Machine Learning course

Lab assignment 1: linear-threshold classifiers

Augustin VIOT

October 17, 2017

Contents

1 Introduction

This document is a repport of the lab assignement done in the course of Machine learning at the University of Genova (UNIGE). It describes the basic methods and implementations for the Simple linear classifier. We will describe the three tasks done in this lab assignement :

1. Program a simple linear classifier
2. Simple linear classifier for 2D binary problems, manual design
3. Simple linear classifier for Iris data, manual design

2 Program a simple linear classifier

For this task, the function was given, so we will simply give an analysis of the function.

```
        function y = linclass(x,w)
                % linear threshold classifier
                %   x: n x m data set to classify
                %   w: 1 x m coefficient vector of hyperplane
                y = 2*(x*w'<0)-1;
        end
```

The function takes two parameters:

- a set of n data, as a n x d matrix (or possibly d+1), observations in rows
- a row vector w of d (or d + 1) coefficients

It returns a vector **y** of n values in {-1, +1}, each representing the classification output for the corresponding input pattern (row of the input data matrix).

We can see that we changed the intial function a little : We added a *2 -1 to the resultuts, because we want this function to return the class given by the classifier w (the target set take is {-1, 1 }).

3 Simple linear classifier for 2D binary problems, manual design

3.1 Dataset1

In this exercice, we had to create a linear classifier to separate the data provided (7 two dimensions points). To do so, we used graphical method to determine an hyperplan that separate linearly the data. Here is the basic code used to charge the data and define a target vector for each dataset:

```
%load a file :
dataset1 = load('dataset1.txt');
dataset2 = load('dataset2.txt');
%we create a vector that contains the target for each dataset
t1=dataset1(:,3);
t2=dataset1(:,3);
```

The target vector is created so it is easier to access directly the target information without having to call the all dataset.

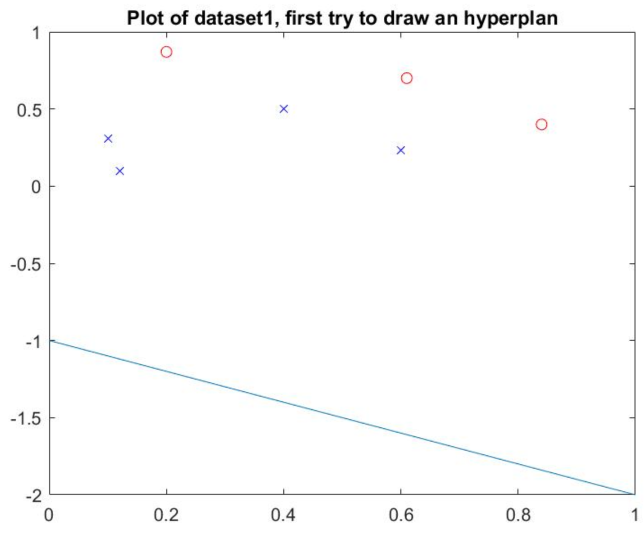
We also created a function **resultLinClass** that calculates the percentage of correctly classifyed points. We will use this value to estimate the classifyer efficency instead of counting the number of correctly classifyed points (so we can reuse the function with other datasets since it is more general than counting the points). Here is the code of this function and a short description of it :

```
function resultLinClass = resultLinClass(dataset, w, t)
        %This function calculates the percentage of points correctly
            classyfied
        %it takes three parameters :
        %       dataset : the dataset we want to check
        %       w                : 1 x m coefficient vector of
            hyperplane used to classify
        %       t                : the vector containing the target of
            each individual

        numberOfCorrectlyClassifyedPoints = sum(linclass(dataset, w)==t
            );
        numberOfPoints = size(dataset, 1);

        %size(A,1) for number of rows. size(A,2) for number of columns
        resultLinClass = numberOfCorrectlyClassifyedPoints /
            numberOfPoints;
end
```

To find the correct classifyer, our strategy was to correct the hyperplan parameters in a way that provides a satisfaying graphic linear separation. We started with w=`[1;1;1]`.

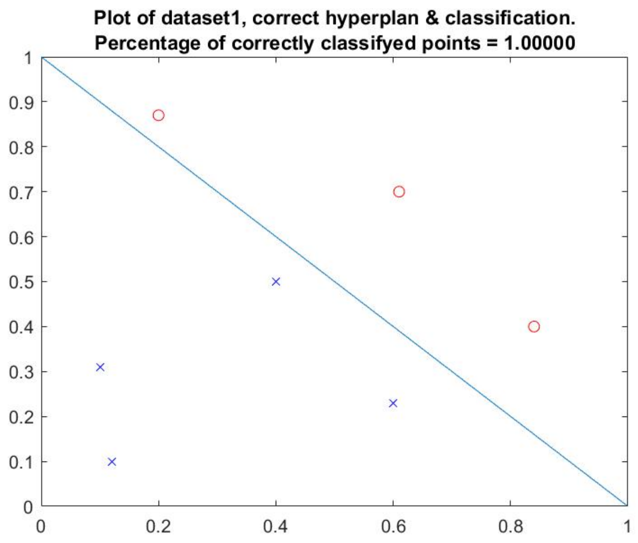
```
w1 = 1;
w2 = 1;
w3 = 1;
w = [w1 w2 w3];
percentageOfClassifyedPoints = resultLinClass(dataset1, w, t1);
disp('the percentage of classifyed with w = [1;1;1] points is  : ');
disp(percentageOfClassifyedPoints);
%result in console :
%        the percentage of classifyed with w = [1;1;1] points is  :
%        0.4286

%let's plot with the line to know how we should change the coefs :

figure(3);
plot(dataset1(t1==1,1),dataset1(t1==1,2),'xb');
hold on;
plot(dataset1(t1==-1, 1), dataset1(t1==-1,2),'or');
title('Plot of dataset1, first try to draw an hyperplan') % add figure
    title
% linspaceForVector = linspace(0,1,1);
x = linspace(0,1,10);
hyperplan = -w3/w2 -(w1*x)/w2;
plot(x, hyperplan);
hold off;
```

This try gave us the following figure :



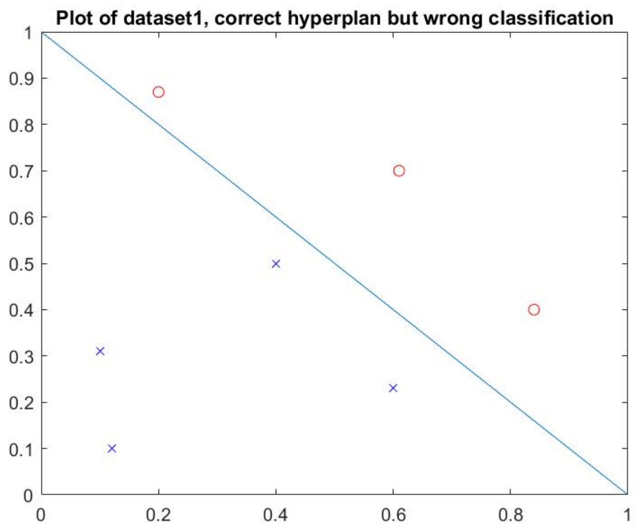
We can see here that the hyperplan does not separate correctly the data. We change manually the hyperplan parameters to obtain a correct hyperplan.

With the parameters w = w=`[1;1;-1]`, we obtain a correct classifyer. Here is the graph produced with this hyperplan :



We computed the percentage of correctly classifyed points with our function **resultLinClass**. The results given by the function was 100%.

During the try and retry tests to get the correct paramters for the hyperplan, we found a confusing configuration. Actually, with w=`[-1;-1;1]`, we obtain a graph that shows a line in the same position as the previous hyperplan (so correct graphical separation), but when we compute the percentage of correctly classifyed points, we obtain 0%. Here is the grap obtained with w=`[-1;-1;1]`.



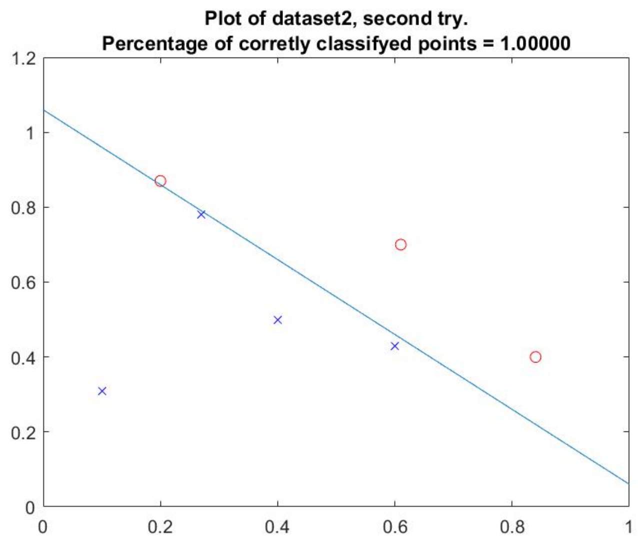
This can be explained by the equation used to classify the points "linclass.m". Actually, if we change the ">" to "<", the problem is solved. It is in fact because of our class names. If the class had opposite names, this classifyer would perfectly work.

3.2 Dataset 2

We followed the same method to find a good classifyer for the second dataset. Here are the parameters we found w=`[1;1;-1.06]`. Here is the code used for this dataset :

```
w1dataset2 = 1;
w2dataset2 = 1;
w3dataset2 = -1.06;
wdataset2 = [w1dataset2 w2dataset2 w3dataset2];
percentageOfClassifyedPointsdataset2 = resultLinClass(dataset2,
    wdataset2, t2);
disp('the percentage of classifyed points is : ');
disp(percentageOfClassifyedPointsdataset2);
figure(9);
plot(dataset2(t2==1,1),dataset2(t2==1,2),'xb');
hold on;
plot(dataset2(t2==-1, 1), dataset2(t2==-1,2),'or');
stringTitle = compose("Plot of dataset2, second try.\nPercentage of
    corretly classifyed points = %.5f",
    percentageOfClassifyedPointsdataset2);
title(stringTitle) % add figure title
x = linspace(0,1,10);
hyperplan = -w3dataset2/w2dataset2 -(w1dataset2*x)/w2dataset2;
plot(x, hyperplan);
hold off;
```

Here is the graph displaying the hyperplan for this dataset :



3.3 Adding a random perturbation

Now we add a random perturbation to our hyperplan. You can find a detailed example and explanation of a random perturbation for only one hyperplan in the code, but here we will only describe the final result. Here is the function used to generate a growing perturbation (with 100 iteration for each value of p).

```matlab
%we are going to store the percentage of correctly classifyed
%poitns is the following variable :
matrixOfErrors = zeros(100,10);
%we have 10 colonmns, each colomns is a p value (0.1, 0.2, ..., 0.10)
%we have 100 lines, each p value has 100 percentage of error to
    calculate

counter = 1;
%we also keep a counter because we can't use p directly as an index to
%access the values inside the matrixOfErrors
disp("this is the w value we will use :");
disp(w);
wNormalized = w / norm(w);

for p = 0.01:0.01:0.1
        for i = 0:100
                randomVector = 2*rand(1,3) -1;
                randomVector = randomVector / norm(randomVector);
                randomVector = p * randomVector;

                wWithPerturbation = wNormalized + randomVector;
                percentageOfClassifyedPoints = resultLinClass(dataset1,
                    wWithPerturbation, t1);
                matrixOfErrors(i+1,counter) =
                    percentageOfClassifyedPoints;
        end
        counter = counter + 1;
end
```

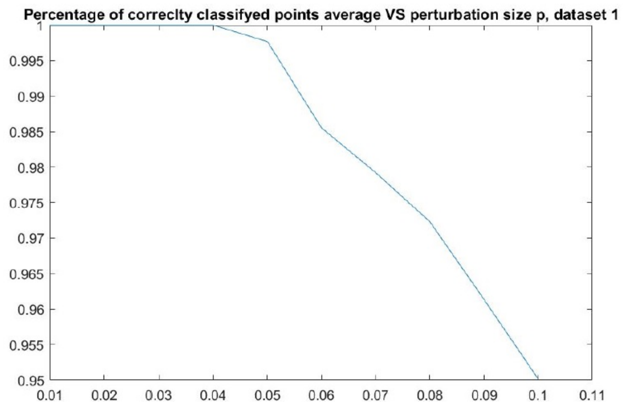
Explanation of the loop : the p loop represents all the perturbation percentage we want to cover, we will go from 0.01 to 0.1 with a stepsize of 0.01. The i loop indicates how many iterations should be compute for each p value. To provide more accurate curve, we used 10000 iterations to generate the following graphs. To generate the random perturbation, we followed the explanation given by the teacher. You can also find precise exlanation and example directly in the code.

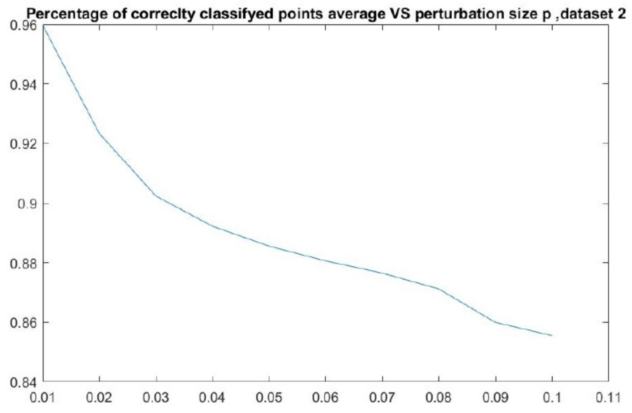
Now we can represent the consequence of the perturbation on the classification.

Comment : We decided to represent the percentage of correctly classifyed points instead of the average number of errors because it makes more sense.

```matlab
                percentageErrorsAverrage = mean(matrixOfErrors);
                %we also create a vector of p value :
                pValuesVector = 0.01:0.01:0.10;
                %then we just have to plot the percentage average VS
                    the p values :
                figure(11);
                plot(pValuesVector, percentageErrorsAverrage);
                hold on;
                title('Classification Error Average VS perturbation
                    size p, \ndataset 1'); % add figure title
                hold off;
```

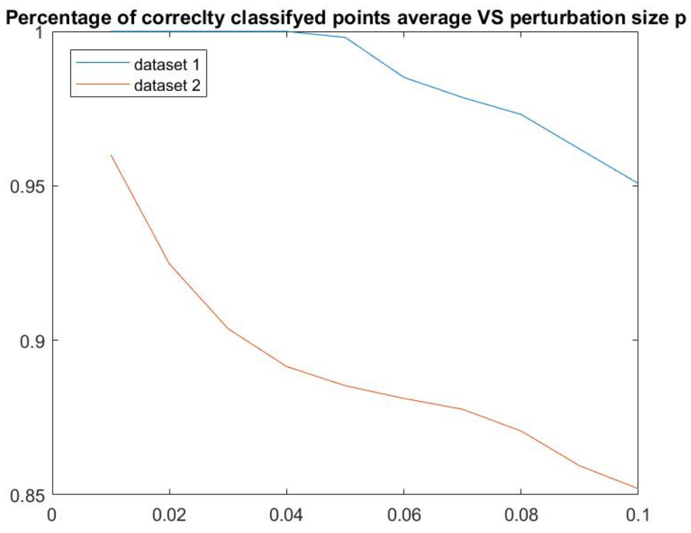
Here are the two graph for each datasets.As said before, the average percentage of classifyed points has been computed using 10000 iteration to get a more accurate graph.





3.3.1 Analyse of the results

We print the two curves on the same graph to make a comparaison :



We can see in this graph that the pertubations have stronger effects on the second dataset classification than the first. This is due to the data contained in each data set. Actually, we can see that for the second dataset, the two class points are really closer to the hyperplan than they are in the first dataset. One consequence of this difference, is that the second hyperplan is more sensitive to parameters changes where the first can handle higher perturbations. This problem can also appear on real data since we can not control how close two classes are.

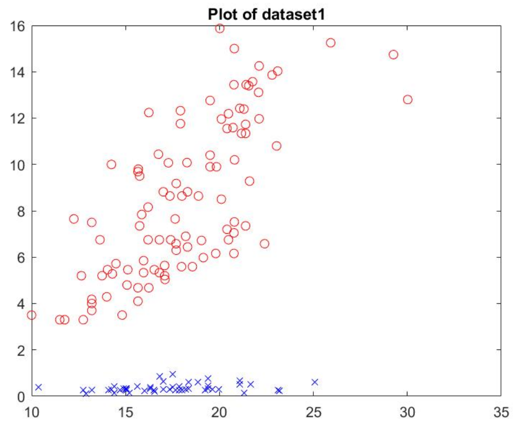
4 Simple linear classifier for Iris data, manual design

For this last task, we used the tools and methods learned before in this lab. We had to implement a linear classifier for a biger dataset (iris dataset).

The first thing we did was to load the data and plot it so we can get an idea of what it look like.

```
        datasetEx2 = load('datasetEx2.txt');
        t=datasetEx2(:,3);
        figure(1);
        plot(datasetEx2(t==1,1),datasetEx2(t==1,2),'xb');
        hold on;
        plot(datasetEx2(t==-1, 1), datasetEx2(t==-1,2),'or');
        title('Plot of dataset1') % add figure title
        hold off;
```

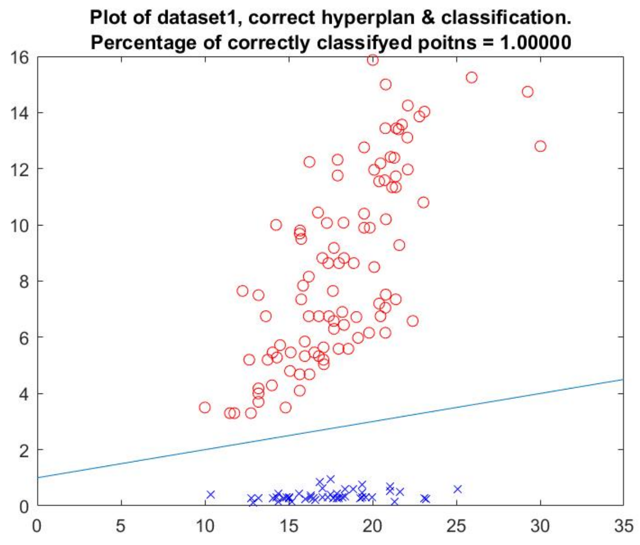
Here is the figure obtained :



We used the same methods as the previous exercice to find a correct hyperplan. Here is the code generating the graph :

```
w1 = -0.1;
w2 = 1;
w3 = -1;
w = [w1 w2 w3];

percentageOfClassifyedPoints = resultLinClass(datasetEx2, w, t);
disp('the percentage of classifyed points (dataset1, correct
    classification )is  : ');
disp(percentageOfClassifyedPoints);
%this is definitly better
figure(6);
plot(datasetEx2(t==1,1),datasetEx2(t==1,2),'xb');
hold on;
plot(datasetEx2(t==-1, 1), datasetEx2(t==-1,2),'or');
title(compose('Plot of dataset1, correct hyperplan & classification.\
    nPercentage of correctly classifyed poitns = %.5f',
    percentageOfClassifyedPoints)); % add figure title
% linspaceForVector = linspace(0,1,1);
x = linspace(0,35,3);
hyperplan = -w3/w2 -(w1*x)/w2;
plot(x, hyperplan);
hold off;
```

Here is the figure obtained :



We found a correct linear classifier for this problem. However, this graphical method only works when we have a two dimensions dataset. Other methods would be required when we get higer than three dimensions.