A Brief Survey on Outlier Classification

Zeyu Jia* Feng Zhu*
1600010603 1600010643

School of Mathematical Science, Peking University

January 19, 2019

Abstract

12

1 Introduction

Anomaly detection (or outlier detection) is the identification of rare items, events or observations which raise suspicions by differing significantly from the majority of the data. Typically, anomalous data can be connected to some kind of problem or rare event such as, e.g., bank fraud, medical problems, structural defects, etc. This connection makes it very interesting to be able to pick out which data point can be considered anomalies, as identifying these events are typically essential from a business perspective. There have been plentiful literatures on machine learning methods for outlier detection, and they generally focus on unsupervised or semi-supervised ones. For a review of them, readers could refer to [2].

However, unlike labelling of pictures or videos, it is not that difficult to label data points in anomaly detection setting. For example, in financial fraud detection, it's easy to obtain whether a deal is fraud or not after it is done. We observe that labeled data for outlier detection is always available for training and testing numerous algorithms. Why do we have to apply unsupervised or semi-supervised learning methods given that the labels are complete? Thus, in this paper, we will mainly focus on "Outlier Classification", where labels are all presented.

Outlier Classification gives rise to other challenging problems.

1. The dataset is highly imbalanced. The ratio between normal data and anomaly ones may be very large, sometimes exceeding 500:1. Such imbalanced proportion between each class will let traditional classifiers tend to classify instances to normal class much easily, and will certainly bring ineffective outcome since classifying anomaly instances into normal will not affect the accuracy very much. Since what we want is to identify outliers accurately, this phenomenon will seriously damage the effect of traditional classifiers.
2. The number of anomaly data is small, even if the size of the whole data set is large. The problem become especially serious when we split the dataset into several parts, because it imposes high volatility on evaluation with regard to anomaly data.

To deal with the problems above, we need to clarify our evaluation metric before going further. We will use *recall*(R) and *precision*(P), which is defined as

$$\begin{aligned} \textit{recall}(\mathrm{R}) &= \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}, \\ \textit{precision}(\mathrm{P}) &= \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}, \end{aligned} \tag{1}$$

where TP, FP, FN are defined as the number of instances in the following table

	True Anomaly	True Normal
Predicted Anomaly	TP	FP
Predicted Normal	FN	TN

*The authors are arranged lexicographically.

To integrate the two metric, we also define our *F-score*. In practice, it is always more important to detect all true anomalies as much as possible, which means that we should put more weight on *recall* than *precision*. Our *F-score*(F) is defined as

$$\textit{F-score}(\mathrm{F}) = \frac{2.5 \times \mathrm{R} \times \mathrm{P}}{1.5 \times \mathrm{P} + \mathrm{R}}. \tag{2}$$

Our paper is organized as follows. Section 2 will briefly introduce several supervised learning techniques. Among them, we pay special attention to (weighted) Decision Tree Classifier, XGBoost, and Ensemble methods. Section 3 will briefly introduce three commonly used unsupervised learning techniques for outlier detection. In Section 4, we will introduce our algorithm, "GMDA", which is originally based on Bayesian decision theory, to combine both supervised and unsupervised learning methods. We also explain its relation and extension with other methods. We conduct a case study of financial fraud detection in Section 5 to produce plentiful results and compare different algorithms. In Section 6, we conclude. The division of our work is in Acknowledgement.

2 Supervised Learning Models

In this section, we will discuss several commonly used supervised learning methods, and their application on outlier classification tasks. These supervised learning methods includes

- Logistic Regression Classifier,
- Gaussian Naive Bayes Classifier,
- Decision Tree Classifier,
- Support Vector Classifier,
- Discriminant Analysis Classifier,
- K-Nearest Neighbor Classifier.

Furthermore, we will also introduce two useful techniques used in classification problem: boosting methods(Adaboost/XGBoost) and other ensemble methods(random forest/bagging/voting). These methods and techniques refer to [6][4], and will be discussed in the following subsections respectively.

2.1 Logistic Regression Classifier

The logistic regression classifier is one of the most simple methods used in 2-class classification problem. Suppose two class are denoted by 0 and 1. Then we will find an proper W and b such that

$$\begin{aligned} P(y=0|X=x,\beta=(W,b)) &= \frac{1}{1+\exp(Wx+b)}, \\ P(y=1|X=x,\beta=(W,b)) &= \frac{\exp(Wx+b)}{1+\exp(Wx+b)}, \end{aligned} \tag{3}$$

and we will maximize the log-likelihood:

$$l(\beta) = \sum_{i=1}^{N} y_i \log P(y=y_i|X=x_i,\beta) + (1-y_i)\log P(y\neq y_i|X=x_i,\beta). \tag{4}$$

By finding the optimal parameter β, then we can use (3) to make prediction.

2.2 Gaussian Naive Bayes Classifier

Gaussian naive Bayes classifier is one of Bayesian classifier using Gaussian distribution as prior. We make the assumption that

$$P(X|y) = \prod_{i=1}^{m} P(x_i|y), \tag{5}$$

where $X = (x_1, \cdots, x_m)$ is the feature vector. This assumption talks about the independence of each element of feature vector. And then we can use MAP to make classification:

$$\hat{y} = \arg\max_y P(y|X) = \arg\max_y \frac{P(y)\prod_{i=1}^{m} P(x_i|y)}{P(X)} = \arg\max_y P(y)\prod_{i=1}^{m} P(x_i|y). \tag{6}$$

And here we assume the likelihood given each y to be Gaussian:

$$P(x_i|y) \sim \mathcal{N}(\mu_y, \sigma_y), \tag{7}$$

where parameters μ_y and σ_y are determined by maximum likelihood estimation. After finishing this estimation, the classification process can be done according to (6).

The underlying assumption of this method is that data in each class satisfies Gaussian distribution, and each features are independent given its label.

2.3 Decision Tree Classifier

The decision tree classifier is a little bit different to other methods. Other methods all aim to find a proper linear (or nonlinear) combination of features. The decision tree, however, will execute on features sequentially. In other words, in each step of decision tree's iteration, only one feature will be taken into consideration.

The training of a decision tree is often divided into two parts: splitting and pruning. In the splitting part, the decision tree will expand their leaves. That is, finding a new feature (often the one with the most reduction of the loss) and then split one of the tree leaf into two new leaves. In the pruning process, some of leaves will merged into one, if merging them can decrease the value of loss plus regularization.

2.3.1 Weighted Decision Tree Classifier

Weighted decision tree is a useful model in tackling cases where number of instances in each class varies a lot.

To balance the number of instances in each class, and to let the classifier be not prone to classify new instance into the major class, we assign a weight to each class. And we aim to let the number of instances in each class multiplied by the weight in that class does not vary a lot. Then we will train a decision tree with weights on these instances. Specifically speaking, suppose the training set is

$$\{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}, \tag{8}$$

where $y_i \in Y = \{1, 2, \cdots, N\}$. If we add weight α_i to class i $(1 \leq i \leq N)$, then training the weighted decision tree with such weights is equivalent to training a decision tree with samples:

$$\begin{array}{c} (x_1, y_1) \times \alpha_{y_1}, \\ (x_2, y_2) \times \alpha_{y_2}, \\ \vdots \\ (x_n, y_n) \times \alpha_{y_n}, \end{array} \tag{9}$$

where we duplicate sample (x_i, y_i) by α_{y_i} times.

Adopting this technique, the inclination to classify a new instance to major classes reduces a lot.

2.4 Support Vector Classifier

The support vector classifier is also termed with support vector machine for classification.

Estimating parameters of common SVM (linear SVM) can be done by solving the following optimization problem:

$$\begin{aligned} &\min_{\beta, \beta_0} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n} \xi_i \\ &\text{subject to } \xi_i \geq 0, y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i, \quad \forall i \end{aligned} \tag{10}$$

We can solve it by writing its dual problem:

$$\begin{aligned} &\max_{\alpha_i} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j x_i^T x_j, \\ &\text{subject to } 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{n} \alpha_i y_i = 0. \end{aligned} \tag{11}$$

In order to handle data that cannot be linear separated, we introduce the kernel function, which elevate the feature x_i into $h(x)$. If $\langle h(x), h(y)\rangle = K(x, y)$, where this $K(x, y)$ is called the kernel, then the dual problem becomes

$$\begin{aligned} &\max_{\alpha_i} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j K(x_i, x_j), \\ &\text{subject to } 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{n} \alpha_i y_i = 0, \end{aligned} \tag{12}$$

and the prediction step can be written as

$$f(x) = h(x)^T\beta + \beta_0 = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i) + \beta_0. \tag{13}$$

We can classify the instance x according to the sign of $f(x)$.

In our implementation, we use the RBF kernel

$$K(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right) \tag{14}$$

to make classification.

2.5 Discriminant Analysis Classifier

The discriminant analysis classifiers we implemented include the linear discriminant analysis classifier (LDA) and quadratic discriminant classifier (QDA).

2.5.1 Linear Discriminant Analysis Classifier

In LDA setting, the underlying assumption is that data in each class satisfies Gaussian distribution. Furthermore, we also assume the covariance matrix in each class shares the same one.

To use this model to classify a new instance, firstly, we need to estimate the probability of each class $\hat{\pi}_k$, the mean of the data distribution of each class $\hat{\mu}_k$, and the covariance matrix Σ:

- $\hat{\pi}_k = N_k/N$, where N_k is the number of instances in class k and N is the total number of instances. $(k = 1, 2)$,
- $\hat{\mu}_k = \sum_{y_i=k} x_i/N_k$,
- $\hat{\Sigma} = \sum_{k=1}^{2} \sum_{y_i=k} (x - \hat{\mu}_k)(x - \hat{\mu}_k)^T/(N - 2)$.

Then we can use the following quantity

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log \pi_k \tag{15}$$

to determine the estimated class of instance x, according to

$$\hat{y} = \arg\max_{k\in\{1,2\}} \delta_k(x). \tag{16}$$

2.5.2 Quadratic Discriminant Analysis Classifier

In LDA's setting, we assume the covariance matrix in each class to be the same, but sometimes this assumption do not always hold, and we need to estimate the covariance matrix for each class. This estimation is called quadratic discriminant analysis.

The estimating method for the covariance in class k is

$$\hat{\Sigma}_k = \frac{1}{N_k - 1} \sum_{y_i=k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T, \tag{17}$$

where N_k is the total number of training data classified into class k, and μ_k is the estimated mean according to the formula in LDA.

In QDA setting, we can use the following quantity

$$\delta_k(x) = -\frac{1}{2}\log|\hat{\Sigma}_k| - \frac{1}{2}(x - \mu_k)^T\Sigma_k^{-1}(x - \mu_k) + \log \pi_k \tag{18}$$

to determine the estimated class of instance x, according to

$$\hat{y} = \arg\max_{k\in\{1,2\}} \delta_k(x). \tag{19}$$

Note that the QDA classifier is similar to the naive Bayes classifier, only lack of the assumption that each feature is independent given the class label. Hence if we assume the covariance matrices $\hat{\Sigma}_k$ to be diagonal, then we will obtain the same results to naive Bayes classifier.

2.6 K-Nearest Neighbor Classifier

The K-Nearest Neighbor Classifier (KNN Classifier) is the simplest kernel smoothing classifier. Its idea is to label a new instance according to the majority among the labels of its nearest k instances.

Specifically, suppose the training set is denoted by $\{(x_i, y_i)\}_{1\leq i\leq n}$ ($y_i \in \{0, 1\}$), and we have a new instance x. If the k nearest neighbors to x among x_i are $x_1 \cdots, x_k$, then we will label it according to

$$\hat{y} = \begin{cases} 0 & \text{if } \sum_{i=1}^{k} y_i \leq k/2, \\ 1 & \text{if } \sum_{i=1}^{k} y_i > k/2. \end{cases} \tag{20}$$

The training process of the KNN method takes very short time. However, in the predicting process, for every new instance, we need to search over the training set to find k nearest neighbor to the new instance, which will be quite time costly if the training set is extremely large.

2.7 Boosting Methods

We will discuss two boosting methods we have implemented in this subsection:

- Adaboost
- XGboost

2.7.1 Adaboost

Adaboost method is one of the most commonly used ensemble method aiming to improve the accuracy on the training set.

The adaboost method can be viewed as a method which improves the performance on the training set by training a model with larger weight on data misclassified and smaller weight on data classified correctly.

Specifically, suppose we have $m-1$ classifier $G_1, \cdots, G_{m-1}$ with weight $\beta_1, \cdots, \beta_{m-1}$. And we have function f_i with $f_i(x) = f_{i-1}(x) + \beta_i G_m(x)$. Then we will train the m-th classifier G_m and its weight β_m to minimize

$$\beta_m, G_m = \arg\min_{\beta, G} \sum_{i=1}^{n} \exp(-y_i(f(x_i) + \beta G(x_i))), \tag{21}$$

and then we let

$$f_m(x) = f_{m-1}(x) + \beta_m G_m(x). \tag{22}$$

2.7.2 XGBoost

This part refers to the paper [3]. XGBoost is the abbreviation of "extreme gradient boosting" method. This is also a method using the boosting tree technique. But instead of using gradient boosting, the main idea of XGBoost is to use both the first and the second order derivative to approximate the loss function. In order to understand the boosting tree technique better, we refine the definition of tree as follows. We assign each leaf with a score, describing how good the current leaf is. And when we use the decision tree for some tasks, we will go through the tree from the the root to leaves, and finally get the score corresponding to that leaf.

To describe the training process, we first define the loss function of the model:

$$\mathcal{L} = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k), \tag{23}$$

where K is the number of trees in this model, Ω is the regularization function, and $\hat{y}_i$ is defined as

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i). \tag{24}$$

Suppose the tree we aim to train at t-th iteration is f_t, where $f_t(x)$ denotes the score we will obtain by putting the instance x into tree f_t. Then if we want to boost the model using this tree f_t, the loss function is

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + const, \tag{25}$$

where $\hat{y}_i^{(t-1)}$ is the predicted value on x_i according to the first $t-1$ trees. Next, in order to minimize this loss, we find the Taylor expansion:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + const, \tag{26}$$

where

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}). \tag{27}$$

Normally, we choose the function l as the squared loss function

$$l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2. \tag{28}$$

Then it is easy to calculate

$$g_i = 2(\hat{y}^{(t-1)} - y_i), \quad h_i = 2. \tag{29}$$

Furthermore, the regularization term $\Omega(f)$ is often chosen to be

$$\Omega(f) = \gamma T + \frac{\lambda}{2} \sum_{j=1}^{T} \omega_j^2, \tag{30}$$

where T is the number of leaves in the tree f, and ω_j is the score according to leaf j.

In order to find the most proper tree model f_t, we only need to find the optimizer of (26).

In comparison with gradient boosting methods, the XGBoost methods use order-2 Taylor expansion, while gradient boosting use order-1 Taylor expansion. (only consider the gradient of the loss functiion) Specifically, in gradient boosting methods, we consider the following estimation of the loss function:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) \right] + \Omega(f_t) + const \tag{31}$$

Hence if we let $f_t(x) = \alpha_t f_t^*(x)$, then we only need to fit the estimator f_t^* to the negative gradient direction, and in such ways we can obtain the largest descent if we increase the parameter α_t from zero.

Therefore, due to the difficulty in obtaining the second order derivative, XGBoost is usually slower than gradient boosting. But the second order derivative can make XGBoost more accurate than gradient boosting, and hence in less iterations, XGBoost methods can achieve similar results to gradient boosting methods.

2.8 Other Ensemble Methods

Methods discussed in this subsection are similar in that they all adopt the idea of bagging or voting to make the classifier more powerful. We will discussed voting method, bagging method and the random forest method here in this subsection.

2.8.1 Voting

Voting is a method balancing different classifiers in order to achieve a better classifier.

Suppose their are n classifier $f_1, \cdots, f_m$. Then we assign weights $w_1, \cdots, w_m$ to each classifier. Then the idea of voting is to choose the label which is most likely been classified. That is, we choose

$$\hat{y} = \max_y \#\{f_m(x) = y, 1 \le i \le m\}. \tag{32}$$

2.8.2 Bagging

Bagging adopts the idea of voting method, uses one model to train several classifiers by bootstrap, and then use these classifiers to vote to get the prediction. This type of methods can reduce the variance, and is easy for parallel computing.

2.8.3 Random Forest

Based on Bagging, Random Forest goes one step further. In this method, the basic model is chosen to be decision tree, and in the bootstrap step, not only does the data apply bootstrap technique, but also the features are selected randomly.

3 Outlier/Novelty Detection Methods

In this section, we will introduce some standard outlier detection methods. As we have mentioned in Introduction, all these methods assume loss of labels and try to empirically figure out whether a collected data point is an “outlier” or a new data point is a “novelty”.

3.1 Robust Covariance Estimation

The estimator obtained from the algorithm is also called Minimum Covariance Determinant (MCD) estimator. It is, to some extent, similar to LDA. The algorithm fits only one Gaussian distribution to model the normal data in a robust way. More rigorously, it tries to find

$$\frac{n_{samples} + n_{features} + 1}{2} \tag{33}$$

observations whose empirical covariance has the smallest determinant and yields a "pure" subset of observations. We then could estimate so-called Mahalanobis distance between any data point x and the center of fitted Gaussian distribution, i.e.,

$$d(x; \mu, \Sigma) = (x - \mu)' \Sigma^{-1} (x - \mu). \tag{34}$$

If it's value is too large, then x is classified as an anomaly, else a normal data point. Commonly, there is no tuning parameters. For more detailed description, readers could refer to [7].

3.2 Isolation Forest

The algorithm borrows ideas form Random Forest. It first randomly select a feature, and randomly select a splitting value between the maximum and minimum values of the selected feature. Then it recursively conduct this process to form a tree. The number of splittings to "isolate" a sample is equivalent to the path length from the root node to the terminating node. Finally, the path length is averaged over a forest of such random trees, and embedded in an anomaly score, which is regarded as a measure of normality. The anomaly score in this algorithm is defined as

$$s(X, n) = 2^{-\frac{E[h(X)]}{c(n)}}, \tag{35}$$

where $h(X)$ is the path length of observation X, $E[h(X)]$ is the average of $h(X)$ through the forest of trees. $c(n)$ is the average path length of unsuccessful search in a Binary Search Tree and n is the number of its external nodes. If we ignore the detailed derivation of $c(n)$, we could regard it as a normalization term. More on the anomaly score and its components can be found in [5].

In fact, random partitioning produces shorter paths for anomalies because anomalies have high probability of being close to the extreme values of a feature. Hence, if for a particular sample, the forest of random trees collectively produce shorter path lengths, it is highly likely to be an anomaly.

The tuning parameters in this algorithm are the number of trees and the proportion of outliers.

3.3 Local Outlier Factor

The algorithm computes a score (called local outlier factor) to reflect the degree of abnormality of the observations. It measures the local density deviation of a given data point with respect to its neighbors. The idea is to detect the samples that have a substantially lower density than their neighbors.

Commonly, the local density is obtained from KNN. The LOF score of an observation is equal to the ratio of the average local density of its k-nearest neighbors and its own local density. A normal instance is expected to have a local density similar to or higher than that of its neighbors, while abnormal data are expected to have much smaller local density.

Now let us describe the algorihm more rigirously. Denote $N_k(X)$ as the set of the first k nearest neighbors of data point X and $d(X, Y)$ as the distance between X and Y. Define $d_k(X)$ as the distance between X and its kth nearest neighbor. The k-reachability distance between X and Y is defined as

$$rd_k(X, Y) = \max\{d_k(Y), d(X, Y)\}. \tag{36}$$

The k-local reachability of X is defined as

$$lrd_k(X) = \frac{|N_k(X)|}{\sum_{Y \in N_k(X)} rd_k(X, Y)}. \tag{37}$$

The k-local outlier factor of X is defined as

$$lof_k(X) = \frac{\sum_{Y \in N_k(X)} lrd_k(Y)}{|N_k(X)| lrd_k(X)}. \tag{38}$$

If $lof_k(X)$ is larger than a threshold, then we classify X is an outlier.

The tuning parameters in this algorithm are the number of neighbors k and the proportion of outliers. For more detailed description of the algorithm, readers could refer to [1].

4 Our Idea: Clustering-Based Modelling Methods

In this part, we will explicitly describe our algorithm on outlier classification. We focus on modelling data distribution with different labels and apply a variant of Bayesian decision theory for classification. Such idea can be easily integrated with other supervised or unsupervised learning technique such as decision tree, ensemble methods, and EM algorithm.

4.1 Bayesian Decision Theory

To begin with, we describe some general theories. We now assume the following conditions.

1. Suppose there are k different classes $G_1, ..., G_k$.
2. For any given data $x \in \mathbb{R}^n$, the prior probability that it belongs to G_k is p_k (Thus we must have $\sum_{i=1}^k p_i = 1$). A criterion D is a partition of $\mathbb{R}^n$, $\{D_1, ..., D_k\}$, such that we classify x into G_i if and only if $x \in D_i (i \in [k])$.
3. Let $L(j|i)$ be the loss when we classify an object in G_i to G_j. Generally, when $i = j$, $L(j|i) = 0$.
4. Let $P(j|i; D)$ be the probability that we classify an object in G_i into G_j under the criterion D.

With these assumptions, we now give the following definition.

Definition 4.1. *Let $g(D) := \sum_{i=1}^k p_i \sum_{j=1}^k P(j|i; D)L(j|i)$ to be the average loss under classification criterion D. If there exists a D^* such that*

$$g(D^*) = \min_D g(D) \tag{39}$$

We call D^* the optimal Bayesian (classification) criterion.

If we add some conditions, we will have the following theorem.

Theorem 4.1. *Suppose the joint distribution density of the ith class G_i is $f_i(X)$, then the optimal Bayesian criterion $D^* = (D_1^*, ..., D_k^*)$ is*

$$D_i^* = \{X|h_i(X) < h_j(X), \forall j \neq i\}, \tag{40}$$

where $h_j(X) = \sum_{i=1}^k p_i L(j|i) f_i(X)$, i.e., the average loss when we classify X into G_j.

Proof. For detailed proof, please see the appendix. □

In our setting, $k = 2$. Then from Theorem 4.1,

$$h_1(X) = p_2 f_2(X) L(1|2), \qquad h_2(X) = p_1 f_1(X) L(2|1) \tag{41}$$

Therefore, the optimal Bayesian solution is

$$\begin{aligned} D_1 &= \{X|W(X) > d\}, \\ D_2 &= \{X|W(X) \leqslant d\}, \end{aligned} \tag{42}$$

where

$$W(X) = \frac{f_1(X)}{f_2(X)}, \qquad d = \frac{p_2 L(1|2)}{p_1 L(2|1)}. \tag{43}$$

Once we obtain the densities, the prior probabilities and the loss, we could make optimal Bayesian decision.

4.2 GMDA: Gaussian Mixture Discriminant Analysis

To determine the density $f(X)$, we have numerous choices. For both fitness of generalization and speed of optimization, we choose to model $f(X)$ using Gaussian Mixture. Many kinds of data in real life can be well approximated by this model and well-performed algorithms such as EM algorithm are widely applied for optimization.

For parameters in d(see 43), prior probabilities p_1 and p_2 could be inferred from data. However, generally, the loss $L(1|2)$ and $L(2|1)$ are determined by decision makers and we need to find a criterion to decide the ratio. Therefore, we should regard d as a tuning parameter.

Based on previous analysis, our algorithm is as follows. We split the data into Training Set, Validation Set and Test Set. We first use Training Set to model the distribution of normal data ($f_1(X)$) and anomaly data ($f_2(X)$). We then use Validation Set to determine an optimal d. Finally, we apply the tuned d to test performance on Test Set. The formal formulation is in Algorithm 1.

Algorithm 1: Gaussian Mixture Discriminant Analysis

Input: T: Training set;
V: Validation set;
n_1: The number of clusters for normal data;
n_2: The number of clusters for anomaly data;
D: The list of possible choices of d;
L: An empty list to restore *F-score*s.
Output: $f_1(X)$: Density of Normal Data;
$f_2(X)$: Density of Anomaly Data;
d: Threshold.

Apply EM algorithm to normal data in *T* to obtain a density function $f_1(X)$ with n_1 clusters;
Apply EM algorithm to anomaly data in *T* to obtain a density function $f_2(X)$ with n_2 clusters;
for d *in* D **do**
    Classify data in *V*. If $\log f_2(X) - \log f_1(X) > d$, classify X as anomaly, else normal;
    Compute *F-score* and add it into *L*;
end
$\rho \leftarrow \arg\max L$;
$d \leftarrow D[\rho]$;
return $f_1(X), f_2(X), d$.

4.3 Relation and Extension with Other Methods

4.3.1 Relation with Naive Bayes

Naive Bayes models data using Gaussian distribution, which is similar to GMDA. However, there is much difference between the two methods.

1. For convenience of optimization, Naive Bayes use one Gaussian distribution, while we use Gaussian Mixture Model.
2. In Naive Bayes, a crucial assumption is independence between variables. However, we do not impose this assumption. In fact, the goal of using Gaussian Mixture Model is to capture the correlation between variables.

4.3.2 Relation with LDA and QDA

Both LDA and QDA (see Section 2.5) model data from different classes seperately. Also, they embed prior probabilities from Bayesian view. However, there is difference between LDA/QDA and GMDA.

1. In LDA/QDA, we model the data using one Gaussian distribution, while in GMDA we model with Gaussian Mixture.
2. LDA/QDA use distribution of different classes to estimate prior probabilities. While in GMDA, we also embed loss of misclassification and use validation sets to tune the threshold.

4.3.3 Relation with Robust Covariance Estimator(RCE)

A big difference between RCE and GMDA is that RCE only models one class of data, i.e., the normal data. Enlightened by RCE, we can combine the two algorithms with each other. We can use normal data in the training set to find a Gaussian Mixture model, and then directly tuning the threshold on validation set without further modelling anomaly data. This is GMDA with normal data only, "GMDA-n".

4.3.4 Relation with Decision Tree

If we look at Algorithm 1 again, we could get further insight. First, $\log f_2(X) - \log f_1(X)$ can be thought as a projection from $\mathbb{R}^n$ to $\mathbb{R}$. Second, the validation process to choose optimal d can be thought as finding a tree with one layer that maximize the *F-score*. This is equivalent to maximize

$$2 \Big/ \left(\frac{\text{TP} + \text{FP}}{\text{TP}} + \frac{\text{TP} + \text{FN}}{\text{TP}} \right) = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}, \tag{44}$$

or equivalently, minimize

$$\frac{\text{FP} + \text{FN}}{\text{TP}}. \tag{45}$$

In Decision Tree setting with two classes, this is, to some extent, similar to minimizing the misclassification rate. This enlighten us to combine Algorithm 1 with Decision Tree Classifier.

Algorithm 2: GMDA with Decision Tree Classifier

Input: T: Training set;
V: Validation set;
n_1: The number of clusters for normal data;
n_2: The number of clusters for anomaly data;
Output: $f_1(X)$: Density of Normal Data;
$f_2(X)$: Density of Anomaly Data;
DTC: A Decision Tree Classifier.

1 Apply EM algorithm to normal data in *T* to obtain a density function $f_1(X) = \sum_{i=1}^{n_1} \pi_i f_{1,i}(X)$;
2 Apply EM algorithm to anomaly data in *T* to obtain a density function $f_2(X) = \sum_{j=1}^{n_2} \pi_i f_{2,j}(X)$;
3 Construct new features based on $f_1(X)$, $f_2(X)$, $f_{1,i}(X)$, $f_{2,j}(X)(1 \leqslant i \leqslant n_1, 1 \leqslant j \leqslant n_2)$;
4 Use the new features above to build a Decision Tree Classifier;
5 Tune the parameters using *V*;
6 **return** $f_1(X), f_2(X)$, *DTC*.

4.3.5 Relation with Ensemble Methods

Once we could build a Decision Tree Classifier based on new features, we could embed it into ensemble algorithms. Bagging, voting, and XGBoosting are all excellent choices for enhancing performance, though parameters remain to be tuned to achieve satisfying results.

5 Case Study: Credit Card Fraud Detection

5.1 Data Description

We use an open dataset on Kaggle. The dataset contains 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, which indicates that the number of frauds is small. The positive class (frauds) account for 0.172% of all transactions, which means that the dataset is highly unbalanced.

The dataset contains only numerical input variables which are the result of a PCA transformation. Due to confidentiality issues, the original features and more background information about the data are not provided. 28 of 30 features are the principal components obtained with PCA, and the only features which have not been transformed with PCA are "Time" and "Amount". Feature "Time" contains the seconds elapsed between each transaction and the first transaction in the dataset. Feature "Amount" is the transaction amount. Feature "Class" is the response variable and it takes value 1 if the transanction is fraud and 0 otherwise.

5.2 Data Preprocessing

We do not make further process on the 28 transformed features. As to Feature "Time", we replace the time elasped between each transaction and the first transaction with the difference between each transaction and its former one. In practice, we do find that such operation generally enhances the performance of various algorithms.

We draw the histogram of each feature corresponding to the response variable. From the charts, we have some confidence that the assumption of our algorithm holds. The distribution can be well approximated by Gaussian Mixture. Of course, rigorous demonstration is left to numerical experiments in the following subsections.

As for data splitting, we set the proportion among training, validation, and test set as 6:2:2. If the training set is small, say, less than half of the whole dataset, then training or modelling is difficult, especially for anomaly data. If the validation set or test set is too small, then our evaluation metric will be vulnerable to a tiny change in anomaly prediction. Our choice is a tradeoff. It's worth noting that we keep the ratio between normal data and anomalies the same in all the 3 sets. This is essential for us to obtain a fair result.

5.3 Implementation Details

In this section, we will discuss the implementation of previously described algorithms, and make comparison among them from experiments. These numerical experiments are carried out on a computer with 2 Intel Core i7-6500U CPUs and 16GiB RAM.

5.3.1 Supervised Learning Methods

For supervised learning methods, we have implemented several methods. The setting of these methods are described below.

- **Decision Tree(DTC):** We choose the criterion for splitting nodes and pruning the tree to be Gini index, and the maximum depth of the tree to be 6. Furthermore, we have done experiments with weighted tree, whose purpose is to balance the number of instances in each class. We have chosen two type of weights: $1:5$ and $1:10$.
- **LDA with bagging(LDA-Bag):** We choose the number of bagging LDA estimators to be 5.
- **QDA with bagging(QDA-Bag):** We choose the number of bagging QDA estimators to be 11.
- **Random Forest(RF):** We choose the criterion for splitting nodes and pruning the tree to be Gini index, and the number of estimators to be .
- **XGBoost(XGB):** We choose the maximum depth of each tree to be 4, regularization term λ to be 0.5 and γ to be 0.
- **Neural Network(NN):** We choose the structure of the neural network to be a 4-layer fully connected neural network, with 5, 4, 3, 3 neurons in each layers respectively.

5.3.2 Outlier Detection Methods

For outlier detection methods, we have implemented three algorithms mentioned above. The parameters are as follows.

- **Robust Covariance Estimation(RCE):** We directly run the fitting process of the algorithm on the training set, which contains both normal and anomaly data. We then tune the threshold for d (see 34) using validation set. The optimal threshold we select is 1.01×10^6.
- **Local Outlier Factor(LOF):** We set the number of neighbors k to be 50 and the proportion of outliers as 0.0008. During the training process, we only use the normal data in the training set because we observe that directly running the fitting process on the whole training set will worsen the performance.
- **Isolation Forest(ILF):** We set the number of trees to be 250 and the proportion of outliers as 0.005. We only use the normal data in the training set, as what we have done in Local Outlier Factor.

5.3.3 GMDA and Related Methods

We have also implemented several algorithms related to GMDA. We now explain them as follows.

- **Gaussian Mixture Discriminant Analysis(GMDA):** We model normal data with $n_1 = 3$ clusters and anomaly data with $n_2 = 3$ clusters in the training set. Note that all of the following methods are based on the two fitted gaussian mixture (normal and anomaly).
- **Gaussian Mixture Discriminant Analysis with normal data only(GMDA-n):** We use the fitted distribution of normal data obtained by GMDA and then tune a threshold using the validation set.
- **Gaussian Mixture with Decision Tree Classifier(GM-dtc):** The algorithm is consistent with 2. The features we select are

$$f_1(X) - f_2(X),\ f_{1,i}(X),\ f_{2,j}(X)\ (1 \leqslant i \leqslant n_1, 1 \leqslant j \leqslant n_2). \tag{46}$$

  We choose Entropy Loss as the criterion. To prevent overfitting, we restrict the maximal depth of the tree to 5.
- **Gaussian Mixture with Bagging(GM-bag):** This algorithm is based on GM-DTC. We choose 11 tree estimators with the same features in (46).
- **Gaussian Mixture with Voting(GM-vote):** This algorithm is based on GM-DTC. To increase diversity, we choose 9 different tree estimators with the same features in (46), but with different weights(see Section 2.3.1), which are

$$100:1,\ 50:1,\ 20:1,\ 10:1,\ 1:1,\ 1:10,\ 1:20,\ 1:50,\ 1:100. \tag{47}$$

- **Gaussian Mixture with XGBoost(GM-xgb):** This algorithm is based on GM-DTC. We choose the maximal depth of each tree to be 5, regularization term λ to be 0.5, and γ to be 0.

5.4 Comparison between Different Methods

We have implemented several previous described algorithms together with algorithms proposed by ourselves, and the results can be viewed in Table 1.

From this table, we observe that tree-based algorithms works better than other supervised learning methods. However, some tree based methods also suffer from relatively low recall. This is due to the fact that anomaly data is very rare, and thus tree-based classifiers tend to classify the data into the normal class. If we add weights to the tree, then the previously described problem will be reduced a little. But the F-score will remain nearly the same. Apart from tree-based methods, from the numerical experiments, other supervised learning methods seldom rival the tree-based methods, and lots of these methods suffer from the problem we discussed before. As for ensemble learning technique, we can see that they indeed improve the performance of basic methods, but the improvement is not significant considering the much time they spend.

In contrast, the Gaussian mixture model methods seems to be affected by this reason less likely. These Gaussian models extract the structure of normal and anomaly data correctly. And using these Gaussian models instead of raw data to make prediction can seriously reduce the effect of data imbalance. And from the table, we can notice that all Gaussian mixture models works pretty well.

Furthermore, when it comes to outlier detection methods, their results are not good enough. The main reason for this is the wrong assumptions we have made. Outlier detection methods usually rely on the assumption that outliers are instances outside the normal data distribution, and there is no explicit distribution for them. But in this dataset, the distribution of outliers seems to have certain structure, which contradicts to the underlying assumption of outlier detection methods.

Methods	Train			Valid			Test			Time	
	R	P	F	R	P	F	R	P	F	T	P
Logistic	58.6	85.2	64.9	61.2	88.2	67.6	59.6	86.8	66.0	8.88	0.01
NB	82.4	5.9	16.5	84.7	6.1	17.1	82.8	5.8	16.2	0.09	0.02
DTC	83.4	98.4	87.5	81.6	90.9	84.3	79.8	91.9	83.2	4.02	0.01
DTC 1:5	84.6	97.8	88.3	80.4	91.3	83.5	78.2	92.2	82.0	4.19	0.01
DTC 1:10	85.8	94.1	88.2	83.4	86.0	84.2	79.8	86.2	81.7	4.16	0.01
LDA	74.2	85.5	77.4	82.7	88.0	84.2	77.8	87.5	80.5	0.70	0.01
LDA-Bag	73.8	85.4	77.1	81.8	87.9	83.6	77.1	87.3	80.0	4.20	0.02
QDA	87.5	5.4	15.4	**90.8**	5.6	16.1	84.8	5.3	15.1	0.34	0.03
QDA-Bag	87.4	6.0	16.9	90.7	6.3	17.6	**84.9**	5.9	16.6	5.30	0.27
RF	**94.4**	**99.6**	**95.9**	80.0	92.8	83.5	77.1	**94.1**	81.6	11.78	0.06
NN	78.0	86.5	80.4	82.7	88.0	84.2	80.8	85.1	82.1	7.54	0.01
XGBoost	87.5	99.2	90.8	82.7	**94.2**	85.9	80.8	**94.1**	**84.5**	37.29	0.13
RCE	74.9	3.5	10.4	79.6	3.9	11.5	80.8	3.8	11.3	20.19	0.05
LOF	50.8	54.5	51.9	55.1	53.5	54.6	50.5	51.5	50.8	368.53	122.77
ILF	47.5	14.1	27.5	46.9	13.6	26.8	47.5	14.3	27.7	368.53	3.96
GMDA	79.0	80.3	79.4	85.7	81.6	84.4	80.8	82.5	81.3	55.55	0.11
GMDA-n	77.6	82.1	78.9	84.7	83.0	84.2	79.8	84.9	81.3	6.69	0.06
GM-dtc	82.7	92.4	85.5	83.7	85.4	84.2	78.8	89.7	81.8	1.99	0.20
GM-bag	83.4	94.6	86.5	84.7	90.2	86.3	79.8	92.9	83.4	8.23	0.25
GM-vote	82.0	95.3	85.7	84.7	88.3	85.8	78.8	94.0	82.9	7.95	0.25
GM-xgb	86.8	98.8	90.2	84.7	91.2	**86.6**	80.8	**94.1**	**84.5**	13.88	0.33

Table 1: Comparison between different methods. Each row represent one kind of algorithm. The first eleven rows are about supervised learning method, and the next six rows are about Gaussian mixture model methods, and the last three rows are about outlier detection methods. We have presented the recall (R), precision (P) and F-score (F) on the training, validation and test sets above, where the definition of recall, precision and F-score are given in the first section. The last two columns are about time costs on training process and prediction process. (on test set)

6 Conclusion

Our work for Outlier Classification ends up here, but new ideas for improvement and continuous passion for research will never fade. We now give a brief summary for what we have done and give some outlooks.

The Outlier Classification mainly consists of two challenges. The first one is that highly unbalanced dataset may lead to deterioration of standard supervised learning technique. The second one is that the small number of outliers(anomalies) may impose annoying influence on our evaluation performance. We propose several techniques to deal with this problem. On the side of evaluation, we restrict ourselves on *recall*, *precision* and a weighted *F-score* as our evaluation metric. To deal with the potential unstable results, we propose to run several times of our algorithms and obtain the average.

On the side of algorithms, based on Bayesian decision theory, we propose GMDA to utilize the advantage of clustering methods and to capture the different distribution between normal and anomaly data. From the view of Naive Bayes/LDA/QDA, we extend the modelling of distribution to Gaussian Mixture, and regard some parameters to be tuned automatically as thresholds by the algorithm. From the view of Decision Tree Classifier and other ensemble methods, the algorithm deal with raw data to obtain highly nonlinear features to enhance performance. The algorithm can also be combined with several other supervised learning techniques.

As to numerical experiments, we conduct a case study to compare our methods with standard ones. Surprisingly, tree-based methods perform relatively well on this setting, which indicates that the unbalanced dataset does not affect peformance too much when we use trees. Our algorithms perform quite well on this dataset. Of course, the success of our algorithms may, to some extent, depend on the organized shape of the dataset. It's worth noting that traditional outlier detection methods perform badly in our setting. This may be caused by lack of information of data labels.

Now we provide some outlooks. For supervised learning methods, we want may further investigate some combined methods such as kernel smoothing. Due to the time limit, we do not implement them. Furthermore, deep-learning based methods are prospective. After all, from our experiments, a simple neural network performs comparable to the best performed algorithms. For our proposed algorithms, apart from directly geenrating features and embed them into trees, we could also split trees first and then apply Gaussian Mixture, though this is left to future work. We also want to mention that for outlier detection methods, we have potential for trying with other algorithms in the literature, since the three algorithms we implemented are the most standard and the simplest.

As we have mentioned in Introduction, Outlier Classification is very common in many business setting. However, there is still room for improvement even with numerous proposed methods. Of course, we hope that we can find more efficient methods, both in time and space. This will be left for future research.

References

[1] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *ACM sigmod record*, volume 29, pages 93–104. ACM, 2000.

[2] Guilherme O. Campos, Arthur Zimek, Jörg Sander, Ricardo J. G. B. Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael E. Houle. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery*, 30(4):891–927, Jul 2016.

[3] T Chen and C Guestrin. Xgboost: A scalable tree boosting system (2016). *arXiv preprint arXiv:1603.02754*.

[4] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, NY, USA:, 2001.

[5] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE, 2008.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[7] Peter J Rousseeuw. Least median of squares regression. *Journal of the American statistical association*, 79(388):871–880, 1984.

A Proof of Theorem 4.1

We have

$$\begin{aligned} g(D^*) &= \sum_{i=1}^{k} p_i \sum_{j=1}^{k} P(j|i; D^*) L(j|i) \\ &= \sum_{i=1}^{k} p_i \sum_{j=1}^{k} L(j|i) \int_{D_j^*} f_i(X) dX \\ &= \sum_{j=1}^{k} \int_{D_j^*} \sum_{i=1}^{k} p_i L(j|i) f_i(X) dX \\ &= \sum_{j=1}^{k} \int_{D_j^*} h_j(X) dX. \end{aligned} \tag{48}$$

For any partition of $\mathbb{R}^n$, $D = \{D_1, ..., D_k\}$, we also have

$$g(D) = \sum_{j=1}^{k} \int_{D_j} h_j(X) dX. \tag{49}$$

Therefore,

$$\begin{aligned} g(D^*) - g(D) &= \sum_{j=1}^{k} \int_{D_j^*} h_j(X) dX - \sum_{t=1}^{k} \int_{D_t} h_t(X) dX \\ &= \sum_{j=1}^{k} \sum_{t=1}^{k} \int_{D_j^* \cap D_t} \left(h_j(X) - h_t(X) \right) dX \\ &\leqslant 0. \end{aligned} \tag{50}$$

In fact, $\forall j, t \in [k]$, we have $h_j(X) \leqslant h_t(X)$ on $D_j^* \cap D_t$ from the definition of D^*. This completes the proof.