Our first classifier

Introduction to Machine Learning

University of Barcelona

December 6, 2015

Authors:
Kazancli, Erol
Leyva, María

Contents

1 Understanding and preprocessing our problem.

2 Data set analysis

2.1 The dataset

Cardinality: 768
Dimensionality: 8
Mean value y: -0.3021
Mean value x: 4.4947 121.6868 72.4052 29.1534 155.5482 32.4575 0.4719 33.2409
We have NaN values, in order to calculate this we have just ignored them using matlab function *nanmean*.

2.2 Dealing with NaN values

We have created three datasets dealing with missing data in different ways:

D_1: Replacing NaN for the average of that attribute

D_2: Replacing NaN for the average of that attribute for that class

D_3: Replacing NaN for the median of that attribute[1] for that class

For this three datasets we have calculated the mean of each attribute.

DataSet	x1	x2	x3	x4	x5	x6	x7	x8
D1	4.4947	121.6868	72.4052	29.1534	155.5482	32.4575	0.4719	33.2409
D2	4.4927	121.6974	72.4281	29.2470	157.0035	32.4464	0.4719	33.2409
D3	4.3776	121.6771	72.3893	29.0898	141.7539	32.4346	0.4719	33.2409

Table 1: Different datasets generated dealing differently with missing values.

3 A simple classifier

3.1 A thresholded regressor

3.1.1 Learning the threshold value.

We are using linear regression, therefore when calculating the weights, the threshold would correspond to -w_0, having an hyperplane looking like this, as we have 8 features in our dataset:

$$f(x) = w_0 + w_1 * x_1 + w_2 * x_2 + w_3 * x_3 + w_4 * x_4 + w_5 * x_5 + w_6 * x_6 + w_7 * x_7 + w_8 * x_8$$

Or, which is the same:

$$\sum_{i=1}^{n}(w_i * x_i) + w_0$$

So our classifier $h(x)$ would be:

$$h(x) = \begin{cases} \text{if } \sum_{i=1}^{n}(w_i * x_i) \geq -w_0, & \text{then class } = 1 \\ \text{else if } \sum_{i=1}^{n}(w_i * x_i) \leq -w_0, & \text{then class } = -1 \end{cases}$$

Or:

$$h(x) = \begin{cases} \text{if } \sum_{i=1}^{n}(w_i * x_i) + w_0 \geq 0, & \text{then class } = 1 \\ \text{else if } \sum_{i=1}^{n}(w_i * x_i) + w_0 \leq 0, & \text{then class } = -1 \end{cases}$$

Therefore:

$$h(x) = sign(\sum_{i=1}^{n}(w_i * x_i) + w_0) = sign(w^T * \widetilde{x})$$

3.1.2 The separating hyperplanes for D_1, D_2, and D_3.

D1

$$normalVector = [0.0455, 0.0129, -0.0028, 0.0003, -0.0002, 0.0274, 0.2510, 0.0050]$$

$$threshold = 3.0278$$

D2

$$normalVector = [0.0557, 0.0099, -0.0015, 0.0104, 0.0019, 0.0161, 0.2295, 0.0018]$$

$$threshold = 2.9337$$

D3

$$normalVector = [0.0597, 0.0104, -0.0015, 0.0092, -0.0015, 0.0176, 0.2248, 0.0017]$$

$$threshold = 2.9354$$

3.1.3 Error rates in training data.

To compare the accuracies of the three data sets with their corresponding regression models, we calculated the ratio of the misclassified data points to the whole data set. The following figures are the numbers found.

	Error rate
D1	0.2214
D2	0.2018
D3	0.2096

Table 2: Error rates for each data set

Comparing the data sets, we can see there is no significant difference, although D2 seems to get sightly better results. This is due to the fact that the NaN replacement method used has given extra information to the data about the class it belongs, which led to a slightly better classification. D3 also performs quite well, because similar to D2 we use this time the median considering the class information, which gives some information to the data about its class.

Method used

The method used is the linear analytical regression with threshold. We have listed in the previous section the corresponding normal vectors and the threshold for the models found for each data set. For the error measure, we calculated the ratio of the misclassified data points to whole data set for each data set with the corresponding regression model.

	Error rate
Training	0.1971
Test	0.2000

Table 3: Training vs test error using D2

3.2 Using test data

Now we are going to calculate D_2 and then we are going to split it into training and test data (4/5 and 1/5, respectively) in order to have data to test our regressor.

We will calculate the thresholded regressor using the training data, and then we calculate the error measures for both sets. The numbers can be seen in the table above:

As can be seen from the table above the training and the test errors is more or less the same as the previous error rate found using the D2 model. This is because we used the same NaN replacement method in both cases. Note that the test error rate is higher than the training error, although by a small amount. This can be explained by the model being more tuned to the characteristics of the training model.

Now we are going to split the data into training and test data in the same proportion as before and apply D_2 method again. This time we are going to do something different. First we will split the data and then we will replace the NaN values with the means found in the training set.

For the training data we will replace the NaN values in the training set by calculating the mean values in this set taking advantage of the class knowledge. We will also use these mean values found in the training set to replace the NaN values in the test data. Normally, we may not be able to do this because it may no be possible to have the class values for the test data. But in this case, since we have the corresponding classes for the test data, we can use this knowledge.

Once we have done this, we train and test and compare the error rates. As we can see, there is no remarkable difference although we have biased the data.

	Error rates
Training	0.1954
Test	0.2065

Table 4: Training vs test error using modified D2

As can be seen from the table above the training error improved slightly. This can be explained by giving it information only about itself, not about the whole set. This pushed the result in the direction of overfitting. Although we have provided some information to the test data set by replacing the NaN values with the means found in the training set, because of biasing the model towards the training set, the error rate for the test set worsened slightly.

3.3 Changing the percentage of testing data

Now we will use different percentages for the training and the test data and compare the error rates found. We assume that a better practice is using a bigger training set than the test set, therefore we test the percentage values 50 or higher for the training set. The rest of the data will be our test data.

It is expected that the training error rate decreases with an increasing sample size and this is what we observe. However, there is one thing which is difficult to explain. The test error rate increases at first and only after a certain percentage point starts to decrease. We normally expect overfitting with lower sample size, therefore test error rates tend to decrease with increasing sample size. This is not what we observe in the beginning, however this might be a random result which might have been caused by the specific NaN replacement method used, which biases the test as well, or the big test sample size. After the 80 percentage point we see the test error decrease as well, which points to a better generalization thanks to the training sample size.

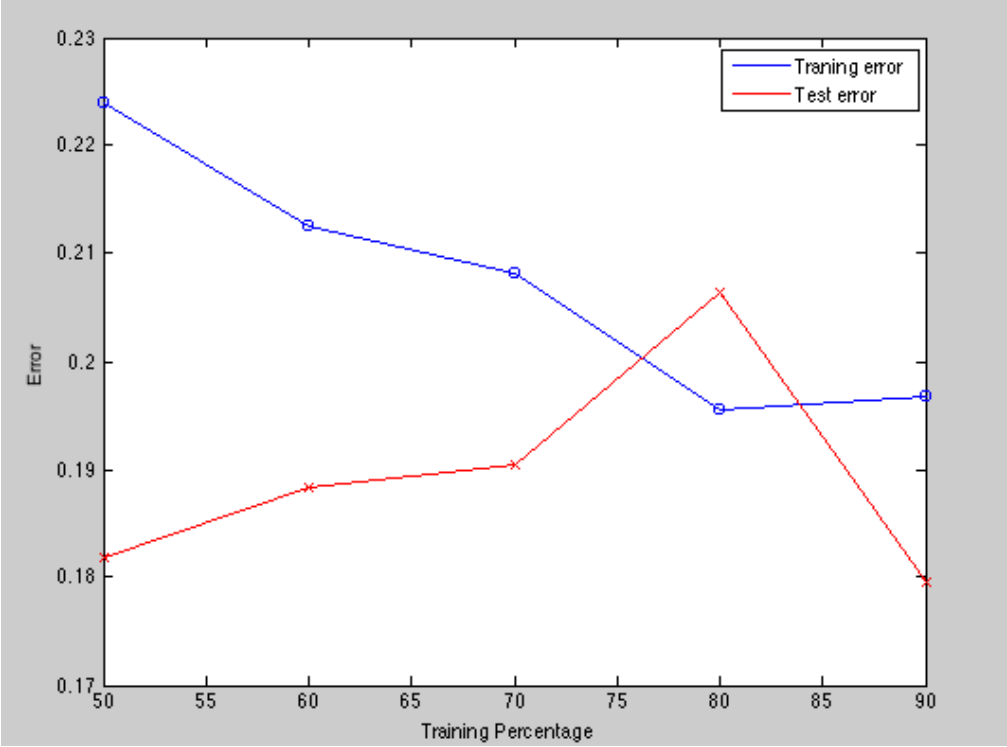

Figure 1: Training and Test error rates vs. Training Percentage

3.4 Upper bound

$$E_{\text{out}}(h) \leq E_{\text{in}}(h) + \sqrt{\frac{d_{\text{VC}}(log(2n/d_{\text{VC}}) + log(2/\delta)}{2n}}$$

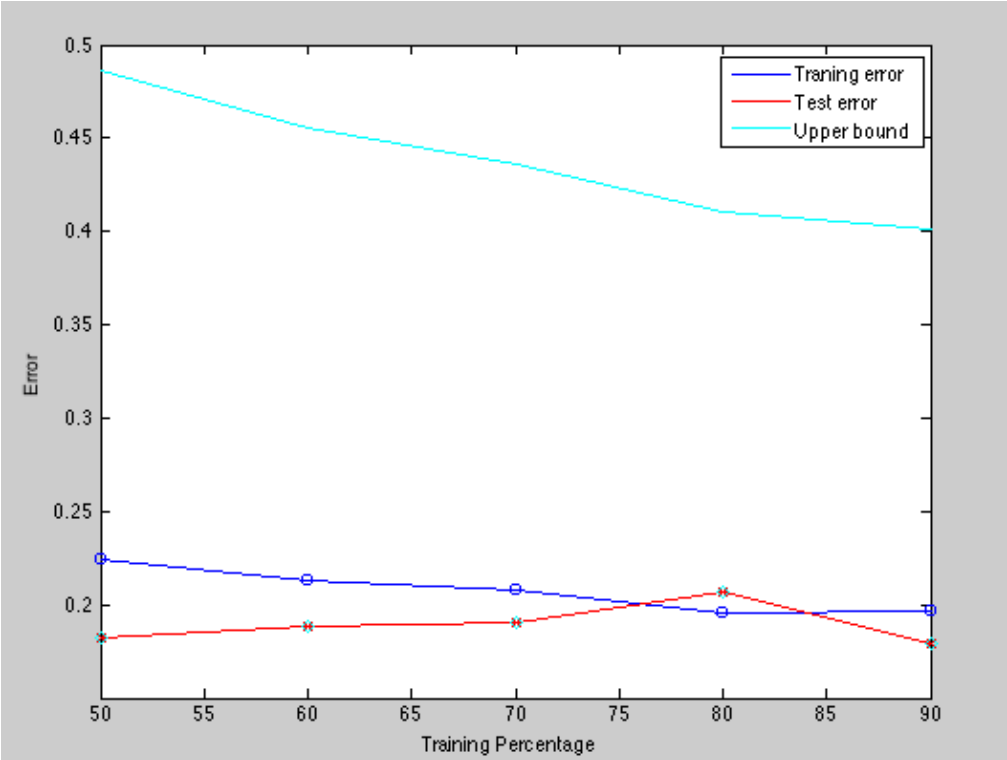

Figure 2: Training, Test and Bound Error rates vs. Training Percentage

As we can see in the graphic, the upper bound decreases as the amount of data used for training increases.

Now we add the upper bound error line for the test data to the graph, which is found by the above equation using dVC as d + 1 (dVC = 9 in our case) and a 95 percent confidence level(delta = 0.05). For all the training and test data the error rates are below the upper bound so it makes it an acceptable classifier. But note that because of the small sample size and high complexity(hinted at by high number of dimensions) for each case, we have a high potential error variance for any test data given.

Using the same equation we find the sample size predicted for different confidence levels and error deviations as follows. (using the 'solve' function in matlab)

Confidence	Error deviation	Optimal N
0.95	0.01	594021
0.5	0.01	581554
0.95	0.05	17407
0.95	0.10	3649

Table 5: Optimal n

As we can see in the attached table, the confidence factor doesn't seem to have a significant influence in the amount of data we need. As the confidence level decreases, the sample size needed slightly decreases, which is expected because the condition on the error has relaxed(we are less confident that the error will be within the specified boundaries). The error deviation, seems to influence much more. When we increase the error deviation the data needed lowers significantly, which is logical if we take into account we're allowing a much bigger error.

References

[1] Edgar Acuña and Caroline Rodriguez, The treatment of missing values and its effect in the classifier accuracy, *Classification, Clustering and Data Mining Applications (2004)*, pp. 639-648