Exercise 2: Our first classifier

Introduction to Machine Learning

University of Barcelona

December 1, 2015

Authors:
Camps, Julià
Serra, Xavier

Contents

1 Question Block 1

1 *Which is the cardinality (number of examples) of the training set?*

Number of instances: 768

2 *Which is the dimensionality of the training set?*

Number of attributes: 8 (it does not include the class attribute)

3 *Which is the mean value of the training set?*

Means: **NaN**, **NaN**, **NaN**, **NaN**, **NaN**, **NaN**, 0.4719, 33.2409

2 Question Block 2

1 *Create a new dataset D1, replacing the NaN values with the mean value of the corresponding attribute without considering the missing values.*

After we previously tried to calculate the means of the dataset, some of the mean values were NaN[1], this was due to the fact that, in the used dataset, some of the instances don't have all their attributes defined.

So we have to guess the missing values in order to be able to process these instances. To do so, the first strategy we are asked to use is replacing NaNs by the mean of all the values that this attribute takes in the rest of instances. After applying it we obtain a new dataset that in our Matlab code we named "D_1".

In order to explain the function: let X be the original matrix of instances without labels, let Y be the labels[2] matrix and let $newX$ be the instances matrix with the missing values converted to the mean value.

The function that we implemented to replace the NaN values was:

```
newX = replaceNaNbyMean(X)
D1 = struct('x',newX,'y',Y);
```

2 *Create a new dataset D2, replacing the NaN values with the mean value of the corresponding attribute without considering the missing values conditioned to the class they belong, i.e. replace the missing attribute values of class +1 with the mean of that attribute of the examples of class +1, and the same for the other class.*

Here we start from the same point that in the previous question, but in this case we are asked to implement a different strategy for solving the NaN values problem. In this case, in order to compute the mean we will only consider those instances with the same label, as the instances with the same label are more likely to have a similar attribute values than others with different labels.

The function that we implemented for replacing the NaN values was $D_2.x = replaceNaNbyMeanOfClass(D_2.x, D_2.y)$. As we can observe from the function, the main difference between this and the one presented in the previous task, is that in this last we need to know the labels of the instances in order to compute the means.

[1]Nan: Matlab notation for referring to "Not a Number", which means that is a missing value.

[2]Label: is the class value, in our code "y", or, from the point of view of the dataset, "dataset.y".

3 *Which are the new mean values of each dataset?* We obtained the following the mean values of the resulting datasets for both strategies:

- D_1
  - $D_1(1) = 4,4947$
  - $D_1(2) = 121,6868$
  - $D_1(3) = 72,4052$
  - $D_1(4) = 29,1534$
  - $D_1(5) = 155,5482$
  - $D_1(6) = 32,4575$
  - $D_1(7) = 0,4719$
  - $D_1(8) = 33,2409$
- D_2
  - $D_2(1) = 4,4927$
  - $D_2(2) = 121,6974$
  - $D_2(3) = 72,4281$
  - $D_2(4) = 29,2470$
  - $D_2(5) = 157,0035$
  - $D_2(6) = 32,4464$
  - $D_2(7) = 0,4719$
  - $D_2(8) = 33,2409$

3 Question Block 3

1. *In this model you have to learn the threshold value. Explain how you can accommodate this parameter.*

   We have set a *ones* row to the x set, in order to generate an extra weight which can be interpreted as an offset. Therefore, our new y will be obtained by:

   $$y = w^T x$$

   , with the w_0 being multiplied by this *ones* row and being simply summed as an offset. Therefore, this w_0 becomes the threshold of the hyperplane.

   The threshold we have obtained for $D1$ is: $w_0 = -3.0278$

   The threshold we have obtained for $D2$ is: $w_0 = -2.9337$

2. *Report the normal vector of the separating hyperplane for each data set $D1$, $D2$, $D3$.*

   The normal vector is obtained by the rest of components of the w vector, that is, $\{w_1, ..., w_8\}$.

   The normal vector we have obtained for $D1$ is:

   $$D1_n = \{0.0455, 0.0129, -0.0028, 0.0003, -0.0002, 0.0274, 0.2510, 0.0050\}$$

   The normal vector we have obtained for $D2$ is:

   $$D2_n = \{0.0557, 0.0099, -0.0015, 0.0104, 0.0019, 0.0161, 0.2295, 0.0018\}$$

3. *Compute the error rates achieved on the training data. Are there significant differences? Report the method used and their parameters.*

   - The error obtained in $D1$ is: 0.2214
   - The error obtained in $D2$ is: 0.2018

   Objectively, we have a 10% difference between both errors. This means that, just by modifying slightly the way we pre-treat the data, we can obtain a reasonable improvement. Therefore, we conclude that, when faced with a new Machine Learning problem, before applying any method, we should use some time imagining how we could process the data.

   The method used in this *Question block* is the analytical solution for linear regression obtained in *Question block 1*, therefore, it does not need any parameter. An interesting test would be to use the iterative solution instead of the analytical one, and see the results with it. Judging from the results obtained in the two previous blocks, we do not expect its behaviour to be much different to this one.

4 Question Block 4

1. *Repeat the learning process in block 3 using just D2 but holding-out the last fifth of the data set for testing purposes, i.e. use the first 4/5-th for training and the last 1/5-th for testing. Follow exactly the following steps in your process:*

   a) *Clear your workspace and close all figures: clear all, close all, clc.*

   For this step we just call the requested Matlab functions.

   b) *Preprocess the data replacing the NaN using the method for creating D2.*

   For this step we used the function $D_2.x = replaceNaNbyMeanOfClass(D_2.x, D_2.y)$ developed for the tasks in block2.

   c) *Split your data in two sets: the first 4/5-th is to be used for training and the last 1/5-th will be used for testing purposes.*

   For doing this step first we just calculated the number of total instances of the dataset and split it at

   $$\frac{4\#Instances}{5}$$

   so that we obtained two new datasets: D_{Train} and D_{Test}.

   d) *Train your model on the training set.*

   In this activity we will use again the analytical regression algorithm $W = (XX^T)^{-1}XY$, as in the previous assignment.

   After training the regression model with the training set we obtain a 8 dimensions hyperplane. For a more detailed explanation, let x be an instance of our dataset, so that $x = [at_1 at_2 at_3 at_4 at_5 at_6 at_7 at_8]$. Then we can define the hyperplane as a function of x, so that $f(x) = W^T x$.
   Finally the for defining the hyperplane as the expanded function of an instance from our dataset x we can see it as

   $$f(x) = w_0 + w_1 at_1 + w_2 at_2 + w_3 at_3 + w_4 at_4 + w_5 at_5 + w_6 at_6 + w_7 at_7 + w_8 at_8$$

   where we find an offset w_0 added to the attributes weighted each one by a weight value.

   e) *Answer the following questions: Which is the error rate on your training data? Which is the error rate on your test data? Are they similar? Did you expect that behavior? Why?*

   - Error rate:

- In training data: 0.1984
- In testing data: 0.2026

- As we can observe they are very similar, with a difference of 0.0042. This result is not the one we were expecting, as we thought there would be a greater difference. The reason for this is that is that the weights are set in order to fit the data in the training set, so it is reasonable to think that it will achieve better results there than in another set. However, a possible explanation would be that the data in the train set is slightly more difficult to classify than the one in the test set, and thus the small difference in the errors of both of them. Therefore, this leads the regression method to fail when trying to estimate the label for an instance, even if this one was used for the regression hyperplane definition.

  In order to visualize better how it's behaving, in tables 1 and 2 we can see the confusion matrix of both subsets.

134	42
80	359

Table 1: Confusion matrix the training dataset with analytical regression.

33	10
21	89

Table 2: Confusion matrix the testing dataset with analytical regression.

5 Question Block 5

1. *Repeat the process in block 4 changing the order of some of the steps. Follow exactly the following steps in your process:*

   a) *Clear your workspace and close all figures: clear all, close all, clc.*

   For this step we just call the requested Matlab functions.

   b) *Split your data in two sets: the first 4/5-th is to be used for training and the last 1/5-th will be used for testing purposes.*

   We simply calculate the number of total instances of the dataset and split it at

   $$\frac{4\#Instances}{5}$$

   so that we obtain two new datasets: D_{Train} and D_{Test}. But, with an important difference compared to the previous block, because in this step, this time, the NaN values haven't been treated yet. So we might have NaN values in the $D_{Train}.x$ matrix and also we might have them in the $D_{Test}.x$ matrix.

   c) *Preprocess the data replacing the NaN using the method for creating D2. But this time use only the data corresponding to the training set.*

   In this step we have to process the NaN values of the D_{Train} dataset only, and leave, by the moment, the D_{Test} dataset apart. For this step we apply the function implemented for block2 $X = replaceNaNbyMeanOfClass(X, Y)$, being $X \equiv D_{Train}.x$ and $Y \equiv D_{Train}.y$, as we have just commented.

   d) *Train your model on the training set.*

   In the previous block we used D_{Train} and also the D_{Test} datasets values for this NaN replacements (before splitting them), but the testing instances in D_{Test} were not used for the training process. Now we train the model with the D_{Train} dataset. This time we thought that it would fit better with the regression model obtained, because for the missing values we only took into account those instances that were going to be used for the training process.
   From this we expect to have a higher accuracy for the training set.

   e) *Replace the NaN values using the means computed on the training data.*

For this step we implemented a new Matlab function for replacing the NaN values in the D_{Test} dataset. In this case, however, we replace them by the mean of the values in D_{Train}, instead of the ones in D_{Test}. Therefore, this function needs to recover the values used for the D_{Train} dataset, so it needs the D_{Train} dataset components in order to retrieve the mean of those instances matching their label with the missing values in the D_{Test} dataset.
The function that we implemented was:

$$D_{Test}.x = replaceNaNbyMeanOfClassTrain(D_{Test}.x, D_{Test}.y, D_{Train}.x, D_{Train}.y)$$

As we have mentioned, we can observe from the function that main difference between this and the one presented in the previous block, is that in this last one we need to know the instances and labels of the the D_{Train} dataset in order to compute the means.

f) *Answer the following questions: Which is the error rate on your training data? Which is the error rate on your test data? Are they similar? Did you expect that behavior? Why?*

- Error rate:
  - In training data: 0.1967
  - In testing data: 0.2026
- As we can observe they are very similar the difference is about 0.0059.
  Although, that the difference is greater than in the previous block, we extract the same conclusions, and this time, were expecting a greater difference than the one obtained, again. The reason for this expectations was that, this time we fitted much more the model on the training set, but the difference, even though it increased as we predicted, remains insignificantly small and the increment was of a single instance correctly classified, as we can see on the confusion matrix of the training data.

  In order to visualize better how it's behaving in tables 3 and 4 we can see the confusion matrix of both subsets.

134	41
80	360

Table 3: Confusion matrix the training dataset with analytical regression.

33	10
21	89

Table 4: Confusion matrix the testing dataset with analytical regression.

g) *Compare these results with the ones in block 4. Do we achieve better or worse results? Why?*

As we have mentioned on the previous task, this time we have fixed the classification of one of the mistaken elements from the training set.
Actually, we implemented a short comparison code, in order to compare the results of both executions. By doing this, we have realized that the only difference between them is that the instance 45 of the training dataset (that corresponds to the instance 45 from the initial dataset as well) is misclassified when we run the regression with NaN treatment of **Block 4**, but it's correctly classified when we do it in **Block 5**.
All in all, we achieve better results that in **Block 4**, but we thought that the difference would be greater than just one instance in the training set, from the 122 that were misclassified initially.

6 Question Block 6

1. *Repeat the process in block 5 changing the percentage of the data for training and testing. Plot a graph with the training and test error rates for each splitting percentage point. Comment the results*

   In Figure 1 we have both the training and the test errors for different percentages of training.

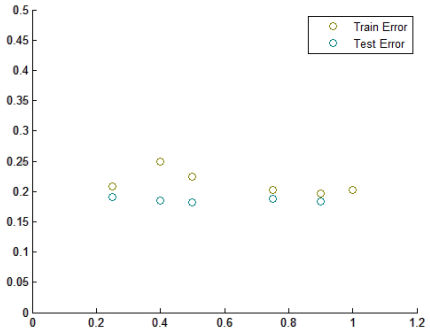

Figure 1: Graph of th evolution of train and test errors

Concretely, the values obtained are shown in Table 5

Percentage for training	**Train error**	**Test error**
0.25	0.2083	0.1919
0.4	0.2500	0.1848
0.5	0.2240	0.1823
0.75	0.2031	0.1875
0.90	0.1965	0.1842
1	0.2018	-

Table 5: Evolution of train and test errors

Surprisingly enough, we can see that the train error is higher than the test one. This is not the normal behaviour, as, normally, it happens the other way around. As already explained, one possible explanation would be that the last part of the data, that we always take as the test set, would be easier to classify than its beginning. In order to prove so, we have decided to take the samples randomly, instead of retrieving the first part as training, and the result is shown below:

Actually, the results obtained this second time behave more similarly to the expected, even though it does not follow exactly the most common behaviour. In this case, the train error is generally lower than

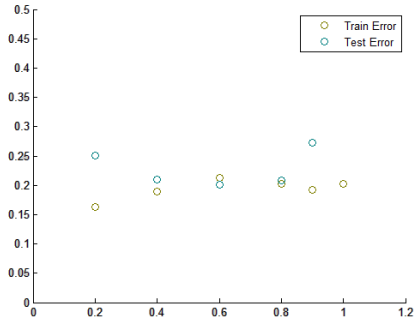

Figure 2: Random data split

the test error (although not always), and it tends to increase as more data is used in the training. On the other hand, the test error has a slight tendency to decrease as less data is used in training (despite a sudden increase at the end). However, as the data split is done randomly, there are still weird behaviours, as the one shown in Figure 3. Further research should be done in order to better understand this behaviour, as it seems the data has some kind of special distribution that makes it difficult to model.

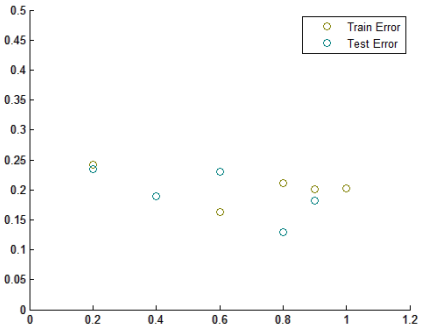

Figure 3: Second random split

2. *Add to the plot the upper bound on the generalization error using the equation of the slides for VC dimension equal to $d + 1$. Discuss the result.*

   We have used the following formula:

$$E_{out} \leq E_{in} + \sqrt{\frac{d_{VC} \times (\log(2n/d_{VC}) + 1) + \log(2/\delta)}{2n}} \qquad (1)$$

We have decided to perform this bound with a confidence of 0.95. This value is broadly used, so we found it a good choice. In this case, we directly provide two plots:

- In the first one we have used the beginning of the dataset as the train set, as we are asked to do
- In the second one, due to the results obtained in the previous section, we have decided to show too the results of the random split of the data.

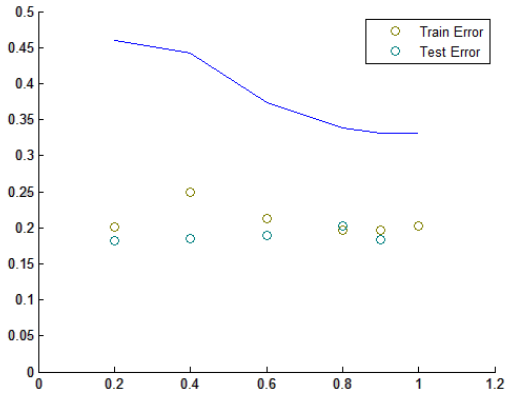

Figure 4: Original bound error

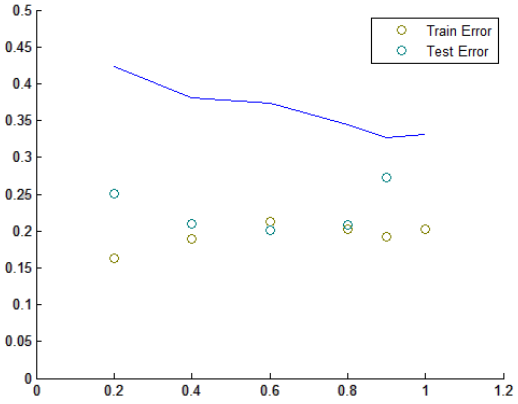

Figure 5: Randomly splitted bound error

As we can see, in both cases the upper bound is well above the test error, so it is working as it should. We can also see a tendency that, the more data we have in the training set, the more accurate the bound becomes. Therefore, we could conclude that using a lot of

data is likely to make our predictions more reliable, or, at least, to **guarantee** a lower maximum error.

3. *How many samples does the bound predict in order to have* 1% *error deviation with a confidence of* 95%*? And with confidence* 50%*? What about* 5% *and* 10% *error deviation with* 95% *confidence? Comment the behavior according to your observations*

   This time we have also used equation 1 in order to obtain the number of samples. However, as the analytical solution required to isolate the x, and we were unable to do so, we decided to use an iterative way. So, in order to find the number of samples for a given deviation error and confidence, we do the following:

   a) We start with number of samples equal to 50
   b) We compute the value of equation 1 for that number of samples
   c) If the obtained value is smaller than the deviation error we are looking for, we stop
   d) Otherwise, we increment the number of samples in 50, and go back to (2)

   So, finally, we will obtain an approximation to the number of samples required in order to obtain the asked deviation error. Using this method, we have obtained the following results:

Deviation error	**Confidence**	**Number of samples**
0.01	0.95	6800
0.01	0.5	100
0.05	0.95	6400
0.1	0.95	6250

Table 6: Number of samples required

The results of this table are straightforward:

- The lower the deviation error the less samples required, although it does not vary greatly (dividing into 10 the deviation error reduces only in 10% the required samples
- The lower the confidence required the less samples required. In this case, dividing into 2 (approximately) the asked confidence results in 60 times less samples.