COL774 – Assignment 2

- **By Nishant Kumar**
2012CS10239

Problem – 1

(a) The SVM dual objective is as follows:

$$maximize_{\alpha} \sum_i \alpha_i - \sum_i \frac{1}{2} \alpha_i \alpha_j y^i y^j K(x^i, x^j)$$

$$subject\ to\ 0 \leq \alpha \leq C\ and$$

$$\sum_i \alpha_i y^i = 0$$

In the linear kernel case, $K(x^i, x^j) = {x^i}^T x^j$

And in the Gaussian kernel case, $K(x^i, x^j) = \exp(-\gamma \left|\left|x^i - x^j\right|\right|^2)$

For training on train-small,

	Value of b	Number of Support Vectors	Accuracy
Linear Kernel	-0.0464	151	91.2%
Gaussian Kernel	-0.372	536	89.4%

For training on train,

	Value of b	Number of Support Vectors	Accuracy
Linear Kernel	-0.0481	452	98.7%
Gaussian Kernel ****	-	1836	98%

****: Since the Gaussian kernel with cvx runs out of memory when run on the whole train dataset, a subset of train was used instead, with 7500 examples.

As can be seen from the above stats, the linear kernel gives better performance than Gaussian kernel almost everytime. This may be the case as the feature space corresponding to the Gaussian kernel has infinite number of dimensions, while for linear kernel its just the original features. Hence Gaussian kernel may be trying to overfit the train data, while the train data may be better explained by the original input features.

Another point worth noting is that the number of support vectors in case of Gaussian kernel is much more as compared to the case of linear kernel.

(d)

Using libSVM, the following accuracies were obtained:

For training on train-small,

	Number of Support Vectors	Accuracy
Linear Kernel	151	91.3%
Gaussian Kernel	530	89.2%

For training on train,

	Number of Support Vectors	Accuracy
Linear Kernel	452	98.7%
Gaussian Kernel	1985	98.7%

LibSVM gives almost the same accuracy and same number of support vectors as in the case of CVX. The only important difference is that since libSVM is a customized solver, it runs amazingly fast as compared to CVX. While on train data, libSVM takes around less than a minute to run, CVX takes hours on end and in case of Gaussian kernel is never able to run on the whole train dataset.

Problem – 2

(b)

To create the mnist_bin38.mat file, all the training examples were chosen from train3 and train8 and randomly shuffled, after adding appropriate outputs to each example.

The neural network used has only one output unit with 0 representing 3 and 1 representing 8 as output.

In the stochastic gradient algorithm, an inner loop iterates over all the examples and keeps updating theta for both the layers. Once after iterating over all the examples, it chooses the last value of theta and calculates, $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\sum_{j=1}^{l}(y_l - o_l)^2$. If in 2 successive iterations, the value of $J(\theta)$ differs by more than ϵ, then break.

Also, the learning rate used was $\alpha_t = \frac{1}{\sqrt{t}}$, where

$t = iteration\ number, updated\ over\ iteraing\ through\ each\ example.$

Using this criteria, we get the following table for initial $\theta \in [-0.1,0.1]$, chosen randomly.

Eps	Time taken(seconds)	Accuracy over test3(%)	Accuracy over test8(%)
10^{-3}	35.04	96.03	96.09
10^{-4}	121.8	97.02	97.5
10^{-5}	630.52	98.41	97.22

Clearly, stopping criteria with $\epsilon = 10^{-4}$ is the best to use in this setting.

(d)

In this case, 10 output units will be needed for the neural network. The outputs correspond to the confidence over classifying that example in a particular class. The one output with the most positive value is the one that is chosen to reflect the class for that example.

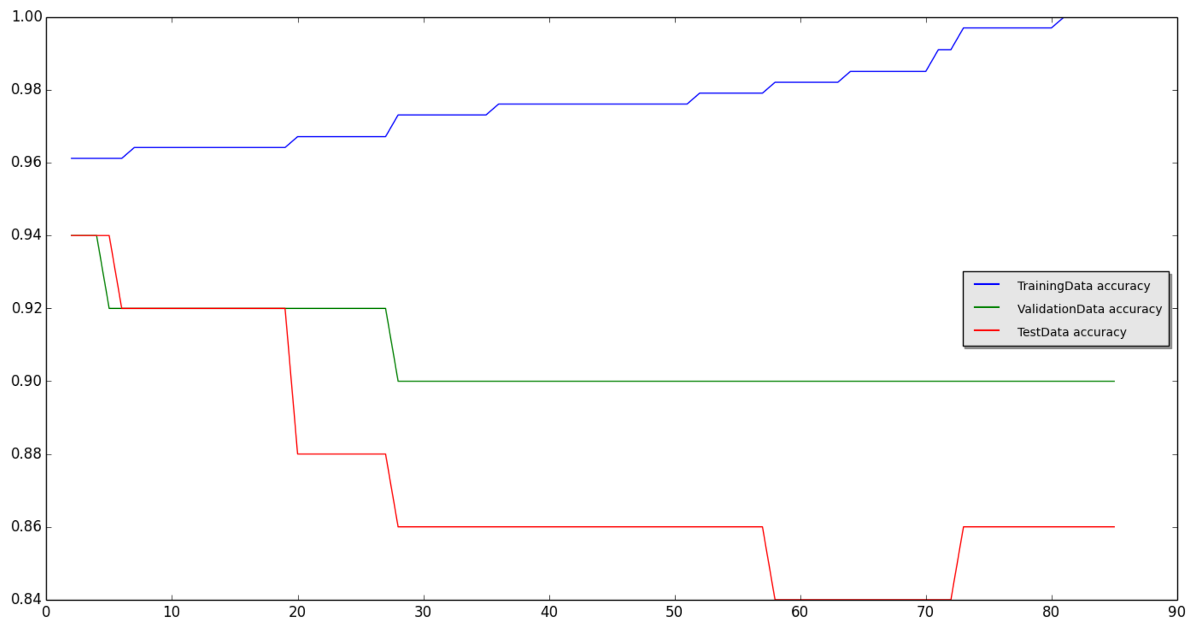
Again to select the example for training in stochastic gradient descent, a file called mnist_bin_shuffled was created in which after adding the apt outputs from all the train classes, all the train data was shuffled.

With the same initial theta $\theta \in [-0.1,0.1]$, we get obtain the following stopping criteria and accuracies over the neural network.

Eps	Time taken(seconds)	Accuracy over all the test data(%)
10^{-3}	210.66	91.27
10^{-4}	471.04	92.79

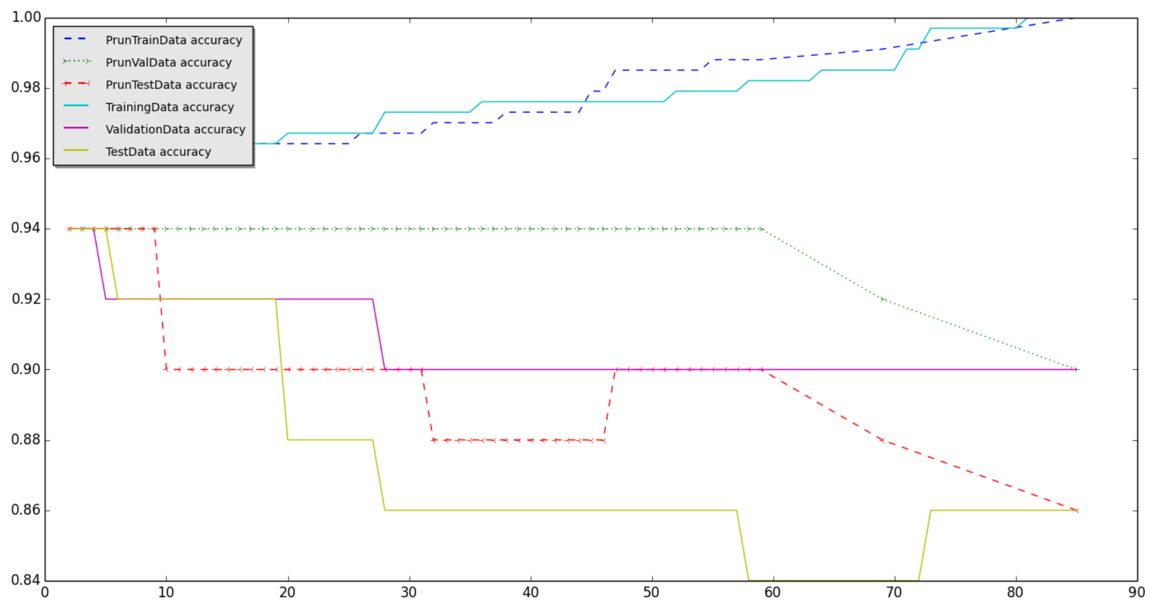
Clearly, the training times of the algorithm are much higher in this setting than in the previous setting.

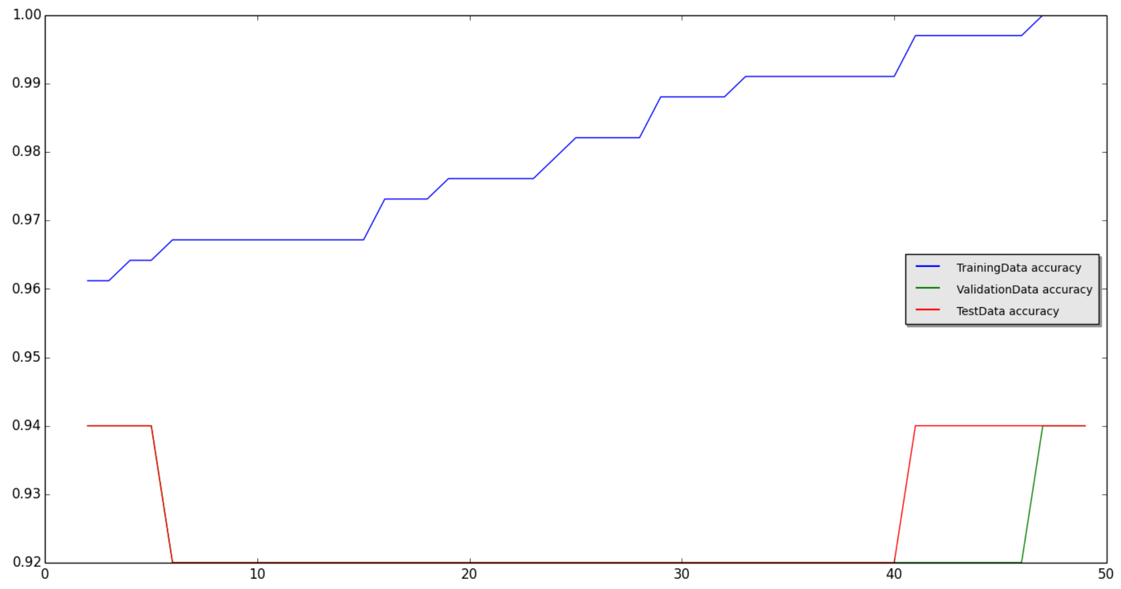
Problem – 3



Above figure : 3(a) : With error as metric and no pruning

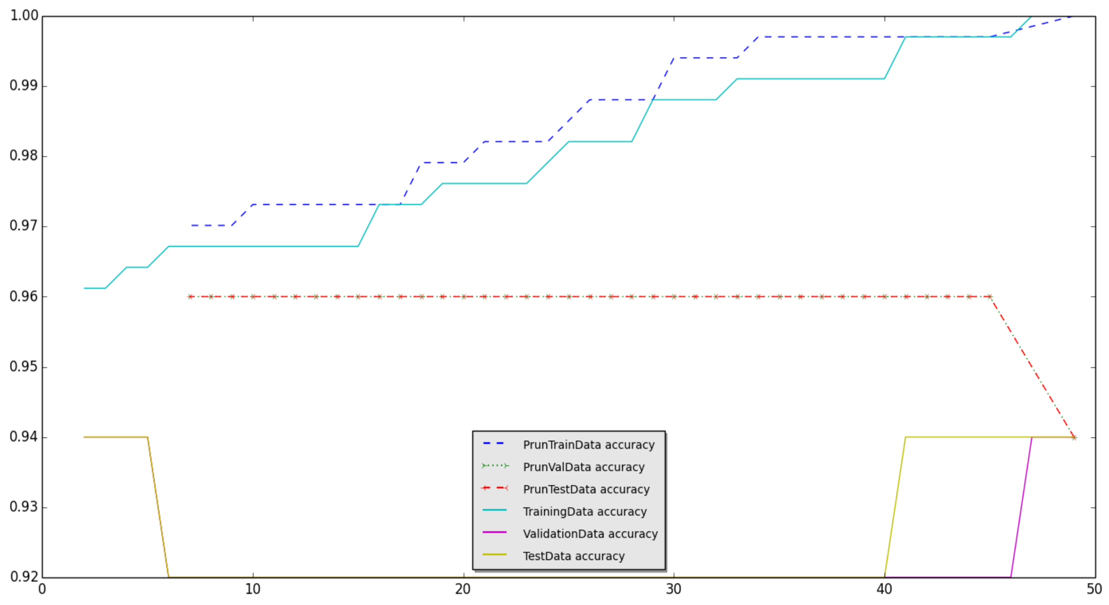
Below figure : 3(b) : With error as metric and pruning





Above figure : 3(C) : With IG as metric and no pruning

Below figure : 3(d) : With IG as metric and pruning



NOTE: Python was used in coding this question

(a) Using net error as the criterion to split on:

On using this criterion the graph as in figure 3(a) was obtained. The tree was constructed on the basis of 3-way split – with each attribute having the possible values of 'y','n' or '?'.

The decision tree grown had a depth of 8 and a total number of nodes of 85.

The x axis has the number of nodes plotted and the y axis has the accuracies of the train, validation and test data plotted. The train accuracy keeps on increasing and finally reaches 1 when the tree has fully grown. The test and validation accuracies keep decreasing, on the other hand. This happens because the decision tree tries to overfit the train data and so train accuracy reaches 1, while the test and validation accuracies decrease.

(b) Using information gain as the criterion to split on:

On using this, the graph as in figure 3(C) was obtained. The tree was constructed on the basis of 3-way split – with each attribute having the possible values of 'y','n' or '?'.

The decision tree grown had a depth of 7 and a total number of nodes of 49.

A similar pattern of overfitting was observed in this case also, as is evident from the graph. But only in this case, there is a small increase in the test and validation accuracies as the tree grows to full size.

Clearly this tree has structure quite different from the one obtained using error as the criterion. But another point worth noting is that in this case the final validation and test accuracies were much higher than in the case above. In this case, the final validation and test accuracies were 0.94 each, whereas in the previous case, it was 0.90 and 0.86 respectively. So, clearly although both lead to overfitting, Information Gain is still a better criteria to split on.

(C) On using post-pruning to prune the decision tree:

As stated in the question, to prune the decision tree, a depth first search was performed among all the nodes of the tree to find the node, s.t. removing it leads to maximum increase in the validation accuracy. And this process was applied in a loop until pruning led to decrease in the validation accuracy.

The only important thing to note in this case was that if only those nodes were removed that led to increase in validation accuracy, then depending on how a single example was being predicted (like always predicting 'democrat' if at some leaf node no. of examples of 'democrats' = no. of examples of 'republicans'), there was no or very less pruning observed. So, to properly observe pruning, those nodes were also removed s.t. removing them led to same validation accuracy.

Hence, pruning the tree obtained in 3(b), we get the final number of nodes as: 49->7 and both the validation and test accuracies increase from the previous 0.94 to 0.96.

As an extra exercise, on running a similar pruning exercise on the decision tree obtained in 3(a), the number of nodes obtained reduced from 85 in the original tree to 2 in the final tree and the validation and test accuracies increased.

(d) On how to treat missing values '?':

One of the easy ways to deal with such cases can simply to throw away the examples with missing attributes. But sometimes this can lead to most of the data getting thrown away. Like for example, in our dataset, most the training examples have some or the attribute as missing.

Another principled way can be to predict the missing values using some model and then use the predicted attribute values to fill in and work through. To predict the new values, one can first form a model to predict attribute values, then use new model with values predicted previously to re-predict, and keep repeating until convergence. Once one is sure that the filled in values are good enough, one can use those values to fill in the data.