

# A Brief Survey on Outlier Classification

Zeyu Jia\*    Feng Zhu\*  
1600010603   1600010643

School of Mathematical Science, Peking University

January 13, 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  $\beta$ , 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

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

where  $X = (x_1, \dots, x_m)$  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:

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

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

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

where parameters  $\mu_y$  and  $\sigma_y$  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 Weight 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:

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

We can solve it by writing its dual problem:

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

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

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

and the prediction step can be written as

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

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

In our implementation, we use the RBF kernel

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

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  $\hat{\Sigma}$ :

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

Then we can use the following quantity

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

to determine the estimated class of instance  $x$ , according to

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

### 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

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

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

In QDA setting, we can use the following quantity

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

to determine the estimated class of instance  $x$ , according to

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

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  $\{(x_i, y_i)\}_{1 \leq i \leq n}$  ( $y_i \in \{0, 1\}$ ), 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

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

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  $G_1, \dots, G_{m-1}$  with weight  $\beta_1, \dots, \beta_{m-1}$ . 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  $\beta_m$  to minimize

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

and then we let

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

### 2.7.2 XGboost

This part refers to the paper [1]. 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:

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

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

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

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

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

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:

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

where

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

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

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

Then it is easy to calculate

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

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

$$\hat{y} = \max_y \# \{f_m(x) = y, 1 \leq i \leq m\}. \quad (26)$$

### 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 Detection Methods

## 4 Clustering-Based Modeling Methods

## 5 Numerical Experiment

## 6 Conclusion

## References

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