Lab2 Report

Muhuan Wu
Student ID: 1822306

Abstract

The essence of pattern recognition is to use the calculation method to divide into certain categories according to the characteristics of samples. This method can quickly extract the characteristic information of samples and make appropriate likelihood estimation to obtain an appropriate probability estimation model. This article will use Parzen Window Method and Modified Quadratic discriminant Function (MQDF) to develop pattern recognition algorithms, and compare the computational efficiency, complexity and effectiveness of these methods. The algorithm of MQDF is based on the article written by F.Kimura in 1987 [1]. The data set used in this experiment is iris plant data set from UCI [2].

1 Introduction

The Parzen Window Method and MQDF algorithms in this article are based on Bayesian theory. The two algorithms estimate the maximum likelihood function of each classification according to the existing training data set, and then put the value to be predicted into each likelihood function to estimate the probability of belonging to each classification, and the classification with the maximum probability is the result of the final prediction. This algorithm is different from the popular method of machine learning in recent years. Different from machine learning, machine learning algorithm first trains the weight of each hidden layer according to the training set, so as to put the data to be predicted into the weight and get the final result. The pattern recognition used in this article has no training process, only the process of establishing probability model for samples. In other words, for large-scale samples, the amount of computation of pattern recognition is slightly huge. However, for small-scale samples, pattern recognition should be an efficient choice.

2 Methodology

As mentioned above, the core prediction method of pattern recognition is to establish the probability model of the maximum likelihood function of the corresponding sample. The result depends on the establishment of probability model. This part will focus on explaining the principle and calculation process of Parzen Window Method and MQDF. This article will uniformly randomly disrupt the iris data set with 17 as the seed, and in order to ensure the stability of the results, the cross validation method will be used. In the cross validation part of this paper, 150 samples of iris will be divided into five cross validation. In other words, there are 120 samples in the training set and 30 samples in the test set for each verification.

2.1 Parzen Window Method

This method uses the principle of Bayes, and the formula is as follows

$$P(w_k|x) \propto P(w_k)P(x|w_k) \tag{1}$$

Where $P(w_k)$ is a prior probability, the formula is as follows:

$$P(w_k) = \frac{n_k}{\sum_{i=1}^{K} n_i} \tag{2}$$

In fact, a prior probability is the proportion of each classification in the total number of samples. In the process of calculation, because the total number of training samples in this experimental test is 120, it is divided into three categories in total. Due to the random disruption of 17 seeds before cross validation, that is, the proportion of each classification is about one third. One third is an approximate number with a certain error.

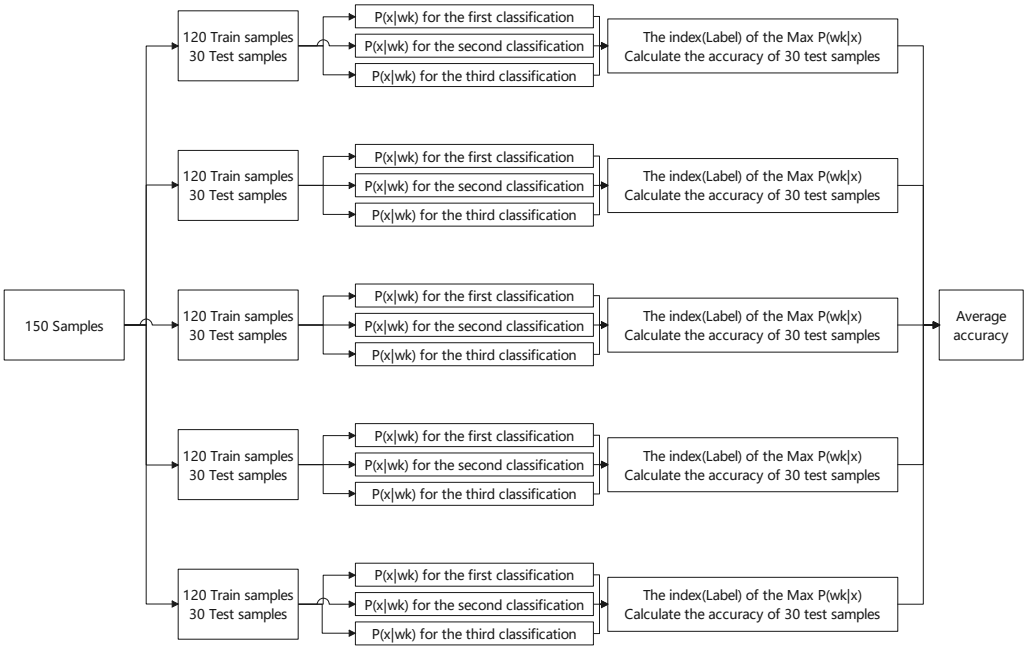
$P(x|w_k)$ is the probability density function, the formula is as follows:

$$P(x|w_k) = \frac{1}{n_k} \sum_{x_i \in w_k} \frac{1}{h^d} \phi(\frac{x - x_i}{h}) \tag{3}$$

where

$$\phi(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}}$$

Because each sample in iris data-set is composed of five numbers, four of which are sample data, and the last one is label type, that is, the dimensions of x and x_i are $(4, 1)$. h is a self-defined hyper-parameter and d denotes the dimension of the sample, which is 4. By changing the value of the hyper-parameter, probability models with different accuracy can be obtained. In other words, this section will calculate the relationship between h and the final accuracy. The specific operation is shown in the figure below:



Each value of h can obtain an accuracy value. Theoretically, for iris data set, h can form a probability distribution with accuracy.

2.2 Modified Quadratic Discriminant Function

Since the data set is the same as the first section, the data set processing method in this section is the same as that in the flowchart above. However, due to different models, the

likelihood function model is different when calculating the final probability. This paper will start from the QDF model, improve based on the QDF model, and analyze and discuss the results of each model.

2.2.1 QDF

The QDF equation is as follows:

$$g_0(x) = (x-\mu)^t \Sigma^{-1}(x-\mu) + log|\Sigma| - 2logP(w) \tag{4}$$

for a class w where μ and Σ denote the mean vector and the covariance matrix for x in the class w.

Because the Σ can be written as:

$$\Sigma_M = \sum_{i=1}^{n} \lambda_i \varphi_i \varphi_i^t$$

The $P(w)$ is omitted and the QDF equation can be as below:

$$g_0(x) = \sum_{i=1}^{n} \frac{1}{\lambda_i}(\varphi_i^t(x-\mu_M))^2 + \sum_{i=1}^{n} log\lambda_i \tag{5}$$

Because the number of samples of each kind in the training samples is different, the number of samples is uncertain. However, since the dimension of the sample is 4, it is assumed that the dimension of the storage sample matrix $train_sample$ is $(x, 4)$, the corresponding eigenvalue and eigenvector can be calculated as follows:

```
import numpy as np
train_cov = np.cov(train_sample, rowvar=False)
cov_eigenvalue, cov_eigenvector = np.linalg.eig(
train_cov)
```

Since there are no super parameters in the QDF model, the accuracy rate is only one number for the five cross validation, which can not form a probability distribution. Therefore, based on the QDF model and Fumitaka's theory [1], this paper will improve the QDF model.

2.2.2 MQDF 1

By employing a kind of pseudo-Bayesian estimate of the covariance matrix:

$$\Sigma_P = \Sigma_M + h^2 I$$

Where I is the n-dimensional identity matrix and h^2 is an appropriate constant, and applying such equation to the QDF model, the MQDF 1 model is shown as below:

$$g_1(x) = \sum_{i=1}^{n} \frac{1}{\lambda_i + h^2}(\varphi_i^t(x-\mu_M))^2 + \sum_{i=1}^{n}(\lambda_i + h^2) \tag{6}$$

In other words, MQDF 1 model also has a super parameter h. by adjusting the size of H, we can get a probability distribution of H about the accuracy. This paper will describe its characteristics in the analysis section.

2.2.3 MQDF 2

The training sample has four characteristic covariance, so the training sample has four characteristic covariance. Therefore, for the model, some eigenvalues and eigenvectors can be selected for training. In other words, MQDF 2 has one more adjustable super parameter. The corresponding equation is shown as below:

$$g_2(x) = \sum_{i=1}^{k} \frac{1}{\lambda_i}(\varphi_i^t(x-\mu_M))^2 + \sum_{i=k+1}^{n} \frac{1}{h^2}(\varphi_i^t(x-\mu_M))^2 + log(h^{2(n-k)} \prod_{i=1}^{k} \lambda_i) \tag{7}$$

By using the equation:

$$\sum_{i=1}^{n}(\varphi_i^t(x-\mu_M))^2 = \| x-\mu_M \|^2$$

The MQDF 2 equation can be written like below:

$$g_2(x) = \frac{1}{h^2}[\| x-\mu_M \|^2 - \sum_{i=1}^{k}(1-\frac{h^2}{\lambda_i})(\varphi_i^t(x-\mu_M))^2] + log(h^{2(n-k)} \prod_{i=1}^{k} \lambda_i) \tag{8}$$

Since the specific operation steps are similar to those above, it will not be repeated here.

3 Results And Analysis

3.1 Parzen Window Method

3.1.1 Results

This article designs h about the accuracy with the equation of Parzen Window. For details, please refer to the appendix of this article. After observing the approximate curve of accuracy and h value change, this paper draws the probability distribution diagram according to the H changes from 0.001 to 10 with step of 0.001 as the abscissa and the accuracy as the ordinate, as shown below:

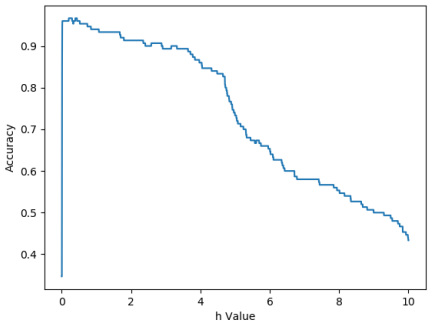


Figure 1: h value about accuracy with Parzen Window Method

3.1.2 Analysis

As shown in the above image, when the value of h is about 1, the accuracy of the whole model is the highest. Whether the value of H decreases or increases, the accuracy will decrease regularly. Because the cross validation function is used in this experiment, the result will fluctuate. This situation is normal, because the accuracy of each verification in cross verification fluctuates up and down, so the final average accuracy fluctuation is normal. If cross validation is not used, the accuracy will not fluctuate in theory, and the value of each change in accuracy should be $\frac{1}{30}$, because the number of samples for the test set is 30. Among the results trained by this model, when the value of h changes, the accuracy may reach a very low level, even less than 0.3. This result is very unsatisfactory and troublesome for the parameter adjustment of H. Therefore, there are the results of the subsequent QDF and MQDF models.

3.2 QDF

3.2.1 Results

In this paper, the qdf model is used to conduct a 5K-fold cross validation by adjusting the initial randomly disrupted seed to 17. Since there are no super parameters to adjust, the results are as follows:

```
print(result)
```

```
[0.9666666666666667, 0.9666666666666667,
0.9666666666666667, 0.9, 0.8666666666666667]
```

```
print(Average_score)
```

```
0.9333333333333332
```

3.2.2 Analysis

It can be seen from the above results that in the process of 5K-fold cross verification, most of the verification results are very high, with the highest accuracy of 96.7% and the lowest of 86.7%. This shows that the effect of QDF equation on the whole iris data set is obvious.

This pattern recognition model is very different from the deep-learning algorithm in recent years. As mentioned above, for a simple training set, pattern recognition can well analyze the sample characteristics of the training set and predict the unknown samples to be classified through such sample characteristics. In this process, there is only a clear calculation process, and there is no gradient descent process like machine learning. In other words, although there are few super parameters in the model, the QDF model can be identified efficiently due to different ways to obtain the sample characteristics

3.3 MQDF 1

3.3.1 Results

The result of MQDF 1 is shown as below:

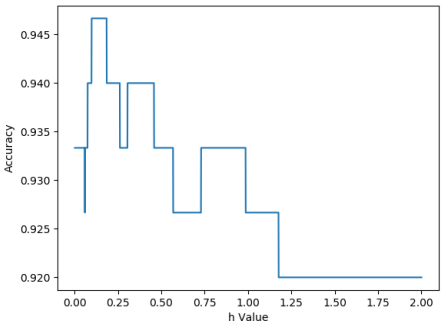


Figure 2: h value about accuracy with MQDF 1

By observing the change trend of accuracy relative to h, this paper draws the relationship between H and accuracy, where h changes from 0.002 to 2 with step of 0.001.

3.3.2 Analysis

It is not difficult to observe from the results above that the accuracy still fluctuates slightly with h changes. This result is caused by cross validation. Different from the previous Parzen

Window Method, the result of MQDF 1 maintains a high accuracy. Even if the value of h changes greatly, the final lowest accuracy is far higher than that of Parzen Window Method. This reason is attributed to the more perfect construction of the model. The MQDF 1 can also be considered as a function derived by substituting the covariance matrix in the QDF by:

$$\Sigma_{p'} = \int (x - \mu_M)(x - \mu_M)^t p_m(x) dx$$

which is the covariance matrix of the density $p_m(x)$ estimated by Parzen Window Method using circular normal density as the kernel. Therefore, with the support of this improved model, the performance of accuracy is much better than Parzen Window Method.

3.4 MQDF 2

3.4.1 Results

As mentioned earlier, MQDF 2 has two superparameters, the pre-defined h and the number of eigenvalues and eigenvectors considered. Since the dimension of the dataset is 4, the value range of the second super parameter is only four numbers. As shown in the figure below, the accuracy change surface of MQDF 2:

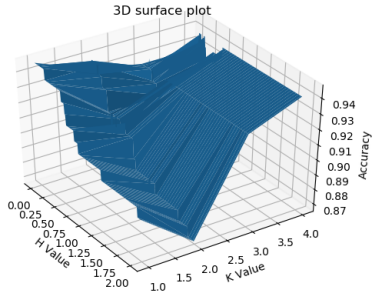


Figure 3: The accuracy change surface of MQDF 2

3.4.2 Analysis

As can be seen from the above figure, just like the law of Parzen Window Method and MQDF 1, the accuracy and h form a probability distribution. When the accuracy is the highest, the value of h is about 0.25 to 1. However, the distribution of the whole accuracy is relatively average, almost above 85%. In terms of the relationship between accuracy and h value, there is little difference between the two models of MQDF.

In terms of the relationship between the observation accuracy and the number of eigenvalues and eigenvectors, the accuracy calculated by using two eigenvalues and eigenvectors is the lowest.

3.5 Analysis about QDF, MQDF 1 and MQDF 2

The sensitivity of MQDF to the estimation error in the covariance matrix is lower than that of QDF using maximum likelihood estimation, and the performance is better. MQDF model can be used even when there is not enough samples or when the covariance matrix is singular, which is a great advantage of MQDF model over QDF model. MQDF2 requires much less computation time and storage than QDF, about $\frac{k}{N}$ times.

But the most important point is that MQDF can even compare with qdf in accuracy on the basis of reducing the number of eigenvalues and eigenvectors. From the above results,

MQDF2 only needs the first three eigenvalues and eigenvectors to achieve the same accuracy as QDF. In other words, the amount of computation of MQDF can be reduced by 25%. Based on the above observation and analysis, the two models of MQDF have the same effect as QDF, but their results are more accurate and can also reduce a large amount of computation of QDF. Therefore, the choice of MQDF is more reasonable.

4 Conclusion

In this report, two model methods are used to build the model of pattern recognition for iris data set. The first method is Parzen Window Method, which uses the traditional Bayesian theory. In the second part of this paper, a more advanced QDF model is used. However, there is room for improvement in the amount of computation, effect and complexity of this model. After learning MQDF from Kimura [1], this report finds that by reducing the number of covariance eigenvalues and eigenvectors considered, it can bring almost constant accuracy effect, and at the same time, it can greatly reduce the amount of computation.

References

[1] F. Kimura, K. Takashina, S. Tsuruoka, and Y. Miyake, "Modified quadratic discriminant functions and the application to chinese character recognition," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-9, no. 1, pp. 149–153, 1987.

[2] "uci machine learning repository: iris data set_2022," 2022. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Iris

Appendix

A Codes for Parzen Window Method

```
def P_x_wk(train_index, x, h, nk, classification):
    sum = 0
    for samples in train_index:
        if samples[4] == classification:
            u = (x - samples[0:4])/h
            parameter = 1/np.sqrt(2*math.pi)
            fei = parameter * math.exp(-(np.square(u[0])+np.square(u[1])+np.square(u[2])+np.square(u[3]))/2)
            sum += fei/(math.pow(h, 4))
    pxwk = sum/nk
    return pxwk

def parzen_window_method(data, h, n_split):
    KF = KFold(n_splits=n_split)
    parzen = []
    for train_index, test_index in KF.split(data):
        train_index, test_index = data.iloc[train_index], data.iloc[test_index]
        train_index, test_index = np.array(train_index), np.array(test_index)

    # fetch the labels of test set
    labels = []
    for samples in test_index:
        labels.append(samples[4])

    # calculate for P_wk of train set
    P_wk = np.array([0, 0, 0])
    for sample in train_index:
        if sample[4] == 0:
            P_wk[0] += 1
```

```
      elif sample[4] == 1:
        P_wk[1] += 1
      elif sample[4] == 2:
        P_wk[2] += 1
      else:
        print('There is something wrong with the labels!')
    num = P_wk[0] + P_wk[1] + P_wk[2]
    P_wk_pro = P_wk/num

    # calculate for scores
    predict = []
    for samples in test_index:
      scores = []
      for i in range(3):
        P_wk_x = P_wk_pro[i] * P_x_wk(train_index=
    train_index, x=samples[0:4], h=h, nk=P_wk[i],
        classification=i)
        scores.append(P_wk_x)
      predict.append(scores.index(max(scores)))

    # predict compares with labels
    count = 0
    for i in range(len(labels)):
      if predict[i] == labels[i]:
        count += 1
    result = count/len(labels)

    # propabilities of each epoch
    parzen.append(result)

  return sum(parzen)/len(parzen)
```

B Codes for QDF

```
def QDF_calculater(test, train):
  x_miu = test-train.mean(axis=0)
  train_cov = np.cov(train, rowvar=False)
  cov_value, cov_vector = np.linalg.eig(train_cov)
  g_0 = 0
  for i in range(4):
    g_0 += np.square((cov_vector[i]*x_miu).sum())/
  cov_value[i]
    g_0 += np.log(cov_value[i])
  return g_0
```

C Codes for MQDF 1

```
def MQDF_calculater(test, train, h):
  x_miu = test-train.mean(axis=0)
  train_cov = np.cov(train, rowvar=False)
  cov_value, cov_vector = np.linalg.eig(train_cov)
  g_1 = 0
  for i in range(4):
    g_1 += (np.square((cov_vector[i]*x_miu).sum()))/(
  cov_value[i]+np.square(h))
    g_1 += np.log(cov_value[i]+np.square(h))
  return g_1
```

D Codes for MQDF 2

```
def MQDF_calculater(test, train, h, k):
    x_miu = test-train.mean(axis=0)
    train_cov = np.cov(train, rowvar=False)
    cov_value, cov_vector = np.linalg.eig(train_cov)
    g_2 = 0
    for i in range(k):
        g_2 -= (np.square((cov_vector[i]*x_miu).sum()))*((
    cov_value[i]-np.square(h))/(cov_value[i]*np.square(h)))
        g_2 += np.log(cov_value[i]*math.pow(h, 2*(4-k)))
    g_2 += np.square((x_miu*x_miu).sum())/np.square(h)
    return g_2
```