$$

The Lagrangian ($\mathcal{L}$) can also be maximized with respect to the dual variables α_i. In fact, solving the dual problem may be easier than the primal problem; because they may have the same values under some conditions (Oded Maimon, 2010). The dual problem can be obtained by substituting back 3.1.7 and 3.1.8 value in Lagrangian function 3.1.6:

$$\mathcal{L} = \frac{1}{2}\left[\Big(\sum_{i=1}^{n}\alpha_i y_i x_i\Big).\Big(\sum_{j=1}^{n}\alpha_j y_j x_j\Big)\right] - \left[\Big(\sum_{i=1}^{n}\alpha_i y_i x_i\Big).\Big(\sum_{j=1}^{n}\alpha_j y_j x_j\Big)\right] + \sum_{i=1}^{n} y_i\alpha_i b + \sum_{i=1}^{n}\alpha_i \tag{3.1.10}$$

And from 3.1.8 the term $\sum_{i=1}^{n} y_i\alpha_i b$ is equal to 0. The first two terms differ by a factor, thus it becomes:

$$\mathcal{L} = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i x_j \tag{3.1.11}$$

$$\text{s.t. } \alpha_i \geq 0, i = 1, \ldots, n, \sum_{i=1}^{n}\alpha_i y_i = 0$$

then the decision function in 3.1.2 will be:

$$f(x) = \text{sign}\left(\sum_{i=1}^{n}\alpha_i y_i (x.x_i) + b\right) \tag{3.1.12}$$

And by using 3.1.9 and the support vectors $x_i, i \in S = \{i : \alpha_i \neq 0\}$ the value of b becomes:

$$b = \frac{1}{S}\sum_{i\in S}\left(y_i - \sum_{j=1}^{n}\alpha_j y_j (x_i.x_j)\right) \tag{3.1.13}$$

- **Kernel Function**

  When the observations are not linearly separable in the input space, they should be projected in higher dimension space, this makes the observations separable by using a hyperplane. The kernel function (K) is used to project the observations from the input space to higher feature space F, where feature space is non-linear mapping $K : \mathbb{R}^N \to \mathbb{R}^M$ where $M > N$ from the input space (Oded Maimon, 2010). In this space a hyperplane can be used to separate the observations, then that hyperplane would be projected back in the input space, and it would have a non-linear curve. Figure 3.3 illustrate the idea.

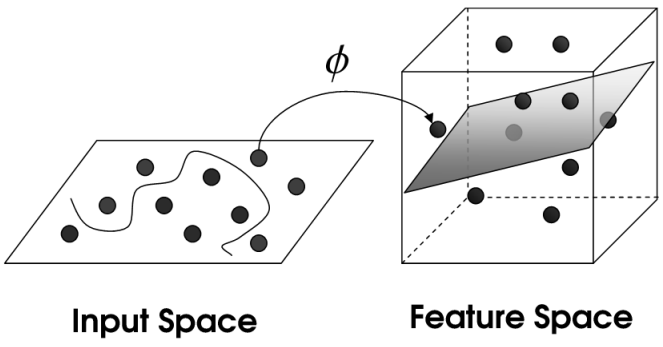

Figure 3.3: Kernel function K projected the observations from input space to feature space (Hyp, Accessed May 2017).

The Kernel computes the inner product of the vectors without knowing the function K on it self (Modise, 2014). There are different choices of kernel function which gives a different classification accuracy. Some of them listed below:

- Linear: $K(x, x_i) = x^T x_i$
- Polynomial: $K(x, x_i) = (\gamma x.x_i + C)^d$
- Sigmoidal: $K(x, x_i) = tanh(\gamma x^T x_i + C)$
- Gaussian Radial Base function (RBF): $K(x, x_i) = \exp\Big(-\gamma||x - x_i||^2\Big)$ where $\gamma \in \mathbb{R}$ is kernel parameter, and $||x - x_i||$ is the dissimilarity measure.

Where C is the parameter that controls the influence of individual observations; It is also called smoothness parameter because it controls the trade-off between smooth decision boundary and classifying trading observations correctly (OCS, Accessed May 2017).

- **Non Separable SVM**

Sometimes the observations may have high noise levels leading the classes to overlap. The previous case was for separable observations, so extending the capabilities of hyperplane classifier so the problem of noise can be solved. (Oded Maimon, 2010). Equation 3.1.4 can be extended by introducing the slack variables ξ_i which allow the training data to violate the constraint in Equation 3.1.6. A soft classifier is used in the non separable case, and controls the classifier capacity and the sum of the slacks $\sum_{i=1}^{n} \xi_i$, the soft classifier minimizes the following function:

$$\min_{w,b} \frac{1}{2}||w||^2 + C\sum_{i=1}^{n} \xi_i \tag{3.1.14}$$
$$\text{s.t. } \xi_i \geq 0, y_i(wx_i + b) \geq 1 - \xi_i \quad \forall i.$$

The value of ξ_i can be one of three options, $\xi_i = 0$, then x_i on the right side of the margin. $0 < \xi_i < 1$, then x_i on the right side of the margin but its distance is less than the margin $\mathcal{M}$. $\xi_i > 1$, then ξ_i is on the wrong side (Modise, 2014). The difference between the separable and the non-separable case is the upper bound C on the Lagrange multipliers α_i, where $C > 0$ is the regularization constant that determines the balance between the empirical risk and the complexity term (Oded Maimon, 2010). The Lagrange function becomes:

$$\mathcal{L}(w, b, \xi_i) = \frac{1}{2}||w||^2 + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i\{y_i(wx_i + b) - (1 - \xi_i)\} - \sum_{i=1}^{n} \beta_i\xi_i \tag{3.1.15}$$

And this has to be minimized with respect to variables w, b, ξ_i:

$$\mathcal{L} = \frac{1}{2}w.w + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n}(\alpha_i y_i w x_i + \alpha_i y_i b - \alpha_i + \alpha_i\xi_i) - \sum_{i=1}^{n} \beta_i\xi_i$$

$$\frac{\partial\mathcal{L}}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{3.1.16}$$

$$\frac{\partial\mathcal{L}}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{3.1.17}$$

$$\frac{\partial\mathcal{L}}{\partial \xi_i} = C - \alpha_i - \beta_i = 0 \rightarrow \beta_i = C - \alpha_i \quad \forall i. \tag{3.1.18}$$

By using Karush-Kuhn-Tucker (KKT) conditions of optimization to minimize $\mathcal{L}$, which include the Equations 3.1.16, 3.1.17, 3.1.18 and:

$$\alpha_i(y_i(wx_i + b) - (1 - \xi_i)) = 0 \tag{3.1.19}$$
$$y_i(wx_i + b) - (1 - \xi_i) \geq 0 \tag{3.1.20}$$
$$\beta_i\xi_i = 0 \tag{3.1.21}$$

Now by using Equation 3.1.18, 3.1.19, 3.1.20 and 3.1.20 the following values obtained:

$$\alpha_i = 0 \rightarrow y_i(wx_i + b) \geq 1$$
$$\alpha_i = C \rightarrow y_i(wx_i + b) \leq 1$$
$$0 < \alpha_i < C \rightarrow y_i(wx_i + b) = 1$$

Now by substituting back in Equation 3.1.15:

$$\mathcal{L} = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i y_i \sum_{j=1}^{n}\{\alpha_j y_j x_j\}.x_i - b\sum_{i=1}^{n}\alpha_i y_i \tag{3.1.22}$$
$$+\sum_{i=1}^{n}\alpha_i(1 - \xi_i) - \sum_{i=1}^{n}(C - \alpha_i)\xi_i$$
$$= \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i.x_j \tag{3.1.23}$$

Now to extend this classifier from input space $\mathbb{R}^N$ to higher dimensional space (feature space F), the polynomial kernel function can be used, thus the Lagrange function becomes:

$$\mathcal{L} = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j(\phi(x_i)).(\phi(x_j)) \tag{3.1.24}$$

3.2 One Class Support Vector Machine

The aim of One Class Support Vector Machine (OCSVM) method is to test if the new data belongs to a defined class or not, unlike previous classifiers where the methods distinguish between classes (Vlasveld, Accessed April 2017). The algorithm represent the observations as points in space, mapped to form separable hyperplane. It uses the origin as the only member of the second class. And it separates the image of the one class from the origin.

In Figure 3.4 the normal observations have been labelled by value $+1$, while the outliers are labelled by value -1.

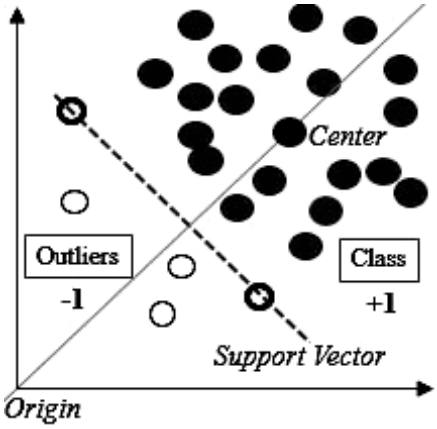

Figure 3.4: One class SVM, whereby the hyperplane separates the normal observations class 1 from anything else (outliers) class -1.

One-Class SVM algorithm: The function in this algorithm returns $+1$ in the small region which is the training data points and -1 elsewhere. The quadratic minimization function is different to the SVM quadratic minimization problem (Scholkopf et al., 1998):

$$\begin{aligned} &\min_{w,\xi_i,\rho} \frac{1}{2}||w||^2 + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i - \rho \\ &\text{s.t. } (w.\phi(x_i)) \geq \rho - \xi_i, \forall i = 1,\ldots,n \\ &\xi_i \geq 0, \forall i = 1,\ldots,n \end{aligned} \tag{3.2.1}$$

The distance to the origin in feature space is presented by ρ. The variable w is the parametrization of the hyperplane to separate the origin from the observations. In the previous formulation C was the smoothness parameter. In this formula ν is the smoothness parameter which is the proportion of outliers expected in the data, and it characterizes the solution (OCS, Accessed May 2017):

- It sets the upper bound on the fractions of outliers;
- It is the lower bound on the number of training examples used as support-vector (Scholkopf et al., 1998).

So it is still a quadratic problem, and by using Lagrange multipliers, the decision function becomes:

$$f(x) = \text{sign}((w.\phi(x_i)) - \rho) = \text{sign}(\sum_{i=1}^{n}\alpha_i K(x, x_i) - \rho)$$

Another method for OCSVM is referred as Tax and Duin method (Tax and Duin, 2004).

4. Experiment

In this section, LOF is compared to OCSVM on detecting outliers. The methodology of the research, input data and evaluation criteria are described and the results of the two algorithms are interpreted.

A comparison of algorithms is done by using a practical example as a test case. While this doesn't unequivocally say which algorithm is better, it will give some evidence towards which one is better for a complex high-dimensional dataset.

The running environment of this experiment consists of a Dell personal computer with quad-core processors running at 3.20 GHz with 8 GB of main memory DDR3. The operating system used is Ubuntu 14.04 LTS. The experiment was conducted using the Python programming language version 3 [1], where both algorithms are implemented using built-in functions imported from the Scikit learn library [2].

4.1 Methodology

The question of this research is: which one of LOF and OCSVM is better in detecting outliers in a highly unbalanced dataset? A highly unbalanced dataset means the number of normal observations is very high compared to the number of outliers. The workflow in Figure 4.1 describes the comparative study steps to answer the previous question.

The LOF algorithm requires one parameter k; which is the number of close neighbors that each normal observation should have, as mentioned in Chapter 2. OCSVM requires two parameters: γ defines how far the influence of an observation reaches, and ν is the proportion of outliers expected in the data. More details are in Section 3.2.

In OCSVM selecting a kernel has a huge impact on the results that are generated. RBF is more flexible than other kernels, so the best possible prediction would be obtained (RBF, Accessed May 2017).

Once the dataset and algorithm requirements are know, the results of LOF and OCSVM need to be compared. The results of these algorithms are lists of detected outliers. True Positive Rate (TPR) and False Positive Rate (FPR) will be used for comparison this is described in Section 4.3. Then the study of detected outliers is interpreted to understand the performance of each algorithm in the dataset.

[1] Python programming language site: https://www.python.org.
[2] Scikit learn library site: http://scikit-learn.org.

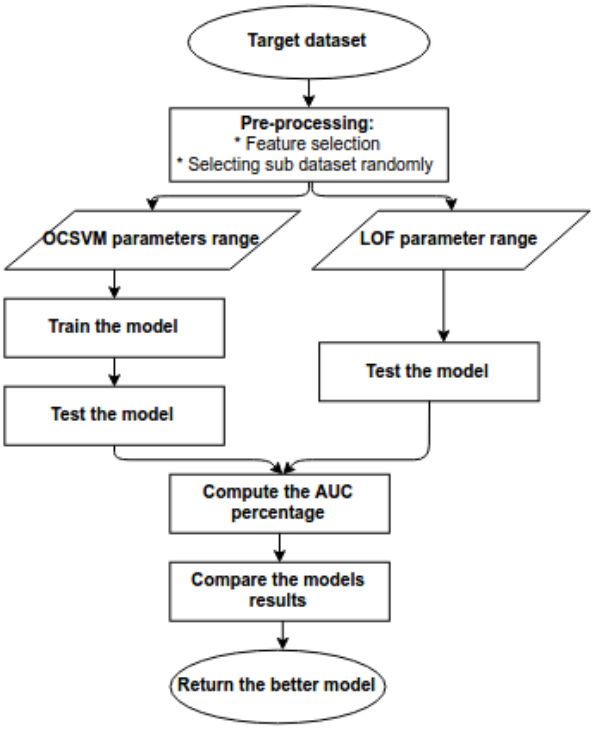

Figure 4.1: Methodology for comparing the performance of LOF and OCSVM algorithms on detecting fraud transaction on a credit card transactions dataset.

4.2 Input Data

A dataset from Kaggle data science repository is used to evaluate the performance of the algorithms. The dataset that has been used contains information of credit card transactions made by credit cards in September 2013 by European cardholders [3]. The aim is to detect the fraudulent transactions, for example transferring or withdrawing unusually large amounts of money from one account. These transactions are considered as outliers. The dataset consists of 284,807 transactions, 492 of which are fraudulent. Each transaction has 32 numerical variables which contain 28 features which are the results of Principal Component Analysis (PCA) transformation. The original features have been obscured to protect anonymity.

Figure 4.2 represents the transactions amount of the observations in the dataset after the preprocessing. Evidently, the number of fraud observations is less than the number of normal observations, which makes the credit card transactions dataset an unbalanced dataset. The fraud transactions (outliers) has a normal behaviour.

The data needs to be preprocessed before applying the algorithms due to running environment limitations. This is performed by selecting the 28 features and by choosing some observations randomly using the Random library in Python [4], and including all the outliers in the data set. More precisely, 40,000 observations are selected randomly from the dataset and all the outliers 492 observations were included in the new dataset.

[3] The credit cards transactions dataset is available in https://www.kaggle.com/dalpozz/creditcardfraud.
[4] The Random library in Python is available in https://docs.python.org/3/library/random.html.

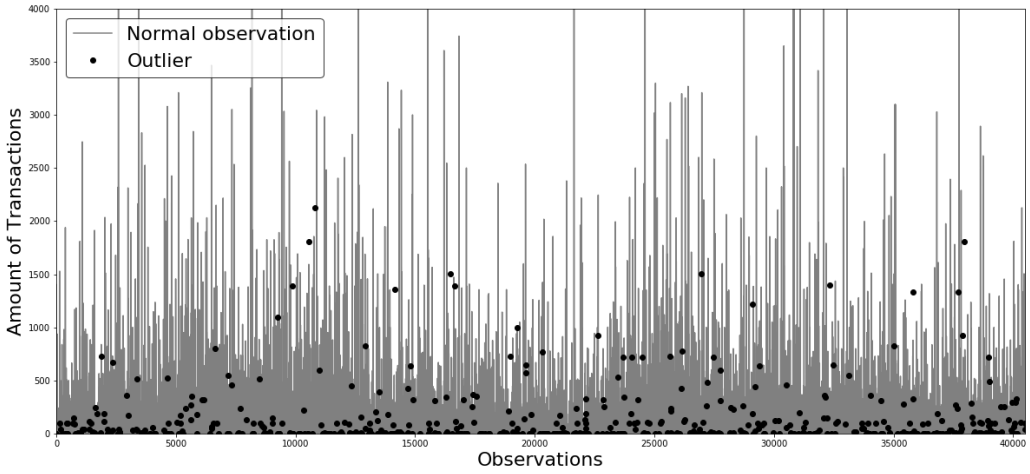

Figure 4.2: Normal and fraud transactions amounts distribution of a credit card fraud dataset

The percentage of normal observations is approximately 98.59%, while it is approximately 1.41% outliers out of 100%. The dataset is classified into two classes: normal observations and outliers. The new dataset was split into classes as shown in Table 4.1 with respect to the algorithm.

In LOF algorithm a range of 10 values has been used for k. The k-lower bound was 10 and the k-upper bound was 100. Also, OCSVM has a range of 10 values for its parameters ν and γ. ν starts from 0.01 and ends with 0.09, γ starts from 0.001 and ends with 0.0001. (These parameters are chosen because they showed the best performance after testing).

LOF algorithm uses the entire dataset to train the classifier based on the density of the observations, using the steps described in Chapter 2. OCSVM algorithm splits the data into two sets, the first set is for classifier training which contains just the normal observations, while the second set is used to test the classifier.

Algorithm	Training	Testing
LOF	40492	40492
OCSVM	30000	10492

Table 4.1: Observations division in LOF and OCSVM. LOF takes all the observations in training and testing while OCSVM divide the dataset into training data and testing data.

4.3 Evaluation Metrics

Evaluating the performance of LOF and OCSVM requires a performance measurements that can scale the output into useful information. One of the performance measurements is a confusion matrix. A confusion matrix is a technique to summarize the performance of a classification algorithm by comparing the actual value of the class with the predicted value (Con, Accessed May 2017). Considering a normal observation as positive and an outlier as negative, there are four possible outcomes from the binary classifier. Figure 4.3 shows the confusion matrix components.

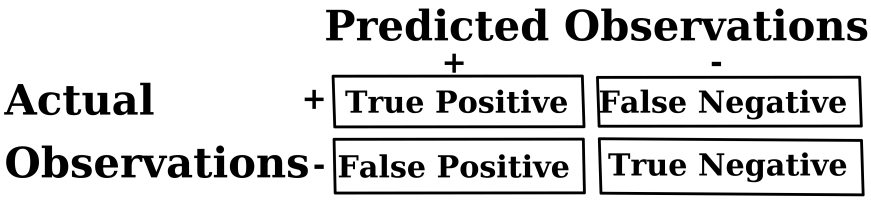

Figure 4.3: Confusion matrix components. Positive sign refers to normal observations and minus sign refers to outliers. The components obtain by comparing actual to predicted values.

Most of the machine learning methods use an accuracy rate to explain the behavior of a classifier. However, an accuracy rate doesn't perform well in unbalanced datasets (ROC, Accessed May 2017).

Confusion Matrix Components:

- True Positive (TP): When the actual value is a normal observation and the predicted value is normal.
- True Negative (TN): When the actual value is an outlier observation and the predicted value is an outlier.
- False Positive (FP): When the actual value is an outlier and the predicted value is a normal observation.
- False Negative (FN): When the actual value is normal and the predicted value is an outlier.

To evaluate the classification of unbalanced dataset two components of the confusion matrix, TP and FP are combined to form a new measurement Receiver Operator Characteristic (ROC) curve.

ROC curve:

An ROC curve is a two-dimensional graph with False Positive Rate (FPR) on the x-axis and True Positive Rate (TPR) on the y-axis, which visualizes the performance of the classifier (ROC, Accessed May 2017).

True Positive Rate: Also called the recall rate or sensitivity rate, The TPR is defined as the proportion of positive observations that are correctly classified as positive (AUR, Accessed May 2017):

$$\text{TPR} = \frac{\text{Positives correctly classified}}{\text{Total actual positives}} = \frac{\text{TP}}{\text{TP+FN}}$$

False Positive Rate: Also called the fallout rate, the FPR is defined proportion of negative observations that are incorrectly classified as positive (AUR, Accessed May 2017):

$$\text{FPR} = \frac{\text{Negatives incorrectly classified}}{\text{Total actual negatives}} = \frac{\text{FP}}{\text{FP+TN}}$$

In Figure 4.5 the top left corner of the graph is the optimal point where the FPR is 0 and TPR is 1 which is the largest Area Under the Curve (AUC) region. AUC is used in order to determine the best model that predicts the classes (AUR, Accessed May 2017).

The good classifier is the one with high TPR and low FPR which results in large AUC region (AUC, Accessed May 2017). In other words, the higher the TPR, the fewer positive observations will be missed. The higher the FPR, the more negative observations will be incorrectly classified (AUR, Accessed May 2017). The AUROC is between 0 and 1, and AUROC $= 1$ means the prediction model perfectly classifies the test data.

4.4 Analysis and Results

The results of LOF for detecting fraud credit card transactions with two features are shown in Figure 4.4. The LOF algorithm was able to detect the fraud transactions (outliers) using test features (2-dimensional space). In Figure 4.5 OCSVM detects the fraud transactions (outliers) in two features (2-dimensional space), and it also defines the region of the normal credit card transactions.

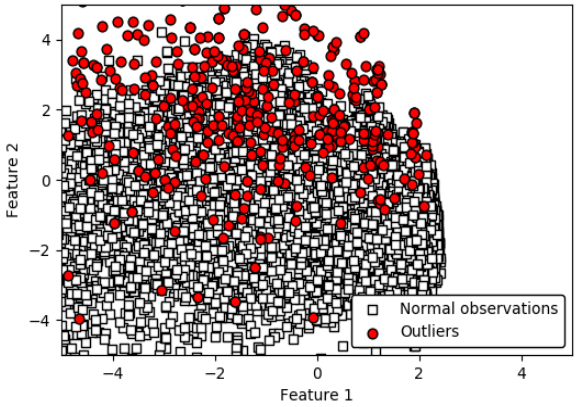

Figure 4.4: LOF results of 2 features with $k = 20$.

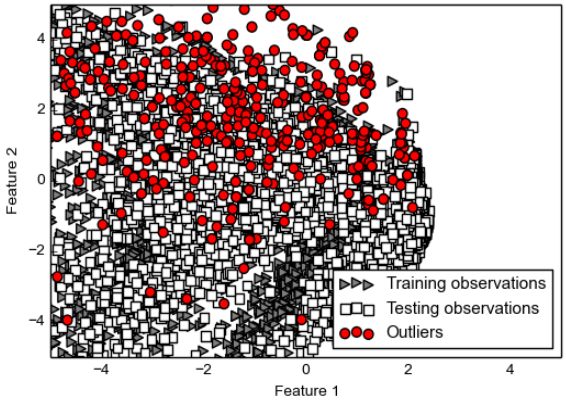

Figure 4.5: OCSVM results of 2 features with ν=0.1, γ=0.1 and RBF kernel.

The behavior of LOF has been changed when 28 features applied. Table 4.2 shows the percentages of the performance measurement results for outlier detection within a range of values for k-nearest neighbors with 28 features. Table 4.3 shows the the percentages of the performance measurement results with different values for γ and ν parameters. the parameters for both algorithms were chosen to be near optimal parameters which is done by testing different ranges of parameter values ranges.

To measure the results, ROC curves have been introduced, which is a performance measurement for unbalanced datasets by comparing the TPR and FPR.

The ROC curve is generated by using ten parameter values for k in LOF algorithm, each one has its own AUC. The same for OCSVM, ten parameter values for ν and γ are used. The values for k, ν and γ are the ones that give highest percentages of TPR and AUC, it is also the values that find the lowest percentage of FPR for both algorithms. Note that using different parameter values will result in different TPR, FPR and AUC percentages.

LOF							
k	TP	FN	FP	TN	TPR	FPR	AUC
10	35939	3986	503	64	0.9001	0.8871	0.5227
20	35952	3973	490	77	0.9004	0.8641	0.5163
30	35967	3958	475	92	0.9008	0.8377	0.5021
40	35980	3945	462	105	0.9011	0.8148	0.5074
50	35984	3941	458	109	0.9012	0.8077	0.5128
60	35982	3943	460	107	0.9012	0.8112	0.5105
70	35977	3948	465	102	0.9011	0.8201	0.5012
80	35981	3944	461	106	0.9012	0.8130	0.4848
90	35980	3945	462	105	0.9011	0.8148	0.4670
100	35990	3935	452	115	0.9014	0.7971	0.4477

Table 4.2: The results of LOF are truncated to 4 digits. When k value increases TPR increases and FPR decreases.

Figure 4.6 shows the highest percentage of AUC for LOF which is 52% for when $k = 10$. From Table 4.2 the higher the k value goes, the lower the AUC will be. In Figure 4.7 the percentage of AUC of OCSVM is higher than LOF. Precisely, AUC = 94%. The value of AUC gets larger by increasing ν value and decreasing the γ value. For both figures, the dashed diagonal line presents the baseline of ROC curve to see the performance of the model when the classes are assigned randomly. Preferable models are the ones above this line (AUR, Accessed May 2017).

OCSVM								
ν	γ	TP	FN	FP	TN	TPR	FPR	AUC
0.01	0.001	9812	113	123	444	0.9886	0.2169	0.9307
0.01	0.0009	9730	195	84	483	0.9803	0.1481	0.9350
0.02	0.0008	9637	288	72	495	0.9709	0.1269	0.9360
0.03	0.0007	9524	401	68	499	0.9595	0.1199	0.9375
0.04	0.0006	9419	506	67	500	0.9490	0.1181	0.9387
0.05	0.0005	9322	603	65	502	0.9392	0.1146	0.9390
0.06	0.0004	9239	686	62	505	0.9308	0.1093	0.9394
0.07	0.0003	9116	809	61	506	0.9184	0.1075	0.9399
0.08	0.0002	9015	910	60	507	0.9083	0.1058	0.9403
0.09	0.0001	8878	1047	59	508	0.8945	0.1040	0.9407

Table 4.3: The results of OCSVM are truncated to 4 digits. The increase in ν value and the decrease in γ, decreases both TPR and FPR values.

Note that the LOF spends 17.1835 seconds to find the testing results of 10 parameter values for k, while OCSVM spends 85.7840 seconds for 10 parameter values for ν and γ.

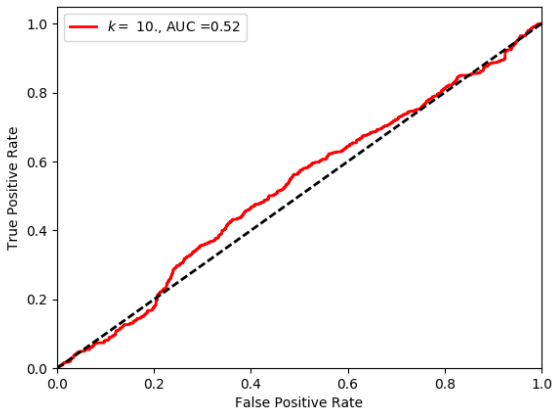

Figure 4.6: LOF model results when parameter value k=10 gives the highest AUC percentage.

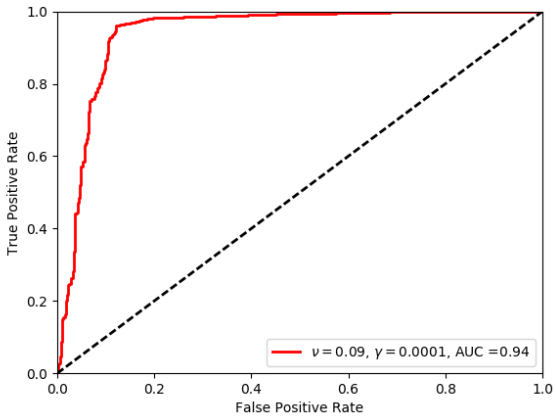

Figure 4.7: Highest AUC (94%) is obtained by setting the parameters ν=0.09 and γ=0.0001.

4.5 Discussion

The experiment showed that both of algorithms detect outliers with different percentages. By comparing the values of the TPR rate and the FPR rate. It has been found that OCSVM algorithm had a higher percentage of TPR and a low percentage of FPR. The LOF algorithm on the other hand showed high FPR, which means it classified a large fraction of the fraud transactions as normal transactions. The TPR for LOF was very low, which results in a small number of actual normal transactions that were classified normal.

The poor performance of LOF was unexpected, even though the range of k-nearest neighbors was big. This might be because of the high dimensionality of the dataset and closeness of the fraud transactions (outliers) to the normal transactions. The poor performance results of LOF is not a fault of algorithm itself. The LOF algorithm might not be well-tested.

To summarise, the OCSVM algorithm took more time comparing to LOF algorithm (factor of $\frac{85.7840}{17.1835} \approx 4.9$), but it showed an excellent performance on detecting outliers, with a percentage starts from 93% to 94% for AUC. [5]

[5] All the codes that used to generate research results and more are available in: https://github.com/sustecha/LOF_OCSVM.git

5. Conclusions

The detection of outliers in indicates something scientifically interesting or significant. There are various algorithms for outlier detection in datasets. Two of them are Local Outlier Factor and One Class Support Vector Machine. Both algorithms have been used to detect outliers in a credit card fraud dataset. These algorithms require labeled normal data, which is essential in semi-supervised learning algorithms. The algorithms assigned a binary label to the observations, classifying themes either normal or fraudulent. Local Outlier Factor and One Class Support Vector Machine have different ways to identify outliers. They use *density and* distance of the observations respectively.

This research proposes a comparison of these two algorithms for the outlier detection process. Both algorithms reach different percentages of performance, and the One Class Support Vector Machine performs better than Local Outlier Factor in the given dataset. Comparison of these algorithms has been done by using the results of a confusion matrix, precisely, the True Positive Rate and False Positive Rate. The algorithm that has a higher True Positive Rate and a False Positive Rate is the better one.

To test the algorithm, a range of parameter values have been used. The Local Outlier Factor True Positive Rate is around 90%, while the False Positive Rate changes from 88.7% to 79.7% and the Area Under the Curve is from 52.2% to 44.7%. The One Class Support Vector Machine algorithm shows different behavior, with True Positive Rates between 98.8% and 89.4%, and False Positive Rates is between 21.6% and, while its 10.4% and the Area Under the Curve is from 93% to 94%. To conclude, the algorithms have shown different True Positive Rates, False Positive Rates and areas of Receiver Operator Characteristic. One Class Support Vector Machine displays a better performance on detecting outliers.

The current results are obtained by applying the Local Outlier Factor algorithm and One Class Support Vector Machine in some transactions that selected randomly from the dataset. Therefore the behavior of the algorithms may change with different transactions and parameters. Further work could be done by applying the algorithms on the whole dataset which will give more precise results. Applying these algorithms in different datasets to see if the results convergence or divergence will give a better understanding of algorithms behavior. Adding more Machine Learning outlier detection algorithms will help to know which one is better to apply in Big Data.

References

Receiver operating characteristic (roc), Accessed May 2017. URL http://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html.

What does auc stand for and what is it?, Accessed May 2017. URL https://stats.stackexchange.com/questions/132777/what-does-auc-stand-for-and-what-is-it.

What is a confusion matrix in machine learning?, Accessed May 2017. URL http://machinelearningmastery.com/confusion-matrix-machine-learning/.

What is the relation between the number of support vectors and training data and classifiers performance?, Accessed May 2017. URL http://stackoverflow.com/questions/9480605/what-is-the-relation-between-the-number-of-support-vectors-and-training-data-and.

Unsupervised machine learning with one-class support vector machines, Accessed May 2017. URL https://thisdata.com/blog/unsupervised-machine-learning-with-one-class-support-vector-machines/.

Why does rbf kernel generally outperforms linear or polynomial kernels?, Accessed May 2017. URL https://www.quora.com/Why-does-RBF-kernel-generally-outperforms-linear-or-polynomial-kernels.

Drawing roc curve, Accessed May 2017. URL https://docs.eyesopen.com/toolkits/cookbook/python/plotting/roc.html.

Introduction to support vector machines. http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html, Accessed May 2017.

Anomlay. Anomaly detection. Wikipedia, the Free Encyclopedia, https://en.wikipedia.org/wiki/Anomaly_detection, Accessed April 2017.

N. Arcolano and D. Rudoy. One class support vector machines methods and applications. School of Engineering presentation, 2008. URL http://www.dainf.ct.utfpr.edu.br/~kaestner/Mineracao/ArquivosExtras2016/Dan.Nick-OneClassSVM.pdf.

V. Barnett and T. Lewis. *Outliers in statistical data*. Number 0-471-93094-6. John Wiley, 1994.

M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF identifying density based local outliers. 2000.

V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey, 2009. URL https://www.vs.inf.ethz.ch/edu/HS2011/CPS/papers/chandola09_anomaly-detection-survey.pdf.

H. Cui. Online outlier detection over data streams, 2005.

U. S. Goldstein M. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PLoS ONE 11(4): e0152173*, 2016. URL https://doi.org/10.1371/journal.pone.0152173.

E. M. Knorr. *Outliers and Data Mining: Finding Exceptions in Data*. PhD thesis, The university of British Columbia, Department of Computer Science, 2002.

E. M. Knox and R. T. Ng. Algorithms for mining distance based outliers in large datasets. pages 392–403, 1998.

LOF. Local outlier factor. Wikipedia, the Free Encyclopedia, https://en.wikipedia.org/wiki/Local_outlier_factor, Accessed April 2017.

D. K. Malak Alshawabkeh, Byunghyun Jang. Accelerating the local outlier factor algorithm on a gpu for intrusion detection systems, 2010.

H. Y. N. Mathew X. Ma and W. Liu. Density-based outlier detection by local outlier factor on large-scale traffic data, 2016. URL http://www.ingentaconnect.com/contentone/ist/ei/2016/00002016/00000014/art00003?crawler=true.

T. B. Modise. A basic introduction to machine learning, 2014. African Institute for Mathematical Sciences, South Africa, 2014.

S. W. MR Tight, EJ Redfern and S. Clark. Outlier detection and missing value estimation in time series traffic count data: Final report of serc project gr/g23180, 1993. URL http://eprints.whiterose.ac.uk/2174/1/ITS295_WP401_uploadable.pdf.

E. A. na and E. Lozano. Parallel algorithms for distance-based and density-based outliers. 2005.

A. Ng. Lecture notes in machine leanring, coursera site, 2017. URL https://www.coursera.org/learn/machine-learning/supplement/d5Pt1/lecture-slides.

NIST. *Engineering statistics*. NIST is an agency of the U.S. Department of Commerce, 2003. URL http://www.itl.nist.gov/div898/handbook/eda/section3/eda35h.htm.

L. R. Oded Maimon. *Data Mining and Knowledge Discovery Handbook*. Number 978-0-387-24435-8. Springer New York Dordrecht Heidelberg London, 2010.

Outlier. Outlier. Wikipedia, the Free Encyclopedia, https://en.wikipedia.org/wiki/Outlier, Accessed April 2017.

B. Scholkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt. *Support Vector Method for Novelty Detection*. Number 0-262-11245-0. MIT Press, 1998.

SCM. Support vector machine. Wikipedia, the Free Encyclopedia, https://en.wikipedia.org/wiki/Support_vector_machine, Accessed April 2017.

S. Seo. A review and comparison of methods for detecting outliers in univariate data sets, 2006.

V. C. Silvia Cateni and M. Vannucci. *Outlier Detection Methods for Industrial Applications, Advances in Robotics, Automation and Control*. Number 78-953-7619-16-9. In Tech Open Access Publisher, 2008.

K. Singh and D. S. Upadhyaya. Outlier detection applications and techniques. *IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 1, No 3*, 2012.

D. M. Tax and R. P. Duin. Support vector data description. *Kluwer Academic Publishers-Plenum Publishers*, 2004.

R. Vlasveld. Introduction to one-class support vector machines, Accessed April 2017. URL http://rvlasveld.github.io/blog/2013/07/12/introduction-to-one-class-support-vector-machines/.