成绩

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 自动化

学 生 学 号 15071129

学 生 姓 名 苗子琛

2018 年 4 月

Experiment1 Perceptron Learning towards Linear Classification

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.

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 $W \in R^{\mathrm{n}}$ and $\mathrm{X} \in R^{\mathrm{n}}$, 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 $\widehat{\mathrm{W}} = (w_1, w_2, \dots, w_n, w_{n+1})^T \in \mathrm{R}^{\mathrm{n+1}}$, and the feature vector may be extended to $\widehat{\mathrm{X}} = (x_1, x_2, \dots, x_n, 1)^T \in \mathrm{R}^{\mathrm{n+1}}$, also, thus the plane of classification can be expressed as $\mathrm{G(X)} = \widehat{\mathrm{W}}^T \widehat{\mathrm{X}} = 0$. The learning rule (algorithm) for the update of weights is designed as

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

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

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

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.

4 Contents and Procedure

Stage 1

According to the principle and theory above, I programmed MATLAB codes as Experiment_1.m and Train_perceptron.m, which the former one generates dataset and calls latter one to train a perceptron.

Initially, I generate linearly separable (by $\mathrm{x}_1 + x_2 = \pm sep, separation = 0.5$) and balanced dataset (100 samples for each of two classes). It is shown in fig.1.

Intuitively, I initialize $\widehat{W}(0)$ by function `W = rand((size(w1, 2)+1), 1);` , i.e. randomly initialize the weight, and set learning rate to be 0.5.

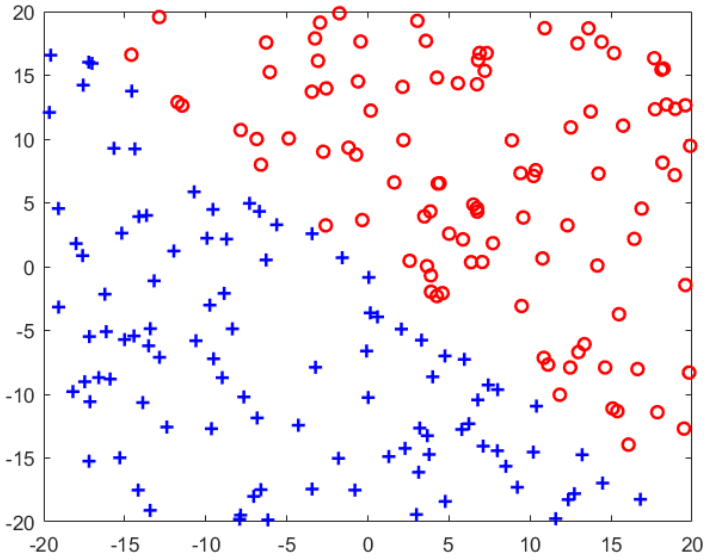
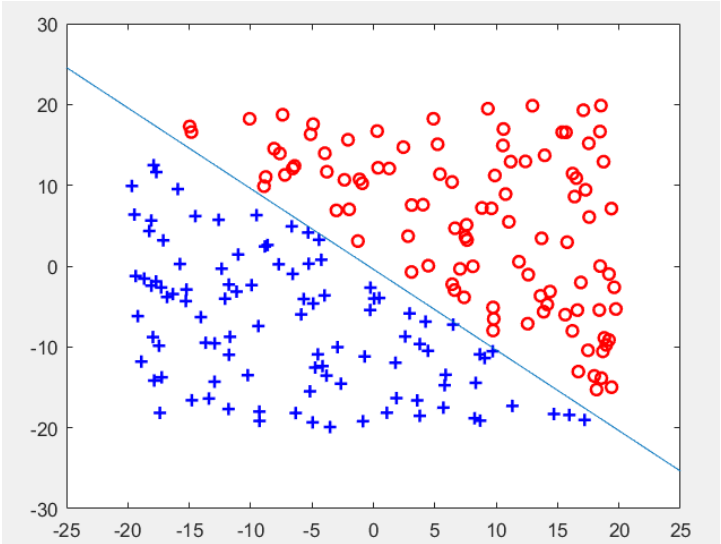


Fig.1 balanced dataset with separation 0.5

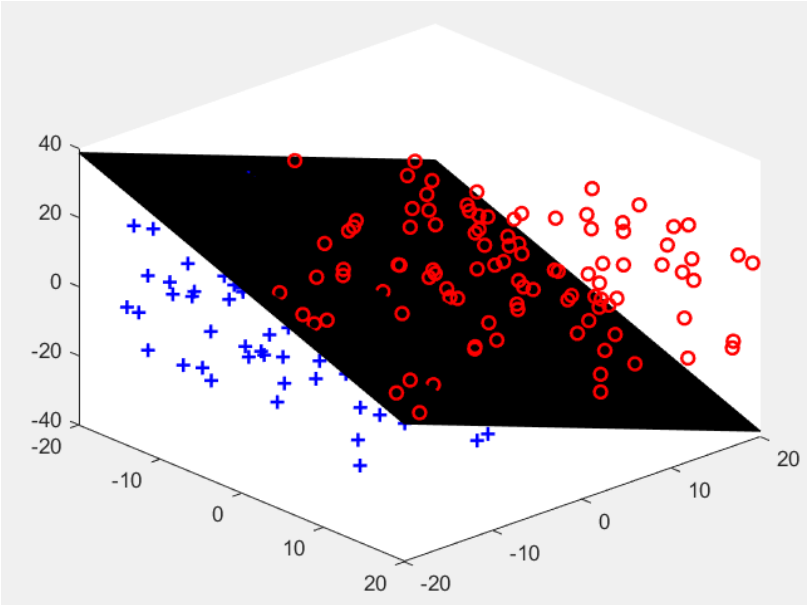
Then I test the code on this dataset and the result is shown in fig.2. The algorithm converges at this time by 7 iterations (one iteration here means use all samples in dataset for weight update). In the end, $\widehat{W} = (6.2267, 6.2143, -0.2)^T$.




```
>> Experiment_1
the perceptron is converged, iterations:7
W is :6.2267        6.2143        -0.2
```

Fig.2 result of perceptron on 3-d dataset

I also conduct perceptron algorithm on 3-dimension dataset. Separation and learning rate are set to 1 and 0.5. The result is shown in fig.3.




```
>> Experiment_1(0.5,  0.5)
the perceptron is converged, iterations:5
W is :30.4487      29.9855      31.3237      -6.59728
```

Fig.3 result of perceptron on 2-d dataset

Stage 2

The separation is set from 0.5 to 5, separated by 0.5. (`for sep = 0.5:0.5:5`). Learning rate is set to 0.3. and result is shown in table 1. Every result was average of 100 tests.

Table 1 results of stage 2

separation	0.5	1	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
iterations	3.2600	2.6200	2.2900	2.1900	2.1200	2.0200	1.9900	1.9800	1.8300	1.8000

From the table above, we can see the perceptron converges faster with a larger separation, which is intuitive and logical: Datasets with larger separation are easier to classify.

Stage 3

Based on Stage 2, we select learning rate ranging from 0.1 to 1 for each separation, the result in shown in table 3. Every result was average of 100 tests.

Table 2 results of stage 3

lr / sep	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.5	3.26	3.19	2.95	2.97	3.00	3.13	3.17	2.74	3.27	3.01
1	2.65	2.39	2.82	2.48	2.61	2.56	2.37	2.49	2.49	2.71
1.5	2.36	2.28	2.22	2.35	2.48	2.23	2.32	2.20	2.29	2.35
2.0	2.08	2.11	2.31	2.12	2.22	2.21	2.13	2.15	2.31	2
2.5	2.07	2.11	2.13	2.13	1.99	2.01	2.06	2.13	2.13	2.05
3.0	2	2.04	1.88	1.90	2.15	2.11	2.05	2.01	1.93	2.02

3.5	1.87	1.95	1.92	1.89	1.91	1.90	1.92	1.89	1.85	1.90
4.0	1.88	1.82	1.90	1.96	1.85	1.99	1.90	1.76	1.95	1.88
4.5	1.90	1.86	1.82	1.81	1.80	1.87	1.95	1.85	1.830	1.82
5.0	1.83	1.76	1.83	1.81	1.78	1.72	1.82	1.80	1.80	1.71

From the table above, we can tell that for each separation, there is a responding optimal learning rate. For example, the optimal learning rate for separation 2.0 is 2, but for separation 2.5, best learning rate is 0.5.

However, this kind of grid search for best learning rate maybe could not find the actually best learning rate. According to material online, the best method for searching learning rate is random search in log field.

5 Experience

Conducting this experiment make me fully understand the perceptron algorithm, especially its update process. Initially, I thought this process is unreasonable, because after transforming data to standard augmented data, i.e. mapping them to one side of the hyper-plane, the algorithm could not converge at all. However, after implementing the method, I got to know that weights update was carried out in one-dimension higher space, and it will surely converge if data is linearly separable.

Additionally, I knew more about the effect of hyper-parameter, learning rate, has to learning process.

6 Code

Train_perceptron.m

```
function [W, num_epoches, is_Converge] = Train_perceptron(w1, w2, num_samples,
learning_rate)
iterations_per_epoch = num_samples;
W = rand((size(w1,2)+1),1);
is_Converge = false;
max_iterations = 100000;

iterations = 0;
num_epoches = 0;
while(is_Converge == false && iterations < max_iterations)
    is_Converge = true;
    %shuffle the data before every epoch
    data_list = randperm(num_samples);
    for step = 1:iterations_per_epoch
        % dequeue data
        data_index = data_list(step);
        if (data_index <= num_samples/2)      %data from w1
            data = [w1(data_index,:) 1];          %standard augmented data
        else
```

```
            data = -[w2(data_index-num_samples/2,:) 1];
        end

        %inference
        y = data*W;

        %update weights
        if (y <= 0)
            W = W+learning_rate*data';
            is_Converge = false;
        end
        iterations = iterations+1;

        if (iterations >= max_iterations)
            disp('the dataset is not linear separable!');
            break;
        end
    end
    num_epoches = num_epoches+1;
    if (is_Converge == true)
        disp(['the perceptron is converged,iterations:',num2str(num_epoches)]);
    end
end
end
```

Experiment_1.m

```
function [iterations] = Experiment_1(sep, learning_rate, dimension)
%% generate dataset
num_samples = 200;
%define seperation line: x1+x2=sep/x1+x2=-sep
w1 = [];
w2 = [];

%generate dataset and assign their identity according to their position
num_gen_w1 = 0;
num_gen_w2 = 0;

%construct balanced dataset
while(1)
    data = rand(1,dimension)*40-20;
    if (sum(data) > sep)
        w1 = [w1; data];
        num_gen_w1 = num_gen_w1+1;
```

```
        end
        if((num_gen_w1 == num_samples/2))
            break;
        end
    end
    while(1)
        data = rand(1,dimension)*40-20;
        if (sum(data) < -sep)
            w2 = [w2; data];
            num_gen_w2 = num_gen_w2+1;
        end
        if((num_gen_w2 == num_samples/2))
            break;
        end
    end
    %               plot(w1(:, 1), w1(:, 2), 'or', 'LineWidth', 1.5);hold on;
    %               plot(w2(:, 1), w2(:, 2), '+b', 'LineWidth', 1.5);hold on;
    %% train the perceptron
    [W, iterations, is_Converge] = Train_perceptron(w1,w2,num_samples,learning_rate);
    if (is_Converge == true)
        if(dimension == 2)
            %display 2d
            plot(w1(:, 1), w1(:, 2), 'or', 'LineWidth', 1.5);hold on;
            plot(w2(:, 1), w2(:, 2), '+b', 'LineWidth', 1.5);hold on;
            X = [-25:0.05:25];
            Y= -1*(W(1)/W(2))*X-(W(3)/W(2));
            plot(X,Y);

        elseif(dimension == 3)
            %display 3-d
            plot3(w1(:, 1), w1(:, 2),w1(:, 3), 'or', 'LineWidth', 1.5);hold on;
            plot3(w2(:, 1), w2(:, 2),w2(:, 3), '+b', 'LineWidth', 1.5)
            X1 = linspace(-20,20,400);
            X2 = linspace(-20,20,400);
            [X1,X2]=meshgrid(X1,X2);
            Y= -1*(W(1)/W(3))*X1-(W(2)/W(3))*X2-(W(4)/W(3));
            surf(X1,X2,Y);
        end
    disp(['W is :',num2str(W')]);
    end
    end
```