Statistical Methods for Machine Learning

<u>Sunspots, Seeds, and Stars</u>

<u>Exam Assignment</u>

April 2013

Evangelos Tzemis

khb107@alumni.ku.dk

I developed the source code using MATLAB

Basic Learning Algorithms

Case 1: Sunspot Prediction

<u>Question 1 (linear Regression):</u> In this part the goal was to find a mapping so as to predict the number of sunspots based on previous observations. Firstly I trained the system using the training data set. In this way I calculated the desired w0 … w6 parameters. Those parameters are making the regression linear. The output values are calculated multiplying the input vector **X** (with a column of 1's added in the beginning) with the weight vector **W**. I used the code I developed in the second assignment.

For the training and the test data I had the following RMS values.

Training: RMS = 14.07

Testing: RMS = 18.77

The six parameters of the model are:

Table 1 - Weight vector W

W0	W1	W2	W3	W4	W5
10.8443	-0.0618	0.1209	-0.0120	-0.5704	1.2845

There are expected results since we have smaller train RMS that test RMS error. It happens because we trained our system with the training data; so normally it will have less Error in those data, than in new test data. However the difference between them is not that wide so we can assume that our regression model adapts satisfactory in the test data.

Question 2 (non-linear regression):

The software I used is the **neural network** library that is included in the MATLAB. Firstly I decided to use a neural network since we had the experience and also some very promising results from the third assignment. However I used the built in software provided by MATLAB since I did not observe any difference in the results I was getting with my own-implemented neural network. I tried to run and train the network with many different parameters as various training and performing functions and number of hidden neurons.

```
% Create a Fitting Network
hiddenLayerSize = 6;
net = fitnet(hiddenLayerSize);
% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 100/100;
net.divideParam.valRatio = 0/100;
net.divideParam.testRatio = 0/100;
net.trainFcn = 'trainlm';  % Levenberg-Marquardt
net.performFcn = 'mse';  % Mean squared error
% Train the Network
[net,tr] = train(net,inputs,targets);
% Test the Network for train
Y = net(inputs);
Y = Y';
Dif = Ytrain-Y;
Dif = Dif(:).^2;
RMSTrainNonL = sqrt(sum(Dif)/size(Tr,1));
% Test the Network for test
YNonLinear = net(TestInp');
Y = YNonLinear';
Dif = Ytest-Y;
Dif = Dif(:).^2;
RMSTestNonL = sqrt(sum(Dif)/size(test,1));
```

As it is obvious from the above code, I first set the parameters of the network and then I train it using the training data and the function **train (net, inputs, targets).** After training we can use different input and get the estimated results by applying the **net** function. So we input the train data to calculate **RmsTrainNonLinear** error and then test data to calculate the **RmsTestNonLinear** error.

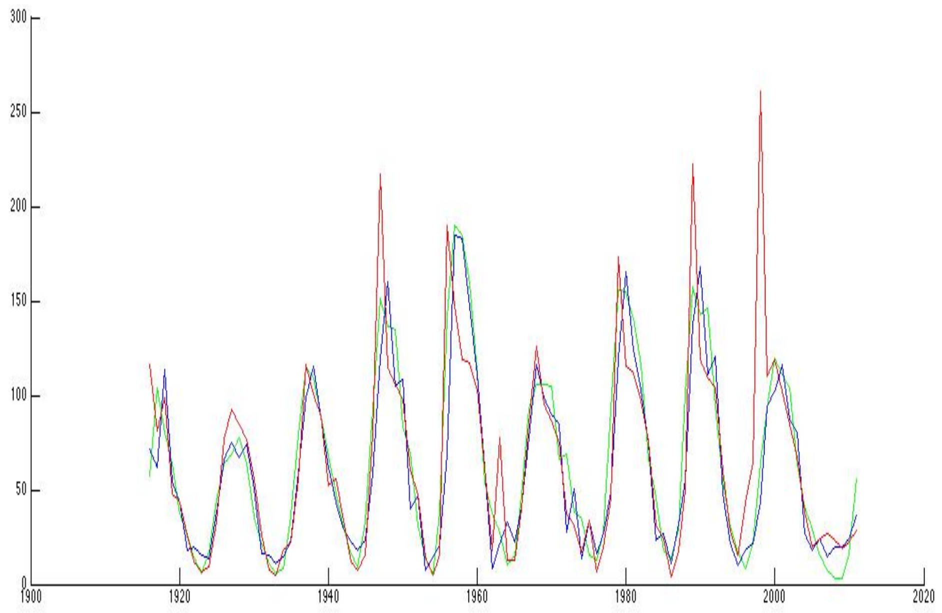
After running the code we get the following results:

trainError = 9.4703 and **testError = [18.9 - 22.3187].**

Test error varies in each run, and it does that because the initial edge weights are randomly initialized. Furthermore, we can see that we have better results in the training set than those with linear regression. However testing Error is slightly worse. No matter the parameters I tried the error got worse than improved. This is happening most likely because of the test data. They are probably more linearly distributed so the linear model can perform better on that. Sometimes a simpler model is more sufficient than a more complex one. This might be an example of this phenomenon.

Question 3 (visualization of the data):

The plot requested is the following one (years on the x-axis, average number of sunspots on the y-axis). The colors used are referring to: GREEN is for actual value, BLUE is for values predicted by the linear regression model and RED is for values predicted by the non-linear regression model.



For the above plot we can easily observe that the output of the linear regression model approaches the actual values in a very good degree. Its simplicity and linearity does not block it from outputting low RMS error results. As far as the non-linear regression's output is concerned, we have that even if it is good it is not better than the linear model. So this plot somehow confirms that test error result we explained above; and by that I mean that since, the non-linear model has higher test error it will adapt less perfectly to the actual (green) graph.

Case 2: Surveying the Sky

Question 4 (binary classification using support vector machines):

The software used is the **LIBSVM** library that we were also instructed to work with on the third assignment. The reason I chose it is on the one hand because I had the experience from the previous assignment and on the other hand cause the results it gave me are really sufficient that a different approach is not really required, or at least it wont have any impact in the results.

- Calculation of Jaakkola's parameters:

```
counter = 1;
G = zeros(803700 , 1);
for i=1:size(train,1)
   if train(i,6) == 0
       for j=1: size(train,1)
          if(train(j,6) == 1)
                G(counter) = norm(train(i,1:5) - train(j,1:5));
                counter = counter + 1;
          end
       end
   end
end
Sjakkolla = median(G)
```

From the above code we can see that instead of having two matrices with the data of the two different classes, I use two if-statements two simulate this behavior. In this the calculation is really fast and it only takes around two seconds, in comparison with other approaches that needed more than a minute.

The **Jaakkola** γ and σ values I calculated are **γ = 0.0328** and **σ = 3.9071**

After calculating the other parameters I created two vectors, one with all Costs and another with all γ values. Then I run a 5 cross-validation test with all possible pairs of C and γ and I chose the ones with the best results.

Those are: **C = 8** and **γ = 0.262033326179241**

Then I trained the svm (**svmtrain**) with those parameters and using the train data. Now that our svm is trained we are allowed to use **svmpredict** function to guess the classification results of any give input data. We did it for training and test data.

Classification accuracy on Train data = 98.6373%

Classification accuracy on Test data = 98.4763%

As we can clearly see the result are really accurate both for the test and the train data.

Question 5 (discussion on overfitting):

After reading through the web blog and understanding all its aspects I analyze how all the different types of overfitting can occur when applying machine learning techniques to the sunspot regression task and the quasar classification task. The arguments are basically focused at the latter one.

- **Traditional overfitting:** We do not have this kind of overfitting because we have enough data in comparison with problem's and the algorithm's complexity.
- **Parameter tweak overfitting:** No, because although I use 5 cross validation to peak the best parameters. I also do that to optimize train data set performance and not the test's one.
- **Brittle measure:** Knowing that all those methods are especially brittle to overfitting I chose to use 5 cross validation to avoid that behavior.
- **Bad statistics:** No. I am not forcing the algorithm in a certain way to reach standard confidence intervals. I exploit the randomness of the input data (they are not given in a specific order). I am not using any randomize methods.
- **Choice of measure:** For my measure of performance I use accuracy cause I think it fits me better and it is also requested from the assignment. I am not trying to pick up one that will give the best results. However apart from accuracy there are also many other measures of performance. One of the best is rock curve, in which the area under the curve shows exactly the algorithms performance.
- **Incomplete Prediction:** since we have only two groups we do not have this kind of overfitting / problem.
- **Human-loop overfitting:** I do not do that since test examples are not available to me. Furthermore I do not modify in any way the test data.
- **Data set selection:** I did not choose my data depending on their accuracy. I use all of them no matter what, presenting the respective accuracy results.
- **Reproblemіng:** No I do not alter anything neither in data nor in the algorithm altering the results.
- **Old datasets:** I have only one data set so in my opinion this kind of overfitting does not occur to my occasion. In case of more than one test data then I should have given priority to the newer ones.

Case 3: Wheat Seeds

Question 6 (principal component analysis):

Description of software used: In this part I used the MATLAB built-in function princomp in the following format: **[COEFF,SCORE,e] = princomp(X).** This function performs principal component analysis to the input data. Firstly I normalized my train data, by subtracting the mean value and dividing by the covariance, and then I inputted only the first seven columns (so without the categorization) to the function. Princomp after performing principal component analysis returns to COEFF a seven by seven matric each column containing coefficients for one principal component. The columns are in order of decreasing component variance.

SCORE is basically the representation of input data in the principal component space. So I did not have to multiply my input data with the first two columns of COEFF to get the new 2D data.

Last but not least 'e' returns all 7 eigenvalues. By plotting the vector e we get the following graph.

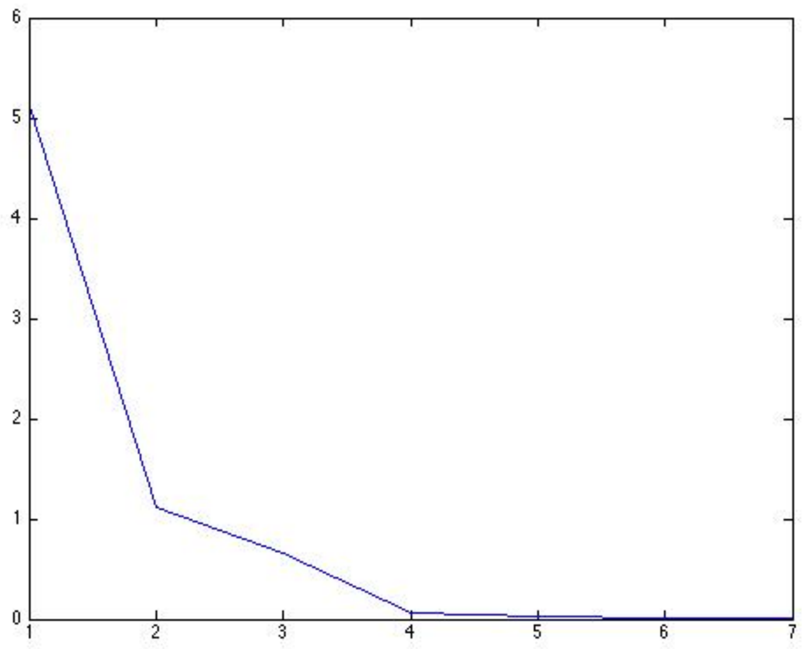


Figure 1 - plot of the eigenspectrum

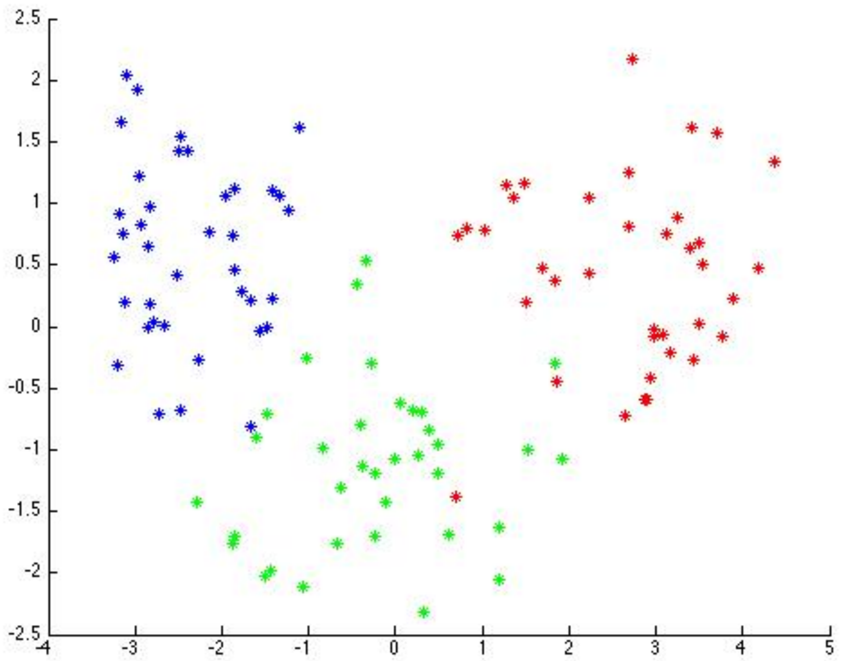


Figure 2-plot of the data projected on the first two principal components. Green for Group 0 Read for Group 1 and Blue for Group 2

Question 7 (clustering):

Description of software used: In this part I used the kmeans MATLAB built-in function.

Kmeans takes as input the data and also the number of teams that we want them to be seperated. It returns an n-by-1 vector IDX containing the cluster indices of each point. By default, kmeans uses squared Euclidean distances. It also returns the centroid locations in a second 3*2 matrix. So the cluster centers that were calculated by the function are presenting in the next table:

Table 2 - Cluster centers

Center	X	Y
1	-2.37288	0.65692
2	-0.41734	-1.163655
3	2.615037	0.391235

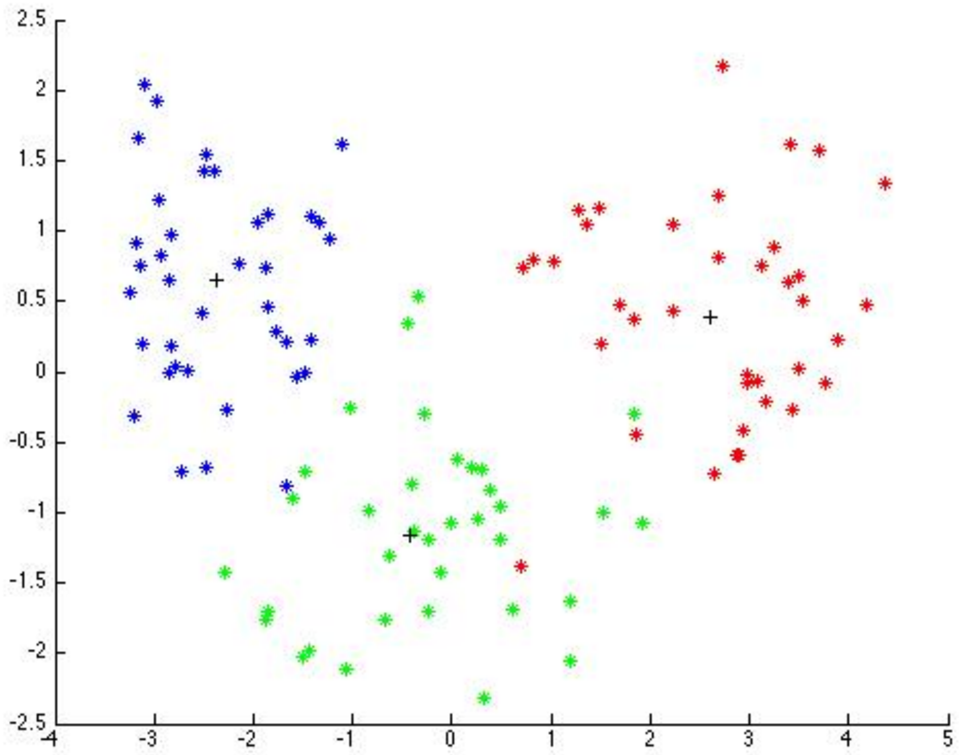


Figure 3 - plot of the data projected on the first two principal components, including the cluster centers

Furthermore in the above graph we can see the cluster centers plotted alongside with the data. From the data distribution we can realize that the results are quite meaningful since the cluster center are more or less in the center of each team.

<u>**Question 8 (multi-class classification):**</u>

For both linear and non-linear Classification I used the methods I have also included and tested in my second assignment.

Linear Classification: In this part I did not use a built in package but I implemented my own LDA.

For LDA I have **test error** = 6% and **train error** = 0,91%.

Non-Linear Classification: In this part I chose to utilize the **knnclassify** function of MATLAB. I also implement my own code for KNN classification but since I was getting the same results as the knnclassify I preferred to submit with the built-in one. This way the code is much more readable. The way I proceeded is by calculating the error for 1,3,5,7 neighbors for train and test data, and I chose the one (maybe different in test and train data) with the less error.

So for KNN I have **test error = 13%** and **train error = 0%** for 1 neighbor. If you think this is not fare then for more neighbors I have **train error = 1.82%.**