北京航空航天大學
BEIHANG UNIVERSITY

Pattern Recognition and Machine Learning Experiment Report

院（系）名称 自动化科学与电气工程学院

专业名称 模式识别与智能系统

学生学号 15031232

学生姓名 刘霜婷

2017 年 4 月 30 日

1 Perceptron Learning towards Linear Classification

1.1 Introduction

Linear perceptron is one of the simplest learning algorithms for a two-class classifier. Given a set of data points in d-dimensions, belonging to two classes, w1 and w2, the algorithm tries to find a linear separating hyper-plane between the samples of the two classes. If the samples are in one, two or three dimensions, the separating hyperplane would be a point, line or a plane respectively. The specific algorithm that we look into is a special case of a class of algorithms that uses gradient descent on a carefully defined objective function to arrive at a solution.

1.2 Principle and Theory

Assume that the samples of the two classes are linearly separable in the feature space. i.e., there exists a plane $G(X) = W^T X + w_{n+1} = 0$, where $W \in R^n$ and $X \in R^n$, such that all samples belonging to the first class are on one side of the plane, and all samples of the second class are on the opposite side. If such planes exist, the goal of the perceptron algorithm is to learn any one such plane, given the data points. Once the learning is completed and the plane is determined, it will be easy to classify new points in the future, as the points on one side of the plane will result in a positive value for $G(X) = W^T X + w_{n+1}$, while points on the other side will give a negative value.

According to the principle of perceptron learning, the weight vector $\mathrm{W} \in \mathrm{R}^{\mathrm{n}}$ can be extended to $\widehat{\mathrm{W}} = (\mathrm{w}_1\ \mathrm{w}_2 \ldots \mathrm{w}_{\mathrm{n}}\ \mathrm{w}_{\mathrm{n+1}})^{\mathrm{T}} \in \mathrm{R}^{\mathrm{n+1}}$ and the feature vector may be extended to $\widehat{\mathrm{X}} = (\mathrm{x}_1\ \mathrm{x}_2 \ldots \mathrm{x}_{\mathrm{n}}\ 1) \in \mathrm{R}^{\mathrm{n+1}}$ also, thus the plane of classification can be expressed as $\mathrm{G(X)} = \widehat{\mathrm{W}}^{\mathrm{T}}\widehat{\mathrm{X}} = 0$. The learning rule (algorithm) for the update of weights is designed as

$$\widehat{\mathrm{W}}(t+1) = \widehat{\mathrm{W}}(t) + \frac{1}{2}\eta\{\widehat{\mathrm{X}}(t) - \widehat{\mathrm{X}}(t)sgn[\widehat{\mathrm{W}}^{\mathrm{T}}(t)\widehat{\mathrm{X}}(t)]\}$$

$$= \begin{cases} W(t) & \widehat{\mathrm{W}}^{\mathrm{T}}(t)\widehat{\mathrm{X}}(t) > 0 \\ W(t) + \eta\widehat{\mathrm{X}}(t) & otherwise \end{cases}$$

where η is the learning rate which may be adjusted properly to improve the convergence efficiency of the learning course.

1.3 Objective

The goals of the experiment are as follows:

(1) To understand the working of linear perceptron learning algorithm.

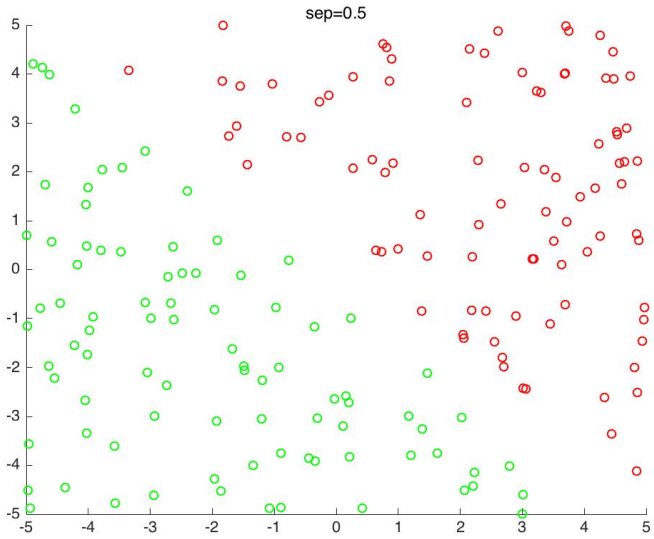
(2) To understand the effect of various parameters on the learning rate and convergence of the algorithm.

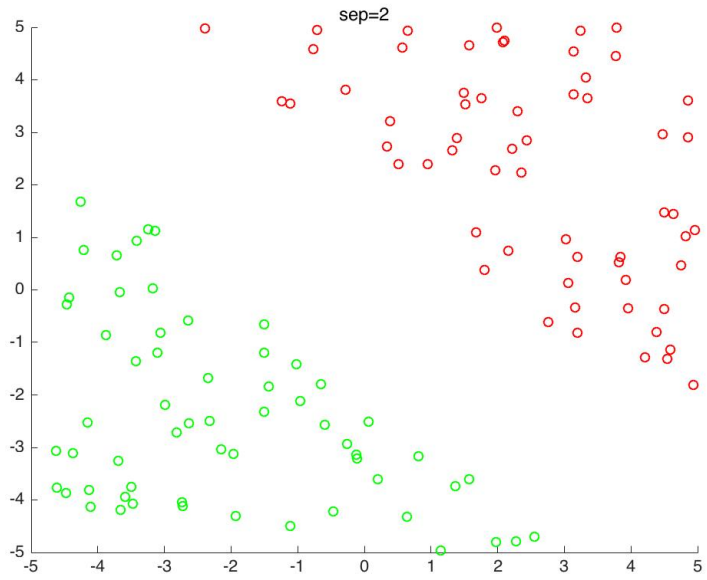
(3) To understand the effect of data distribution on learnability of the algorithm.

1.4 Contents and Procedures

Stage 1

(1) The first step is to create a linearly separable pattern dataset with more than 50 samples for each one of two classes. In this experiment, I wrote a function to generate two separate class of dataset in 2-dimension plane. The data ranges from -5 to 5. The amount of data and the distance and separation between two classes of patterns are able to adjust. The function “data_generation(number,sep)” is just used for generate dataset of two classes, where “number” represents the amount of random data and “sep” is a measurement of distance of the two classes of dataset. In the experiment, I generate the dataset randomly. The number of each class is more than 50. Here is an example of two classes of dataset with separation of 0.5 and 2, respectively.





(2) In my experiment, the initial weight vector is set to be $\hat{w}(0) = [0,0,0]^T$, the dataset has the factor "sep=1,number=200", and the learning is set to be 0.5. The process of perception learning is achieved through the function of "perception" (shown in the code). The result is as follows:

The number of iteration is 8. It is necessary to note that the number of iteration is not the same because of the randomness of the datasets. However, in terms of certain dataset, the number of iteration is unchangeable.

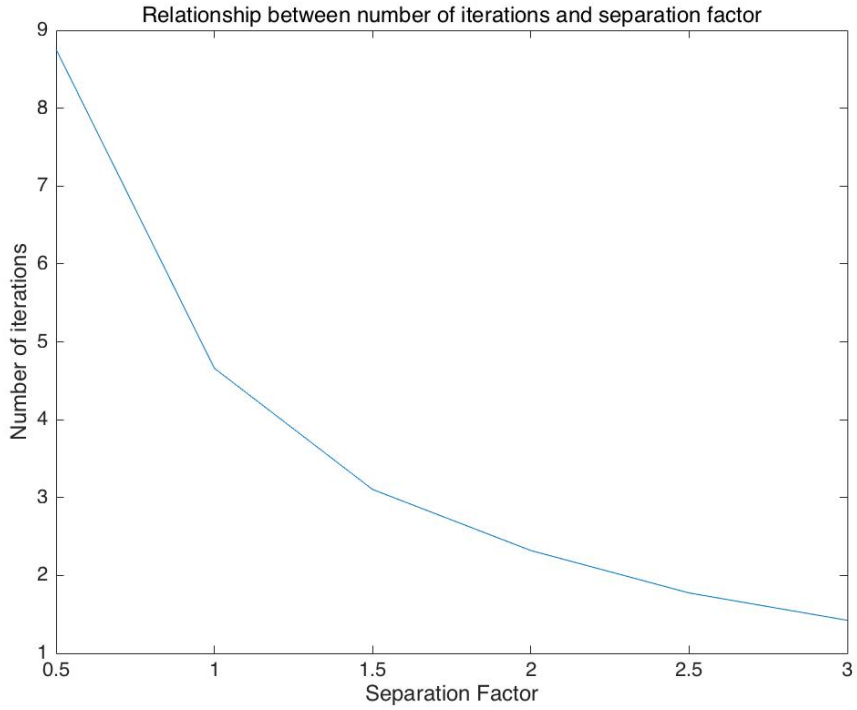
Stage 2

To study the relationship between the separation factor and the number of iterations, I change the separation factor from 0.5 to 3.0 with the increasing step of 0.5. To improve the reliability of the result, each experiment repeats 1000 times with different separation factors. By changing the separation factor and calculate the result, we can obtain the following table.

Separation Factor	0.5	1.0	1.5	2.0	2.5	3.0
Number of Iteration	8.755	4.660	3.104	2.320	1.775	1.424

As the table shows, with the increase of separation factor, the number of iteration goes down, which means that the perception learning process converge faster. Then we could draw a conclusion that the dataset with larger separation factor is easier to be classified.

The curve of the number of iteration and the separation factor is as follows:



Stage 3

Based on the analysis in Stage 2, we obtain the relationship between the separation factor and the number of iteration. As for each separation factor, we adjust the learning rate, ranging from 0.1 to 1.0, with the increasing step of 0.1. We repeat the experiment 1000 times and

calculate the average number of iteration. By changing the learning rate and calculate the result, we can obtain the following table.

Rate / SF	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.5	8.753	8.76	8.757	8.781	8.558	8.627	8.832	8.488	8.964	8.67
1.0	4.712	4.696	4.717	4.701	4.448	4.742	4.743	4.696	4.674	4.613
1.5	3.237	3.069	3.058	3.042	3.058	3.106	3.13	3.026	3.088	3.165
2.0	2.252	2.246	2.238	2.314	2.237	2.204	2.233	2.278	2.239	2.222
2.5	1.76	1.76	1.743	1.689	1.768	1.753	1.775	1.707	1.746	1.784
3.0	1.413	1.421	1.431	1.409	1.401	1.426	1.404	1.442	1.386	1.405

When the learning rate increase, the number of iterations is slightly decrease and then increase.

Since learning rate represents how much *w* will change each time, the increase of learning rate means that the *w* will change more each iteration, which leads to the decrease of number of iterations it needs to get convergence. While when learning rate is too large, the fluctuation is too large. The value of *w* will change too much each iteration, which means it will get harder to get convergence. Therefore, find a suitable learning rate is important for reduce the time of executing the program.

In this experiment, the number of iterations is slightly decrease and then increase when the learning rate increase. However, the influence of learning rate is not obvious. This may be because the characteristic of dataset. The dataset has small data range and number of samples. Also the data is idealize——all the data is separable. Since the real dataset is more complicated, it is important to set a suitable learning rate.

1.5 Experience

In the first experiment in pattern recognition, I gained a lot from conducting the experiment, after which I had a better understanding of linear perception and knew how could apply it as classifier. With the help of MATLAB, I obtained a better view of how the linear classifier could separate two or more classes since that the theory we learned in classes seemed to be a little abstract. Also, through comparison, I have a deeper comprehension of the influence of learning rate in classification problems. I believe what I learn in this experiment will be of great help in my future study.

1.6 Codes

mian.m

```
% This is the main function
clc;
%----------this part is for stage 1-----------
number=200;
A=[];
sep=1;
rate=0.5;
[class_1,class_2]=data_generation(number,sep);
[iter]=perception(rate,class_1,class_2);
% ----------------------------------------

%----------this part is for stage 2-----------
% number=200;
% rate=0.5
% A=[];
%  for sep=0.5:0.5:3
%          iter_sum=0;
%          for i=1:1000
%              [class_1,class_2]=data_generation(number,sep);
%              [iter]=perception(rate,class_1,class_2);
%              iter_sum=iter_sum+iter;
%          end
%          iter_aver=iter_sum/1000;
%          A=[A iter_aver];
%          fprintf('separation factor is:%2f\n,the number of iterations
is:%2f\n',sep,iter_aver);
%  end
% ----------------------------------------

%----------this part is for stage 3-----------
% number=200;
% A=[];
%  for sep=0.5:0.5:3
%      for rate=0.1:0.1:1
%          iter_sum=0;
%          for i=1:1000
%              [class_1,class_2]=data_generation(number,sep);
%              [iter]=perception(rate,class_1,class_2);
%              iter_sum=iter_sum+iter;
%          end
%          iter_aver=iter_sum/1000;
%          A=[A iter_aver];
%          fprintf('separation factor is:%2f\n,learing rate is:%2f\n,the
number of iterations is:%2f\n',sep,rate,iter_aver);
%      end
%  end
% ----------------------------------------
```

data_generation.m

```
% This function is used to generate dataset
 function [class_1,class_2]=data_generation(number,sep)
class_1=[];
```

```
class_2=[];
% classify the random data
for i=1:number
    data=rand(1,2)*10-5;% generate random data,ranging from -5 to 5
    if data(1)+data(2)>sep
        class_1=[class_1;data];
    elseif data(1)+data(2)<(-1)*sep;
        class_2=[class_2;data];
    end
end
% plot the data
scatter(class_1(:,1),class_1(:,2),'r');
hold on;
scatter(class_2(:,1),class_2(:,2),'g');
hold on;
end
```

perception.m

```
%  This function is used for perception learning
function [iter] = perception(rate,class_1,class_2)
%randomize the data's order
dataset_rand=[];
c1_length=length(class_1(:,1));
c2_length=length(class_2(:,1));
class1=[class_1 ones(c1_length,1)];
class2=[class_2 ones(c2_length,1)];
m=[class1;(-1)*class2];
m_length=length(m(:,1));
list=randperm(m_length);
for i=1:m_length
    x(i,:)=m(list(i),:);
end
% initialization
w=[];
w(1,:)=[0 0 0];
count=1; % represent the amount of times that the value of w doesn't change
during the iterations
iter=0; % represent the number of iterations
f_exit=0; % the flag to decide the exit condition
% find the appropriate w
while(f_exit==0)
    for t=1:m_length
        result=x(t,:)*(w(t,:))';
        if result>0
            w(t+1,:)=w(t,:);
            count=count+1;
        elseif result<=0
            w(t+1,:)=w(t,:)+rate*x(t,:);
            count=0;
            iter=iter+1;
        end
    end
    w(1,:)=w(m_length+1,:);
    if (count>=m_length)
        f_exit=1;
    else
        f_exit=0;
    end
%     decide whether the dataset is linearly separable
    if(iter>1500)
```

```
        fprintf('The dataset linearly unseparable.');
        break;
    end
    A(1)=w(t+1,1);
    A(2)=w(t+1,2);
    A(3)=w(t+1,3);
end

% plot the result
xp=-5:5;
yp=-A(3)/A(2)-A(1)*xp/A(2);
plot(xp,yp);
 end
```