Experiment 1

Perceptron Learning toward Linear Classification

1. Introduction

Linear Classification is one of the most common classification problem and the most simple and theoretically complete problem. The problem is described as following: "given a set of data points in d-dimensions which belong to two classes, ω_1 and ω_2, try to find a linear separating hyper-plane between the samples of the two classes to effectively divide the data into two subsets ".

If there exists a hyper-plane that can separate all the points labeled ω_1 on one side of the plane while all the points labeled ω_2 on the other side, then the data is linearly separable. If there is not such a hyper-plane, the data is linearly non-separable.

In order to evaluate the classifying quality, we can define a cost function to calculate the degree of misclassification. Perceptron Learning is a kind of algorithm that minimize the cost function based on the dataset and a series of parameter updating rules.

In the following part, we will discuss the detailed algorithm of Perceptron Learning, then build and test the model we established by a series of well-designed data to get better understanding about the following questions:

(1) The working of linear perceptron learning algorithm.

(2) The effect of various parameters on the learning rate and convergence of the algorithm.

(3) The effect of data distribution on learnability of the algorithm.

2. Principle and Theory

2.1 Linear Discriminant Functions

Assume the points in the data set are n dimensional vectors $\mathbf{x} = (x_1, x_2, \dots, x_n) \in R^n$, define function $G(\mathbf{x}) = \mathbf{w}^t \boldsymbol{x} + w_0$, where $\mathbf{w} = (w_1, w_2, \dots, w_n) \in R^n$. The equation $\mathrm{G}(\mathbf{x}) = 0$ is actually a hyper-plane in n dimension space. The space will be divided into two parts so do all the sample points $\mathbf{x}$.

In order to get more uniform in expression, we extend $\mathbf{x}$ and $\mathbf{w}$ to be a n+1 dimensional vector by defining $\mathbf{x} = (\mathrm{x}_0, \mathrm{x}_1, \mathrm{x}_2, \dots, \mathrm{x}_\mathrm{n}) = (1, \mathbf{x})$ while $\mathbf{w} = (w_0, \mathbf{w})$ so the define function can be written as $\mathrm{G}(\mathbf{x}) = \mathbf{w}^\mathbf{t}\boldsymbol{x}$**.**

Assume we have m sample points $\mathbf{x}^{(\mathbf{i})} \in \mathbf{R}^{\mathbf{n+1}}, \mathrm{i} = 1,2,3, \dots, \mathrm{m}$(all the samples are extended by adding an element “1”). Mathematically, we define all the $\mathbf{x}^{(\mathbf{i})}$ that satisfy $\mathrm{G}\left(\mathbf{x}^{(\mathbf{i})}\right) > 0$ be the positive class while $\mathrm{G}\left(\mathbf{x}^{(\mathbf{i})}\right) < 0$ to be the negative one. Our goal is to find a proper weight vector $\mathbf{w} \in \mathbf{R}^{\mathbf{n+1}}$ to classify as much points labeled ω_1 into positive class as possible while as much points labeled ω_2 into negative class.

2.2 Cost function

Intuitively, it is reasonable to consider both the amount of correctly classified samples and the quality they are classified, which means how far they are from the boundary.

In order to evaluate the quality of classification function we define cost function where set X is the collection of misclassified samples.

$$\mathrm{J}(\mathbf{w}) = \sum_{x \in X} -\boldsymbol{w}^{\boldsymbol{t}}\, \boldsymbol{x}$$

If $\mathrm{X} = \emptyset$ then we consider $\mathrm{J}(\mathbf{w}) = 0.$ The reason why it is defined in this way can be explained by a 2-D example.

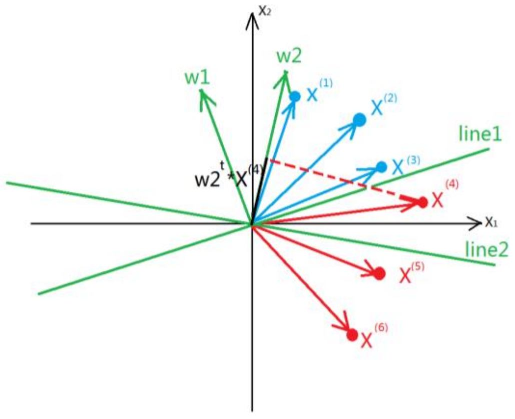


Figure 1 intuition of cost function

On the plane there are two sets of sample points with red one labeled ω_1 while blue ones labeled ω_1. The line 1 is much better as a separation line than line 2, obviously. The weight vector is actually the normal vector of the separation line. So if there are misclassified samples $\boldsymbol{x}$, the inner product $\boldsymbol{w^t x}$ is actually the projected length of $\boldsymbol{x}$ on $\mathbf{w}$ multiplied by the norm of $\mathbf{w}$. (The projected length is marked as a black line segment in figure 1). The value can be used to assess how badly the sample has been separated by the hyper-plane because the further away the misclassified point is from the right side, the larger this product will be. Therefore, if we minimize the value, we are actually minimizing the classification error. That is probably the reason why it is called "cost function". By declining the cost by a series of iterations, we can finally get to the optimum.

2.3 Updating rules

Starting from a random value (or some predefined uniform initial value), the weight vector can finally converge to the value we need according to the dataset by a series of updating. According to the principle of gradient descent, if the iterations t is seen as a variable of the changing weighting vector **w**, then the **w** in the next iteration should be

$$\mathrm{w(t+1)} = \mathrm{w(t)} - \eta\nabla \mathrm{J}(\mathbf{w}) = \mathrm{w(t)} + \eta\sum_{x\in X}\boldsymbol{x}$$

which means the weight **w** is added to the sum of a series of misclassified sample vectors after scaled by η. To be more accurate, there are two categories of algorithm used to handle the misclassified samples. Firstly, we can update the value of **w** every time we find a misclassified sample until there aren't any misclassified samples or the level of misclassifying are within a threshold. Secondly, we can update the value of **w** after all the misclassified samples defined by this **w** are found. And the update change of **w** will be the sum of all the misclassified samples **x** weighted by factor η. The former one is called single-sample perceptron and the latter one is batch perceptron. We will implement both of them in the following part and get better understanding of their difference.

3. Model test

In the former part of the paper, we displayed the detailed algorithm of learning perceptron. However, the difference of single-sample and batch perceptron as well as the influence of learning rate η are not clear. In addition, the effect of the amount of separation as well as separability are unrevealed. In this part, we will first generate a series of 2-class labeled data and train the model by the data to get better understanding of those questions.

3.1 Generate data

In order to get intuitive understanding of the data, we are going to test the model by a set of 3-D data at first. Because almost all the data collected for all kinds of purposes are distributed in a Gaussian way, we generate two kinds of data that belongs to two classes by randomly pick values from a 3-D Gaussian distribution whose parameters are predefined and can be modified. The following figure showed one of the dataset generated by the previously mentioned method with both 100 positive labeled data and negative ones. The separability of generated data can vary by modifying the Gaussian distribution parameter $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. In a more intuitive view, the distance between those two point clouds and the shape and span of them can be modified intentionally to test the model.

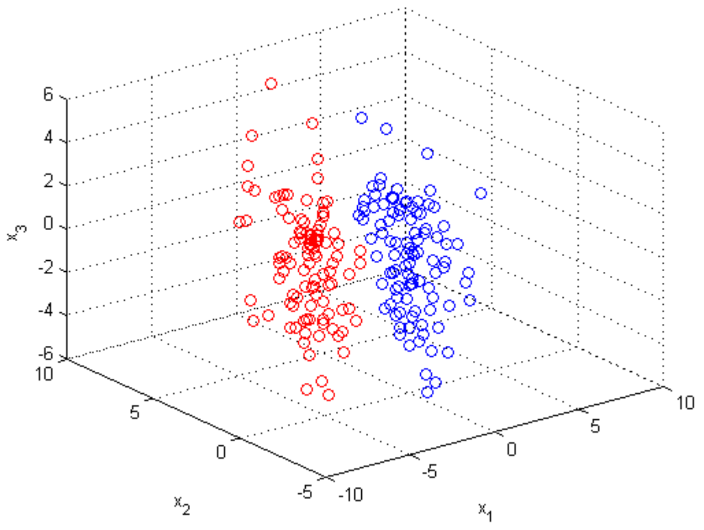


Figure 2 3D data example

3.2 The influence of learning rate

Firstly, we tested the declining of cost function by varying leaning rate η using single sample perceptron. In this test, we use a data set with both 500 positive labeled data and negative ones generated by the method described above.

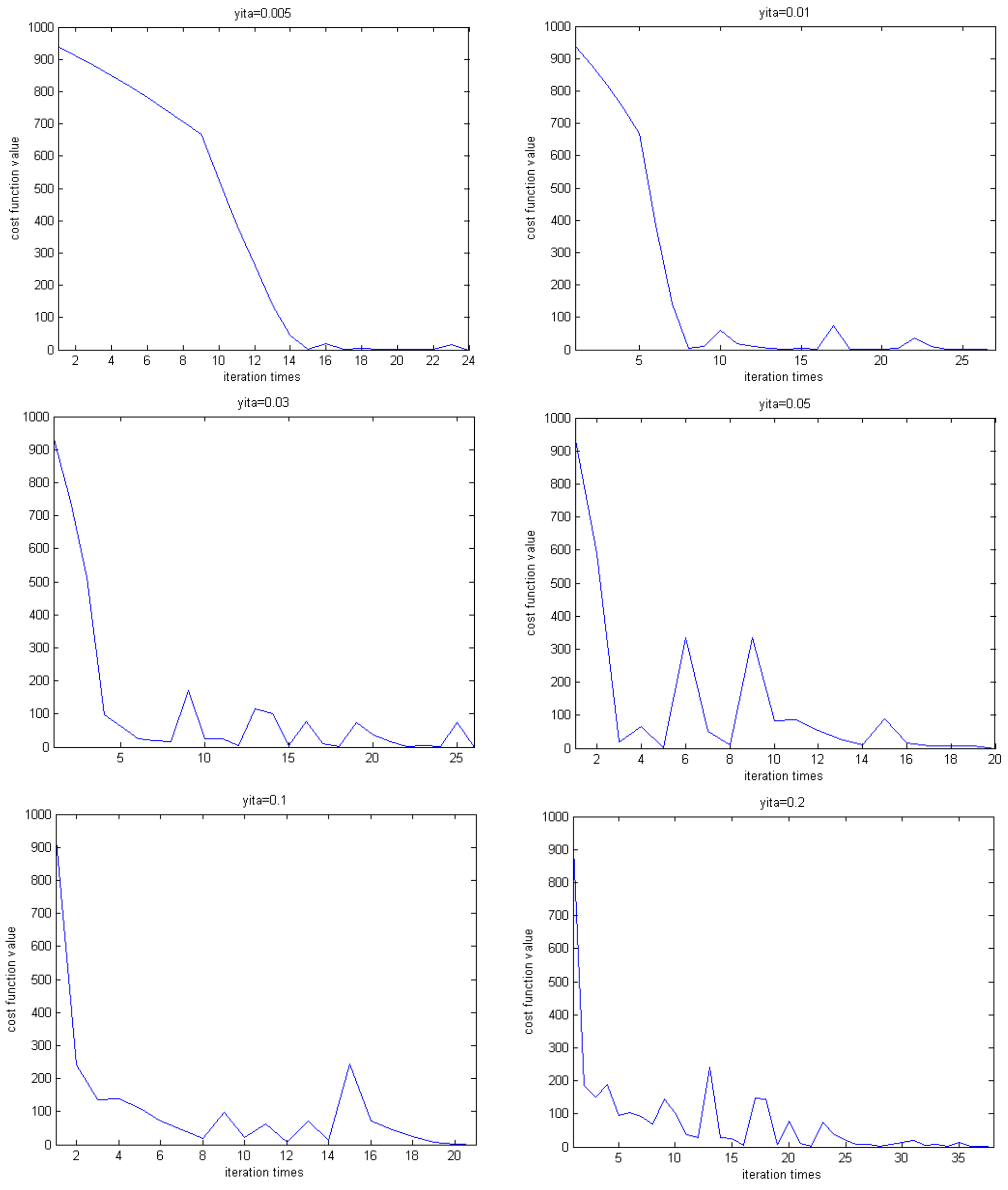


Figure 3 The influence of learning rate

There are figures with η varied from 0.005 to 0.2. We find following points:

- With η increases, the cost function decline more and more rapidly. When yita=0.01, the learning procedure spent approximately 15 iterations while the cost decrease more than 700 in the first iteration can became stable after only 5 iterations when yita=0.2.
- When the cost is already in a relatively low level, it goes up and down repeatedly with iteration time increases, which means the separating plane is adjusting its positon to better fit all the data. However, the cost is not always decreasing because the sample sequence is random and the first misclassified sample it met is not usually the best direction where the plane should go to fit all the data. However, the curve is decreasing to zero in general.
- There isn't a clear correlation between η and final iterations times. When η increases, the iterations times costed to get the final plane doesn't drop significantly. We suppose the reason is that when η is higher there are multiple overshoots when the plane is near the final position. Therefore, even though the cost drop rapidly at first few iterations, it spend more time adjusting back and forth near the optima and doesn't perform better in overall speed.

The classifying result is presented in the following figure (from different angles).

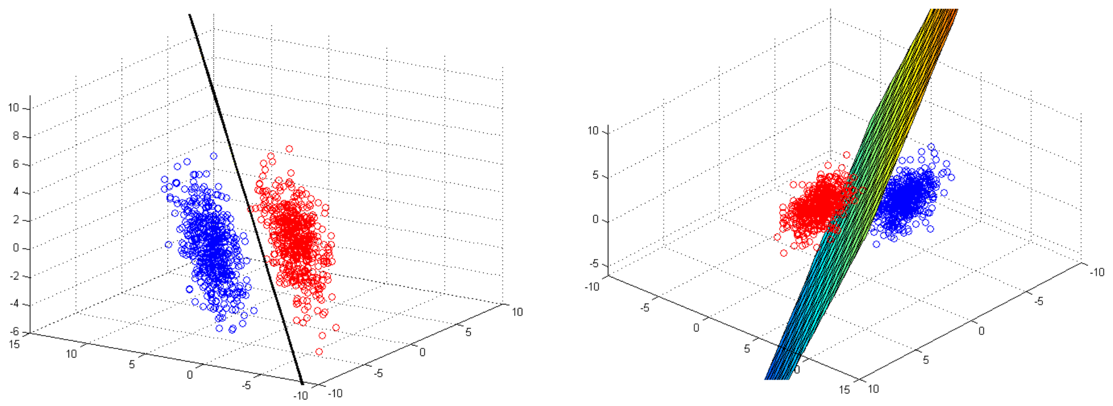
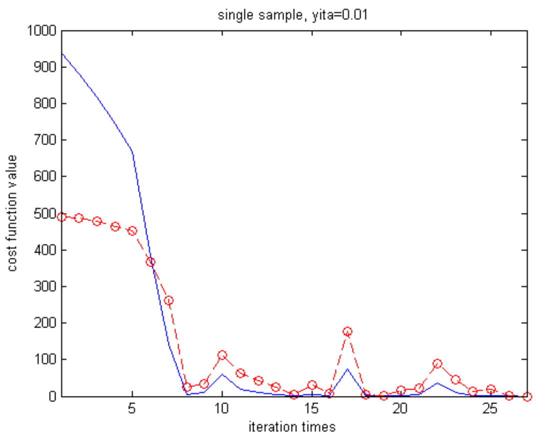


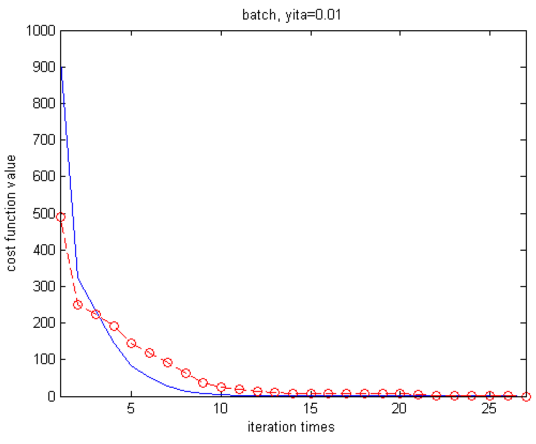
Figure 4 3D separation result

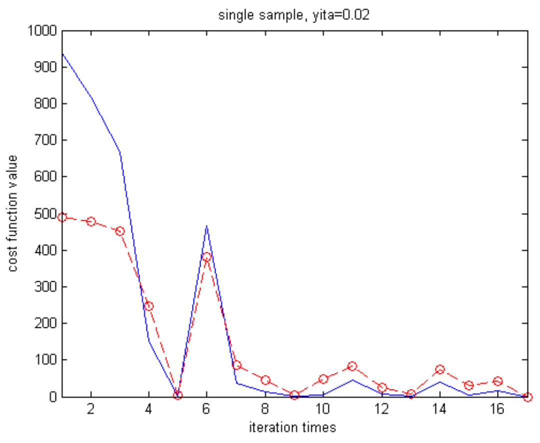
3.3 The influence of updating mode

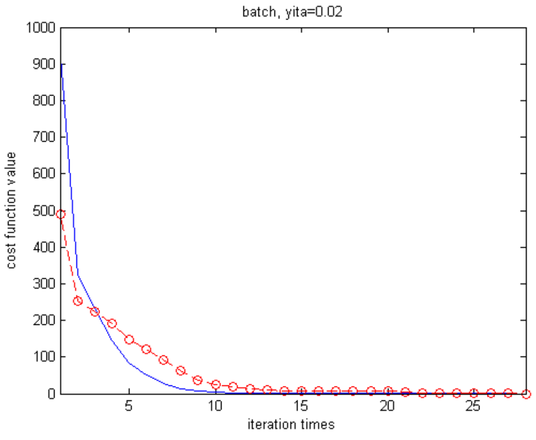
In the previous part, we introduced two different algorithms, the single-sample perceptron and the batch perceptron. In this part, we tested both two models with several different learning rate.

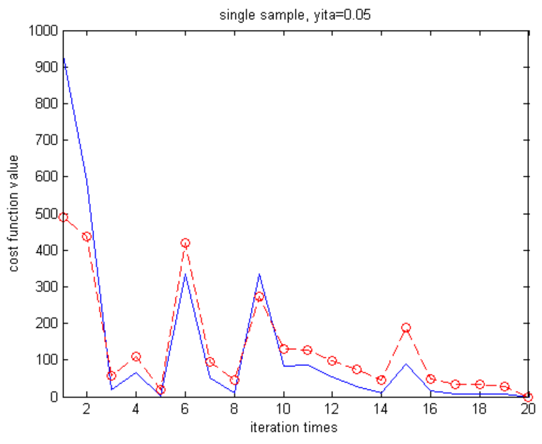
In the following figures, the red circle represents the number of misclassified samples and the blue lines are cost function value. We tested η from 0.01 to 0.5 with both of the two model. There are points could be conclude from the figures.

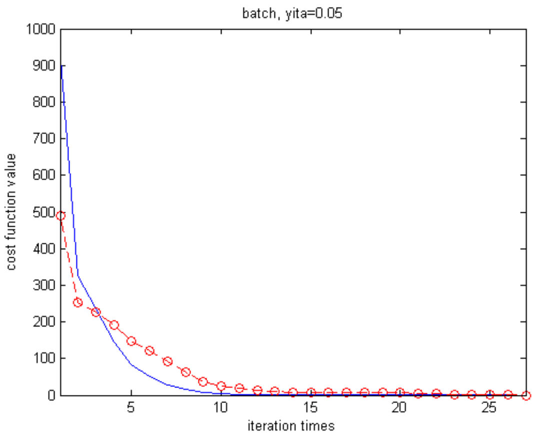












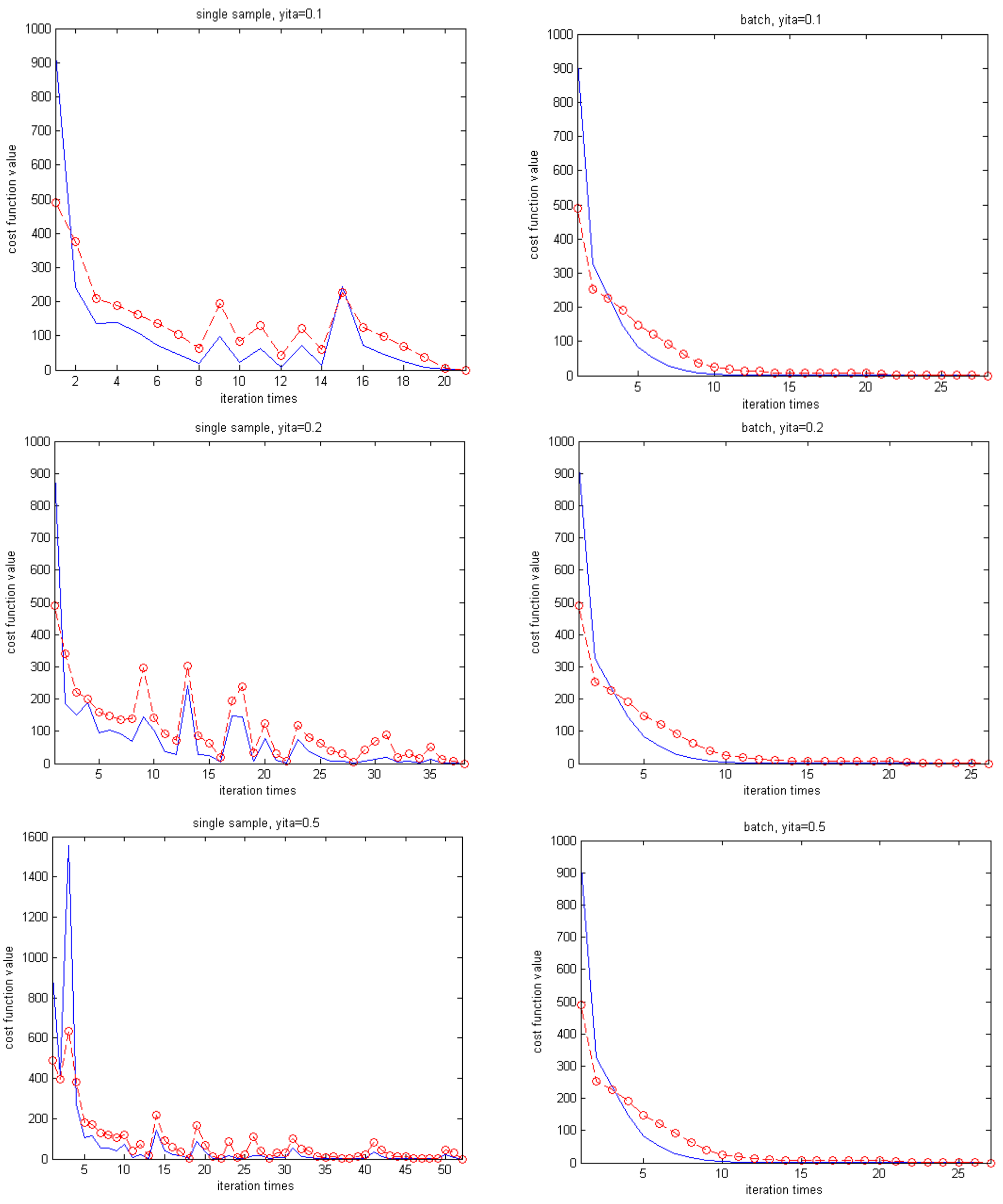


Figure 5 influence of update mode and learning rate

- The cost function of batch perceptron drop more smoothly and is monotone decreasing as iteration time increasing. The reason is that the perceptron update its weight function by adding all the misclassified samples to the original one. Therefore, the randomness of directions is decreased significantly by adding up all the random offset. So basically the weighting vector is moving to the optima in every iteration without random overshoot.

- It seems that the change of η have little effect on batch perceptron. The reason is we normalize cost function by dividing the norm of **w** at every iteration. Therefore, the effect of η on changing rate have been cancelled.

3.4 The influence of separability

In this part will generate dataset with different separability by modifying the mean value and standard deviation matrix and test the performance of our perceptron.

We assume the dataset for both label generated from multivariate normal distribution has a same covariance matrix Σ. We define the separability to be

$$\text{separability} = \frac{\|\boldsymbol{\mu_1} - \boldsymbol{\mu_2}\|}{|\Sigma|}$$

We could imagine that if those two point cloud have **μ** far away from each other while |Σ| is small which means the cloud is compact and almost all the points are within a tiny span, the separability should be large which means those two classes are easily separated into two groups. That is the reason we define it.

Again, we plot the cost curve with iteration times and scale it to check the first 10 iterations. In the following figure the batch perceptron with $\eta = 0.1$ are implemented to see the influence of separability.

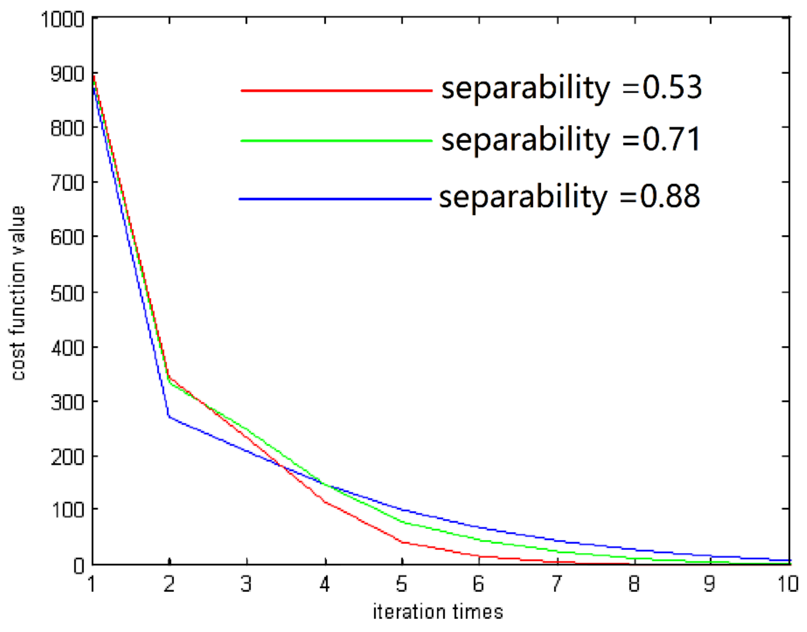


Figure 6 the influence of separability

The blue, green, red lines represent different level of seprability. We could find when the separability increases, the cost function drops more quicker at first while converge slower after four to five iterations. It is because the data distribution is diffused into a larger range when the separability is larger, so the separation plane should go further to divide them and thus spend longer time in the following iterations after drop significantly at the beginning.

3.5 The influence of data size and dimensions

In all the experiments above, we tested the model with a dataset with 500 positive label and 500 negative label all in 3 dimensions. In the following part, we are going to change the number of dimensions and data samples to test the model.

Firstly, we test the influence of dimensions with datasets with fixed 1000 sample numbers. Because the cost function value changed with iteration times can show how quick the hyper-plane has converged to the optima, we still use cost function value to compare learning procedure based on dataset with different dimensions. Again, we implement the batch perceptron only and the learning rated is set to be 0.1.

The following figure showed the change of cost function value with iteration times with different data dimensions varied from 10 to 100.

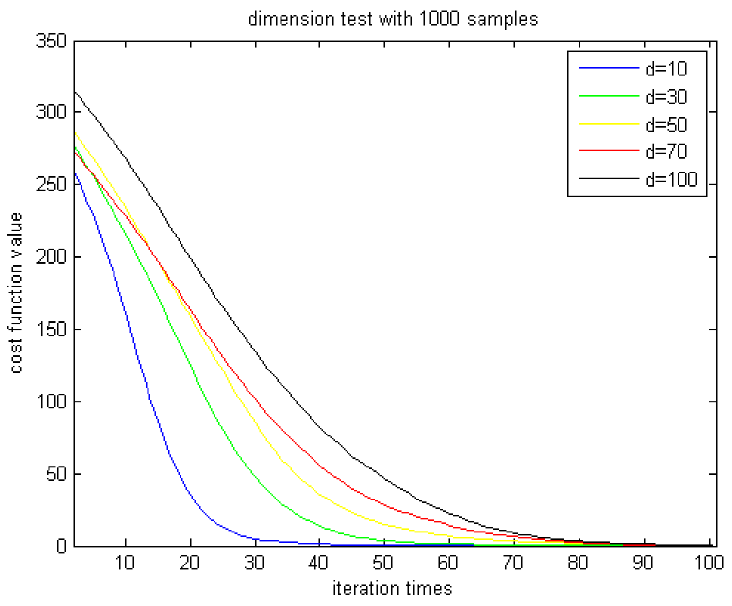


Figure 7 influence of dimension

Obviously, the increase in data dimension can slow down the convergence to some extent. When the dataset is 10-dimension the cost function will go below 5 after 30 iterations while it spent 76 iterations for 100-dimension to go below 5. Also there are subtle differences after 100 iterations, the remained cost function are 0.3136, 0.2173, 0.2427, 0 and 0 for dimension from 10 to 100 respectively.

Then we tested the model with a fixed 50 dimension data varied in sample numbers from 200 to 10000. All the data are composed with half of positive label data as well as half of negative ones.

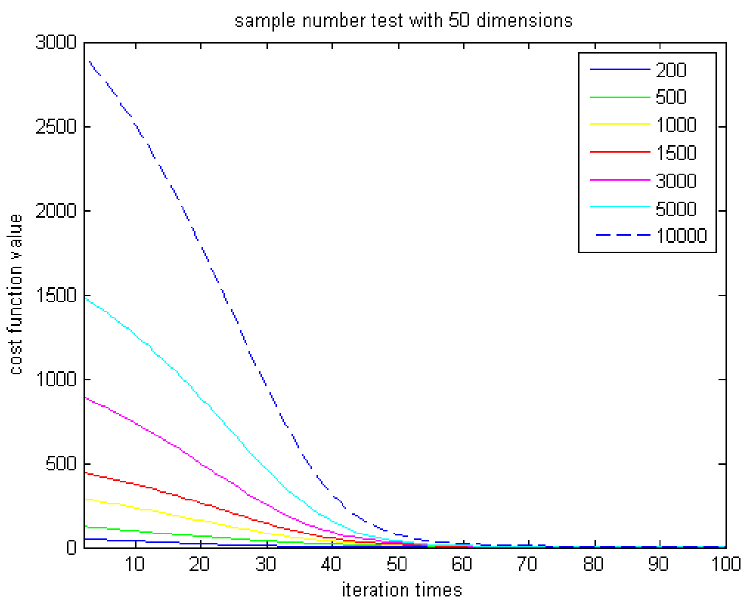


Figure 8 influence of data size

In the figure, we could find that the influence of sample numbers are mainly reflected in the initial value of cost function and the cost function values are approximately in proportion to the data size. For example, when there are 5000 samples, the initial value begin from 1500 while it begin from around 3000 when there are 10000 samples.

Finally, I find that the proportion of positive label samples also influence the performance of perceptron. Therefore, another test is performed with different proportion of positive and negative labeled samples. The data size and dimensions are fixed to 5000 samples 50 dimensions.

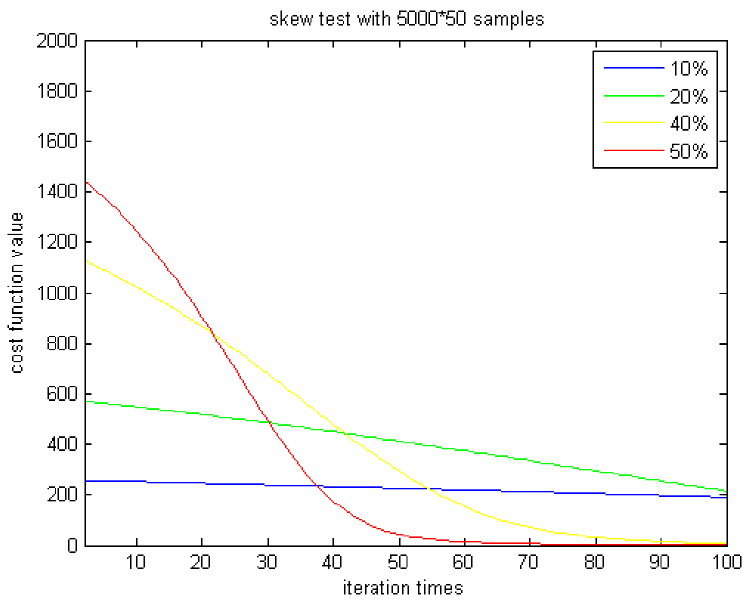


Figure 9 influence of positive-negative proportion

According to the figure, when the proportion of positive label samples drop down to 10%, the training procedure performed extremely slowly and only decrease 56.7 in 100 iterations. However, when the proportion increases, the speed of convergence is increased greatly. The result gives us a clear information that the skewed data are difficult to handle in training a model. So further research and advanced method such as F score should be used to assess the quality of classification to get quicker convergence to optima.

4. Conclusion

In this experiment, we built two linear perceptron with two slightly different algorithms and tested their performance by observing the cost function curve to evaluate the speed and quality of convergence.

Moreover, by testing the perceptron with well-designed data, we found that the learning rate (η), update mode (batch or single-sample), separability, data size and dimension as well as positive label data rate all have different influence on the performance of the perceptron. The test helped me understand more about how the Perceptron Learning works and how to implement it effectively.

Experiment 2

Synthetical Design of Bayesian Classifier

1. Introduction

Different from linear perceptron, Bayesian Classifier consider the feature vector as a random vector belongs to specific distributions. We learn the distribution pattern and parameter from the data. The more data we get the more certain we are about the distribution.

Given the probability distribution function, the probability of observing a particular feature vector **x** under condition ω_i is known as likelihood function which is

$$\mathrm{P}(\mathbf{x}|\omega_i) \sim \mathrm{Distribution}(\mathrm{parameters})$$

However, we need to know the probability of occurring ω_i under the condition of observed feature $\mathrm{P}(\omega_i|\mathbf{x})$ (posterior probability) in order to take decisions based on the knowledge we have learned. According to Bayesian rules, the posterior probability can be derived from prior probability $\mathrm{P}(\omega_i)$ and likelihood function

$$\mathrm{P}(\omega_i|\mathbf{x}) = \frac{\mathrm{P}(\mathbf{x}|\omega_i)\,\mathrm{P}(\omega_i)}{\sum_{i=1}^{c} \mathrm{P}(\mathbf{x}|\omega_i)\,\mathrm{P}(\omega_i)}$$

Where c is the number of class. Based on the posterior probability we can easily make decision like take action α_i correspond to condition ω_i when $\mathrm{i} = arg[\max_i \mathrm{P}(\omega_i|\mathbf{x})]$. We can also introduce more complex decision-making strategies considering the difference of loss caused by different mistakes.

According to the brief introduction above, we could build a Bayesian Classifier and test it from different perspective and depth.

2. Principle and Theory

2.1 Likelihood function

The most important information we learn from data is the distribution of feature vector (or known as likelihood function). However, the first problem is which kind of distribution we are going to assume that the sample comes from and how to get the corresponding parameters. Normally, we first study the distribution of data with statistic methods and assume the distribution according to our experience. For example, in practice most random samples come from a normal distribution. Basically, we assume it to be normal distribution because we have complete knowledge and theory about this distribution no matter how many dimensions the feature is. If it is not normal distribution, we usually transfer it to distribution similar to normal by linear transformation to make the study easier.

The p.d.f. of normal distribution with d dimensions is as follows

$$\mathrm{p}(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\boldsymbol{t}}\boldsymbol{\Sigma}^{-\mathbf{1}}(\boldsymbol{x}-\boldsymbol{\mu})\right]$$

2.2 Decision-making strategy

In order to take the different of loss or risk into consideration, we define risk function as follows

Let the series of decision actions as$\{\mathrm{a}_1, \mathrm{a}_2, \mathrm{a}_3, \dots, \mathrm{a}_\mathrm{c}\}$, if the coefficient of risk or loss when you take decision a_i under the condition of ω_j is $\lambda(\mathrm{i},\mathrm{j})$, then the conditional risk of decision action a_i can be computed by

$$\mathrm{R}(\mathrm{a}_\mathrm{i}|\boldsymbol{x}) = \sum_{j=1}^{c} \lambda(\mathrm{i},\mathrm{j})\mathrm{P}(\omega_\mathrm{j}|\mathbf{x}), \mathrm{i} = 1,2,\dots,\mathrm{c}$$

The principle of decision-making is to minimize the risk so the action finally should be taken according to the algorithm is

$$\mathrm{a}^*(\boldsymbol{x}) = \arg(\min_i \mathrm{R}(\mathrm{a}_\mathrm{i}|\boldsymbol{x}))$$

3. Model establish and test

3.1 Gaussian condition two class Bayesian classifier

Given the data in the guidebook, we firstly plot the histogram to get an intuitive feeling of how those samples looks like. We can find the value of those two classes are from two separate range without mixed.

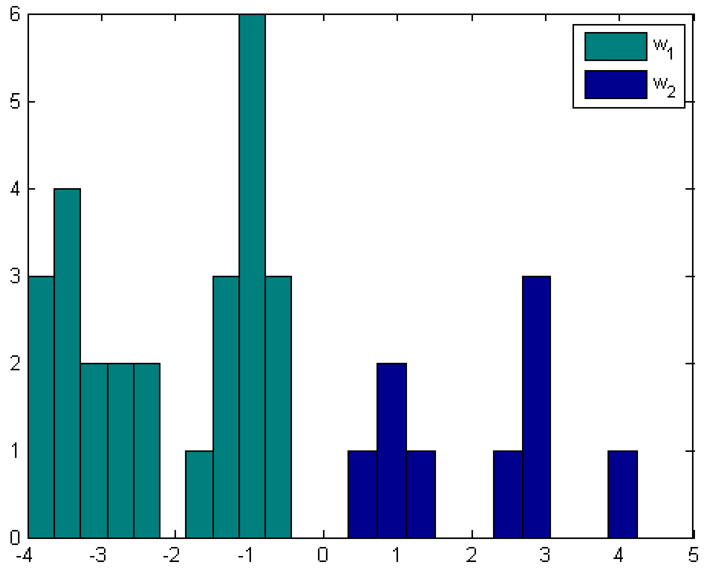


Figure 10 data histogram

The expectation and variance are

$$\mu_1 = -2.1226 \ \mu_2 = 2.1019$$

$$\sigma_1^2 = 1.4758 \ \ \sigma_1^2 = 1.3727$$

Therefore, the likelihood function is

$$\mathrm{P}(\mathrm{x}|\omega_\mathrm{i}) = \frac{1}{(2\pi)^{1/2}\sigma_i}\exp\left[-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right], i = 1,2$$

Then the posterior probability is

$$\mathrm{P}(\omega_\mathrm{i}|\mathbf{x}) = \frac{\mathrm{P}(\mathbf{x}|\omega_\mathrm{i})\,\mathrm{P}(\omega_\mathrm{i})}{\sum_{i=1}^{2}\mathrm{P}(\mathbf{x}|\omega_\mathrm{i})\,\mathrm{P}(\omega_\mathrm{i})}$$

The loss parameters λ for different decision action are given in table 1

Table 1 the loss parameters for different decision

real class		ω_1	ω_2	
decision action	a_1	0	1	loss parameters
	a_2	6	0	

For every single new coming feature x, the risk of taking action a_i when feature vector x is observed is as follows

$$R(a_i|x) = \sum_{j=1}^{2} \lambda(i,j) P(\omega_j|x)$$

We can get the value of risk function $R(a_i|x), i = 1,2$ of all samples from dataset as figure 11.

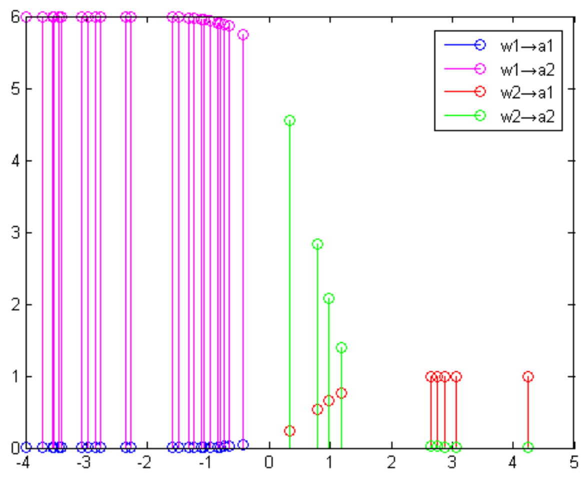


Figure 11 risk function of sample data

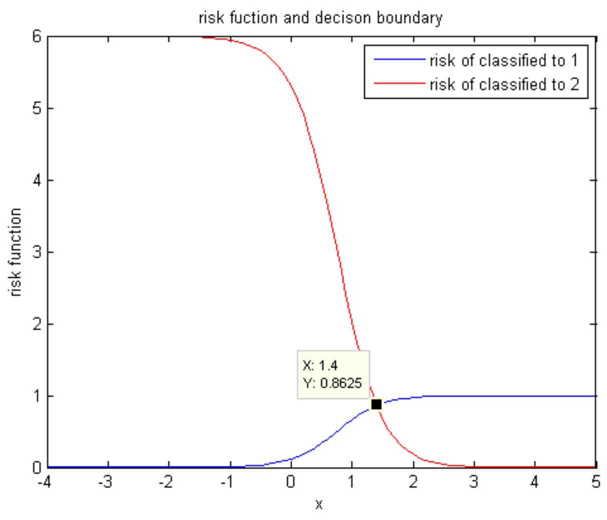


Figure 12 risk function curve

The intensive magenta lines on the left side of figure 11 show that the samples originally belonged to class ω_1 have much higher risk when classified to ω_2 than ω_1(magenta circles are much higher than blue ones). On the right side, those five sample greater than 2 should classified to ω_2 as red circles are higher than green ones. However, because the high risk coefficient classified to ω_2, the left 4 samples tend to be classified to ω_1 rather than where the originally belonged to. And the closer they are to the left, the stronger tendency they have to be classified to ω_1. The risk of classified to two classed are drawn in figure 12. To minimize the risk, we make decision according to which line is lower. So we will decide the sample to be class 1 when the observed x is lower than 1.403 while to be class 2 otherwise. So the decision boundary can be determined as 1.403.

Obviously, the risk curve and decision boundary will change when the loss parameter λ change. In the follow figure, I tested the classifier with different $\lambda(1,2)$ with the sum $\lambda(1,2) + \lambda(2,1)$ fixed.

$$\lambda(1,2) + \lambda(2,1) = 1$$

$$\lambda(1,1) = \lambda(2,2) = 0$$

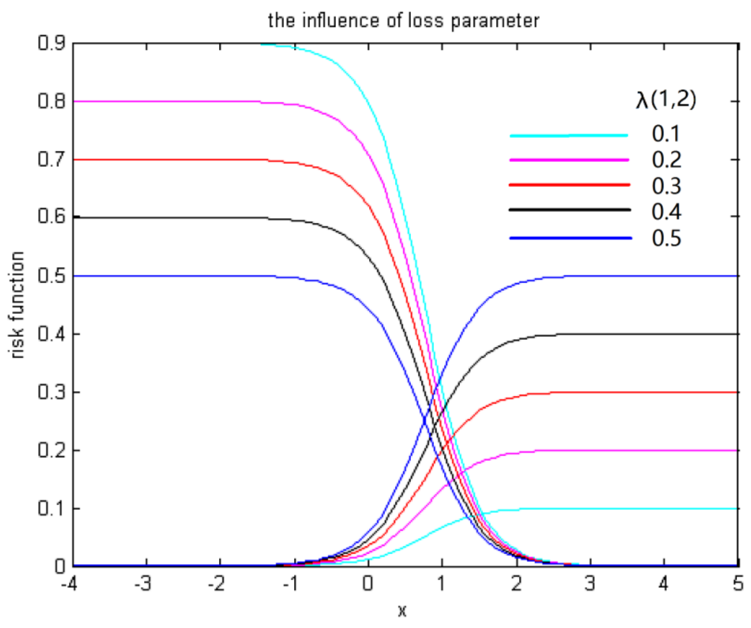


Figure 13 influence of loss parameter

According to the figure, when the loss parameter $\lambda(1,2)$ increase (which means we suffer more risk to classify sample from class 2 wrongly into class 1), the decision boundary will move towards left (we tend to believe the samples in the intermediate range belong to class 2). That is a reasonable respond to the rise of cost to classify a sample into class 1.

3.2 Multi-classes and high dimension Bayesian classifier

In this part, we are going to generate a high dimension multi-class sample from a Gaussian distribution and train the model to classify new examples from the same distribution. Similar to experiment 1, we are interested in the effect of data size and dimensions as well as separability. Test on those three parameters will help us find some hint on how the Bayesian classifier works.

3.2.1 Data intuition

Firstly, to get better intuition of how the data looks like, I plot a 3-D scatter diagram with 5 classes and 100 sample points in each class. To modify separability of the data, I changed the distribution parameters and got two diagram with different data separability, which could be directly perceived through figure 14.

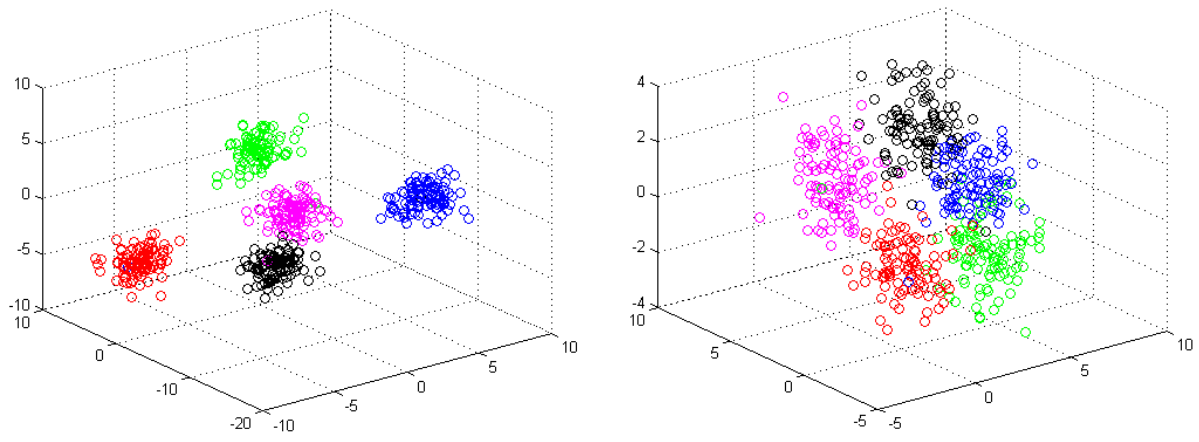


Figure 14 multi-class 3-d data scatter diagram

3.2.2 Influence of separability on classification result

Then I trained the Bayesian Classifier with the data and tested it with the exact same data. In order to simplify the parameters, we assume all the loss from different classification error are uniform which means there is no particular tendency to avoid several specific kinds of misclassification.

To measure the separability, we define degree of separability for class i as

$$\mathrm{d_s}(i) = \frac{\min\limits_{j, j \neq i} \left\| \mu_i - \mu_j \right\|}{\|\Sigma\|}$$

The following figure showed the relation between classification accuracy (number of samples correctly classified divided by number of samples belonged to this class) towards degree of separability. I chose a data set with 20 classes and got 20 points in figure 15.

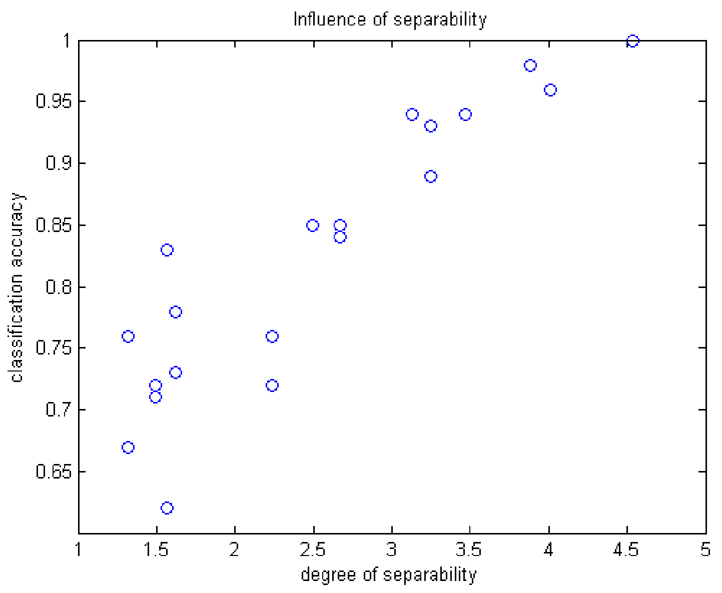


Figure 15 influence of separability

Though they are not strictly in linear correlation, we could still see the positive correlation property between degree of separability and the classification accuracy. It make sense because the larger degree of separability is, the more easily they can be separated and the higher the classification accuracy should be.

3.2.3 Influence of data dimension on classification accuracy

Then we changed the data dimension and tested the classification accuracy with 50 class for each fixed data dimension. The figure is as follows.

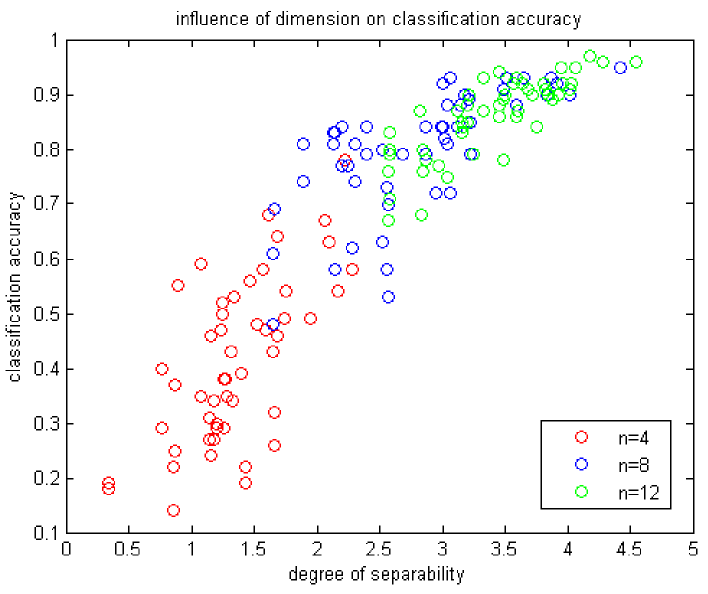


Figure 16 influence of dimension

Three data dimension 4,8,12 is tested in the experiment, it can be seen that the relation of d_s and classification accuracy still hold the positive correlation property. Though the range of d_s are different because of the change of dimension (because generally the larger the number of dimension is, the larger the norm of mean value is), they are still in the approximately same relation curve. So actually, dimension change did not influence the classifying ability significantly.

3.2.4 Influence of data size

In this test, we changed the data size with dimension fixed to three. Again, we plot the relation between d_s and classification accuracy to display the change of classifying ability of my classifier.

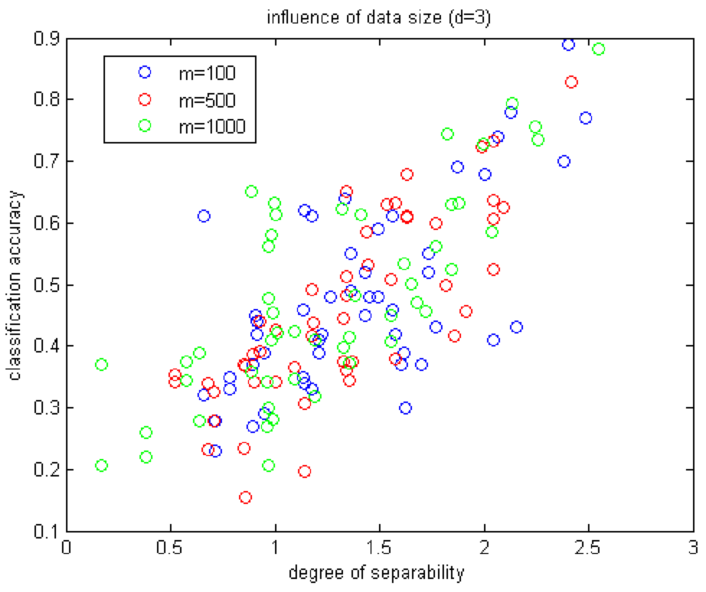


Figure 17 influence of data size

Even though the data size are changed in a large range (from 100 to 1000), we cannot find clear distinction of distribution pattern between different data size (different color) in the figure. We suppose it means the data size also will not influence the classifying ability significantly.

4. Conclusion

In this experiment, we built and tested a Bayesian Classifier according to the idea of Bayesian Decision Theory. We learned the distribution pattern of feature vector and then represent the probability of getting a specific feature vector by likelihood function. Then the posterior probability can be derived from likelihood function and prior probability to help us make judgments about which class this sample comes from. Then different loss parameter are integrated into the risk function to embody our consideration on the difference in loss caused by different mistakes.

Then we tested the performance of the classifier on multi-class multi-dimensional data set and found that the classification accuracy is relied on seperability while data dimension and size have no significant influence on classification accuracy.

I learned how the Bayesian Classifier works and how parameters influence the performance of classification by a series of experiments. The experience give me a glimpse of how to implement databased method in classifying problem and will definitely help me go further on the road of learning Pattern Classification algorithm and applications.