A Brief Survey on Outlier Classification

Zeyu Jia* Feng Zhu*
1600010603 1600010643

School of Mathematical Science, Peking University

January 18, 2019

Abstract

12

1 Introduction

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, and random forest/bagging/voting methods. These methods and techniques 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

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

and we will maximize the log-likelihood:

$$l(\beta) = \sum_{i=1}^{N} y_i \log P(y=y_i|X=x_i,\beta) + (1-y_i)\log P(y \neq y_i|X=x_i,\beta). \tag{2}$$

By finding the optimal parameter β, then we can use (1) to make prediction.

*The authors are arranged lexicographically.

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

$$P(X|y) = \prod_{i=1}^{m} P(x_i|y), \tag{3}$$

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:

$$\hat{y} = \arg\max_y P(y|X) = \arg\max_y \frac{P(y)\prod_{i=1}^{m} P(x_i|y)}{P(X)} = \arg\max_y P(y)\prod_{i=1}^{m} P(x_i|y). \tag{4}$$

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

$$P(x_i|y) \sim \mathcal{N}(\mu_y, \sigma_y), \tag{5}$$

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 (4).

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. The 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 features (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

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 multiply instances in each class by a weight. After doing that, we aim to let instance weight in each class multiplied by the number of instances in that class not vary a lot. (This has the same effect as duplicating instances in minor classes several times.) Then we will train a decision tree with these instances and weights.

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:

$$\begin{aligned} &\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n}\xi_i \\ &\text{subject to } \xi_i \geq 0, y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i, \quad \forall i \end{aligned} \tag{6}$$

We can solve it by writing its dual problem:

$$\begin{aligned} &\max_{\alpha_i} \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j, \\ &\text{subject to } 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{n}\alpha_i y_i = 0. \end{aligned} \tag{7}$$

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

$$\begin{aligned} &\max_{\alpha_i} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j), \\ &\text{subject to } 0 \le \alpha_i \le C, \quad \sum_{i=1}^{n} \alpha_i y_i = 0, \end{aligned} \tag{8}$$

and the prediction step can be written as

$$f(x) = h(x)^T \beta + \beta_0 = \sum_{i=1}^{n} \alpha_i y_i K(x, x_i) + \beta_0. \tag{9}$$

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

In our implementation, we use the RBF kernel

$$K(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right) \tag{10}$$

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

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k \tag{11}$$

to determine the estimated class of instance x, according to

$$\hat{y} = \arg \max_{k \in \{1,2\}} \delta_k(x). \tag{12}$$

2.5.2 Quadratic Discriminant Analysis Classifier

In LDA's setting, we assume the covariance matrix in each class to be the same one, but sometimes this assumption cannot 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

$$\hat{\Sigma}_k = \frac{1}{N_k - 1} \sum_{y_i=k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T, \tag{13}$$

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

$$\delta_k(x) = -\frac{1}{2} \log |\hat{\Sigma}_k| - \frac{1}{2} (x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) + \log \pi_k \tag{14}$$

to determine the estimated class of instance x, according to

$$\hat{y} = \arg \max_{k \in \{1,2\}} \delta_k(x). \tag{15}$$

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 its nearest k instances label.

Specifically, suppose the training set is denoted by $\{(x_i, y_i)\}_{1\le i\le n}$ ($y_i \in \{0,1\}$), 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

$$\hat{y} = \begin{cases} 0 & \text{if } \sum_{i=1}^k y_i \le k/2, \\ 1 & \text{if } \sum_{i=1}^k y_i > k/2. \end{cases} \tag{16}$$

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

$$\beta_m, G_m = \arg\min_{\beta, G} \sum_{i=1}^n \exp(-y_i(f(x_i) + \beta G(x_i))), \tag{17}$$

and then we let

$$f_m(x) = f_{m-1}(x) + \beta_m G_m(x). \tag{18}$$

2.7.2 XGboost

This part refers to the paper [2]. 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.

Specifically speaking, we first define the loss function of the model:

$$\mathcal{L} = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k), \tag{19}$$

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

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i). \tag{20}$$

When we aim to train the t-th tree f_t, the loss function is

$$\mathcal{L}^{(t)} = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + const, \tag{21}$$

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:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^n \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + const, \tag{22}$$

where

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}). \tag{23}$$

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

$$l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2. \tag{24}$$

Then it is easy to calculate

$$g_i = 2(\hat{y}^{(t-1)} - y_i), \quad h_i = 2. \tag{25}$$

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

$$\Omega(f) = \gamma T + \frac{\lambda}{2} \sum_{j=1}^{T} \omega_j^2, \tag{26}$$

where T is the number of leaves in the tree f, and ω_j is the corresponding score with respect to node j.

In order to find the most proper tree model f_t, we only need to find the optimizer of (22).

2.8 Random Forest / Bagging / Voting 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 Method

The voting method 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

$$\hat{y} = \max_y \#\{f_m(x) = y, 1 \leq i \leq m\}. \tag{27}$$

2.8.2 Bagging Method

The bagging method 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 Method

The well-known random forest method is based on bagging method, but 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 applied 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. All these methods assume loss of labels and try to empirically figure out whether a collected data point is an "outlier"(Outlier Detection) or an new data point is a "novelty"(Novelty Detection).

3.1 Robust Covariance Estimator

The estimator is also called Minimum Covariance Determinant (MCD) estimator. It is, to some extent, similar to LDA. However, since it assumes lack of label, the algorithm only fits one Gaussian distribution to model the normal data. To fit the data robustly, it tries to find

$$\frac{n_{samples} + n_{features} + 1}{2} \tag{28}$$

observations whose empirical covariance has the smallest determinant and yields a "pure" subset of observations from which we can estimate so-called Mahalanobis distance for any data point x, i.e.,

$$d(x; \mu, \Sigma) = (x - \mu)' \Sigma^{-1} (x - \mu). \tag{29}$$

If it's value is too large, then x is classified as an anomaly, else a normal data point. For more detailed description, readers could refer to [4].

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

$$s(X, n) = 2^{-\frac{E[h(X)]}{c(n)}}, \tag{30}$$

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 external nodes. More on the anomaly score and its components can be found in [3].

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.

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 his k-nearest neighbors and its own local density. A normal instance is expected to have a local density similar to 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 reachability distance between X and Y is defined as

$$rd_k(X, Y) = \max\{d_k(Y), d(X, Y)\}. \tag{31}$$

The local reachability of X is defined as

$$lrd_k(X) = \frac{|N_k(X)|}{\sum_{Y \in N_k(X)} rd_k(X, Y)}. \tag{32}$$

The local outlier factor of X is defined as

$$lof_k(X) = \frac{\sum_{Y \in N_k(X)} lrd_k(Y)}{|N_k(X)| lrd_k(X)}. \tag{33}$$

If $lof_k(X)$ is larger than a threshold, then we classify X is an outlier. In practice, the proportion of outliers is given by decision makers as an input and the threshold is computed by the algorithm. 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 seperately 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 and EM algorithm.

4.1 Bayesian Decision Theory

We assume the following conditions.

1. Suppose there are k different classes $G_1, ..., G_k$.
2. For any given data x, the prior probability that it belongs to G_k is p_k (Thus we must have $\sum_{i=1}^{k} p_i = 1$).
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 obtain 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*

$$g(D^*) = \min_D g(D) \tag{34}$$

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*

$$D_i^* = \{X|h_i(X) < h_j(X), \forall j \neq i\}, \tag{35}$$

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,

$$h_1(X) = p_2 f_2(X) L(1|2), \qquad h_2(X) = p_1 f_1(X) L(2|1) \tag{36}$$

Therefore, the optimal Bayesian solution is

$$\begin{aligned} D_1 &= \{X|W(X) > d\}, \\ D_2 &= \{X|W(X) \leqslant d\}, \end{aligned} \tag{37}$$

where

$$W(X) = \frac{f_1(X)}{f_2(X)}, \qquad d = \frac{p_2 L(1|2)}{p_1 L(2|1)}. \tag{38}$$

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 38), 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-scores*.
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
ρ ← arg max L;
return f_1(X), f_2(X), D[ρ].
```

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 Naive Bayes and GMDA.

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 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

$$2 \Big/ \left(\frac{\text{TP} + \text{FP}}{\text{TP}} + \frac{\text{TP} + \text{FN}}{\text{TP}} \right) = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}, \tag{39}$$

or equivalently, minimize

$$\frac{\text{FP} + \text{FN}}{\text{TP}}. \tag{40}$$

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.

```
Apply EM algorithm to normal data in T to obtain a density function f_1(X) = Σ_{i=1}^{n_1} π_i f_{1,i}(X);
Apply EM algorithm to anomaly data in T to obtain a density function f_2(X) = Σ_{j=1}^{n_2} π_i f_{2,j}(X);
Construct new features: log f_2(X)/f_1(X), log f_{1,i}(X), log f_{2,j}(X), log f_{2,j}(X)/f_{1,i}(X) (1 ⩽ i ⩽ n_1, 1 ⩽ j ⩽ n_2);
Use the new features above to build a Decision Tree Classifier;
Tune the parameters using V;
return f_1(X), f_2(X), DTC.
```

5 Numerical Experiment

In this section, we will discuss the implementation of previously described algorithms, and make comparison between them from numerical experiments.

5.1 Supervised Learning Methods

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

Table 1:

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
Tree	83.4	98.4	87.5	81.6	90.9	84.3	79.8	91.9	83.2	4.02	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	74.9	85.7	77.9	82.7	88.0	84.2	77.8	87.5	80.5	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	87.1	5.7	16.2	90.8	6.0	17.1	84.8	5.7	16.0	5.30	0.27
RF	96.3	100.0	97.4	81.6	93.0	84.8	74.7	93.7	79.7	11.78	0.06
NN	78.3	84.9	80.2	82.7	84.4	83.2	78.8	82.1	79.8	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
GMDA	79.3	81.0	79.8	83.7	82.8	83.4	79.8	83.2	80.8	53.76	0.13
GM-dtc	85.1	92.3	87.2	83.7	87.2	84.7	80.8	90.9	83.7	4.27	0.24
GM-bagging	85.4	94.4	88.0	81.6	87.9	83.5	75.8	90.4	79.7	24.36	0.30
GM-voting	85.8	94.4	88.2	82.7	89.0	84.5	77.8	93.9	82.1	23.12	0.29
GMDA-n	77.6	82.1	78.9	84.7	83.0	84.2	79.8	84.9	81.3	7.33	0.07

- **Decision Tree:** 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.
- **LDA with bagging:** We choose the number of bagging LDA estimators to be 5.
- **QDA with bagging:** We choose the number of bagging QDA estimators to be 11.
- **Random Forest:** We choose the criterion for splitting nodes and pruning the tree to be Gini index, and the number of estimators to be .
- **XGBoost:** 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:** 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.

5.2 Outlier Detection Methods

5.3 Gaussian Mixture Models Methods

5.4 Comparison between Different Methods

We have implemented several previous described algorithms together with algorithms proposed by ourselves on .

6 Conclusion

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

[3] 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.

[4] 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

$$\begin{aligned} g(D^*) &= \sum_{i=1}^{k} p_i \sum_{j=1}^{k} P(j|i; D^*) L(j|i) \\ &= \sum_{i=1}^{k} p_i \sum_{j=1}^{k} L(j|i) \int_{D_j^*} f_i(X) dX \\ &= \sum_{j=1}^{k} \int_{D_j^*} \sum_{i=1}^{k} p_i L(j|i) f_i(X) dX \\ &= \sum_{j=1}^{k} \int_{D_j^*} h_j(X) dX. \end{aligned} \tag{41}$$

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

$$g(D) = \sum_{j=1}^{k} \int_{D_j} h_j(X) dX. \tag{42}$$

Therefore,

$$\begin{aligned} g(D^*) - g(D) &= \sum_{j=1}^{k} \int_{D_j^*} h_j(X) dX - \sum_{t=1}^{k} \int_{D_t} h_t(X) dX \\ &= \sum_{j=1}^{k} \sum_{t=1}^{k} \int_{D_j^* \cap D_t} \left(h_j(X) - h_t(X) \right) dX \\ &\leqslant 0. \end{aligned} \tag{43}$$

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.