NEC - Exercise 2

Comparison of SVM with BP and Linear Regression

Carlos García

January 2017

Contents

Exercise 2: Comparison of SVM with BP and Linear Regression

Objective

Compare the classification results with the following algorithms:

- Support Vector Machines (SVM), using free software
- Back-Propagation (BP), implemented by the student
- Multiple Linear Regression (MLR), using free software

1 Main concepts

As Backpropagation and MLR concepts were already included in first exercise, I'm here just including a brief summary about what the Support Vector Machines are.

1.1 Support Vector Machines (SVM)

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples [1].

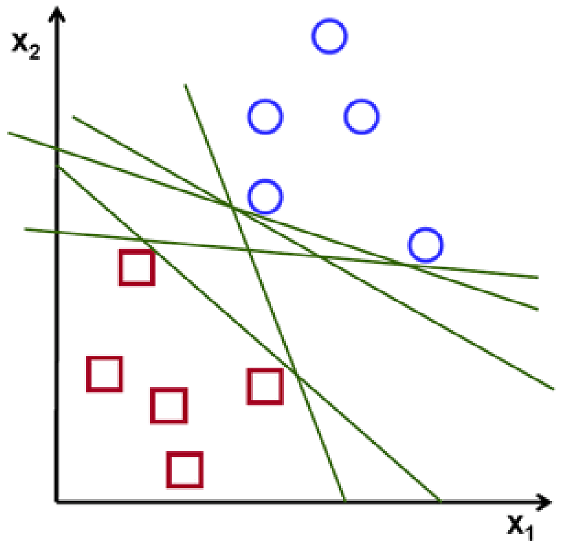
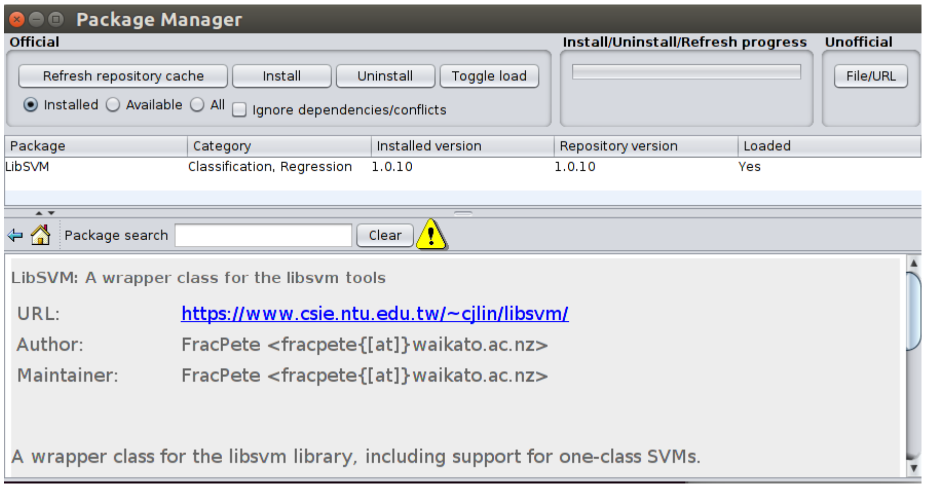


Figure 1: SVM representation [1]

More formally, a support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier [2].

2 Support Vector Machines (SVM), using free software

For this test I'm using again the free software Weka, we first must install the SVM package called LibSVM (Figure 2):



Package	Category	Installed version	Repository version	Loaded
LibSVM	Classification, Regression	1.0.10	1.0.10	Yes

LibSVM: A wrapper class for the libsvm tools

URL: https://www.csie.ntu.edu.tw/~cjlin/libsvm/

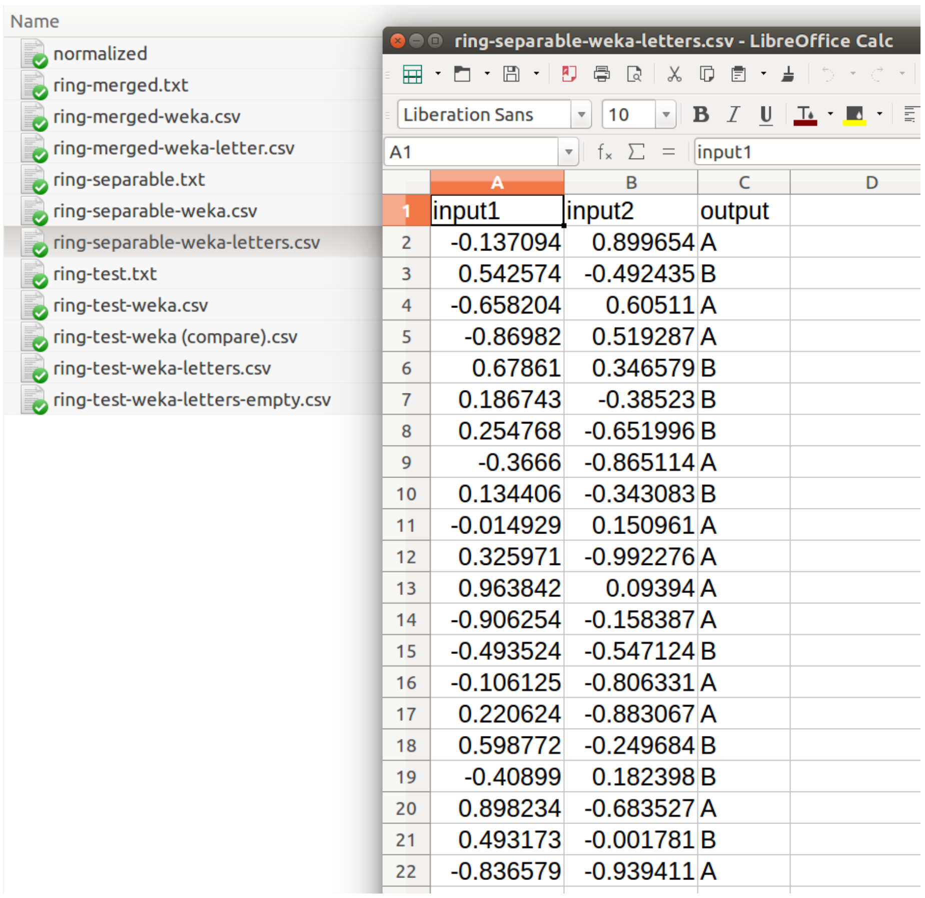
Author: FracPete <fracpete{[at]}waikato.ac.nz>

Maintainer: FracPete <fracpete{[at]}waikato.ac.nz>

A wrapper class for the libsvm library, including support for one-class SVMs.

Figure 2: Install LibSVM package in Weka.

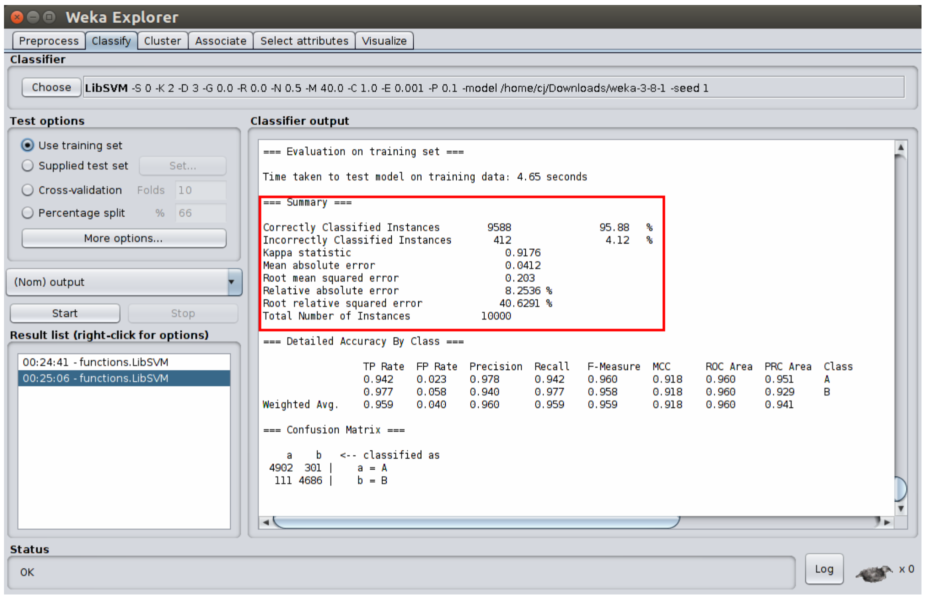
As this package cannot use numbers as classes, we must create a csv file with the data and change the last column for using A and B instead of 0 and 1 (Figure 3).


Name
normalized
ring-merged.txt
ring-merged-weka.csv
ring-merged-weka-letter.csv
ring-separable.txt
ring-separable-weka.csv
ring-separable-weka-letters.csv
ring-test.txt
ring-test-weka.csv
ring-test-weka (compare).csv
ring-test-weka-letters.csv
ring-test-weka-letters-empty.csv
ring-separable-weka-letters.csv - LibreOffice Calc
Liberation Sans
10
A1
input1

	A	B	C	D
1	input1	input2	output	
2	-0.137094	0.899654	A	
3	0.542574	-0.492435	B	
4	-0.658204	0.60511	A	
5	-0.86982	0.519287	A	
6	0.67861	0.346579	B	
7	0.186743	-0.38523	B	
8	0.254768	-0.651996	B	
9	-0.3666	-0.865114	A	
10	0.134406	-0.343083	B	
11	-0.014929	0.150961	A	
12	0.325971	-0.992276	A	
13	0.963842	0.09394	A	
14	-0.906254	-0.158387	A	
15	-0.493524	-0.547124	B	
16	-0.106125	-0.806331	A	
17	0.220624	-0.883067	A	
18	0.598772	-0.249684	B	
19	-0.40899	0.182398	B	
20	0.898234	-0.683527	A	
21	0.493173	-0.001781	B	
22	-0.836579	-0.939411	A	

Figure 3: Change class column of data from numbers to letters.

Then we import the “ring-separable-weka-letters” file on Weka explorer and after that we go to “Classify” tab, , try learning only using training set (Figure 4), and then in “Supplied test set” we can include the file “ring-test-weka-letters.csv” (Figure 5), then we press the start button to begin the training (Figure 6).



Classifier: LibSVM -S 0 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -model /home/cj/Downloads/weka-3-8-1 -seed 1

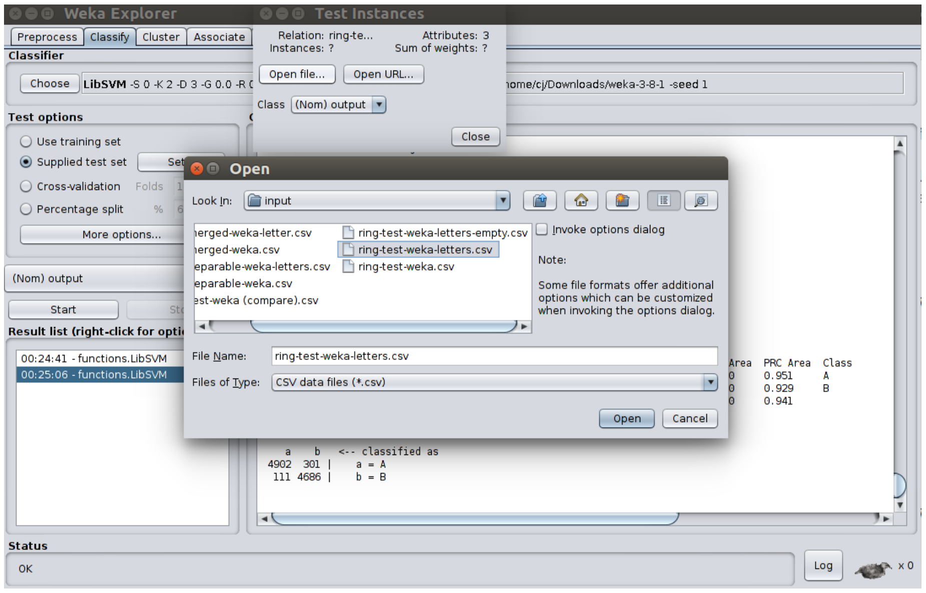
Test options: (● Use training set)

Result list (right-click for options):
00:24:41 - functions.LibSVM
00:25:06 - functions.LibSVM

```
=== Evaluation on training set ===

Time taken to test model on training data: 4.65 seconds

=== Summary ===

Correctly Classified Instances        9588               95.88   %
Incorrectly Classified Instances       412                4.12   %
Kappa statistic                          0.9176
Mean absolute error                      0.0412
Root mean squared error                  0.203
Relative absolute error                  8.2536 %
Root relative squared error             40.6291 %
Total Number of Instances            10000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.942    0.023    0.978      0.942    0.960      0.918    0.960     0.951     A
                 0.977    0.058    0.940      0.977    0.958      0.918    0.960     0.929     B
Weighted Avg.    0.959    0.040    0.960      0.959    0.959      0.918    0.960     0.941

=== Confusion Matrix ===

    a    b   <-- classified as
 4902  301 |    a = A
  111 4686 |    b = B
```

Status: OK

Figure 4: SVM only using training set (separable).



Test Instances: Relation: ring-te... Attributes: 3 Instances: ? Sum of weights: ?

Open — Look In: input

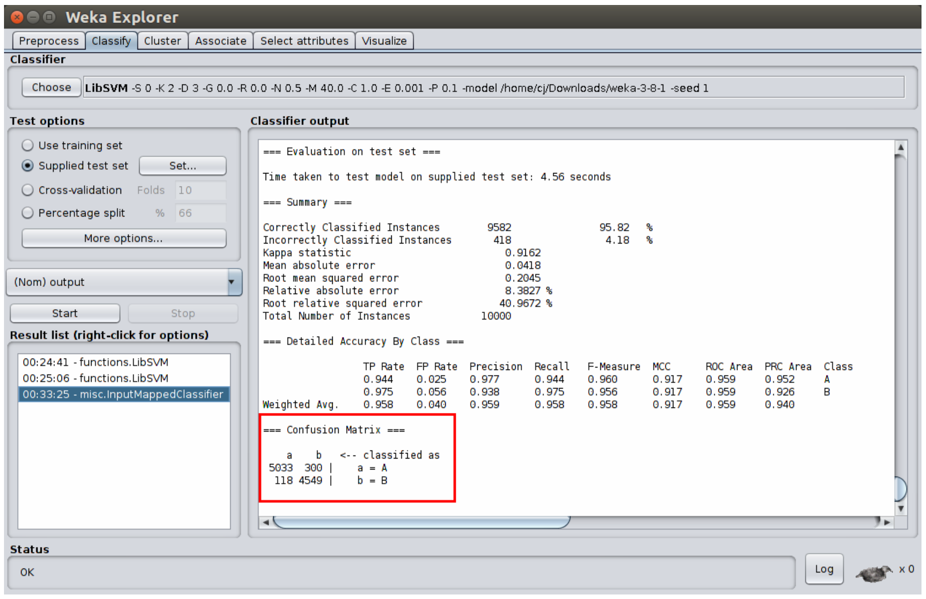
```
erged-weka-letter.csv          ring-test-weka-letters-empty.csv
erged-weka.csv                 ring-test-weka-letters.csv
eparable-weka-letters.csv      ring-test-weka.csv
eparable-weka.csv
est-weka (compare).csv
```

Note: Some file formats offer additional options which can be customized when invoking the options dialog.

File Name: ring-test-weka-letters.csv

Files of Type: CSV data files (*.csv)

Figure 5: SVM adding test set (separable).



```
LibSVM -S 0 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -model /home/cj/Downloads/weka-3-8-1 -seed 1
```

```
=== Evaluation on test set ===

Time taken to test model on supplied test set: 4.56 seconds

=== Summary ===

Correctly Classified Instances        9582               95.82   %
Incorrectly Classified Instances       418                4.18   %
Kappa statistic                          0.9162
Mean absolute error                      0.0418
Root mean squared error                  0.2045
Relative absolute error                  8.3827 %
Root relative squared error             40.9672 %
Total Number of Instances            10000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.944    0.025    0.977      0.944    0.960      0.917    0.959     0.952     A
                 0.975    0.056    0.938      0.975    0.956      0.917    0.959     0.926     B
Weighted Avg.    0.958    0.040    0.959      0.958    0.958      0.917    0.959     0.940

=== Confusion Matrix ===

    a    b   <-- classified as
 5033  300 |    a = A
  118 4549 |    b = B
```

Figure 6: SVM train result using test set (separable).

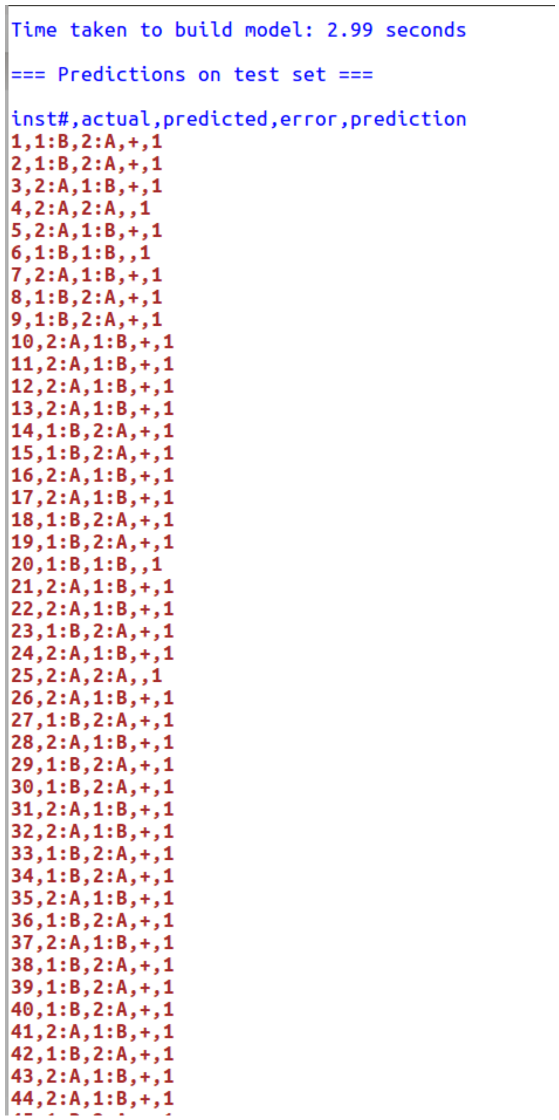
Having the confusion matrix in Weka we can easily compute the classification error according to the formula:

$$E = 100 \cdot \frac{n_{01} + n_{10}}{n_{00} + n_{01} + n_{10} + n_{11}}$$

$$E = 100 \cdot \frac{300 + 118}{5033 + 300 + 118 + 4549}$$

$$E = 100 \cdot \frac{418}{10000} = 100 \cdot 0.0418 = 4.18\%$$

In Weka we can also get the details about the classification errors by going to MoreOptions/OutputPredictions/Choose/CSV, the output for this are the 10000 real values and the predictions by the software:



```
Time taken to build model: 2.99 seconds

=== Predictions on test set ===

inst#,actual,predicted,error,prediction
1,1:B,2:A,+,1
2,1:B,2:A,+,1
3,2:A,1:B,+,1
4,2:A,2:A,,1
5,2:A,1:B,+,1
6,1:B,1:B,,1
7,2:A,1:B,+,1
8,1:B,2:A,+,1
9,1:B,2:A,+,1
10,2:A,1:B,+,1
11,2:A,1:B,+,1
12,2:A,1:B,+,1
13,2:A,1:B,+,1
14,1:B,2:A,+,1
15,1:B,2:A,+,1
16,2:A,1:B,+,1
17,2:A,1:B,+,1
18,1:B,2:A,+,1
19,1:B,2:A,+,1
20,1:B,1:B,,1
21,2:A,1:B,+,1
22,2:A,1:B,+,1
23,1:B,2:A,+,1
24,2:A,1:B,+,1
25,2:A,2:A,,1
26,2:A,1:B,+,1
27,1:B,2:A,+,1
28,2:A,1:B,+,1
29,1:B,2:A,+,1
30,1:B,2:A,+,1
31,2:A,1:B,+,1
32,2:A,1:B,+,1
33,1:B,2:A,+,1
34,1:B,2:A,+,1
35,2:A,1:B,+,1
36,1:B,2:A,+,1
37,2:A,1:B,+,1
38,1:B,2:A,+,1
39,1:B,2:A,+,1
40,1:B,2:A,+,1
41,2:A,1:B,+,1
42,1:B,2:A,+,1
43,2:A,1:B,+,1
44,2:A,1:B,+,1
```

Figure 7: SVM prediction details (separable)

In Weka we also have the option to print the predicted values and have a visual representation of the errors, for this we simply make right click in the important iten on ResultList panel:

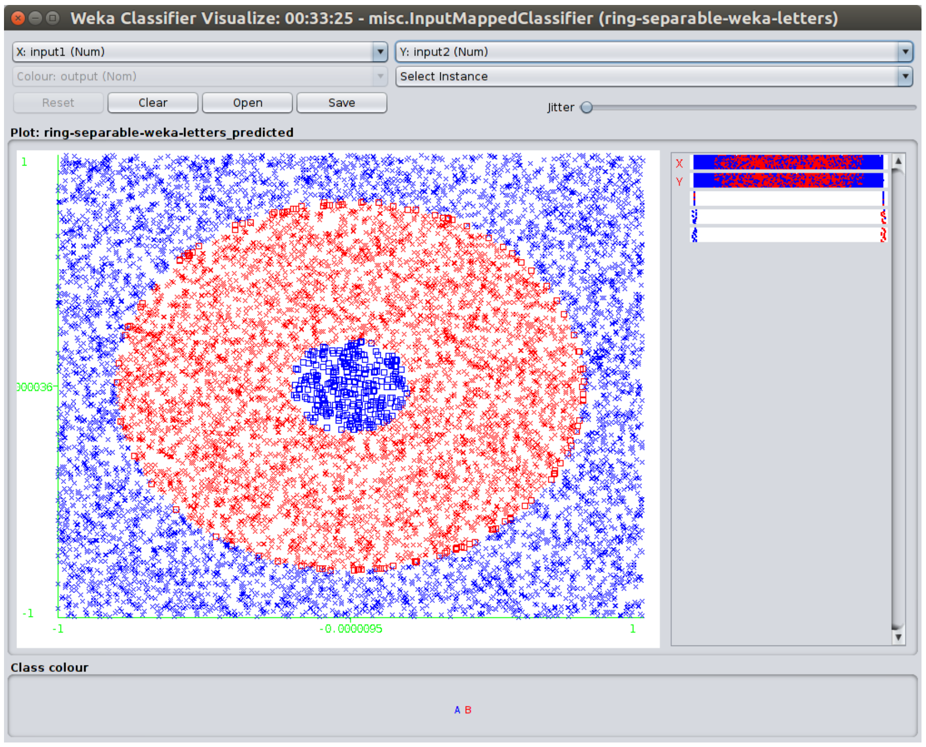
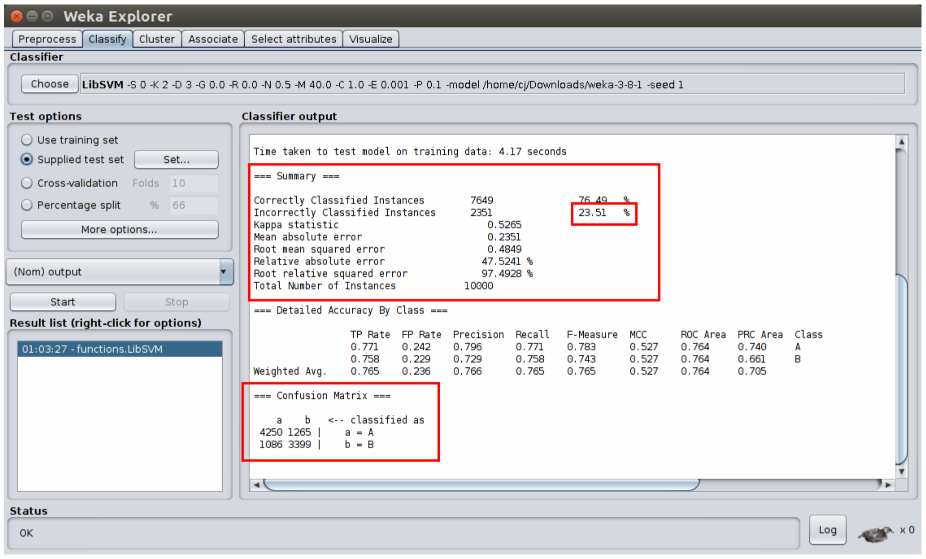

Weka Classifier Visualize: 00:33:25 - misc.InputMappedClassifier (ring-separable-weka-letters)
X: input1 (Num)
Y: input2 (Num)
Colour: output (Nom)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot: ring-separable-weka-letters_predicted
1
000036
-1
-1
-0.0000095
1
X
Y
Class colour
A B

Figure 8: Weka plot result (separable)

Now following the same steps using the "ring-merged-weka-letters" file we can get the following results:



```
LibSVM -S 0 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -model /home/cj/Downloads/weka-3-8-1 -seed 1
```

Use training set
[✓ Supplied test set] Set...
Cross-validation Folds 10
Percentage split % 66
More options...

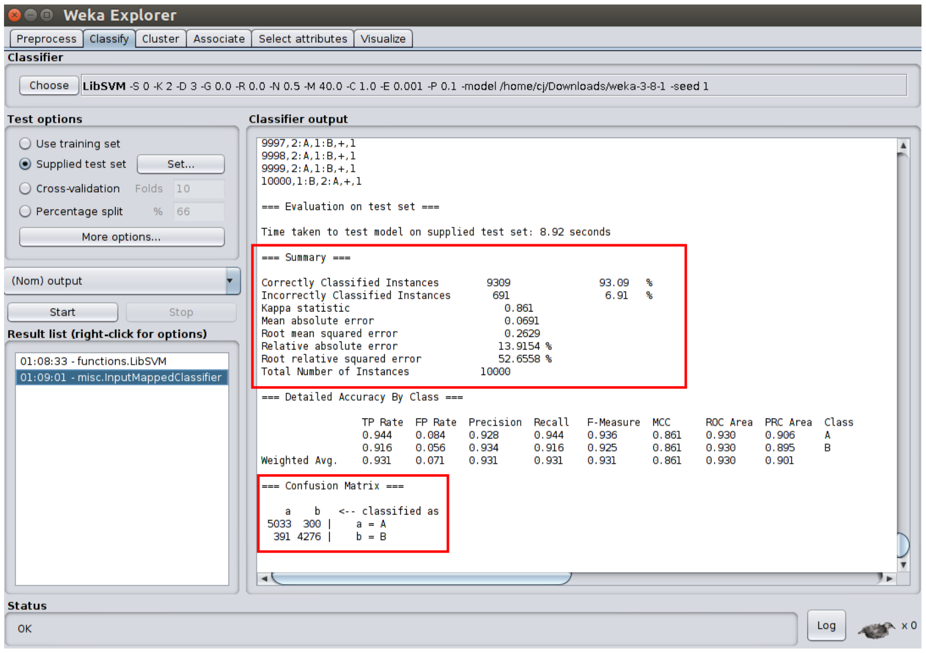
(Nom) output

01:03:27 - functions.LibSVM

OK

```
Time taken to test model on training data: 4.17 seconds

=== Summary ===

Correctly Classified Instances        7649               76.49   %
Incorrectly Classified Instances      2351               23.51   %
Kappa statistic                          0.5265
Mean absolute error                      0.2351
Root mean squared error                  0.4849
Relative absolute error                 47.5241 %
Root relative squared error             97.4928 %
Total Number of Instances            10000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.771    0.242    0.796      0.771    0.783      0.527    0.764     0.740     A
                 0.758    0.229    0.729      0.758    0.743      0.527    0.764     0.661     B
Weighted Avg.    0.765    0.236    0.766      0.765    0.765      0.527    0.764     0.705

=== Confusion Matrix ===

    a    b   <-- classified as
 4250 1265 |    a = A
 1086 3399 |    b = B
```

Figure 9: SVM only using training set (merged).



```
LibSVM -S 0 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -model /home/cj/Downloads/weka-3-8-1 -seed 1
```

Use training set
[✓ Supplied test set] Set...
Cross-validation Folds 10
Percentage split % 66
More options...

(Nom) output

01:08:33 - functions.LibSVM
01:09:01 - misc.InputMappedClassifier

OK

```
9997,2:A,1:B,+,1
9998,2:A,1:B,+,1
9999,2:A,1:B,+,1
10000,1:B,2:A,+,1

=== Evaluation on test set ===

Time taken to test model on supplied test set: 8.92 seconds

=== Summary ===

Correctly Classified Instances        9309               93.09   %
Incorrectly Classified Instances       691                6.91   %
Kappa statistic                          0.861
Mean absolute error                      0.0691
Root mean squared error                  0.2629
Relative absolute error                 13.9154 %
Root relative squared error             52.6558 %
Total Number of Instances            10000

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.944    0.084    0.928      0.944    0.936      0.861    0.930     0.906     A
                 0.916    0.056    0.934      0.916    0.925      0.861    0.930     0.895     B
Weighted Avg.    0.931    0.071    0.931      0.931    0.931      0.861    0.930     0.901

=== Confusion Matrix ===

    a    b   <-- classified as
 5033  300 |    a = A
  391 4276 |    b = B
```

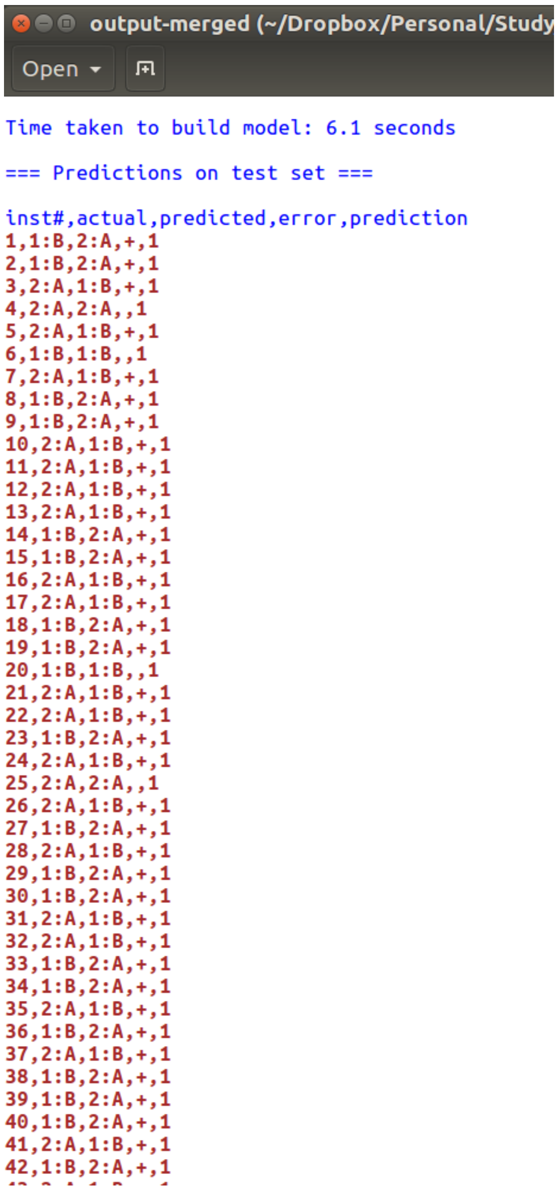
Figure 10: SVM train result using test set (merged).

Again, having the confusion matrix in Weka we can easily compute the classification error according to the formula:

$$E = 100 \cdot \frac{n_{01} + n_{10}}{n_{00} + n_{01} + n_{10} + n_{11}}$$

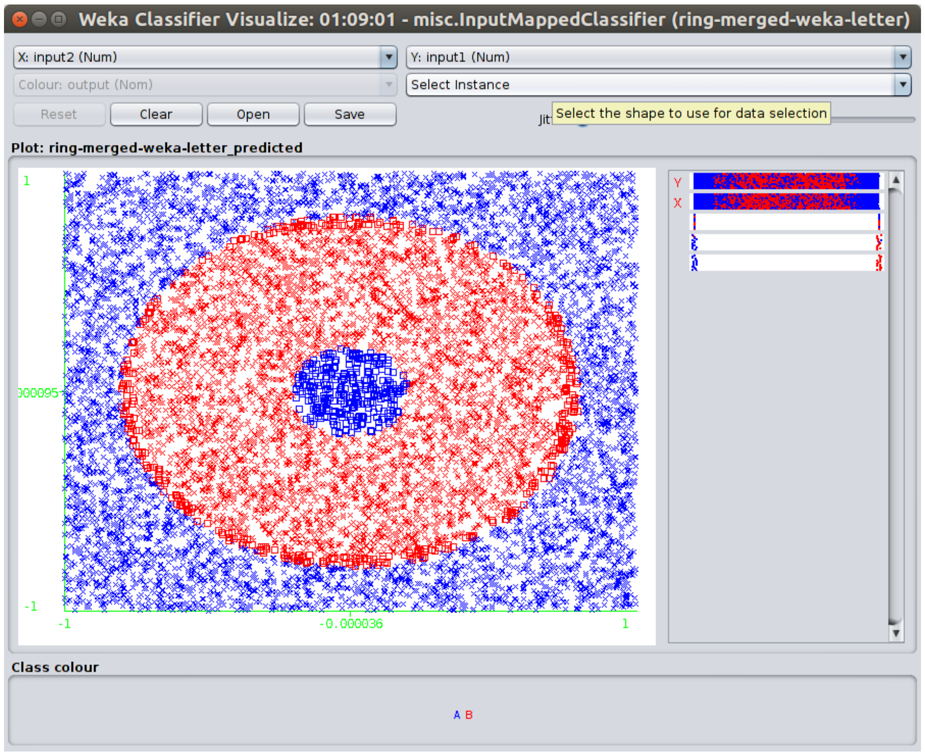
$$E = 100 \cdot \frac{300 + 391}{5033 + 300 + 391 + 4276}$$

$$E = 100 \cdot \frac{691}{10000} = 100 \cdot 0.0691 = 6.91\%$$



```
Time taken to build model: 6.1 seconds

=== Predictions on test set ===

inst#,actual,predicted,error,prediction
1,1:B,2:A,+,1
2,1:B,2:A,+,1
3,2:A,1:B,+,1
4,2:A,2:A,,1
5,2:A,1:B,+,1
6,1:B,1:B,,1
7,2:A,1:B,+,1
8,1:B,2:A,+,1
9,1:B,2:A,+,1
10,2:A,1:B,+,1
11,2:A,1:B,+,1
12,2:A,1:B,+,1
13,2:A,1:B,+,1
14,1:B,2:A,+,1
15,1:B,2:A,+,1
16,2:A,1:B,+,1
17,2:A,1:B,+,1
18,1:B,2:A,+,1
19,1:B,2:A,+,1
20,1:B,1:B,,1
21,2:A,1:B,+,1
22,2:A,1:B,+,1
23,1:B,2:A,+,1
24,2:A,1:B,+,1
25,2:A,2:A,,1
26,2:A,1:B,+,1
27,1:B,2:A,+,1
28,2:A,1:B,+,1
29,1:B,2:A,+,1
30,1:B,2:A,+,1
31,2:A,1:B,+,1
32,2:A,1:B,+,1
33,1:B,2:A,+,1
34,1:B,2:A,+,1
35,2:A,1:B,+,1
36,1:B,2:A,+,1
37,2:A,1:B,+,1
38,1:B,2:A,+,1
39,1:B,2:A,+,1
40,1:B,2:A,+,1
41,2:A,1:B,+,1
42,1:B,2:A,+,1
```

Figure 11: SVM prediction details (merged)


Weka Classifier Visualize: 01:09:01 - misc.InputMappedClassifier (ring-merged-weka-letter)

X: input2 (Num)

Y: input1 (Num)

Colour: output (Nom)

Select Instance

Select the shape to use for data selection

Plot: ring-merged-weka-letter_predicted

Class colour

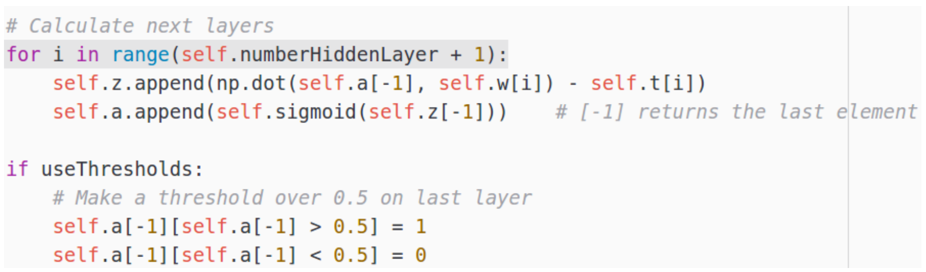
Figure 12: Weka plot result (merged)

3 Back-Propagation (BP), implemented by the student

3.1 Code changes

When working with the Back-propagation, we are using the same implementation made for the first exercise, but in this case we need to make some adjust in the code:

1. Include a new parameter to "forward" method, because while learning it will be the same than before, but when testing and computing the classification error it must apply a threshold to the output in last layer, this way we force the output to be only 0 or 1.

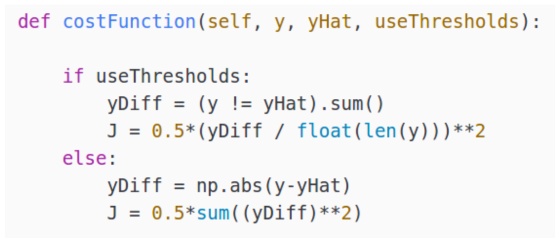


```
# Calculate next layers
for i in range(self.numberHiddenLayer + 1):
    self.z.append(np.dot(self.a[-1], self.w[i]) - self.t[i])
    self.a.append(self.sigmoid(self.z[-1]))    # [-1] returns the last element

if useThresholds:
    # Make a threshold over 0.5 on last layer
    self.a[-1][self.a[-1] > 0.5] = 1
    self.a[-1][self.a[-1] < 0.5] = 0
```

Figure 13: Forward method using threshold.

2. Include a new parameter to "costFunction" method, this is because after having the threshold in Forward method while testing, we will also need to compute the classification error as

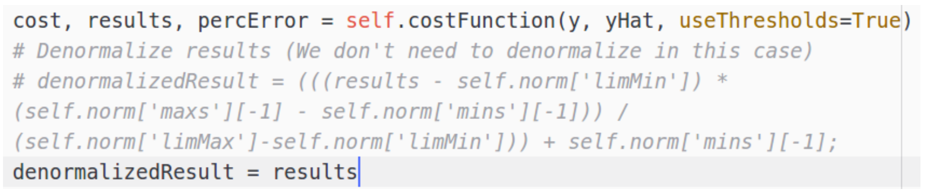
$$E = 100 \cdot \frac{n_{01} + n_{10}}{n_{00} + n_{01} + n_{10} + n_{11}}$$



```
def costFunction(self, y, yHat, useThresholds):

    if useThresholds:
        yDiff = (y != yHat).sum()
        J = 0.5*(yDiff / float(len(y)))**2
    else:
        yDiff = np.abs(y-yHat)
        J = 0.5*sum((yDiff)**2)
```

Figure 14: CostFunction method using threshold.

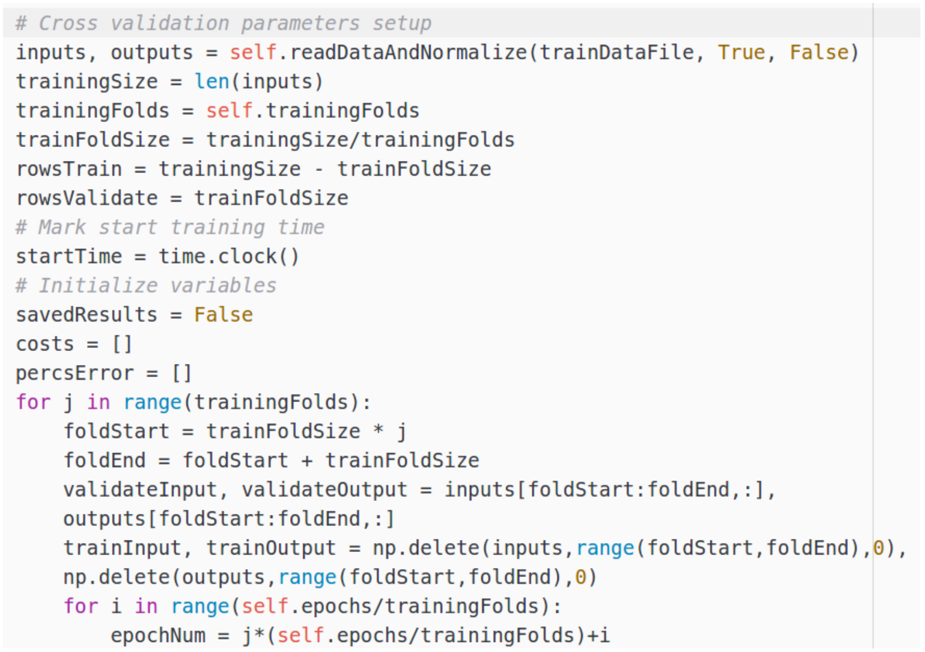
3. Cancel the processes of normalize and denormalize because in this example all the values are already normalized. This was canceled in methods "readDataAndNormalize" and



```
cost, results, percError = self.costFunction(y, yHat, useThresholds=True)
# Denormalize results (We don't need to denormalize in this case)
# denormalizedResult = (((results - self.norm['limMin']) *
(self.norm['maxs'][-1] - self.norm['mins'][-1])) /
(self.norm['limMax']-self.norm['limMin'])) + self.norm['mins'][-1];
denormalizedResult = results
```

Figure 15: Caption [**?**]

4. This classification is more complicated than first one, that's why in this case I'm improving the cross validation, so I'm adding a "trainingFolds" parameter to the network and according to this value the system split the data and split the epochs, this resulting parts are called "folds", for every fold we iterate the part of the data we use as training and validating set.



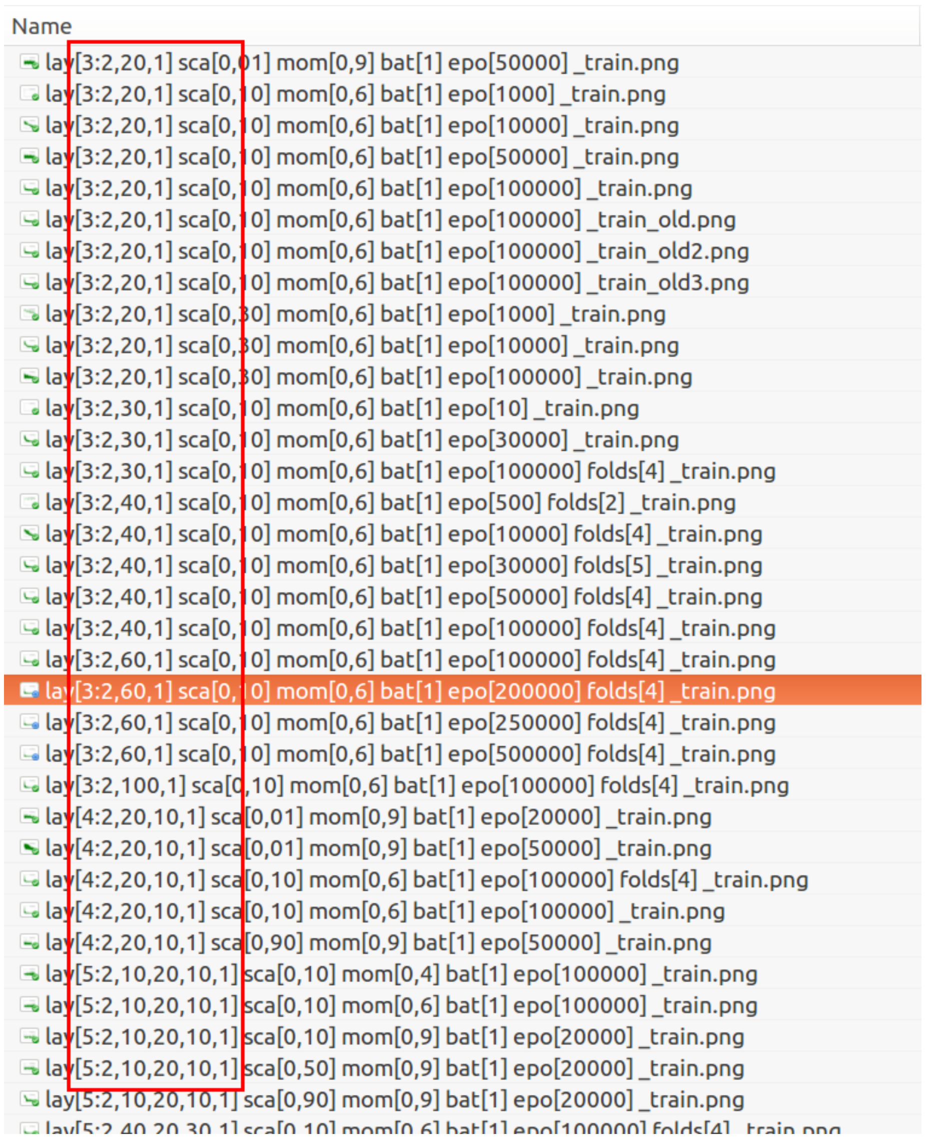
```
# Cross validation parameters setup
inputs, outputs = self.readDataAndNormalize(trainDataFile, True, False)
trainingSize = len(inputs)
trainingFolds = self.trainingFolds
trainFoldSize = trainingSize/trainingFolds
rowsTrain = trainingSize - trainFoldSize
rowsValidate = trainFoldSize
# Mark start training time
startTime = time.clock()
# Initialize variables
savedResults = False
costs = []
percsError = []
for j in range(trainingFolds):
    foldStart = trainFoldSize * j
    foldEnd = foldStart + trainFoldSize
    validateInput, validateOutput = inputs[foldStart:foldEnd,:],
    outputs[foldStart:foldEnd,:]
    trainInput, trainOutput = np.delete(inputs,range(foldStart,foldEnd),0),
    np.delete(outputs,range(foldStart,foldEnd),0)
    for i in range(self.epochs/trainingFolds):
        epochNum = j*(self.epochs/trainingFolds)+i
```

Figure 16: Caption [**?**]

3.2 Architecture changes

Additionally to last code changes, we also need to include some network architecture changes, because this data is more complex and we need our network to deal with this complexity by increasing the number of neurons and maybe also the number of layers.

This are some of the architectures I tried before making some conclusions:



```
Name
lay[3:2,20,1] sca[0,01] mom[0,9] bat[1] epo[50000] _train.png
lay[3:2,20,1] sca[0,10] mom[0,6] bat[1] epo[1000] _train.png
lay[3:2,20,1] sca[0,10] mom[0,6] bat[1] epo[10000] _train.png
lay[3:2,20,1] sca[0,10] mom[0,6] bat[1] epo[50000] _train.png
lay[3:2,20,1] sca[0,10] mom[0,6] bat[1] epo[100000] _train.png
lay[3:2,20,1] sca[0,10] mom[0,6] bat[1] epo[100000] _train_old.png
lay[3:2,20,1] sca[0,10] mom[0,6] bat[1] epo[100000] _train_old2.png
lay[3:2,20,1] sca[0,10] mom[0,6] bat[1] epo[100000] _train_old3.png
lay[3:2,20,1] sca[0,30] mom[0,6] bat[1] epo[1000] _train.png
lay[3:2,20,1] sca[0,30] mom[0,6] bat[1] epo[10000] _train.png
lay[3:2,20,1] sca[0,30] mom[0,6] bat[1] epo[100000] _train.png
lay[3:2,30,1] sca[0,10] mom[0,6] bat[1] epo[10] _train.png
lay[3:2,30,1] sca[0,10] mom[0,6] bat[1] epo[30000] _train.png
lay[3:2,30,1] sca[0,10] mom[0,6] bat[1] epo[100000] folds[4] _train.png
lay[3:2,40,1] sca[0,10] mom[0,6] bat[1] epo[500] folds[2] _train.png
lay[3:2,40,1] sca[0,10] mom[0,6] bat[1] epo[10000] folds[4] _train.png
lay[3:2,40,1] sca[0,10] mom[0,6] bat[1] epo[30000] folds[5] _train.png
lay[3:2,40,1] sca[0,10] mom[0,6] bat[1] epo[50000] folds[4] _train.png
lay[3:2,40,1] sca[0,10] mom[0,6] bat[1] epo[100000] folds[4] _train.png
lay[3:2,60,1] sca[0,10] mom[0,6] bat[1] epo[100000] folds[4] _train.png
lay[3:2,60,1] sca[0,10] mom[0,6] bat[1] epo[200000] folds[4] _train.png
lay[3:2,60,1] sca[0,10] mom[0,6] bat[1] epo[250000] folds[4] _train.png
lay[3:2,60,1] sca[0,10] mom[0,6] bat[1] epo[500000] folds[4] _train.png
lay[3:2,100,1] sca[0,10] mom[0,6] bat[1] epo[100000] folds[4] _train.png
lay[4:2,20,10,1] sca[0,01] mom[0,9] bat[1] epo[20000] _train.png
lay[4:2,20,10,1] sca[0,01] mom[0,9] bat[1] epo[50000] _train.png
lay[4:2,20,10,1] sca[0,10] mom[0,6] bat[1] epo[100000] folds[4] _train.png
lay[4:2,20,10,1] sca[0,10] mom[0,6] bat[1] epo[100000] _train.png
lay[4:2,20,10,1] sca[0,90] mom[0,9] bat[1] epo[50000] _train.png
lay[5:2,10,20,10,1] sca[0,10] mom[0,4] bat[1] epo[100000] _train.png
lay[5:2,10,20,10,1] sca[0,10] mom[0,6] bat[1] epo[100000] _train.png
lay[5:2,10,20,10,1] sca[0,10] mom[0,9] bat[1] epo[20000] _train.png
lay[5:2,10,20,10,1] sca[0,50] mom[0,9] bat[1] epo[20000] _train.png
lay[5:2,10,20,10,1] sca[0,90] mom[0,9] bat[1] epo[20000] _train.png
lay[5:2,40,20,30,1] sca[0,10] mom[0,6] bat[1] epo[100000] folds[4] _train.png
```

Figure 17: Different architectures tried

Note that the red square shows the structure of the layers in that test, the first number before the colon symbol (:), tells the number of layers we have, and the following numbers are the numbers of neurons in every layer from input to output.

Next we can see some of this results, specially the last ones:

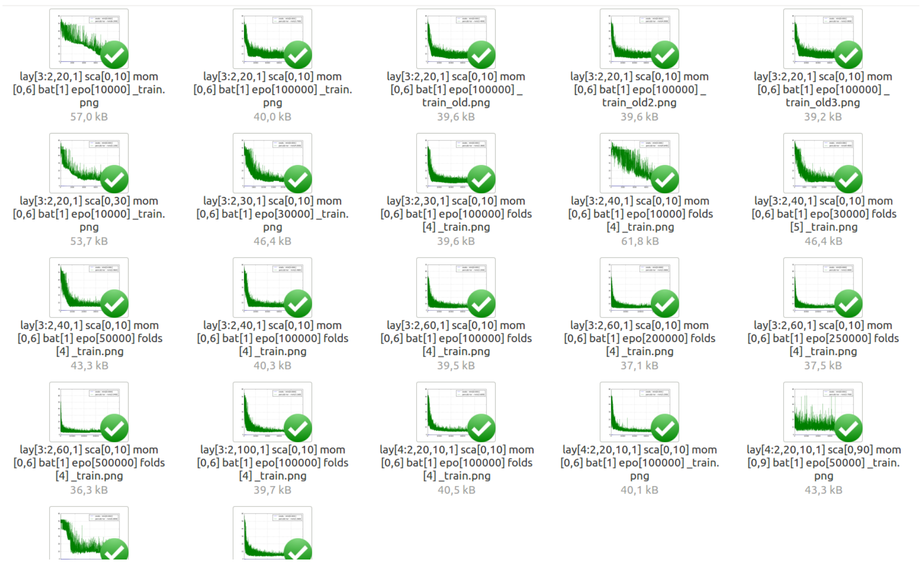

lay[3:2,20,1] sca[0,10] mom [0,6] bat[1] epo[10000] _train.png 57,0 kB
lay[3:2,20,1] sca[0,10] mom [0,6] bat[1] epo[100000] _train.png 40,0 kB
lay[3:2,20,1] sca[0,10] mom [0,6] bat[1] epo[100000] _train_old.png 39,6 kB
lay[3:2,20,1] sca[0,10] mom [0,6] bat[1] epo[100000] _train_old2.png 39,6 kB
lay[3:2,20,1] sca[0,10] mom [0,6] bat[1] epo[100000] _train_old3.png 39,2 kB
lay[3:2,20,1] sca[0,30] mom [0,6] bat[1] epo[10000] _train.png 53,7 kB
lay[3:2,30,1] sca[0,10] mom [0,6] bat[1] epo[30000] _train.png 46,4 kB
lay[3:2,30,1] sca[0,10] mom [0,6] bat[1] epo[100000] folds [4] _train.png 39,6 kB
lay[3:2,40,1] sca[0,10] mom [0,6] bat[1] epo[10000] folds [4] _train.png 61,8 kB
lay[3:2,40,1] sca[0,10] mom [0,6] bat[1] epo[30000] folds [5] _train.png 46,4 kB
lay[3:2,40,1] sca[0,10] mom [0,6] bat[1] epo[50000] folds [4] _train.png 43,3 kB
lay[3:2,40,1] sca[0,10] mom [0,6] bat[1] epo[100000] folds [4] _train.png 40,3 kB
lay[3:2,60,1] sca[0,10] mom [0,6] bat[1] epo[100000] folds [4] _train.png 39,5 kB
lay[3:2,60,1] sca[0,10] mom [0,6] bat[1] epo[200000] folds [4] _train.png 37,1 kB
lay[3:2,60,1] sca[0,10] mom [0,6] bat[1] epo[250000] folds [4] _train.png 37,5 kB
lay[3:2,60,1] sca[0,10] mom [0,6] bat[1] epo[500000] folds [4] _train.png 36,3 kB
lay[3:2,100,1] sca[0,10] mom [0,6] bat[1] epo[100000] folds [4] _train.png 39,7 kB
lay[4:2,20,10,1] sca[0,10] mom [0,6] bat[1] epo[100000] folds [4] _train.png 40,5 kB
lay[4:2,20,10,1] sca[0,10] mom [0,6] bat[1] epo[100000] _train.png 40,1 kB
lay[4:2,20,10,1] sca[0,90] mom [0,9] bat[1] epo[50000] _train.png 43,3 kB

Figure 18: Several learning curves while training with different architectures.

After checking all this results, taking into account the learning curve, validating errors and test errors, I consider that the best architecture is as follows:

- Learning rate: 0.1
- Momentum: 0.6
- Batch size: 1 (on-line)
- Number of epochs: 200,000
- Cross validation while training, fold system
- Fold number: 4 (7500 for training and 2500 for validation)
- Architecture: 3 layers
  - Layer 1: 2 neurons (input)
  - Layer 2: 60 neurons (hidden layer)
  - Layer 3: 1 neuron (output)

3.3 Backprop results

3.3.1 ring-separable

The results were as following:

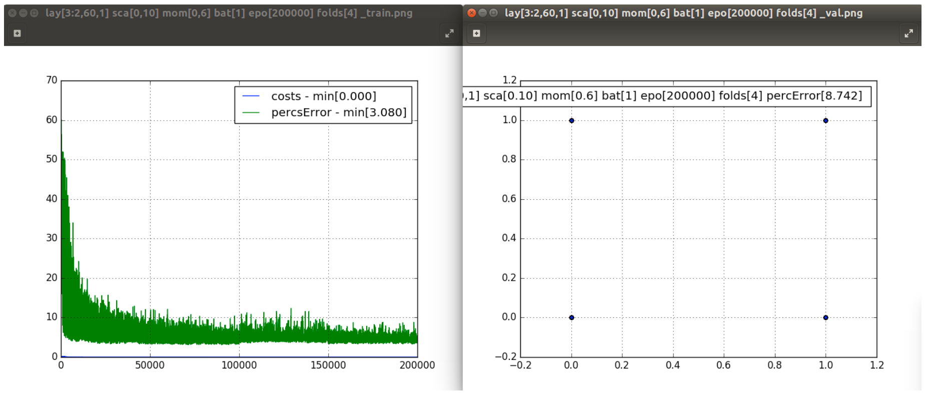
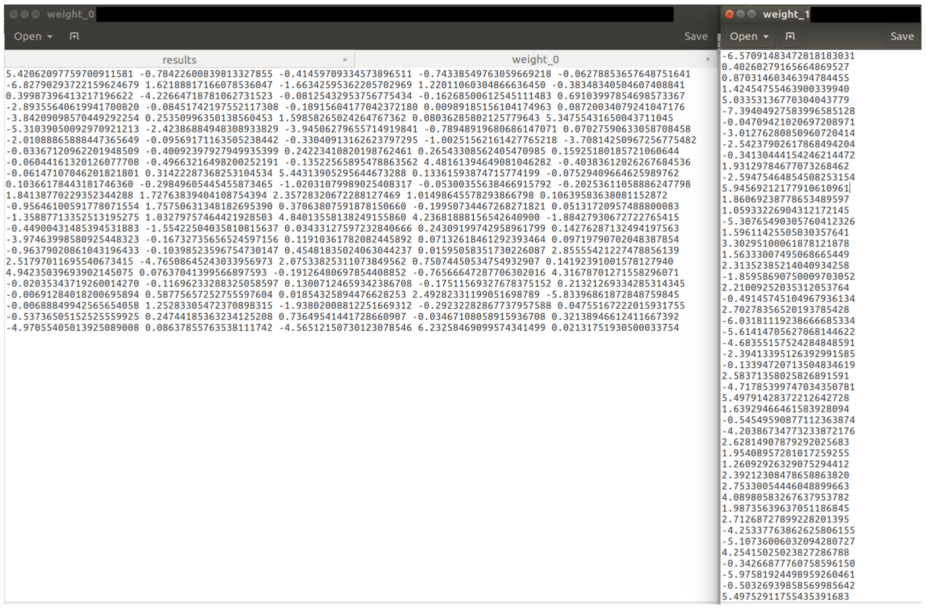


Figure 19: Learning progress and percentage error while validating using separable file.

The image in the right maybe is not very representative because of the nature of the data, but included because it shows the resulting error value computed as:

$$E = 100 \cdot \frac{n_{01} + n_{10}}{n_{00} + n_{01} + n_{10} + n_{11}}$$

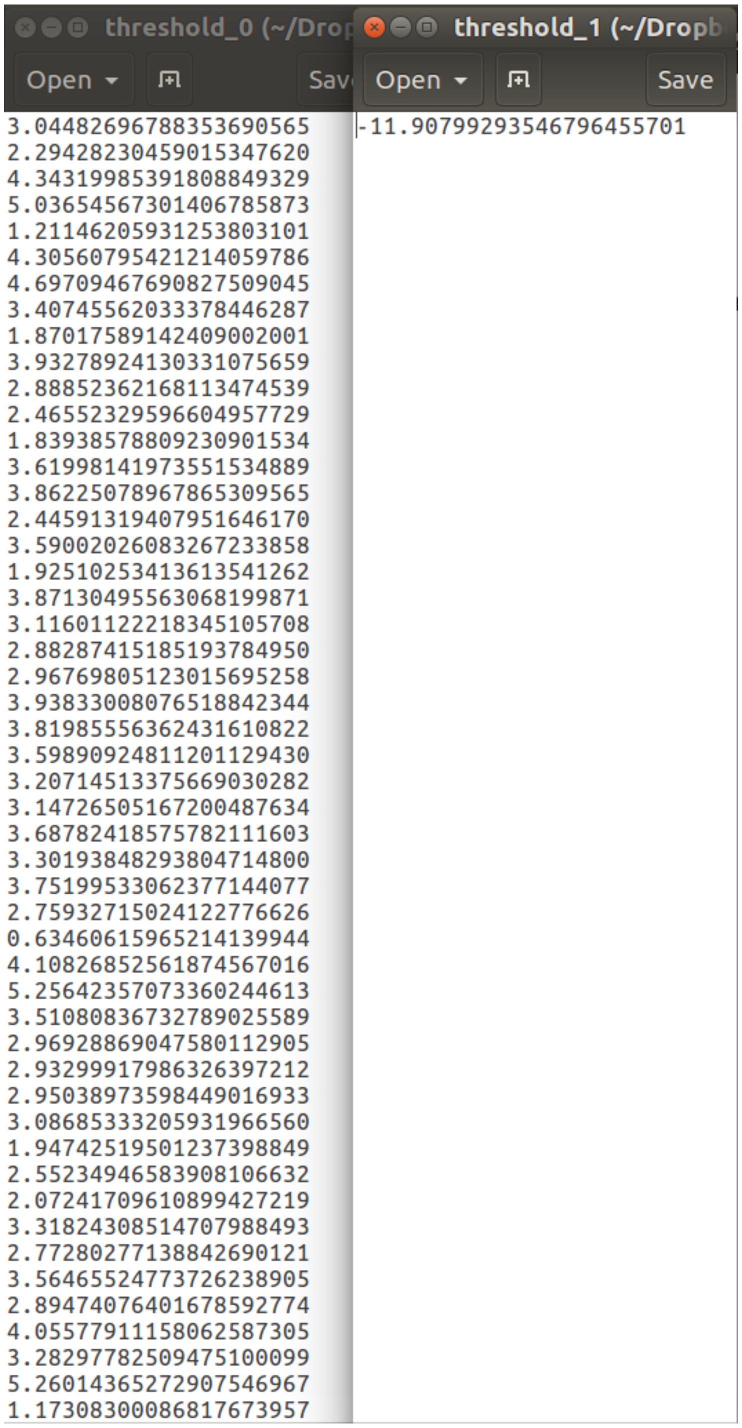
Additionally I include the final weights and thresholds:



```
5.42062097759700911581 -0.78422600839813327855 -0.41459709334573896511 -0.74338549763059669218 -0.06278853657648751641
-6.82790293722159624679 1.62188817166078536047 -1.66342595362205702969 1.22011060304866636450 -0.38348340504607408841
0.39987396413217196622 -4.22664718781062731523 -0.08125432953756775434 -0.16268500612545111483 0.69103997854698573367
-2.89355640619941700820 -0.08451742197552117308 -0.18915604177042372180 0.00989185156104174963 0.08720034079241047176
-3.84209098570449292254 0.25350996350138560453 1.59858265024264767362 0.08036285802125779643 5.34755431650043711045
-5.31039050092970921213 -2.42386884948308933829 -3.94506279655714919841 -0.78948919680686147071 0.07027590633058708458
-2.01088865888447365649 -0.09569171163505238442 -0.33040913162623797295 -1.00251562161427765218 -3.70814250967256775482
-0.03367120962201948509 -0.40092397927949935399 0.24223410820198762461 0.26543308562405470985 0.15925180185721060644
-0.06044161320126077708 -0.49663216498200252191 -0.13522565895478863562 4.48161394649081046282 -0.40383612026267684536
-0.06147107046201821801 0.31422287368253104534 5.44313905295644673288 0.13361593874715774199 -0.07529409664625989762
0.10366178443181746360 -0.29849605445455873465 -1.02031079989025408317 -0.05300355638466915792 -0.20253611058886247798
1.84138770229352344288 1.72763839404108754394 2.35728320672288127469 1.01498645578293866798 0.10639583638081152872
-0.95646100591778071554 1.75750631348182695390 0.37063807591878150660 -0.19950734467268271821 0.05131720957488800083
-1.35887713352513195275 1.03279757464421928503 4.84013558138249155860 4.23681888156542640900 -1.88427930672722765415
-0.44900431485394531883 -1.55422504035810815637 0.03433127597232840666 0.24309199742958961799 0.14276287132494197563
-3.97463998580925448323 -0.16732735656524597156 0.11910361782082445892 0.07132618461292393464 0.09719790702048387854
-0.96379020861043196433 -0.10398523596754730147 0.45481835024063044237 0.01595058351730226087 2.85555421227478856139
2.51797011695540673415 -4.76508645243033956973 2.07533825311073849562 0.75074450534754932907 0.14192391001578127940
4.94235039693902145075 0.07637041399566897593 -0.19126480697854408852 -0.76566647287706302016 4.31678701271558296071
-0.02035343719260014270 -0.11696233288325058597 0.13007124659342386708 -0.17511569327678375152 0.21321269334285314345
-0.00691284018200695894 0.58775657252755597604 0.01854325894476628253 2.49282331199051698789 -5.83396861872848759845
-0.00688849942565654058 1.25283305472370898315 -1.93802008812251669312 -0.29232282867737957588 0.04755167222015931755
-0.53736505152525559925 0.24744185363234125208 0.73649541441728660907 -0.03467108058915936708 0.32138946612411667392
-4.97055405013925089008 0.08637855763538111742 -4.56512150730123078546 6.23258469099574341499 0.02131751930500033754
```

```
-6.57091483472818183031
0.40260279165664869527
0.87031460346394784455
1.42454755463900339940
5.03353136770304043779
-7.39404927583996585128
-0.04709421020697208971
-3.01276280850960720414
-2.54237902617868494204
-0.34130444154246214472
1.93129784677073268462
-2.59475464854508253154
5.94569212177910610961
1.86069238778653489597
1.05933226904312172145
-5.30765490305760412326
1.59611425505030357641
3.30295100061878121878
1.56333007495068665449
2.31352385214040934258
-1.85958690750009703052
2.21009252035312053764
-0.49145745104967936134
2.70278356520193785428
-6.03181119238666685334
-5.61414705627068144622
-4.68355157524284848591
-2.39413395126392991585
-0.13394720713504834619
2.58371358025826891591
-4.71785399747034350781
5.49791428372212642728
1.63929466461583928094
-0.54549590877112363874
-4.20386734773233872176
2.62814907879292025683
1.95408957281017259255
1.26092926329075294412
2.39212308478658863820
2.75330054446048899663
4.08980583267637953782
1.98735639637051186845
2.71268727899228201395
-4.25337763862625806155
-5.10736006032094280727
4.25415025023827286788
-0.34266877760758596150
-5.97581924498959260461
-0.50326939858569985642
5.49752911755435391683
```

Figure 20: Final weights after learning process (separable).



```
3.04482696788353690565
```

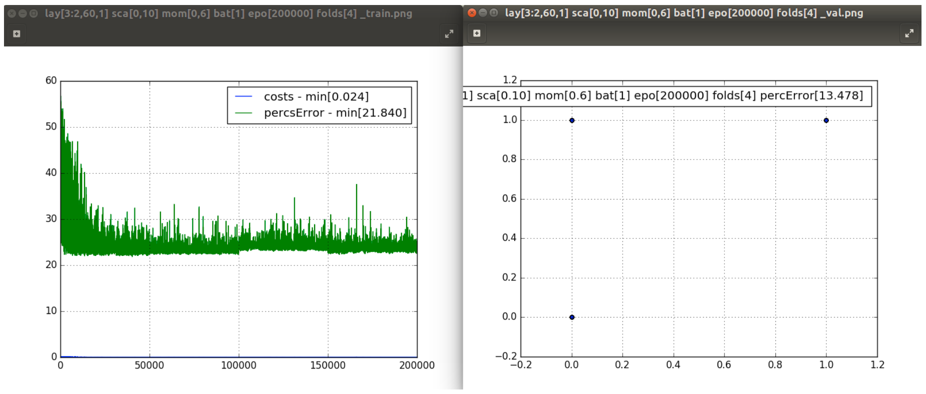
```
3.04482696788353690565
2.29428230459015347620
4.34319985391808849329
5.03654567301406785873
1.21146205931253803101
4.30560795421214059786
4.69709467690827509045
3.40745562033378446287
1.87017589142409002001
3.93278924130331075659
2.88852362168113474539
2.46552329596604957729
1.83938578809230901534
3.61998141973551534889
3.86225078967865309565
2.44591319407951646170
3.59002026083267233858
1.92510253413613541262
3.87130495563068199871
3.11601122218345105708
2.88287415185193784950
2.96769805123015695258
3.93833008076518842344
3.81985556362431610822
3.59890924811201129430
3.20714513375669030282
3.14726505167200487634
3.68782418575782111603
3.30193848293804714800
3.75199533062377144077
2.75932715024122776626
0.63460615965214139944
4.10826852561874567016
5.25642357073360244613
3.51080836732789025589
2.96928869047580112905
2.93299917986326397212
2.95038973598449016933
3.08685333205931966560
1.94742519501237398849
2.55234946583908106632
2.07241709610899427219
3.31824308514707988493
2.77280277138842690121
3.56465524773726238905
2.89474076401678592774
4.05577911158062587305
3.28297782509475100099
5.26014365272907546967
1.17308300086817673957
```

```
-11.90799293546796455701
```

Figure 21: Final thresholds after learning process (separable).

3.3.2 ring-merged

I kept this same architecture for "separable" and "merged" cases so we can make a comparison at the end. The results when using the merged file are as following:


lay[3:2,60,1] sca[0,10] mom[0,6] bat[1] epo[200000] folds[4] _train.png

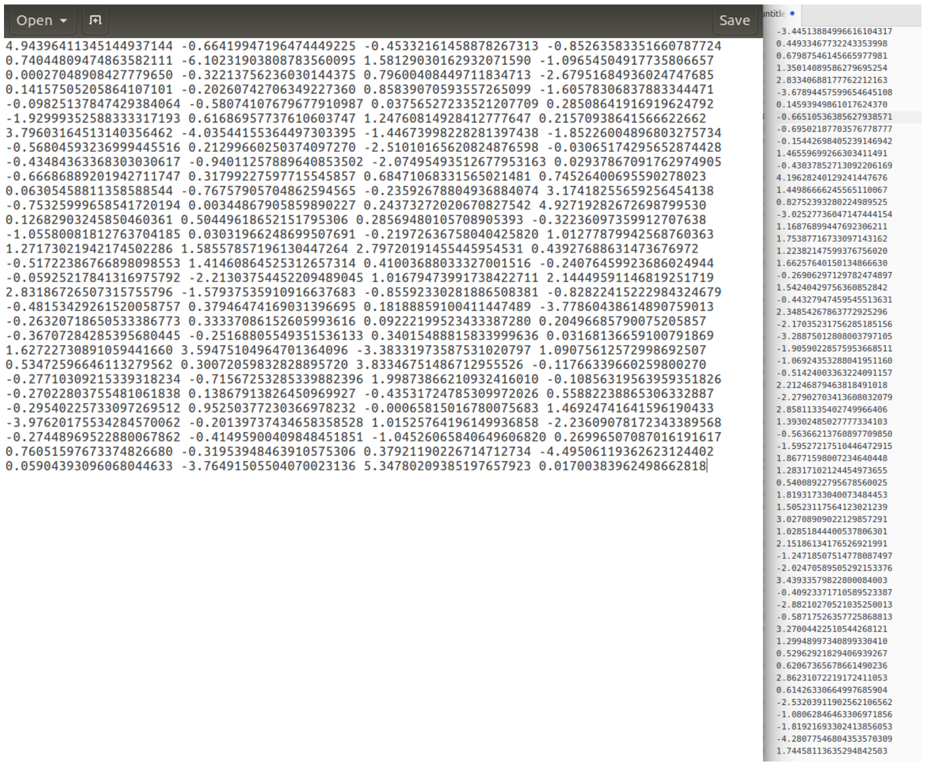
lay[3:2,60,1] sca[0,10] mom[0,6] bat[1] epo[200000] folds[4] _val.png

costs - min[0.024]

percsError - min[21.840]

1] sca[0.10] mom[0.6] bat[1] epo[200000] folds[4] percError[13.478]

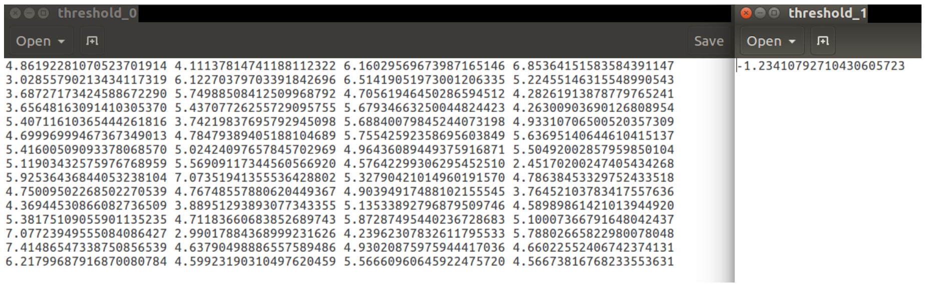
Figure 22: Learning progress and percentage error while validating using merged file.



```
4.94396411345144937144 -0.66419947196474449225 -0.45332161458878267313 -0.85263583351660787724
0.74044809474863582111 -6.10231903808783560095 1.58129030162932071590 -1.09654504917735806657
0.00027048908427779650 -0.32213756236030144375 0.79600408449711834713 -2.67951684936024747685
0.14157505205864107101 -0.20260742706349227360 0.85839070593557265099 -1.60578306837883344471
-0.09825137847429384064 -0.58074107679677910987 0.03756527233521207709 0.28508641916919624792
-1.92999352588333317193 0.61686957737610603747 1.24760814928412777647 0.21570938641566622662
3.79603164513140356462 -4.03544155364497303395 -1.44673998228281397438 -1.85226004896803275734
-0.56804593236999445516 0.21299660250374097270 -2.51010165620824876598 -0.03065174295652874428
-0.43484363368303030617 -0.94011257889640853502 -2.07495493512677953163 0.02937867091762974905
-0.66686889201942711747 0.31799227597715545857 0.68471068331565021481 0.74526400695590278023
0.06305458811358588544 -0.76757905704862594565 -0.23592678804936884074 3.17418255659256454138
-0.75325999658541720194 0.00344867905859890227 0.24373272020670827542 4.92719282672698799530
0.12682903245850460361 0.50449618652151795306 0.28569480105708905393 -0.32236097359912707638
-1.05580081812763704185 0.03031966248699507691 -0.21972636758040425820 1.01277879942568760363
1.27173021942174502286 1.58557857196130447264 2.79720191455445954531 0.43927688631473676972
-0.51722386766898098553 1.41460864525312657314 0.41003688033327001516 -0.24076459923686024944
-0.05925217841316975792 -2.21303754452209489045 1.01679473991738422711 2.14449591146819251719
2.83186726507315755796 -1.57937535910916637683 -0.85592330281886508381 -0.82822415222984324679
-0.48153429261520058757 0.37946474169031396695 0.18188859100411447489 -3.77860438614890759013
-0.26320718650533386773 0.33337086152605993616 0.09222199523433387280 0.20496685790075205857
-0.36707284285395680445 -0.25168805549351536133 0.34015488815833999636 0.03168136659100791869
1.62722730891059441660 3.59475104964701364096 -3.38331973587531020797 1.09075612572998692507
0.53472596646113279562 0.30072059832828895720 3.83346751486712955526 -0.11766339660259800270
-0.27710309215339318234 -0.71567253285339882396 1.99873866210932416010 -0.10856319563959351826
-0.27022803755481061838 0.13867913826450969927 -0.43531724785309972026 0.55882238865306332887
-0.29540225733097269512 0.95250377230366978232 -0.00065815016780075683 1.46924741641596190433
-3.97620175534284570062 -0.20139737434658358528 1.01525764196149936858 -2.23609078172343389568
-0.27448969522880067862 -0.41495900409848451851 -1.04526065840649606820 0.26996507087016191617
0.76051597673374826680 -0.31953948463910575306 0.37921190226714712734 -4.49506119362623124402
0.05904393096068044633 -3.76491505504070023136 5.34780209385197657923 0.01700383962498662818
```

```
-3.44513884996616104317
0.44933467732243353998
0.67987546145665977981
1.35014089586279695254
2.83340688177762212163
-3.67894457599654645108
0.14593949861017624370
-0.66510536385627938571
-0.69502187703576778777
-0.15442698405239146942
1.46559699266303411491
-0.43037852713092206169
4.19628240129241447676
1.44986666245565110067
0.82752393280224989525
-3.02527736047147444154
1.16876899447692306211
1.75387716733097143162
1.22382147599376756020
1.66257640150134866630
-0.26906297129782474897
1.54240429756360852842
-0.44327947459545513631
2.34854267863772925296
-2.17035231756285185156
-3.28875012808003797105
-1.90590228575953668511
-1.06924353288041951160
-0.51424003363224091157
2.21246879463818491018
-2.27902703413608032079
2.85811335402749966406
1.39302485027777334103
-0.56366213760897709850
-1.59527217510446472915
1.86771598007234640448
1.28317102124454973655
0.54008922795678560025
1.81931733040073484453
1.50523117564123021239
3.02708909022129857291
1.02851844400537806301
2.15186134176526921991
-1.24718507514778087497
-2.02470589505292153376
3.43933579822800084003
-0.40923371710589523387
-2.88210270521035250013
-0.58717526357725868813
3.27004422510544268121
1.29948997340899330410
0.52962921829406939267
0.62067365678661490236
2.86231072219172411053
0.61426330664997685904
-2.53203911902562106562
-1.08062846463306971856
-1.81921693302413856053
-4.28077546804353570309
1.74458113635294842503
```

Figure 23: Final weights after learning process (merged).



```
4.86192281070523701914 4.11137814741188112322 6.16029569673987165146 6.85364151583584391147
3.02855790213434117319 6.12270379703391842696 6.51419051973001206335 5.22455146315548990543
3.68727173424588672290 5.74988508412509968792 4.70561946450286594512 4.28261913878779765241
3.65648163091410305370 5.43707726255729095755 5.67934663250044824423 4.26300903690126808954
5.40711610365444261816 3.74219837695792945098 5.68840079845244073198 4.93310706500520357309
4.69996999467367349013 4.78479389405188104689 5.75542592358695603849 5.63695140644610415137
5.41600509093378068570 5.02424097657845702969 4.96436089449375916871 5.50492002857959850104
5.11903432575976768959 5.56909117344560566920 4.57642299306295452510 2.45170200247405434268
5.92536436844053238104 7.07351941355536428802 5.32790421014960191570 4.78638453329752433518
4.75009502268502270539 4.76748557880620449367 4.90394917488102155545 3.76452103783417557636
4.36944530866082736509 3.88951293893077343355 5.13533892796879509746 4.58989861421013944920
5.38175109055901135235 4.71183660683852689743 5.87287495440236728683 5.10007366791648042437
7.07723949555084086427 2.99017884368999231626 4.23962307832611795533 5.78802665822980078048
7.41486547338750856539 4.63790498886557589486 4.93020875975944417036 4.66022552406742374131
6.21799687916870080784 4.59923190310497620459 5.56660960645922475720 4.56673816768233553631
```

```
-1.23410792710430605723
```

Figure 24: Final thresholds after learning process (merged).

3.3.3 Plotting results

Now plotting the coordinates for results we get the following outputs:

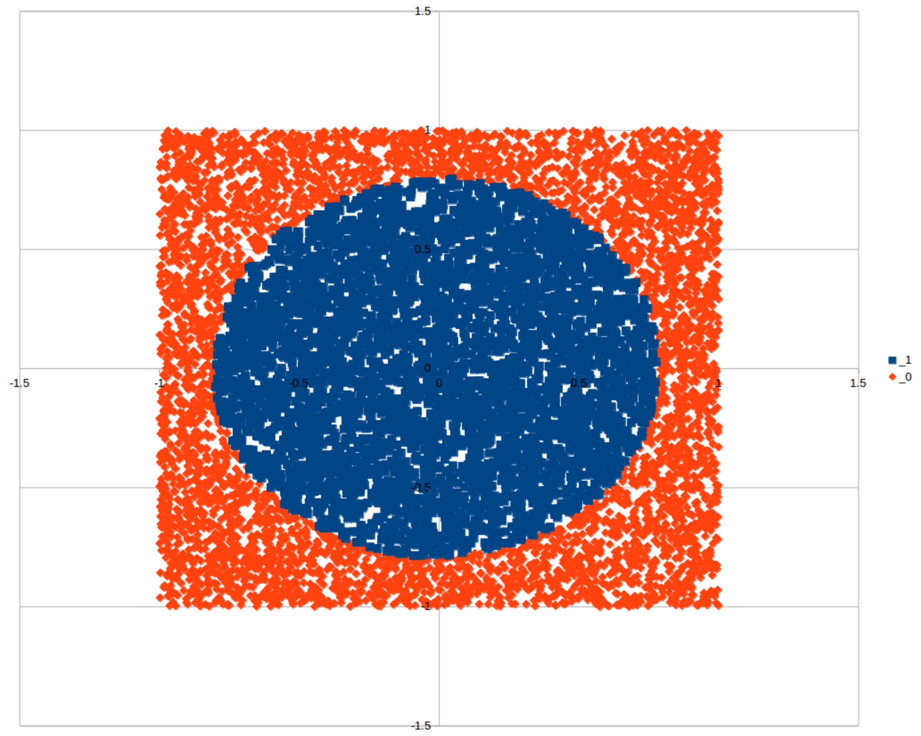


Figure 25: Old results using one hidden layer (separable).

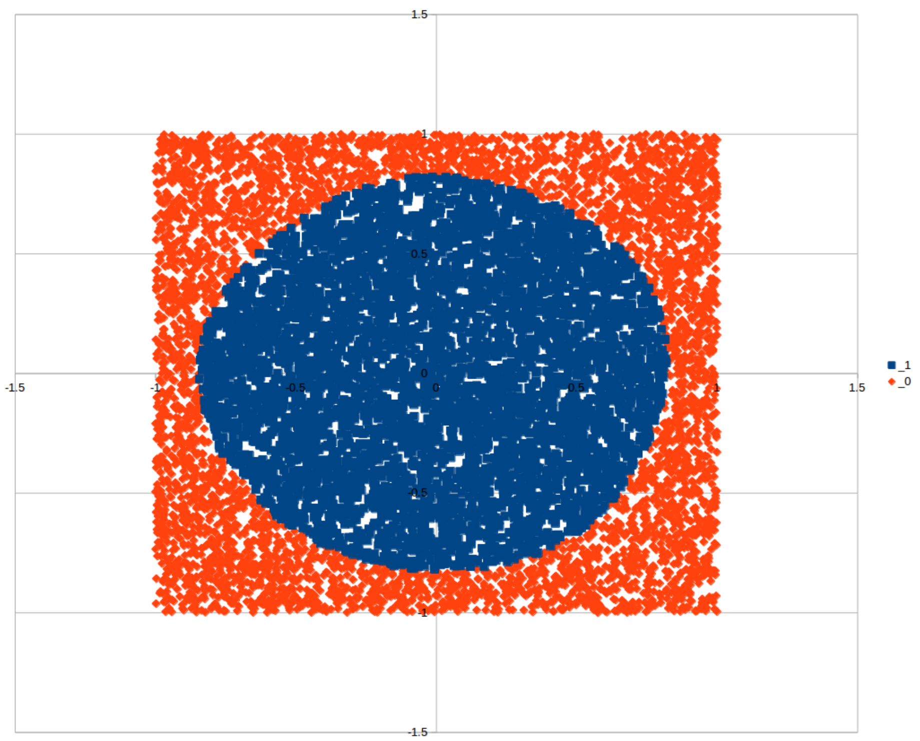


Figure 26: Old results using one hidden layer (merged).

If we analyze this plots, even when learning results seemed good, it is obvious that our algorithm is not working properly, then I have changed the architecture of the network including more hidden layers and so handling better the complexity of the data, the new architecture is like this:

- Learning rate: 0.1
- Momentum: 0.6
- Batch size: 1 (on-line)
- Number of epochs: 100,000
- Cross validation while training, fold system
- Fold number: 4 (7500 for training and 2500 for validation)
- Architecture: 6 layers
  - Layer 1: 2 neurons (input)
  - Layer 2: 30 neurons (hidden layer)

– Layer 3: 40 neurons (hidden layer)
– Layer 4: 40 neurons (hidden layer)
– Layer 5: 30 neurons (hidden layer)
– Layer 6: 1 neuron (output)

With this new Network we got the following results:

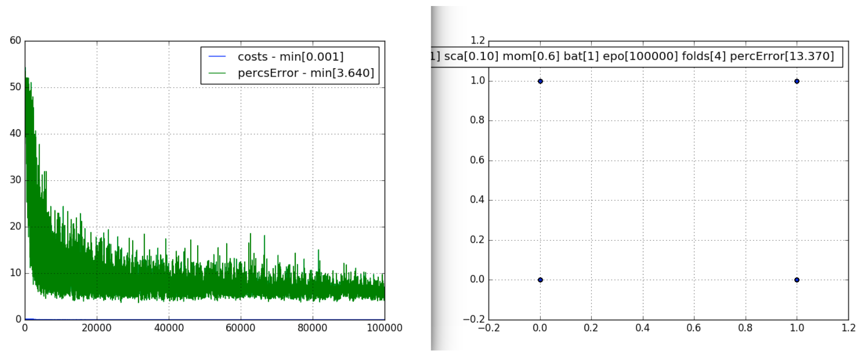


Figure 27: New results (separable).

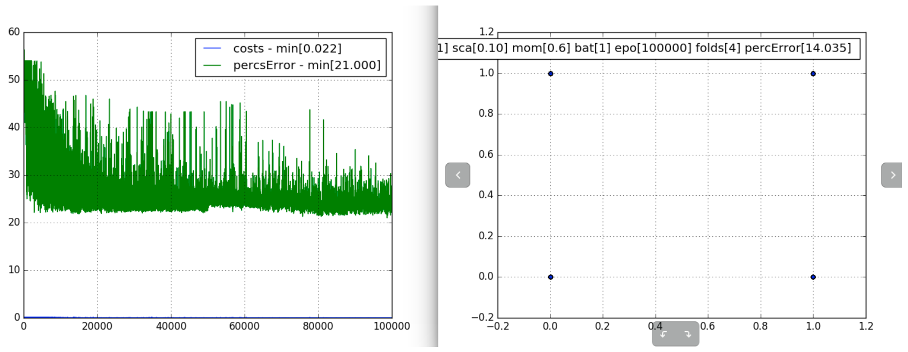


Figure 28: New results (merged).

Again plotting the coordinates for results we get the following outputs:

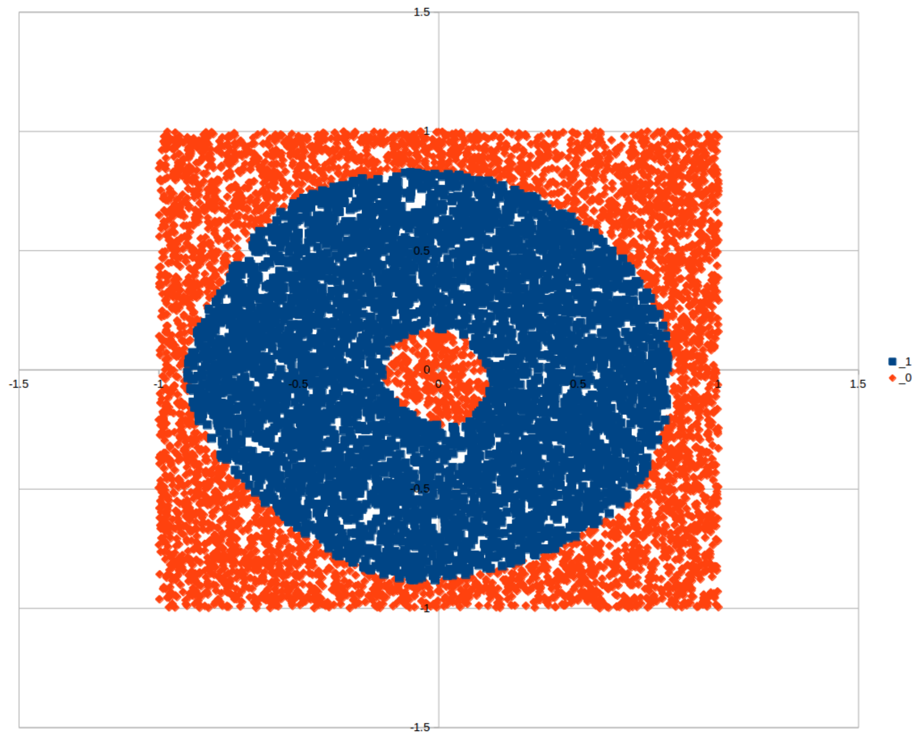


Figure 29: New results using multiple hidden layers (separable).

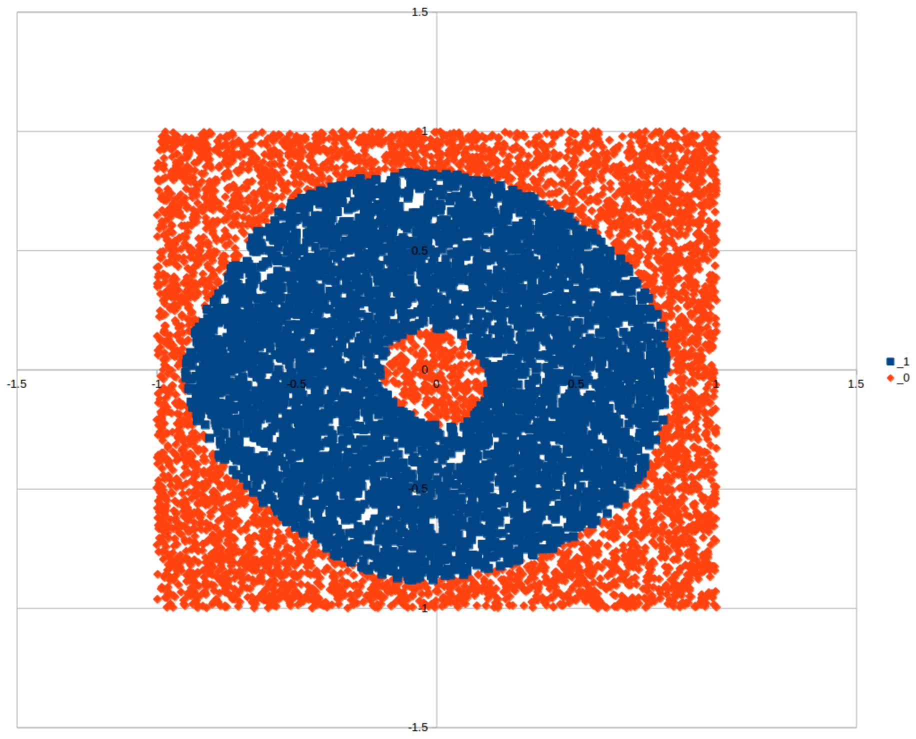
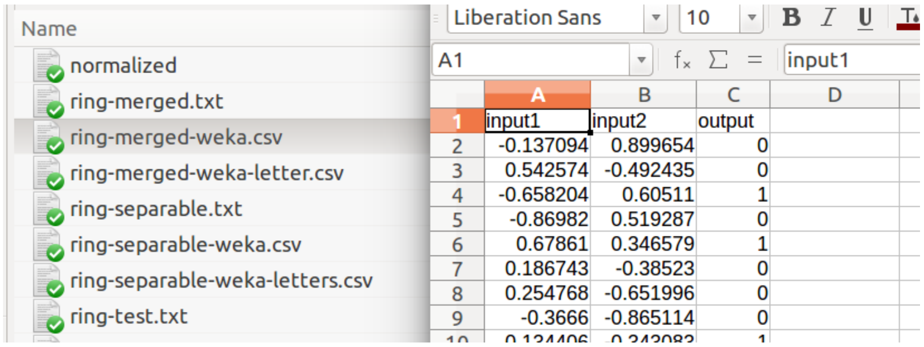


Figure 30: New results using multiple hidden layers (merged).

For this case, and thanks to results plot, we can see an important difference on how the leaning has been improved.

4 Multiple Linear Regression (MLR), using free software

For the case of MLR, just as before we are using Weka software, so we first must assign the training data, in this case we can use the csv file without the letters, but using the original 0 and 1:



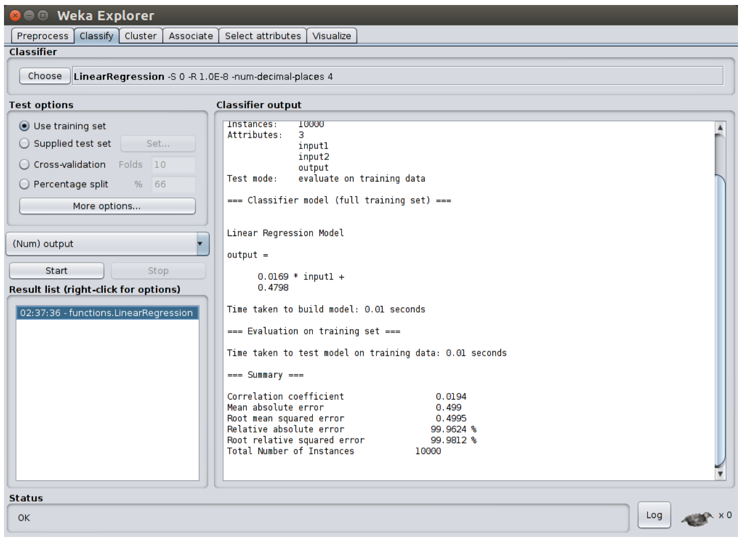
Name
- normalized
- ring-merged.txt
- ring-merged-weka.csv
- ring-merged-weka-letter.csv
- ring-separable.txt
- ring-separable-weka.csv
- ring-separable-weka-letters.csv
- ring-test.txt

A1 | input1

	A	B	C	D
1	input1	input2	output	
2	-0.137094	0.899654	0	
3	0.542574	-0.492435	0	
4	-0.658204	0.60511	1	
5	-0.86982	0.519287	0	
6	0.67861	0.346579	1	
7	0.186743	-0.38523	0	
8	0.254768	-0.651996	0	
9	-0.3666	-0.865114	0	

Figure 31: Training data for MLR.

Then we try a simply training using only training data:



Classifier: LinearRegression -S 0 -R 1.0E-8 -num-decimal-places 4

Test options: [✓ Use training set] Supplied test set; Cross-validation Folds 10; Percentage split % 66

(Num) output

Result list (right-click for options): 02:37:36 - functions.LinearRegression

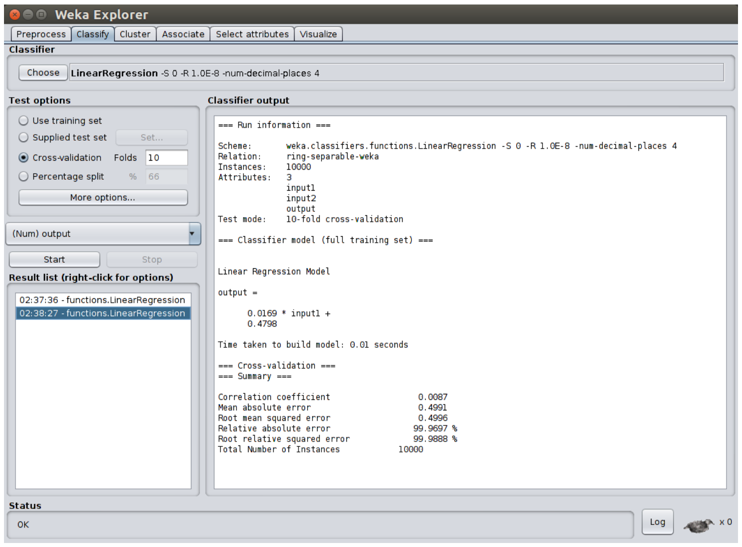
Classifier output

```
Instances:    10000
Attributes:   3
              input1
              input2
              output
Test mode:    evaluate on training data

=== Classifier model (full training set) ===


Linear Regression Model

output =

      0.0169 * input1 +
      0.4798

Time taken to build model: 0.01 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.01 seconds

=== Summary ===

Correlation coefficient                  0.0194
Mean absolute error                      0.499
Root mean squared error                  0.4995
Relative absolute error                 99.9624 %
Root relative squared error             99.9812 %
Total Number of Instances            10000
```

Status: OK

Figure 32: MLR simple training (separable).

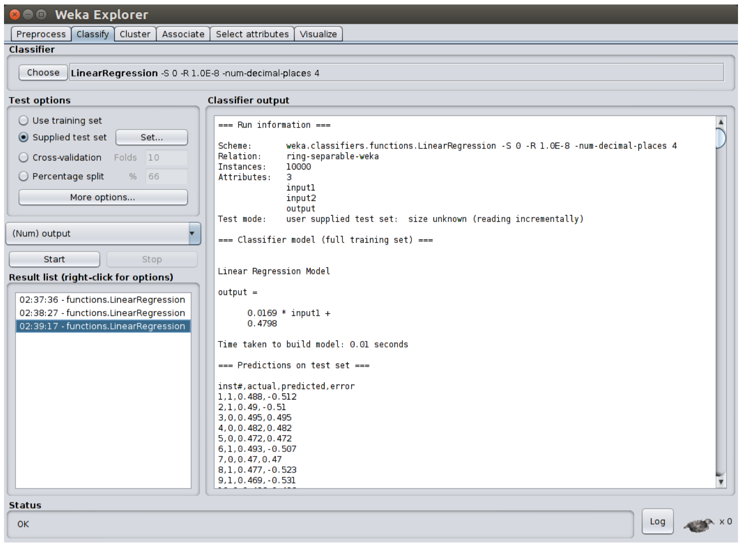
Then another training just like before but using a cross validation with 10 folds:



```
=== Run information ===

Scheme:       weka.classifiers.functions.LinearRegression -S 0 -R 1.0E-8 -num-decimal-places 4
Relation:     ring-separable-weka
Instances:    10000
Attributes:   3
              input1
              input2
              output
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===


Linear Regression Model

output =

      0.0169 * input1 +
      0.4798

Time taken to build model: 0.01 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.0087
Mean absolute error                      0.4991
Root mean squared error                  0.4996
Relative absolute error                 99.9697 %
Root relative squared error             99.9888 %
Total Number of Instances            10000
```

Figure 33: MLR training using 10 folds cross validation (separable).

After that, I tried a new training using the test data and also print all predictions values:

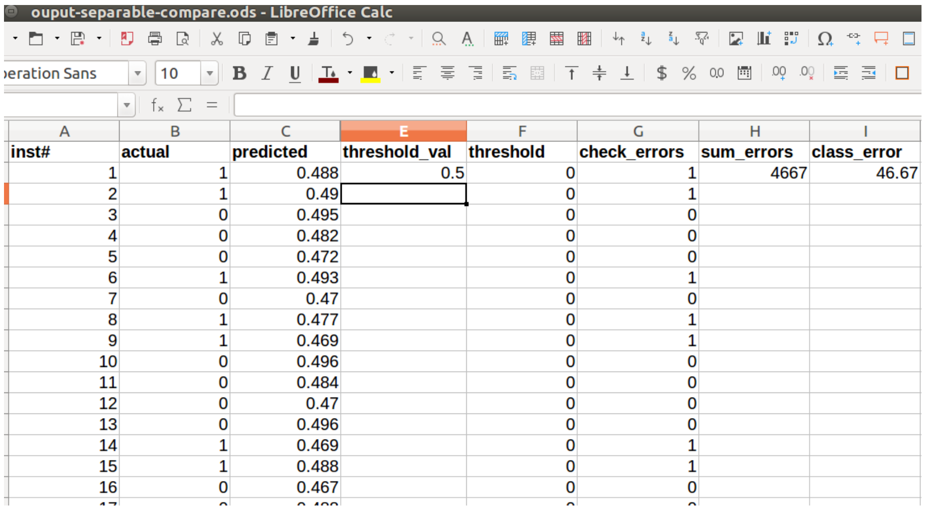


```
=== Run information ===

Scheme:       weka.classifiers.functions.LinearRegression -S 0 -R 1.0E-8 -num-decimal-places 4
Relation:     ring-separable-weka
Instances:    10000
Attributes:   3
              input1
              input2
              output
Test mode:    user supplied test set:  size unknown (reading incrementally)

=== Classifier model (full training set) ===


Linear Regression Model

output =

      0.0169 * input1 +
      0.4798

Time taken to build model: 0.01 seconds

=== Predictions on test set ===

inst#,actual,predicted,error
1,1,0.488,-0.512
2,1,0.49,-0.51
3,0,0.495,0.495
4,0,0.482,0.482
5,0,0.472,0.472
6,1,0.493,-0.507
7,0,0.47,0.47
8,1,0.477,-0.523
9,1,0.469,-0.531
```

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
LinearRegression -S 0 -R 1.0E-8 -num-decimal-places 4
Test options
Use training set
[✓ Supplied test set] Set...
Cross-validation Folds 10
Percentage split % 66
More options...
(Num) output
Start
Stop
Result list (right-click for options)
02:37:36 - functions.LinearRegression
02:38:27 - functions.LinearRegression
02:39:17 - functions.LinearRegression
Classifier output
Status
OK
Log
x 0

Figure 34: MLR including training data (separable).

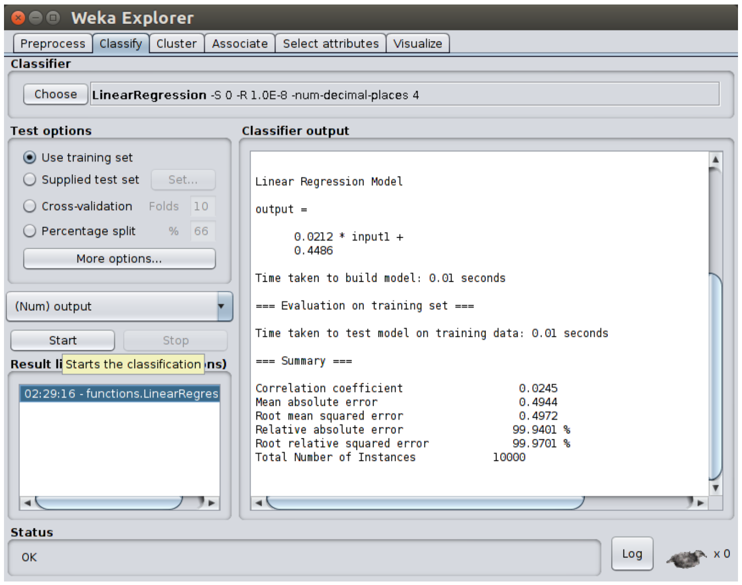
Using prediction values I created a table to compare if values are bigger or smaller than the median (using median as threshold because all values are smaller than 0.5) and so try to detect the approximation of the function.



inst#	actual	predicted	threshold_val	threshold	check_errors	sum_errors	class_error
1	1	0.488	0.5	0	1	4667	46.67
2	1	0.49		0	1		
3	0	0.495		0	0		
4	0	0.482		0	0		
5	0	0.472		0	0		
6	1	0.493		0	1		
7	0	0.47		0	0		
8	1	0.477		0	1		
9	1	0.469		0	1		
10	0	0.496		0	0		
11	0	0.484		0	0		
12	0	0.47		0	0		
13	0	0.496		0	0		
14	1	0.469		0	1		
15	1	0.488		0	1		
16	0	0.467		0	0		

Figure 35: MLR computing classification error (separable).

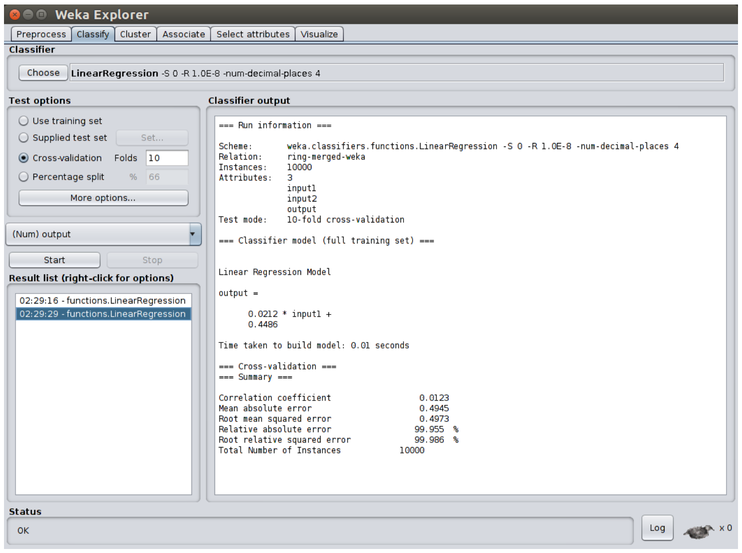
Now following the same steps using MLR with “ring-merged-weka” file:



Classifier: LinearRegression -S 0 -R 1.0E-8 -num-decimal-places 4

```
Linear Regression Model

output =

      0.0212 * input1 +
      0.4486

Time taken to build model: 0.01 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0.01 seconds

=== Summary ===

Correlation coefficient                  0.0245
Mean absolute error                      0.4944
Root mean squared error                  0.4972
Relative absolute error                 99.9401 %
Root relative squared error             99.9701 %
Total Number of Instances            10000
```

Figure 36: MLR simple training (merged).



LinearRegression -S 0 -R 1.0E-8 -num-decimal-places 4

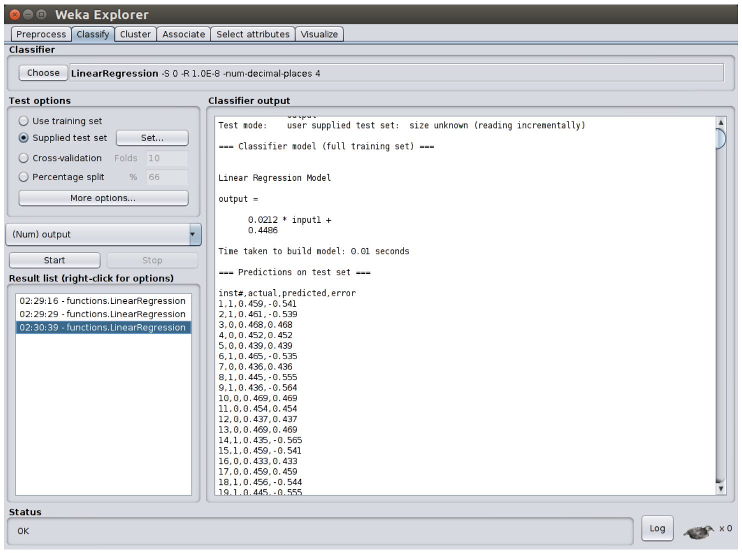
Cross-validation [✓] Folds 10

Percentage split % 66

```
02:29:16 - functions.LinearRegression
02:29:29 - functions.LinearRegression
```

```
=== Run information ===

Scheme:       weka.classifiers.functions.LinearRegression -S 0 -R 1.0E-8 -num-decimal-places 4
Relation:     ring-merged-weka
Instances:    10000
Attributes:   3
              input1
              input2
              output
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===


Linear Regression Model

output =

      0.0212 * input1 +
      0.4486

Time taken to build model: 0.01 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.0123
Mean absolute error                      0.4945
Root mean squared error                  0.4973
Relative absolute error                 99.955  %
Root relative squared error             99.986  %
Total Number of Instances            10000
```

Figure 37: MLR training using 10 folds cross validation (merged).



LinearRegression -S 0 -R 1.0E-8 -num-decimal-places 4

Supplied test set [✓]

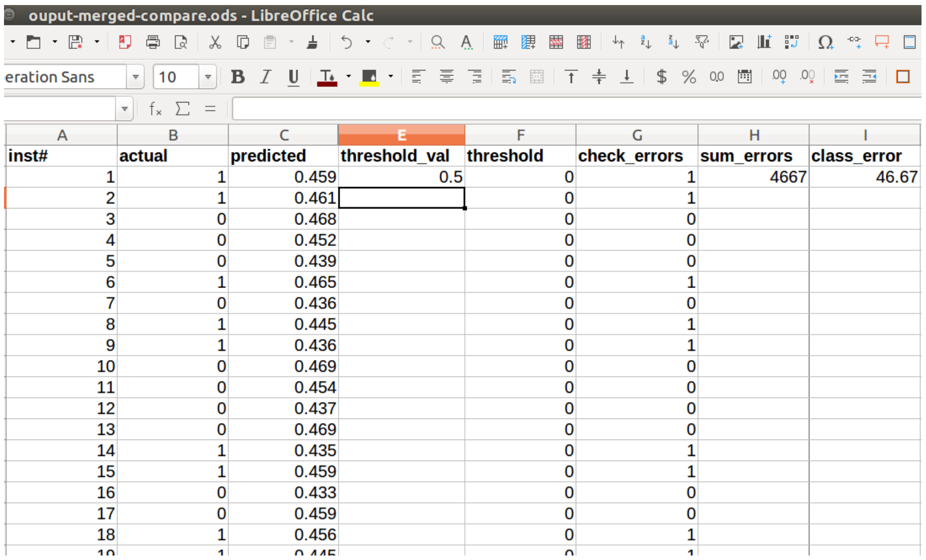
Cross-validation Folds 10

Percentage split % 66

```
02:29:16 - functions.LinearRegression
02:29:29 - functions.LinearRegression
02:30:39 - functions.LinearRegression
```

```
Test mode:    user supplied test set:  size unknown (reading incrementally)

=== Classifier model (full training set) ===


Linear Regression Model

output =

      0.0212 * input1 +
      0.4486

Time taken to build model: 0.01 seconds

=== Predictions on test set ===

inst#,actual,predicted,error
1,1,0.459,-0.541
2,1,0.461,-0.539
3,0,0.468,0.468
4,0,0.452,0.452
5,0,0.439,0.439
6,1,0.465,-0.535
7,0,0.436,0.436
8,1,0.445,-0.555
9,1,0.436,-0.564
10,0,0.469,0.469
11,0,0.454,0.454
12,0,0.437,0.437
13,0,0.469,0.469
14,1,0.435,-0.565
15,1,0.459,-0.541
16,0,0.433,0.433
17,0,0.459,0.459
18,1,0.456,-0.544
19,1,0.445,-0.555
```

Figure 38: MLR including training data (merged).

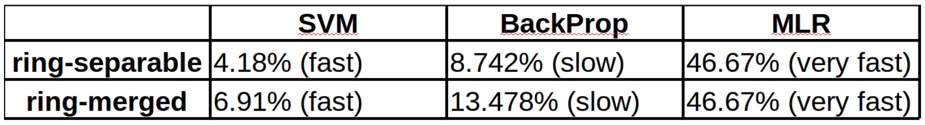


A	B	C	E	F	G	H	I
inst#	actual	predicted	threshold_val	threshold	check_errors	sum_errors	class_error
1	1	0.459	0.5	0	1	4667	46.67
2	1	0.461		0	1		
3	0	0.468		0	0		
4	0	0.452		0	0		
5	0	0.439		0	0		
6	1	0.465		0	1		
7	0	0.436		0	0		
8	1	0.445		0	1		
9	1	0.436		0	1		
10	0	0.469		0	0		
11	0	0.454		0	0		
12	0	0.437		0	0		
13	0	0.469		0	0		
14	1	0.435		0	1		
15	1	0.459		0	1		
16	0	0.433		0	0		
17	0	0.459		0	0		
18	1	0.456		0	1		

Figure 39: MLR computing classification error (merged).

5 Comparison of results

Before starting with the comparison I first want to create a table with the classification error values obtained using each method:



	SVM	BackProp	MLR
ring-separable	4.18% (fast)	8.742% (slow)	46.67% (very fast)
ring-merged	6.91% (fast)	13.478% (slow)	46.67% (very fast)

Figure 40: Comparing classification method for every method.

Watching the results, the first conclusion we can make is that the Multiple Linear Regression is not well suited for this problem, its prediction looks almost like random values, with an error close to 50% in both cases, maybe this is because linear regressions are meant to describe linear relationships between variables, so, if there is a nonlinear relationship, we will have a bad model.

Usually the neural networks have a good performance when data is well separated and easy to segment, and this is not an exception, even when we needed to adapt our code and rethink the architecture and parameters, the results were pretty good and satisfactory, the problem with this solution is the time of execution, this is probably due to the huge amount of information, so it is important having this into account when using this approach.

Finally, I think the winner in this case, having less classification error values and best performance were the Support Vector Machines, this technique worked very well in both situations and I guess it is due to the ease of creating hyperplanes compared with the complexity of training a neural network. Of course I think SVM worked specially good in this case because of the nature of the data, that is to say, all this points could be easily isolated using lines in the plane.

References

[1] OpenCV Tutorials Contributors, ”Introduction to Support Vector Machines”, Feb 02 2017

[2] Wikipedia contributors. ”Support vector machine.” Wikipedia, The Free Encyclopedia. Web. 3 Feb. 2017.