Algorithms for Anomaly Detection

Cheng Peng

STA551 Foundations of Data Science

Contents

1 Introduction - Real-World Fraud Detection

Credit card fraud is a type of identity theft that occurs when someone illegally uses another person's credit card or account information for an unauthorized transaction. Fraud can happen as a result of a stolen, misplaced, or counterfeit credit card.

According to Nilson Report [https://nilsonreport.com/content_promo.php?id_promo=16], card fraud losses worldwide reached $28.65 billion in 2020. By 2025, the United States is projected to reach $12.5 billion in card fraud losses.

Among all card fraud, credit card fraud contributes a significant portion. Unlike the process of identifying credit default in which we have a lot of information about cardholders, in the fraud domain, we have no information about fraudsters except for the amount, location, and time of the fraudulent transactions. This makes the detection and identification of credit card fraud more challenging.

There are different types of credit card fraud. The most frequently occurring types of credit card fraud include lost and stolen, counterfeit, card-not-present (CNP), account takeover, application fraud, card-never-received, etc. The different strategies of prevention for different types of fraud have been developed by financial industries over the years. These include educating cardholders on how to protect their personal information,

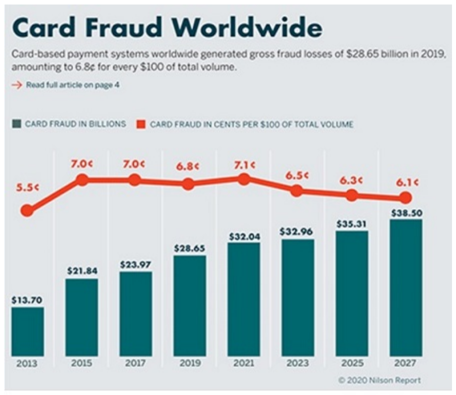


Figure 1: Figure 1. The trend of worldwide card fraud loss.

increasing security measures such as using chip-enabled cards and enhanced web portals for online transactions, monitoring card shipment, etc.

Fraud detection, on the other hand, is a more complex and difficult process. This is where analytics come into play.

In the rest of this note, we will

- overview of the statistical models and machine learning algorithms that are used in credit card fraud detection;
- introduce the statistical foundation of the proposed algorithm;
- detail the proposed new algorithms based on sequential data;
- evaluate the performance of the proposed algorithms
- introduce the potential improvement and generalization of the proposed algorithms.

2 Anomaly Detection Use Case: Fraud Operation

In essence, identifying fraud is an anomaly detection process that identifies unusual patterns that do not fit normal behavior in the data generation process. In addition to fraud detection, anomaly detection has many applications in business such as intrusion detection (identifying strange patterns in network traffic), health monitoring (spotting a malignant tumor in an MRI scan), fault detection in operating environments, etc.

Three different types of anomalies can be found in credit card fraud.

Global Anomalies – if a data point is too far from the rest, it falls into the category of point anomalies (outliers). For example, if all historical fuel costs are below $75, but the most recent transaction is $149.50. The most recent transaction is abnormal.

Contextual Anomalies – If the event is anomalous for specific circumstances (context), then we have contextual anomalies. As data becomes more and more complex, it is vital to use anomaly detection methods for the context. This anomaly type is common in sequence data. For example, - a card was used for fuel roughly 4-5 times a month historically, but the same card was used at gas pumps 5 times within the last 3 hours.

Collective Anomalies. The collective anomaly denotes a collection of anomalous concerning the multiple feature variables, but not individual objects. For example: if one card was skimmed at a pump, the fraudsters then make many duplicate cards and sell them to card other fraudsters. The counterfeit cards may be used in very different geographic locations at approximately the "same time". Each transaction associated with this card may be normal. However, we will see the abnormality if we look at these transactions collectively.

Sometimes, people break anomaly detection algorithms down into two subclasses: outlier detection and novelty detection.

- **Outlier detection**: the input data set contains examples of both standard events and anomaly events. These algorithms seek to fit regions of the training data where the standard events are most concentrated, disregarding, and therefore isolating the anomaly events. Such algorithms are often trained in an unsupervised fashion (i.e., without labels). We sometimes use these methods to help clean and pre-process datasets before applying additional machine-learning techniques.
- **Novelty detection**: Unlike outlier detection, which includes examples of both standard and anomaly events, novelty detection algorithms have only the standard event data points (i.e., no anomaly events) during training time. During training, these algorithms use only labeled examples of standard events (supervised learning). At the time of testing/prediction, novelty detection algorithms must detect when an input data point is an outlier.

Various machine learning algorithms and statistical models could be used for credit card fraud detection dependent on the amount of information on the types of fraud. But they are broadly classified into supervised and unsupervised methods. Some of the classical methods are outlined in the following subsections.

3 Supervised and Unsupervised Anomaly Detection

Depending on whether the labels are available, anomaly detection techniques can be categorized into one of the following three modes:

3.1 Supervised Anomaly Detection

Techniques trained in supervised mode assume the availability of a training data set that has labeled instances for normal as well as anomaly classes. A typical approach in such cases is to build a predictive model for normal vs. anomaly classes. Any unseen data instance is compared against the model to determine which class it belongs to.

Two major issues arise in supervised anomaly detection.

- **Imbalance Labels**: The anomalous instances are far fewer compared to the normal instances in the training data. Issues that arise due to imbalanced class distributions have been addressed in the data mining and machine-learning literature
- **Inaccuracy of Label**: obtaining accurate and representative labels, especially for the anomaly class is usually challenging. This is particularly true in the world of credit card fraud operations.

Other than these two issues, the supervised anomaly detection problem is similar to building predictive models such as logistic regression and decision tree algorithms.

3.2 Semi-Supervised Anomaly Detection.

These types of methods usually assume that the training data has labeled instances for only the normal class. Since they do not require labels for the anomaly class, they are more widely applicable than supervised techniques. The typical approach used in such techniques is to build a model for the class corresponding to normal behavior and use the model to identify anomalies in the test data.

A limited set of anomaly detection techniques exist that assume the availability of only the anomaly instances for training. Such techniques are not commonly used, primarily because it is difficult to obtain a training data set that covers every possible anomalous behavior that can occur in the data.

Sometimes, we have to use supervised and unsupervised methods to label some unlabeled examples to reach the minimum sample size for using supervised anomaly detection methods.

3.3 Unsupervised Anomaly Detection.

These types of methods operate in unsupervised mode and do not require training data, and thus are most widely applicable. The techniques in this category make the implicit assumption that normal instances are far more frequent than anomalies in the test data. If this assumption is not true then such techniques suffer from a high false alarm rate.

Many semi-supervised techniques can be adapted and used in an unsupervised mode by using a sample of the unlabeled data set as training data. Such adaptation assumes that the test data contains very few anomalies and the model learned during training is robust to these few anomalies.

In the following sections, we only select two representative anomaly detection algorithms: outlier detection and novelty detection.

4 Local Outlier Factor Method: Outlier Detection

The LOF [proposed by Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jorg Sander in 2000] is the most well-known local anomaly detection algorithm whose idea is carried out in many nearest-neighbor-based algorithms.

4.1 Definitions of Some `Distances`

Before introducing the steps for calculating the LOF score, we define the following technical terms

Normal Distance (ND) - any of the valid "statistical distances" such as Euclid, Minkowski, Manhattan, etc.

k-distance (kD): For the pre-selected k, k-distance is defined to be the distance of a (new) point to its kth neighbor. For example, if k was 3, the k-distance of A, denoted by `k-distance(A)`, would be the distance of point A to its `third closest` point which is D (B is the first closest neighbor, C is the second closest neighbor), see the following Figure 3.

Reachability Distance (RD) is defined to be the maximum distance of two points and the k-distance of the `second` point. For example, the reachability distance between A and D is given by

`Reachability-Distance(A, D) = max{k-distance (D), normal_distance (A, D)}`

Where `normal-distance(A, D)` could be any "statistical distance" that is used in `k-distance(D)`.

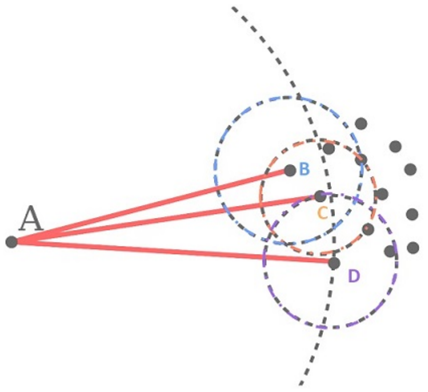


Figure 2: Figure 2. Illustration of calculating LOF score.

Local Reachability Density (LRD) refers to how far we need to go from the point we are at to reach the next point or set of points. For example, For all k =3 closest neighbors of A, the reachability distances are calculated and the values found are summed and divided by the value of k =3. When the inverse of this value is taken, we calculate the density we are looking for.

```
Local-reachability-density(A) = 1 / ( sum ( Reachability_Distance (A , n ) ) / k)
```

n is the number of points/neighbors centered at point A.

Local Outlier Factor (LOF) Score

The local reachability densities found are compared to the local reachability densities of A's nearest k neighbors. The density of each neighbor is summed up and divided by the density of A. The value found is divided by the number of neighbors i.e. k.

```
LOF(A) =[(LRD(1st. neighbor) + LRD(2nd. neighbor) + ... + LRD(kth. neighbor))/LRD(A)]/k
```

Use of LOF Score - A value of approximately 1 indicates that the object is comparable to its neighbors. A value below 1 indicates a denser region (which would be an inlier), while values significantly larger than 1 indicate outliers.

- **LOF(k) ~ 1** means Similar density as neighbors,
- **LOF(k) < 1** means Higher density than neighbors (Inlier),
- **LOF(k) > 1** means Lower density than neighbors (Outlier)

4.2 A Toy Example

The calculation of the LOF score is not difficult. However, it involves multiple steps and several "distance"-based measures. In this section, we use a toy example with 4 points depicted in the following figure.

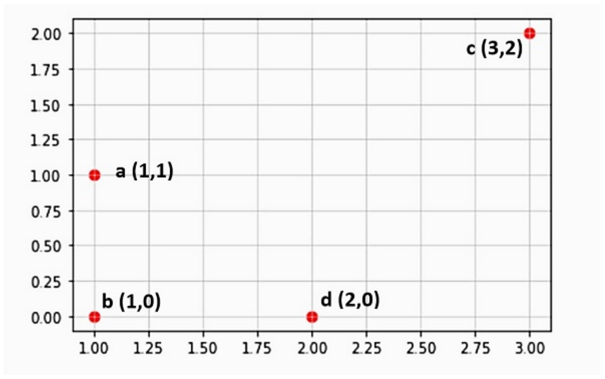


Figure 3: Figure 3. Toy data set.

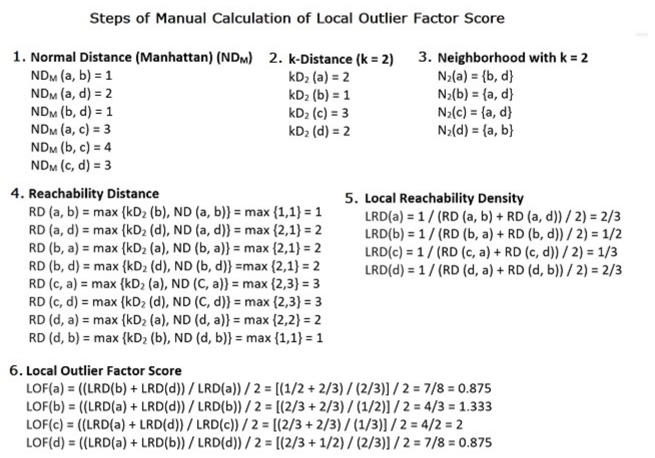
The following figure shows the steps for calculating the LOF score for each of the four points in the toy data set.

We can see that the LOF score of point c is 2. Point c is a potential outlier.

4.3 Implementation of LOF in R

Several R packages have a function to calculate LOF scores. We use **lof()** in **{dbscan}** to calculate LOF scores of data points in the well-known **iris** data.

lof() can calculate the LOF score in a high dimensional space. The original iris data has 4 numerical variables (sepal and petal widths and lengths). We will calculate the LOF scores based on these variables (in



Steps of Manual Calculation of Local Outlier Factor Score

1. **Normal Distance (Manhattan) (ND_M)**
   - ND_M (a, b) = 1
   - ND_M (a, d) = 2
   - ND_M (b, d) = 1
   - ND_M (a, c) = 3
   - ND_M (b, c) = 4
   - ND_M (c, d) = 3
2. **k-Distance (k = 2)**
   - kD_2 (a) = 2
   - kD_2 (b) = 1
   - kD_2 (c) = 3
   - kD_2 (d) = 2
3. **Neighborhood with k = 2**
   - N_2(a) = {b, d}
   - N_2(b) = {a, d}
   - N_2(c) = {a, d}
   - N_2(d) = {a, b}
4. **Reachability Distance**
   - RD (a, b) = max {kD_2 (b), ND (a, b)} = max {1,1} = 1
   - RD (a, d) = max {kD_2 (d), ND (a, d)} = max {2,1} = 2
   - RD (b, a) = max {kD_2 (a), ND (b, a)} = max {2,1} = 2
   - RD (b, d) = max {kD_2 (d), ND (b, d)} =max {2,1} = 2
   - RD (c, a) = max {kD_2 (a), ND (C, a)} = max {2,3} = 3
   - RD (c, d) = max {kD_2 (d), ND (C, d)} = max {2,3} = 3
   - RD (d, a) = max {kD_2 (a), ND (d, a)} = max {2,2} = 2
   - RD (d, b) = max {kD_2 (b), ND (d, b)} = max {1,1} = 1
5. **Local Reachability Density**
   - LRD(a) = 1 / (RD (a, b) + RD (a, d)) / 2) = 2/3
   - LRD(b) = 1 / (RD (b, a) + RD (b, d)) / 2) = 1/2
   - LRD(c) = 1 / (RD (c, a) + RD (c, d)) / 2) = 1/3
   - LRD(d) = 1 / (RD (d, a) + RD (d, b)) / 2) = 2/3
6. **Local Outlier Factor Score**
   - LOF(a) = ((LRD(b) + LRD(d)) / LRD(a)) / 2 = [(1/2 + 2/3) / (2/3)] / 2 = 7/8 = 0.875
   - LOF(b) = ((LRD(a) + LRD(d)) / LRD(b)) / 2 = [(2/3 + 2/3) / (1/2)] / 2 = 4/3 = 1.333
   - LOF(c) = ((LRD(a) + LRD(d)) / LRD(c)) / 2 = [(2/3 + 2/3) / (1/3)] / 2 = 4/2 = 2
   - LOF(d) = ((LRD(a) + LRD(b)) / LRD(d)) / 2 = [(2/3 + 1/2) / (2/3)] / 2 = 7/8 = 0.875

Figure 4: Figure 4. Steps for calculating LOF scores.

4-dimensional space). To visualize the LOF score, we also perform a PCA and use the first two PCs (which account for about 95% of the total variation) to calculate LOF scores.

A Cautionary Note on LOF Scores with PCA - The purpose of calculating LOF scores based on the first two-component analysis is to visualize the outliers in a 2-dimensional plot. The original variables must not be scaled in the PCA to obtain comparable LOF scores. The translation of the original variable will give the same LOF score.

```r
log.iris = log(iris[,-5])    # drop the categorical variable in the original
                             # data set and transform all numerical to the
                             # log-scale
ir.pca <- prcomp(log.iris, center = TRUE, scale = FALSE)
# use the first two PCs to define a data frame for LOF
pca.iris = data.frame(ir.pca$x[, 1:2])
### Calculate the two LOF scores with the original variables and PCs respectively.
lof.pca <- lof(pca.iris, minPts = 30)  #minPts = k value
lof.orig <- lof(log.iris, minPts = 30)
## 2D plot of LOF score based on PCs
plot(pca.iris, pch = "x",    # point symbol
     main = "LOF Based on PCA",
     #asp = 1,
     cex = 0.5)                # aspect ratio - ratio of 'y/x'
points(pca.iris,
       cex = (lof.pca-1)*1.5,      # point size according to the LOF score
       pch = 21,
       col = "purple")
text(pca.iris[lof.pca > 1.8,],
     labels = round(lof.pca, 1)[lof.pca > 1.8],
     pos = 1,    # 1, 2, 3 and 4 => below, left , above, and right
     cex = 0.7,
     col = "blue")
```

```r
plot(lof.pca, lof.orig, pch ="x",
     main = "LOF Scores Comparison: PCA vs Original Variable",
     xlab = "PCA LOF Score",
     ylab = "Original LOF Score",
     cex = 0.6)
```

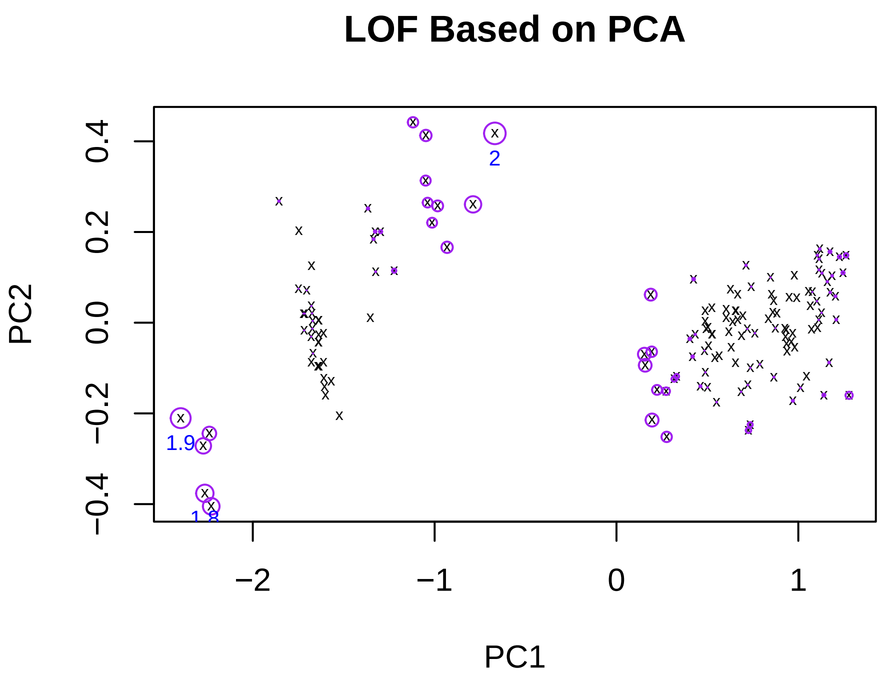


Figure 5: Figure 5. LOF scores based on the first 2 principal components.

```
points(lof.pca[lof.pca > 1.8], lof.orig[lof.pca > 1.8],
       pch = 21,
       cex = lof.pca*1.5,
       col = "purple")
text(lof.pca[lof.pca > 1.8], lof.orig[lof.pca > 1.8],
     labels = round(lof.pca, 1)[lof.pca > 1.8],
     pos = 1,    # 1, 2, 3 and 4 => below, left , above, and right
     cex = 0.7,
     col = "blue")
```

The above two figures show that the LOF method identifies the same outliers based on the original four variables (in the 4-dimensional feature space) and the first two PCs (2-dimensional space). No variable scaling was used in the PCA.

5 One-class SVM: Novelty Detection

Many machine learning models throw inaccuracies in the modeling of outlier elements. So it becomes a most basic requirement for us to determine if the new observation comes under the same existing distribution or if the new observation should be determined as different. In other words, there is only one class in the practical problem. The methods introduced in the following subsections are based on the non-linear kernel function. We will not introduce the mathematics behind these methods. But I will give a brief description of how the kernel function works.

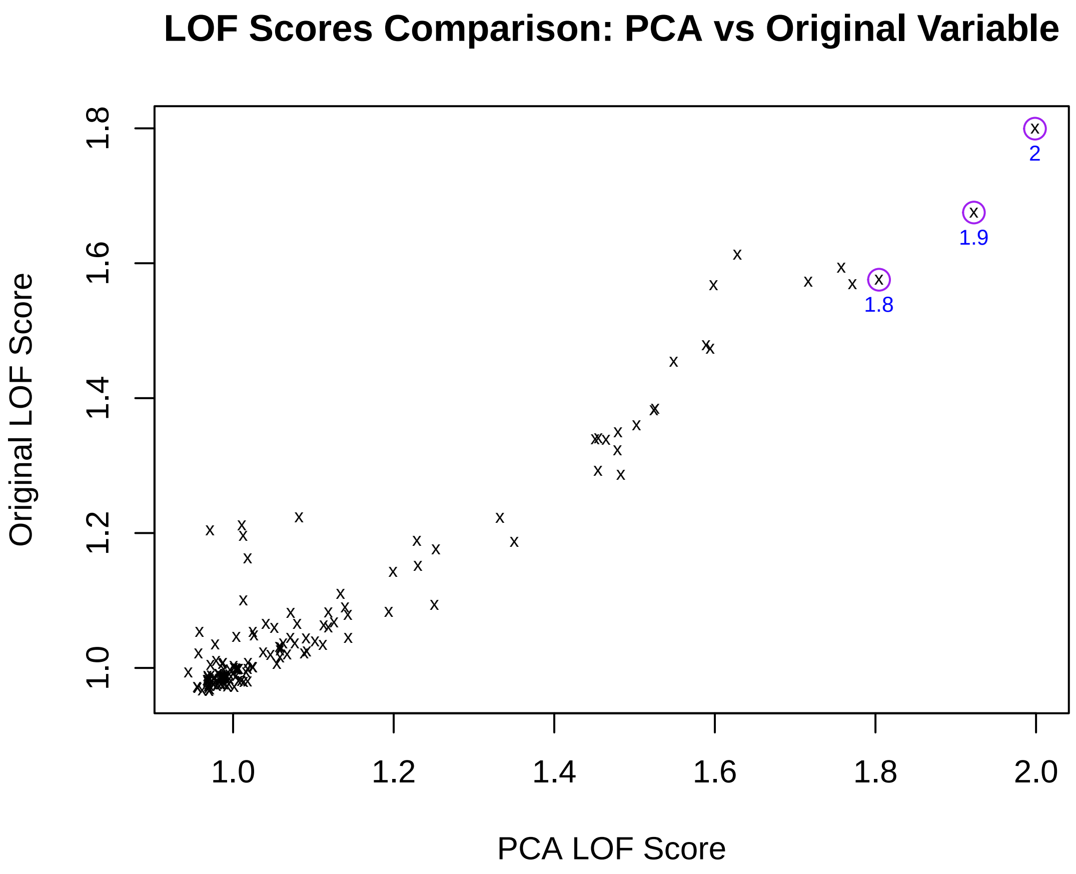
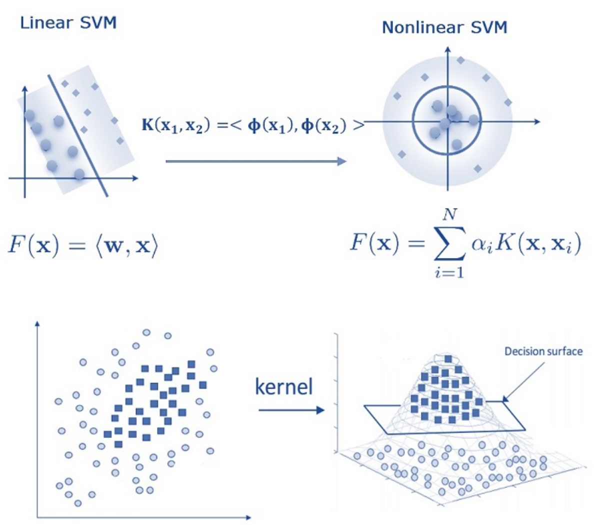


Figure 6: Figure 6: Scatter plot of LOF scores based on the original variables and the first 2 PCs.

5.1 Kernel Function

A kernel function is a weighing function. It defines an inner product (dot product) based on transformed feature variables. It can map a lower-dimensional nonlinear classification problem to a higher-dimensional space to obtain a linear classification problem.



Linear SVM
Nonlinear SVM
$K(x_1, x_2) = < \phi(x_1), \phi(x_2) >$
$F(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$
$F(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}, \mathbf{x}_i)$
kernel
Decision surface

Figure 7: Figure 7. Kernel function.

There are different kernel functions used in various applications. The commonly used kernel functions are linear, quadratic, exponential, sigmoid, Gaussian radial-based function (RBF), etc.

5.2 Description of Support Vector Data Description (SVDD)

Support Vector Data Description (SVDD, developed by Tax and Duin in 2004) is a one-class classification technique that is useful in applications where data that belong to one class are abundant but data about any other class are scarce or missing. SVDD can

- Identifies minimum radius hyper-sphere around "normal" data
- Works on multivariate data
- Does not require the assumption of normality
- Fits flexible surfaces using a kernel function
- Minimizes the chance of accepting outliers

It creates a minimum-radius hyper-sphere around the training data set and scores new observations by calculating the distance to the hyper-sphere center. Observations with distances greater than the minimum radius are flagged as anomalies. It has been used in the areas such as fraud detection, equipment health monitoring, and process control where the majority of the data belong to one class.

SVDD's foundation is built based on the primal-dual algorithms (a method for solving linear programs inspired by the Ford–Fulkerson). It uses a kernel-based approach to define a score function for each observation and obtain a decision boundary on whether new incoming data is an outlier.

To avoid mathematical expression, the following figure illustrates the

5.3 One-Class Support Vector Machine (OC-SVM)

The OC-SVM formulation is equivalent to the SVDD formulation when the **RBF kernel** function is used.

In this subsection, we use the R function **svm()** in package **{e1071}** to perform one-class support vector

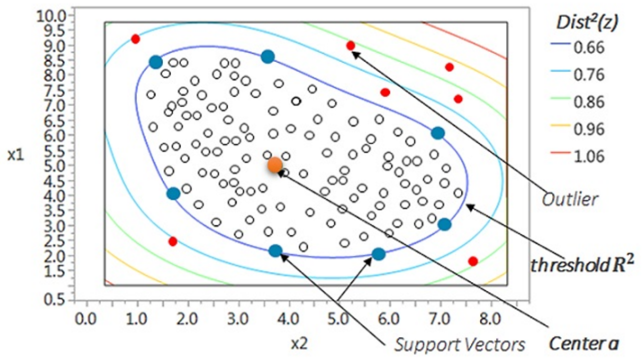


Figure 8: Figure 8. Illustration of SVDD.

machine analysis using the iris data. We need to define a training data set. For simplicity, we use all setosa records to train the OC-SVM and then use the entire iris data to calculate the accuracy metrics.

```
data(iris)
df <- iris
##
df <- subset(df ,  Species=='setosa')           # choose only one of the classes
x <- subset(df, select = -Species)              # make x variables
y <- df$Species                                 # make y variable(dependent)
model <- svm(x, y, type ='one-classification') # train an one-classification model
# print(model)
summary(model) #print summary
```

```
##
## Call:
## svm.default(x = x, y = y, type = "one-classification")
##
##
## Parameters:
##    SVM-Type:  one-classification
##  SVM-Kernel:  radial
##       gamma:  0.25
##          nu:  0.5
##
## Number of Support Vectors:  27
##
##
##
##
## Number of Classes: 1
```

```
# Test on the whole set
pred <- predict(model, subset(iris, select = -Species)) #create predictions
actual = (iris[,5]== 'setosa')    # actual labels
confusion.matrix = table(pred, actual)
confusion.matrix
```

```
##          actual
## pred      FALSE TRUE
##   FALSE    100   25
##   TRUE      0   25
```

Question: Can we mimic the case study in LOF to use the first two principal components to perform OC-SVM?