AHSANULLAH UNIVERSITY OF SCIENCE AND TECHNOLOGY

Pattern Recognition Lab

Experiment Number 02

"Implementing the Perceptron algorithm for finding the weights of a Linear Discriminant function"

Submitted by-
MD. SHAMIM TOWHID
ID: 12.02.04.097
Section: B2
Year: 4th Semester: 2nd
Date: 30-07-2016

Implementing the Perceptron Algorithm for Finding the Weights of a Linear Discriminant Function

Md. Shamim Towhid
Computer Science and Engineering Department
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
shamim.towhid@gmail.com

***Objectives*—the objective of this experiment is to apply perceptron algorithm to find the weights of a linear discriminant function. Perceptron algorithm is an incremental way for finding the weights of a linear discriminant function. In this experiment we will apply this algorithm to find weights of a given linear discriminant function. In this algorithm we start with a random weights and gradually we will forward to the actual weights. Though this algorithm has some drawbacks we will apply it for its simplicity. There are two implementation of this algorithm: one is batch processing (also known as many at a time), and the other is one at a time. We will evaluate our sample data with both process and in the result analysis section we will compare the performance of this two methods with different learning rate.**

Keywords—discriminant functions; pattern recognition; perceptron algorithm; MATLAB code; FI function;

I. Introduction

To think about perceptron algorithm first we have to think about minimum distance to class mean classifier because perceptron algorithm can be thought of a simplified version of minimum distance to class mean classifier. In minimum distance to class mean classifier we know the decision rule is as follows-

If g(x) >0 then x € ω_1

If g(x) <0 then x € ω_2

Now,

$g(x) = w^t x + w_0$

Augmented form is $g(x) = w^t x$

We know here w is the weight matrix. X can be easily found, so if we calculate the weight matrix then we can easily differentiate between two classes with a decision boundary. We can use gradient descent method where number of incorrect samples denotes by y axis and weight is plotted in x axis. Now with gradient descent method we can gradually moves to the downward to minimize the error. Where the error is zero or nearly to zero then the w value is our desired weight values. Here the challenge is what will be the step size that we will use to go downward gradient. It can be written by the following equation-

$W(t+1) = w(t) + \eta \vee J$

Here $\vee J$ is known as perceptron criterion function. Number of misclassified samples can't be the perceptron criterion function in gradient descent method because there may be some flat region in the graph where the gradient descent method will stuck.

So we will use the distance from our decision boundary to the misclassified samples as perceptron criterion function.

Another important concept we will need in this experiment and that is the FI function. Sometimes there is no way we could draw a decision boundary between two classes. Then we have to go to a higher dimension and we hope that there will exist a solution. The FI function helps us to move to a higher dimension.

In this experiment we are given the following two classes-

ω1 = {(1,1), (1,-1), (4,5)}
ω2 = {(2,2.5), (0,2), (2,3)}

First we will plot the above two classes and will observe that if there is a way to draw a decision boundary between them. After that we will move to a higher dimension by using the following given FI function-

Y = [x1^2 x2^2 x1*x2 x1 x2 1]

Then we will use perceptron algorithm to draw a decision boundary between the classes.

II. Implementation

A. *Plotting the sample points and observing*

We simply plot the given sample points first. We use different marker and color for different classes to distinguish them. We use green color for class one and red color for class two. The output is as follows-

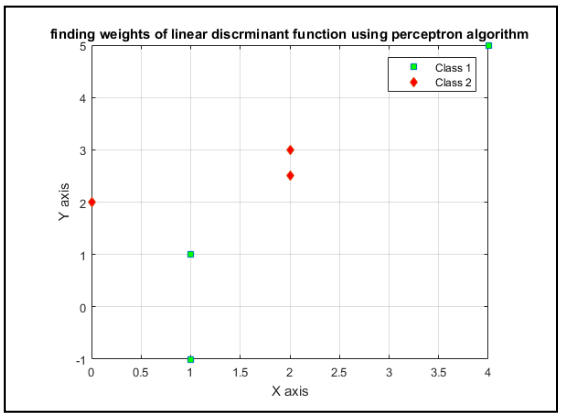

Figure 1: plotting the given sample points

Now observing the figure above we can see that there is no way we can draw a decision boundary to distinguish between the classes. Every time there will be some error. So we will have to use a FI function to move to a higher dimension.

B. *Calculating high dimensional sample points*

We have to calculate the higher dimensional sample points for each point of the given class. The given FI function moves our sample points to a six dimensional space.

Before using gradient descent technique we have to normalize any one of the class. Normalization or reflection is a process to shift one class completely to the opposite. It can be done only in two class problem. Here we normalize class two. After normalizing we get the following higher dimensional sample points-

1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
1.0000	1.0000	-1.0000	1.0000	-1.0000	1.0000
16.0000	25.0000	20.0000	4.0000	5.0000	1.0000
-4.0000	-6.2500	-5.0000	-2.0000	-2.5000	-1.0000
0	-4.0000	0	0	-2.0000	-1.0000
-4.0000	-9.0000	-6.0000	-2.0000	-3.0000	-1.0000

Now we can use gradient descent method to find the weight matrix.

C. *Applying gradient descent method*

From our previous discussion we came to know that here we will use distance between misclassified samples and decision boundary as perceptron criterion function. Thus-

$J_p = - w^t y$ ------------------------------ (i)

Here y is only misclassified samples and minus (-) sign is taken because we normalize one class here. After differentiating equation one with respect to w we obtain,

$\vee_w J_p = -y$ ------------------------------- (ii)

Now, $w(t+1) = w(t) + y\eta$

$= w(t) + y$ [here $\eta = 1$]

Since we are using one at a time method we will process one point at a time. Since $J_p = -w^t y$ so $g(y) = w^t y$ must be greater than zero.

Now, we will evaluate one point at a time and calculate g(y) for that point and update weight whenever $g(y) <= 0$.

Thus we will get a set of weight values for which all the point will give positive values. According to the algorithm this set of values is a valid weight values. So we can draw decision boundary with the weight values.

D. *Drawing the decision boundary*

After calculating the weight values we can simply draw a decision boundary that will distinguish the classes in higher dimension. We simply fixed a x axis range for the boundary and then put the values of x in the equation of the decision boundary, we will get a set of y values. After that we can draw a boundary.

III. Result Analysis

The final output of this program is as follows-

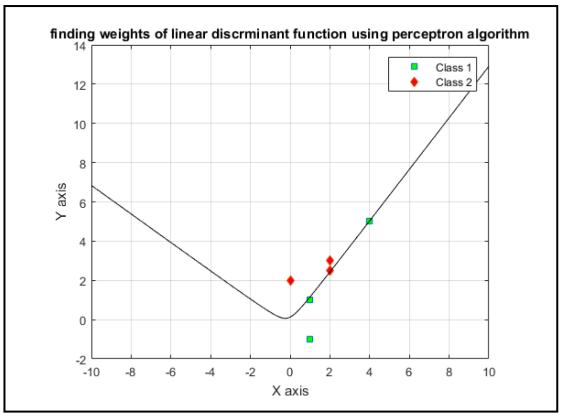

Figure 2: Final output

From the figure we can see that the decision boundary differentiate between two classes. The red marker denotes class two and the green marker in the figure denotes class 1. If we use many at a time process we will get a similar output but the weight vector will be different because many at a time process works differently. It simply calculates all the g(x) value first then it checks how many g(x) is less than zero. Then it simply adds the value of y whose corresponding value of g(x) is less than zero. Then it calculates all the value of g(x) again and it continues until all the values of g(x) become positive. In the above output figure it is not clear that the boundary differentiate between the two given classes. To see it more clearly if we zoom the above figure and obtain the following-

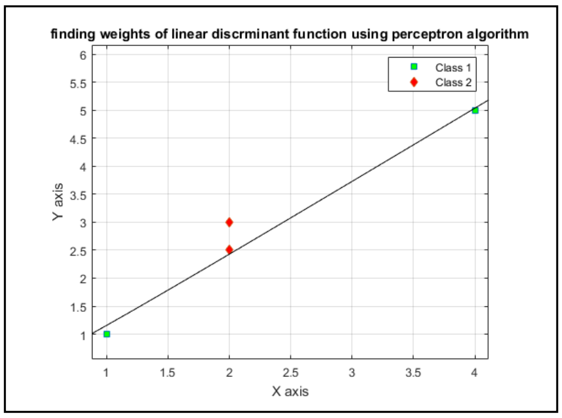

Figure 3: Zooming result of the previous output

From the above figure it clear that the decision boundary divide the two classes accurately.

In this experiment the step size or learning rate (η) is a very important factor. It ranges 0 < η <= 1

If we vary this factor then the step to calculate the weight will vary also. A comparison table is given below between one at a time method and many at a time method with different learning rate. We know that the best value for the learning rate is-

η = 1/k [where k is the number of total sample points, here k=6]

so the best value for learning rate in this experiment is 1/6 = 0.16666 ~ 0.17

Value of ETA (η)	One at a time	Many at a time
0.1	147	166
0.2	132	114
0.3	112	118
0.4	124	123
0.5	105	146
0.6	105	146
0.7	94	106
0.8	94	106
0.9	94	106
1.0	94	106
0.17	127	154

Figure 3: Comparison table

From the above comparison table we can see that the smaller the step size the greater number of step it will take to find the weight matrix.

Another important observation in this experiment is the FI function is chosen randomly so sometimes it is possible that the higher dimensional sample points can't give any solution.

The weight matrix also initialized first randomly. Here we initialized it with all zero values then gradually it will update itself towards solution.

To visualize it more clearly and compare between one at a time and many at a time we plot the values of step taken against ETA for each method and obtain the following 2D graph-

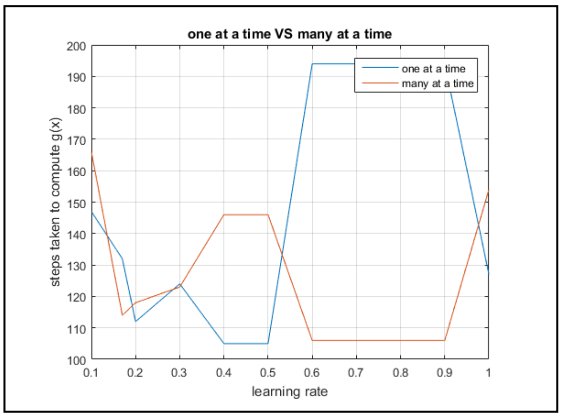

Figure 4: One at a time VS Many at a time

Actually the result and the iteration value depends on the learning rate and the initial guess and the FI function used. So the iteration number varies differently for each method. Since there is no actual method for choosing FI function this algorithm cannot give solution each time. The iteration number also depends on the initial guess, the more close the initial guess to the solution the less steps required to reach the solution. Also the learning rate should small enough that it does not jump over the solution state of the weight vector. If so then it will fall in an infinite loop that cannot give any solution.

MATLAB CODE:

One at a time:

```
class1=[1 1;1 -1;4 5];
class2=[2 2.5;0 2;2 3];
%plotting the sample points
plot(class1(:,1),class1(:,2),'s','MarkerF
aceColor','g');
hold on;
plot(class2(:,1),class2(:,2),'d','MarkerF
aceColor','r');

%generating high dimensional sample
points using fi function
%y=[x1^2 x2^2 x1*x2 x1 x2 1]
for m=1:3
    for n=1:6
        if n==1
```

```
            x(m,n)=class1(m,n)^2;
        elseif n==2
            x(m,n)=class1(m,n)^2;
        elseif n==3

x(m,n)=class1(m,1)*class1(m,2);
        elseif n==4
            x(m,n)=class1(m,1);
        elseif n==5
            x(m,n)=class1(m,2);
        else
            x(m,n)=1;
        end
    end
end

for m=1:3
    for n=1:6
        if n==1
            y(m,n)=class2(m,n)^2;
        elseif n==2
            y(m,n)=class2(m,n)^2;
        elseif n==3

y(m,n)=class2(m,1)*class2(m,2);
        elseif n==4
            y(m,n)=class2(m,1);
        elseif n==5
            y(m,n)=class2(m,2);
        else
            y(m,n)=1;
        end
    end
end
%disp(x);
%disp(y);


%normalization of class 2
y=-y;
%disp(y);

%Applying perceptron algorithm to find
the weight coefficients
final=[x;y];
disp(final);
weight=[0 0 0 0 0 0]; %initialization
g=0;
counter=0;
flag=true;
control=6;
while flag
    for m=1:6
        g=g+final(m,:)*weight';
        if g<1
                g=0;

weight=weight+final(m,:)*0.17;%assuming
alpha(learning rate) = 1
                %disp(weight);
                control=control+1;
        else
            control=control-1;
            g=0;
        end
    end
    if control==0
        flag=false;
    else
        control=6;
    end
    counter=counter+1;
end
disp(counter);
disp(weight);
%drawing decision boundary
syms x1 x2;
s=sym(weight(1)*x1*x1+weight(2)*x2*x2+wei
ght(3)*x1*x2+weight(4)*x1+weight(5)*x2+we
ight(6));
s2=solve(s,x2);

xvals1=(-10:0.01:10);
xvals2(1,:)=subs(s2(2),x1,xvals1);
plot(xvals1,xvals2(1,:),'k');

legend('Class 1','Class 2');
title('finding weights of linear
discrminant function using perceptron
algorithm');
grid;
xlabel('X axis');
ylabel('Y axis');
hold off;
```

Code for many at a time:

```
while flag
    for m=1:6
        g(m)=g(m)+final(m,:)*weight';
    end
    for m=1:6
        if g(m)<1

weight=weight+final(m,:)*1.0;%assuming
alpha(learning rate) = 1
                control=control+1;
                counter=counter+1;
        else
            control=control-1;
        end
    end
    if control==0
        flag=false;
    else
        control=6;
    end
    g = zeros(size(g));
    %counter=counter+1;
end
```

Code for Comparison Graph:

```
x=[0.1 0.17 0.2 0.3 0.4 0.5 0.6 0.7 0.8
0.9 1.0];
one_at_a_time=[147 132 112 124 105 105
194 194 194 194 127];
many_at_a_time=[166 114 118 123 146 146
106 106 106 106 154];

plot(x,one_at_a_time,x,many_at_a_time);
legend('one at a time','many at a time');

title('one at a time VS many at a time');
grid;
xlabel('learning rate');
ylabel('steps taken to compute g(x)');
```

IV. Conclusion

Since the algorithm stops whenever it gets a weight matrix immediately the decision boundary will be very close to one particular class and very far from another one. The solution space is very large in this case. Thus there is another kind of solution to this problem and that is we will stop with positive values after a threshold values. This will give a margin with the decision boundary and we will draw our actual boundary with the margin, thus the performance can be improve.