COL774

Report of Assignment 2

### Devansh Dalal (2012CS10224)

Note: All the questions are mainly done in Matlab 2013a or 2014b.

*Q1.* Spam Classification

1. **Using CVX package**

The optimization of SVM dual is implemented in function *p2*. By derivation and comparing to the original dual, the weight vector w and the intercept term b are given by:

And in matrix form, the Q matrix can be given by

Similarly, b and c were obtained to be

The number of support vectors with C=1 obtained are displayed in the table shown below. The complete output of the function with the indices of the respective support vectors are present in the files ‘a\_small(Linear) .txt’, ‘a\_small(Rbf) .txt’, ‘a\_bigcvxLin.txt’,’ a\_bigcvxRbf.txt’ respectively.

|  |  |  |
| --- | --- | --- |
| Type of Kernel/File | Number of features | Number of support vectors |
| Linear Kernel (with train-small) | 996 | 152 |
| Gaussian Kernel (with train-small) | 996 | 258 |
| Linear Kernel (with train) | 1000 | 452 |
| Gaussian Kernel (with train) | 1000 | 752 |

Variation of Number of Support Vectors

**Observations**:

1. **Weight vector w and the intercept term b**:

By calculating the weights and intercepts, the accuracies were obtained as follow:

|  |  |  |
| --- | --- | --- |
| Size of input | Intercept Term | Accuracy (%) |
| Linear Kernel (with train-small) | 0.0832 | 91.3 % (913/1000) |
| Linear Kernel (with train) | -0.08522 | 98.7 % (987/1000) \* |
| Gaussian Kernel ( with train-small ) | 0.5047 | 87 % (870/1000) |
| Gaussian Kernel ( with train ) | 0.3394 | 96.9 % (969/1000) \* |

Accuracies and Intercept terms with the size of File in linear and Gaussian Kernels

**Observations**:

* The above result justifies that with bigger training set the SVM for spam classification learnt much better as compared to train-small file.
* Linear kernel performs marginally better than the Gaussian kernel as clear from the accuracy values.

1. **Using the Gaussian kernel:**

The number of support vectors and the accuracies obtained have already been shown in the tables above. Now the Q matrix is changed such that

where

Whereas and are not changed.

The Gaussian and Linear kernel for the big file took almost 6 hr each on GCL machines for this part.

1. **Using the LibSvm Library:**

The outputs for this part are stored in files ‘d\_gauss\_libsvm\_big.txt’, ‘d\_gauss\_libsvm\_small.txt’, ‘d\_lin\_libsvm\_small.txt’ and ‘d\_linear\_libsvm\_big.txt’.

|  |  |  |
| --- | --- | --- |
| Size of input | Accuracy (%) | Number of support vectors |
| Linear Kernel (with train-small) | 91.3 % (913/1000) | 152 |
| Linear Kernel (with train) | 98.7 % (987/1000) | 452 |
| Gaussian Kernel ( with train-small ) | 89.2% (892/1000) | 530 |
| Gaussian Kernel ( with train ) | 98.7 % (987/1000) | 1982 |

Accuracies and SVs with different parameters using LIBSVM

**Observations**:

* The LibSVM performs better as compared to CVX implementation of the same problem. This is because CVX is general purpose quadratic optimization software but LibSVM is highly optimized for support vector learning.
* The number of support vector using CVX and LibSVM were almost found to be same in number as given in plot. This cross verifies the CVX implementation also.

*Q2. Digit Recognition*

1. **Visualization**:

The *visualize()* function takes a row vector of size 784 as input and shows the corresponding image of size 28 x 28 pixels. A few examples are shown below.



1. **Binary Classifier for digit 3 and 8**:

I have implemented a general **N** output layer network in file *nn.m* and it is called using the *run38()* functions which parses the data present in **minist\_bin38.mat** and call *nn()* for binary classification using neural network. I have added the basis to the input and the output of hidden layer and initialized the initial values of and with random doubles in range -0.1 to 0.1 to allow network to learn faster in the beginning and then fine tune at later iterations. And the training examples are shuffled properly before starting neural learning.

Stopping criteria

Where

**Observations:**

* I have used the difference in the average values of as the stopping criteria mainly but I don’t check this condition until a threshold number of iterations of main loop have been executed. And there is a limit of the maximum number of iterations also in the same way.
* A better approach is that we maintain a validation set of training examples apart from training and testing set of points. Then our algorithm would be to learn until a desired accuracy on validation set is obtained.

1. **Accuracies and training times**:

|  |  |  |
| --- | --- | --- |
| No of iterations | Accuracy( % ) | Time to learn |
| 1 | 66.330645 | 11.433763 sec |
| 2 | 82.560484 | 23.371798 sec |
| 3 | 92.641129 | 31.634243 sec |
| 4 | 93.497984 | 44.624940 sec |
| 5 | 94.556452 | 53.550198 sec |
| 10 | 96.471774 | 2 minutes |
| 50 | 96.8750 | 10 minutes |
| 100 | 97.983871 | 20 minutes |
| 5000 | 99.495968 | More than 10 hrs |

variations of Accuracies for Binary classification with iterations (4GB ram, i5)

|  |  |  |  |
| --- | --- | --- | --- |
| Eps | No of iterations | Accuracy (%) | Time |
| 0.1 | 1 | 80.89717 | 6.925768 sec |
| 0.01 | 8 | 97.4729 | 26.472261 sec |
| 0.005 | 9 | 97.78226 | 30.887746 sec |
| 0.001 | 18 | 98.286290 | 61.373354 sec |
| 0.0005 | 23 | 98.084677 | 79.551893 sec |
| 0.0001 | 45 | 98.689516 | 169.947914 |

Accuracies for Binary classification with stopping criteria (4GB ram, i5)

1. **Multiclass classification for all 10 digits**:

In all the digits classification, I have used 10 different outputs for all the 10 different digits. Also for getting same accuracy as with binary classification this neural network takes longer time.

|  |  |  |
| --- | --- | --- |
| No of iterations | Accuracy( % ) | Time to learn |
| 1 | 85.2700 | 15.29 sec |
| 10 | 91.8900 | 145.5 sec |
| 100 | 95.2600 | 27.95 min |
| 500 | 96.1200 | 2 hours |
| 5000 | 96.6000 | Around 1 Day |

Accuracies and times for Mnist\_all dataset( 8GB ram, i7,8 cores)

**Observations:**

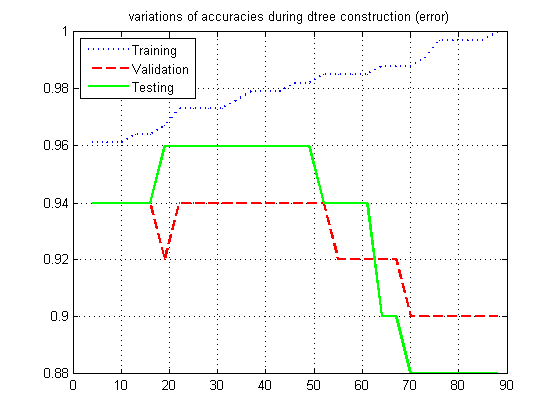
* Multiclass neural network takes longer times and more number of iterations to obtain a particular accuracy as compared to binary classifier.
* We could have done it with only 4 outputs but in that setting learning is not good as compared to learning with 10 different outputs.
* After around 100 iterations the network doesn’t learn much uptill 5000 iterations. Basically excessive learning may lead to over fitting in such cases.

*Q 3.* Decision Tree Learning

The algorithms for Decision tree construction and choosing the best attributes are implemented by the function **growTree()** and file **chooseBest** respectively. Check function checks the accuracy of the input testing examples with respect to current number of nodes in the tree which are given by function **tsize**. The raw input is parsed in appropriate format using the python script *parse.py*.

1. **With error as the splitting criteria:**

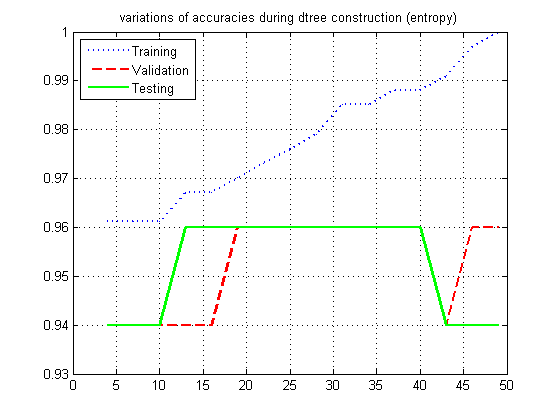
In this part *chooseBest* function gives the attribute with the minimum error or maximum accuracy. The accuracies values are plotted with respect to the current size of the tree excluding the leaf nodes.



|  |  |
| --- | --- |
| Parameters | Observations |
| Tree size | 88 nodes |
| training accuracy | 1 |
| validation accuracy | 0.900 |
| testing accuracy | 0.8600 |

1. **With Entropy as the splitting criteria :**

In this part the splitting criteria is the information gain. The corresponding graph is plotted similar to part(a).



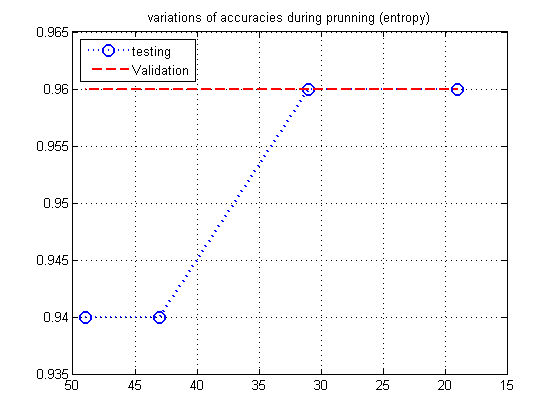
|  |  |
| --- | --- |
| Parameters | Observations |
| Tree size | 49 nodes |
| training accuracy | 1 |
| validation accuracy | 0.9400 |
| testing accuracy | 0.9400 |

**Observations:**

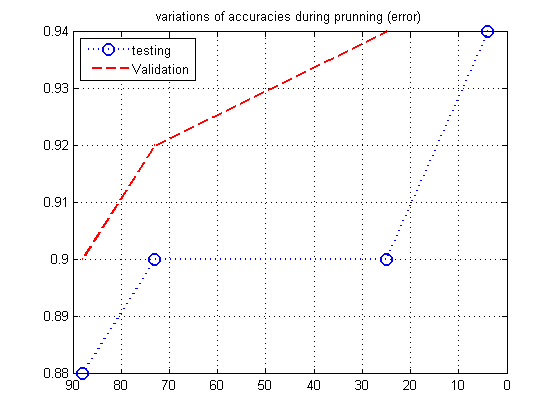
* Training accuracy increases continuously with increase in the size of tree successively for both parts. While the validation and testing accuracies goes down after some specific size of tree due to over fitting.
* The testing and validation set accuracies goes down more when ‘error’ is the splitting criteria as compared to ‘Information gain’ as the stopping criteria verifying that ‘error’ is not a good splitting criteria

1. **Post-pruning:**

The prune() function applies the required part as asked in this part and the variation in accuracy values are plotted for ‘entropy’ and ‘error’ based constructed trees.



|  |  |  |
| --- | --- | --- |
| Parameters | ‘error’ | ‘information gain’ |
| Tree size(after pruning) | 4 nodes | 19 nodes |
| Final validation accuracy | 0.9400 | 0.9600 |
| Final testing accuracy | 0.9400 | 0.9600 |



**Observations:**

* The validation and test set accuracies increases during pruning for both kind of trees constructed above and becomes equal to 0.96 and 0.94 respectively.
* The size of the trees after pruning decreases to such small values due to small number of examples in the validation set. So validation set should be large enough not to under fit the data.

1. **Dealing with ‘?’s:**

The strategy of dealing with unknown attributes can be:

* Assign the value that is most common among training examples at node n to the missing attributes in some examples.
* Let A be a missing attribute for some examples and the current node splits about A also. Then calculate what fraction of examples with known value of A have ‘y’ value. For each example, assign the value of missing attribute according to the probability of ‘y’ and ‘n’ at that node.
* We can extend 2nd point, we push different fractions of a training example with unknown attribute into 2 branches according to the probability calculated above and then incorporate such examples in information gain based calculations to have better decision tree construction as compared to just ignoring them.