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 15031204

学生姓名 张家奇

2018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 $\mathrm{G}(\mathrm{X}) = \widehat{\mathrm{W}}^{\mathrm{T}}\widehat{\mathrm{X}} = 0$. 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.

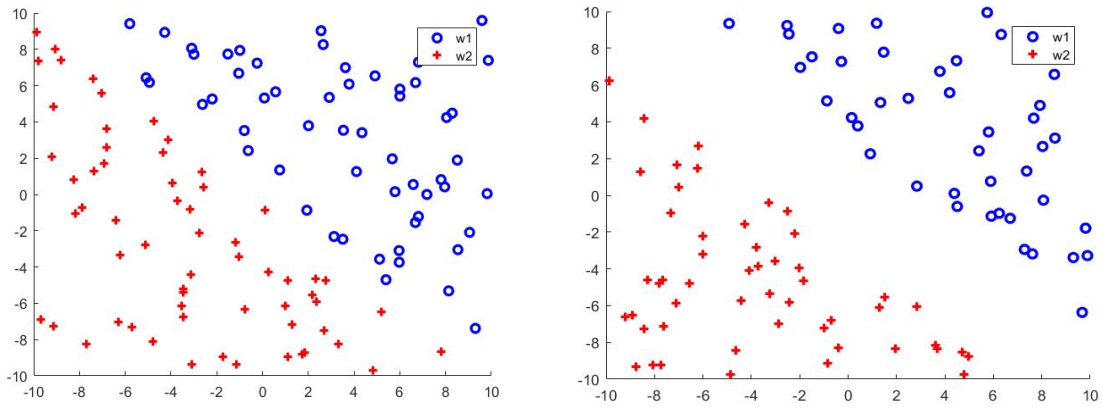
(3) To understand the effect of data distribution on learnability of the algorithm.

1.4 Contents and Procedures

Stage 1

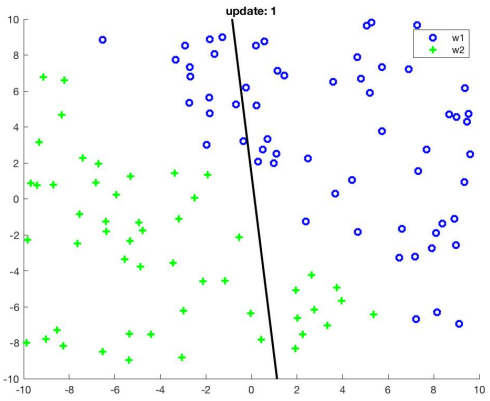
(1) I conducted this experiment based on Matlab. The function 'generate_dataset' is programmed to create a 2-d linearly separable dataset, with arbitrary separation between the two classes of more than 50 samples each. The first parameter of this function regulates the separation between the two datasets, while the second parameter (optional) initializes the Matlab built-in random number generator (rng) for the convenience of result comparisons. Please refer to the code attached to the end of this report.

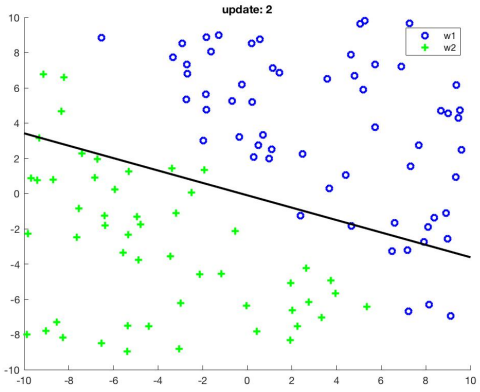
Here is an example of two datasets with separation of 0.5 and 3, respectively.

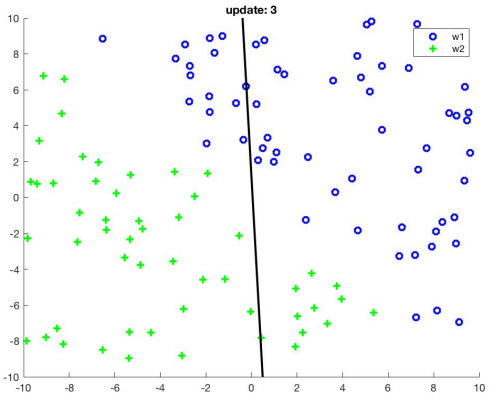


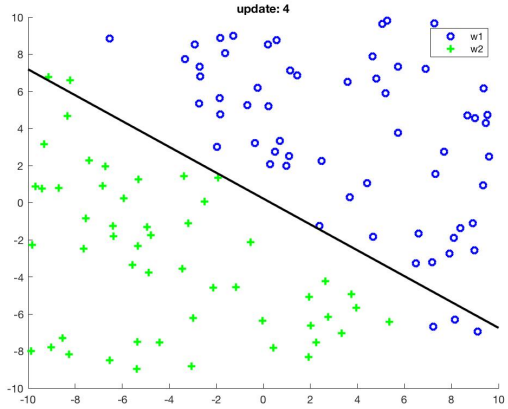
(2) The function 'classify_perceptron' uses a linear perceptron to classify the dataset. It calls the function 'generate_dataset' aforementioned, randomly initializes the weight vectors uniformly between -1 and 1. Both the dataset and the separating plane are plotted by step during the learning procedure, and the number of updates on the weight vector is returned after convergence.

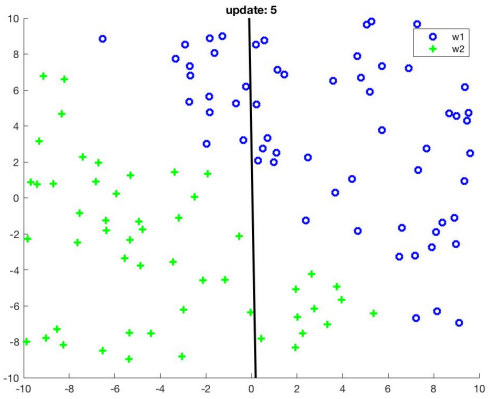
Below is a typical example of convergence on a dataset of 0.5 separation and 0.5 learning rate. For the convenience for visualization, I carefully chose a set of parameters, so that the perceptron converged after 8 iterations (which usually takes more). The weight vector converged at $\widehat{W} = (5.6789, 5.8076, -0.7460)^T$.

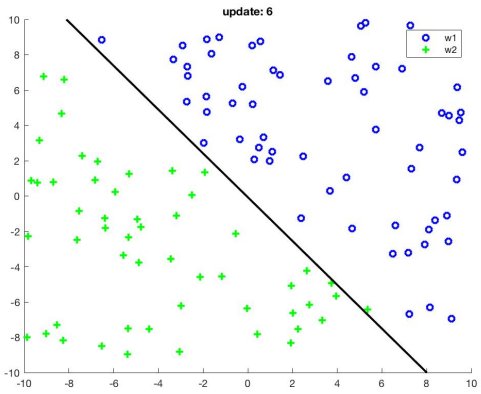

update: 1
w1
w2

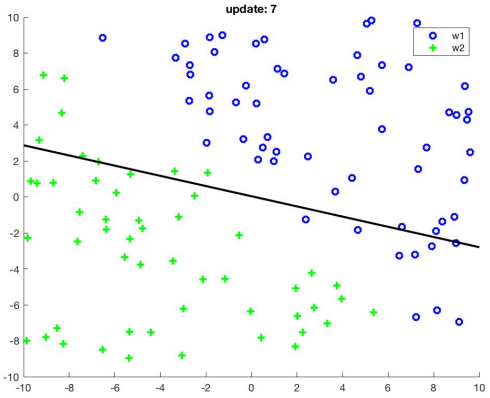

update: 2
w1
w2

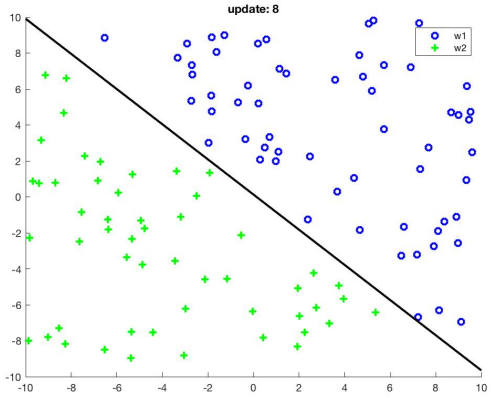

update: 3
w1
w2


update: 4
w1
w2


update: 5
w1
w2


update: 6
w1
w2


update: 7
w1
w2


update: 8
w1
w2

Stage 2

Varying the dataset separation from 0.5 to 5, the perceptron converged after the following number of updates on the weight vector, with a learning rate of 0.5. Each number is an average over 1000 random-initialized tests.

Separation	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
#Updates	10.37	7.01	5.05	4.01	3.21	2.70	2.33	1.97	1.70	1.48

As seen in the table above, the perceptron converges faster with a larger separation between datasets, which is intuitive and logical: Datasets with larger separation are easier to classify.

Stage 3

Varying both the dataset separation and learning rate, the following form was generated, which depicts the number of updates before convergence. Each number is an average over 1000 random-initialized tests.

Rate / Sep	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0.5	13.18	12.24	11.03	10.88	10.37	11.20	11.10	10.68	11.06	11.37
1.0	7.80	7.41	7.24	7.00	7.01	6.96	6.81	6.79	7.14	6.75
1.5	5.65	5.09	5.11	5.00	5.05	5.20	5.13	5.27	4.89	5.14
2.0	4.44	4.08	4.14	4.06	4.01	4.07	4.04	3.98	3.96	4.24
2.5	3.71	3.35	3.32	3.19	3.21	3.31	3.23	3.11	3.18	3.26
3.0	3.09	2.79	2.68	2.78	2.70	2.78	2.73	2.72	2.68	2.60
3.5	2.50	2.38	2.29	2.29	2.33	2.33	2.30	2.32	2.33	2.27
4.0	2.25	2.00	1.94	1.92	1.97	1.90	1.90	1.98	1.91	1.98
4.5	1.97	1.73	1.69	1.70	1.70	1.72	1.68	1.66	1.72	1.64
5.0	1.69	1.51	1.47	1.47	1.48	1.45	1.44	1.46	1.52	1.50

As depicted in the table above, the number of updates decreases with an increasing learning rate.

However, a moderate learning rate should be used. In some peculiar cases where a certain sample locates far off from other samples, the plane of classification may be pulled towards it abruptly with a large learning rate, impeding convergence.

1.5 Experience

Conducting this experiment enhanced my understanding over the concept of linear perceptron. Linear perceptron manages to find a separating plane of two linear-separable datasets, and is easy to implement.

However, most datasets are not linear separable, such as an XOR relation, in which case the linear perceptron fails to find a separating plane. In addition, even if such planes exist, linear perceptron does not guarantee to find the best one. However, it is the basis of both SVMs and artificial neural networks, which later overcame these weaknesses.

1.6 Code

```
function [w1, w2] = generate_dataset(sep, rng_init)
if(nargin >= 2)
    rng(rng_init)
end
w1 = [];
w2 = [];
cnt = 0;
while(1)
    cnt = cnt + 1;
    pt = rand(1, 2) * 20 - 10;
    if(pt(1) + pt(2) > sep)
        w1 = [w1; pt];
    elseif(pt(1) + pt(2) < -1 * sep)
        w2 = [w2; pt];
    end
    if(cnt > 120)
        break
    end
end
```

Detailed experiment results and specific code are on my github homepage:

https://github.com/MagicOwO/BUAA-Pattern_Recognition_and_Machine_Learning

```
function iter_cnt = classify_perceptron(dataset_sep, learning_rate, ~)
% Generate dataset
[w1, w2] = generate_dataset(dataset_sep);
% Sequence for updates
seq_list = randperm(size(w1, 1) + size(w2, 1));
% Set random number generator for test
% rng(0)
% Plot original dataset
% f = figure(1);
hold on
if(nargin >= 3)
    pause off
else
    pause on
end
plot(w1(:, 1), w1(:, 2), 'ob', 'LineWidth', 2)
plot(w2(:, 1), w2(:, 2), '+g', 'LineWidth', 2)
```

```
legend('w1','w2')
% Classify with perceptron
w = 2 * rand(3, 1) - 1;
converged = 0;
iter_cnt = 0;
epoch = 0;
x = [-10 : 0.01 : 10];
y1 = -1 * (w(1) / w(2)) * x - (w(3) / w(2));
h = plot(x, y1);

while(converged == 0)
    converged = 1;
    epoch = epoch + 1;
    for i = 1 : length(seq_list)
        seq = seq_list(i);
        sample = [];
        % If sample belongs to w1
        if(seq <= length(w1))
            sample = [w1(seq, :), 1];
        % If sample belongs to w2
        else
            sample = -1 * [w2(seq - length(w1), :), 1];
        end
        y = sample * w;
        if y < 0
            iter_cnt = iter_cnt + 1;
            w = w + learning_rate * sample';
            converged = 0;
            y1 = -1 * (w(1) / w(2)) * x - (w(3) / w(2));
            delete(h)
            h = plot(x, y1, 'k', 'LineWidth', 2);
            axis([-10 10 -10 10]);
            title(['update: ', num2str(iter_cnt)]);
            pause(0.01)
        end
    end
    if(iter_cnt > 10000)
        disp('Dataset linearly unseparable. Perceptron does not converge.')
        return
    end
end

disp(['The perceptron converged with ',num2str(epoch) , ' epochs of ',
num2str(iter_cnt), ' updates'])
disp(w')
end
```