Kernel Methods and Pattern Analysis
Assignment 2

Parth Joshi (CS09B051), Sudharshan GK (CS13M050) & Umesh Kanoja (CS13M024)

April 8, 2014

1 Methodology

1. **Computing Width for Gaussian Radial Basis Functions :** The width for the case of Gaussian radial basis functions was set based on the number of dimensions and the range of the input space, approximated from the given data. Specifically, if α is the maximum distance between two points in the given dataset and m is the number of basis functions, the width is set to be $\frac{\alpha}{2m}$ [SH].

2. **Data Preprocessing :** We have made use of the MATLAB neural network toolbox for most of the portions of the assignment that dealt with neural networks. The API provided by the toolbox provides options to normalize the input data in the range $[-1, 1]$. For the cases where we made use of our own code (RBFNN, Bayes classifier) the datasets were all normalized to the range $[0, 1]$ as part of pre-processing.

3. **Use of Scaled Conjugate Gradient algorithm :** Conjugate gradient methods (CGMs) are general-purpose second-order techniques for numerical optimization of functions of several variables. Second order means that they take the the second derivative (or the Hessian matrix) of the objective function into account. While in gradient descent, we usually proceed in the direction of *steepest* slope of the error function, the basic difference in CGMs is that we proceed in a direction that is *conjugate* to the directions of the previous steps [SCG]. Scaled conjugate gradient differs from other conjugate gradient methods in that for computational efficiency it approximates the second derivative as an *average* rather than *instantaneous* rate of change of the first derivative. Since if the Hessian is indefinite, the performance of the algorithm is poor, it regulates the indefiniteness of the Hessian using an empirical parameter. The two parameters used by this algorithm are σ, which is the weight perturbation for approximating the Hessian and λ, which is the regularization term for making the Hessian positive definite. Since the MATLAB toolbox provides several different conjugate gradient algorithms and this algorithm has been observed to be considerably faster than other CGMs in practice [SCG], we have made use of this as the conjugate gradient algorithm for this assignment.

4. **Use of the Rumelhart-McClelland Momentum Based Updates :** In class, the gradient descent with momentum based weight update rule for the m^{th} iteration was given as follows

$$\Delta\overline{w}_m = -\eta\overline{\nabla}\overline{w} + \alpha\Delta\overline{w}_{m-1}$$

where η is the learning rate and α the momentum factor. However, MATLAB's toolbox implements the version of the momentum update as originally given by Rumelhart-McClelland, in which α is used as an exponential smoothing factor

$$\Delta\overline{w}_m = -\eta\left[(1-\alpha)\overline{\nabla}\overline{w} + \alpha\Delta\overline{w}_{m-1}\right] \tag{1}$$

This results in an exponentially decaying weighted average over past weight changes [SK, pg 176].

5. **Early stopping :** The validation data that we are given can often be used to determine an early stopping criterion that attempts to ensure good generalization on unseen data. Early stopping terminates the training process when the error on the validation data keeps increasing for a number of consecutive epochs — a sign of over-fitting [SK, pg 192]. We have mentioned it explicitly wherever this is used as the termination criterion for our learnt model.

6. **Reciever Operating Characteristics :** We have made use of ROC curves in some cases to compare the performance of different classifiers. These curves plot the hit rate against the false alarm rate and gives a visual representation of the *discriminability* between the different classes [DH, pg 49].

Kindly refer to A Few Notes on the Implementation for some of the other details and the issues faced in the implementation.

Part I

Classification

2 Bivariate Datasets

2.1 Linearly separable classes

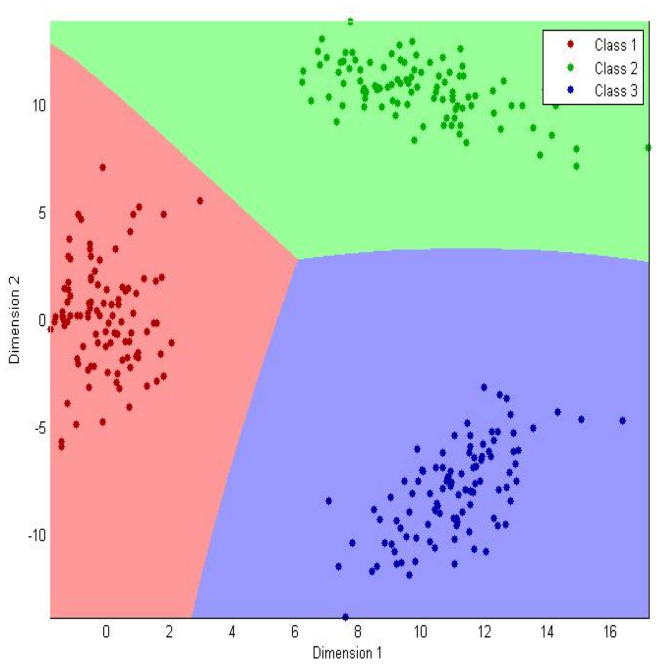
The linearly separable dataset we recieved consists of three classes. The following three models were used for classification

- A Bayes classifier using Gaussian class conditional densities. Each class is modelled using a single Gaussian. Full covariance matrices are considered for each class.
- A one-vs-one (1V1) binary perceptron classifier for multiple classes. To get the class that a particular point belongs to, we used the following rule. We first compute the ${}^3C_2$ discriminant functions $g_{ij}(x)$ for every pair of classes i and j for each point x. Each discriminant function gives us a vote for one particular class. Now, to get the net discriminant function value $g_k(x)$ for any class, we take all those $g_{ij}(x)$, $i = k$ or $j = k$ that vote for class k and take their average. The class with the largest such $g_k(x)$ is said to be the class to which point x belongs. All the weights are initialized to zero before training.
- Multi-layer feed forward neural network (MLFFNN) classifier with the following configuration parameters
  - 1 hidden layer consisting of 4 nodes
  - Hyperbolic tangent activation function at the hidden layer and logistic sigmoidal function at the output layer
  - Gradient descent with learning rate $\eta = 50$ and momentum factor $\alpha = 0.5$
  - Initialization of weights using random values in the range $[-1, 1]$
  - The training terminated after the gradient dropped below the threshold of 10^{-5}

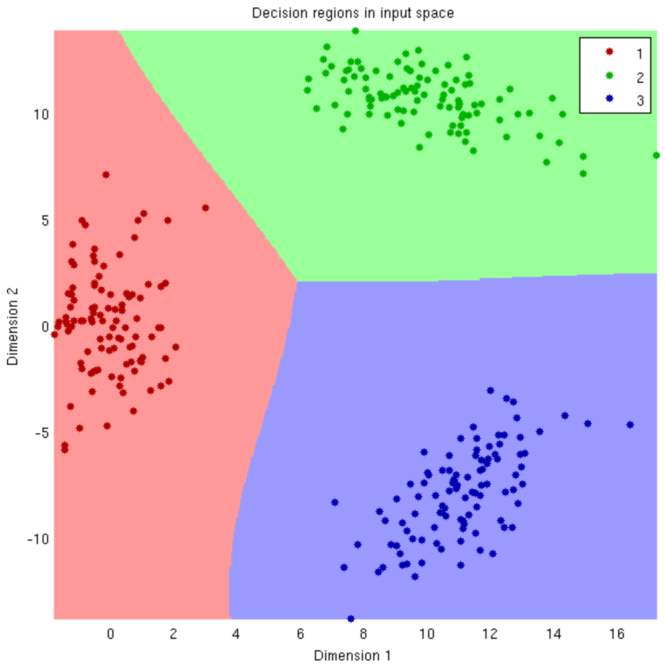
  The decision region plots obtained from each of the three models are shown in figure 1. The confusion matrix and classification accuracy for all models is the same and is shown in figure 2.

Figure 1: Decision region plots for different classifiers on the linearly separable dataset. The points in the test dataset are superimposed on the decision regions

(a) Bayes classifier



(b) MLFFNN classifier



(c) Perceptron classifier

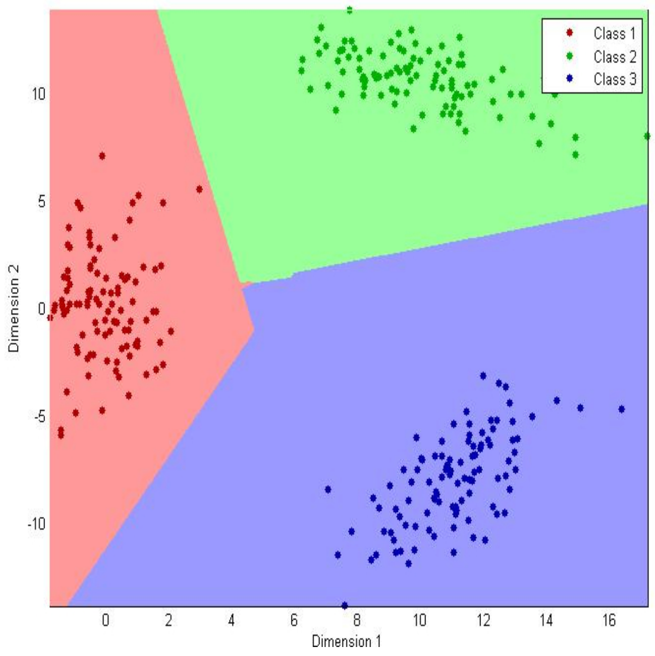
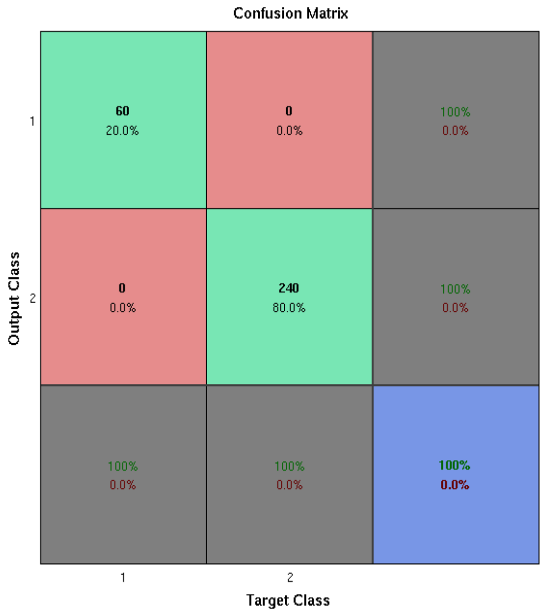


Figure 2: Confusion matrix and classification accuracy of MLFNN, Bayes classifier and perceptron on linearly separable data



Observations

- The linearly separable case is the easiest to classify and all three models are able to give 100% accuracy on the given data.
- The decision surfaces obtained from the perceptron are always linear. Those obtained from the Bayes classifier are quadratic. However, the surfaces obtained from the MLFFNN model are S-shaped due to the sigmoidal nature of the activation functions (figure 1b).
- The algorithm converges faster as we increase the learning rate. We experimented with η values as large as 215 with 0 momentum factor and found that the algorithm was able to converge to the solution. However, the solutions with very large η did not give very good generalization and the decision regions so obtained were found to be very close to the points in the test set for the classes. Note that with larger values of η, the solution diverged and network paralysis was observed with the performance remaining unchanged from iteration to iteration.

Inferences

- Any of the above models can be used for classifying linearly separable classes with good results.
- Although the binary perceptron is able to provably classify 2 linearly separable classes correctly, its extension to multiple classes in the form of 1V1 leaves much to be desired.
  - Firstly, it requires the learning of ${}^kC_2$ discriminant functions, which can be prohibitive if k is large.
  - Secondly, each individual model is a binary classifier and learns only how to discriminate between those two classes without taking the other classes into account at all.
  - Thirdly, the decision logic used to combine the outputs of these individual binary perceptrons into a single classifier is complex and heuristic in nature and is not guaranteed to generalize well for larger k where there will be many ambiguous regions. In fact, even for the given dataset, for some

of the runs starting with different random initial weights, the classifier managed to misclassify one or two points in the test dataset after training. There are other generalizations of perceptrons to multiple classes that involve simultaneous learning of k discriminant functions for k classes that might work better in such cases [DH, pg 265].

- The observations with regard to the learning rate were a cause of surprise since even with large values of η, we were unable to see any oscillations in the error curve. The model either converged very quickly to the solution or the weights became so large that the solution diverged. Increasing the learning rate too much caused the activation function to always remain in its saturating region resulting in no change in the weights from epoch to epoch. Some of our experiments with different values of the parameters η and α are detailed in Fluctuations in the error as we approach the minima.

2.2 Non-linearly separable classes

The given dataset consists of two classes. The following classifier models were built for this dataset

- Bayes classifier using Gaussian class conditional densities. From figure 4a, it can be seen that the two classes present different distributions. While the inner class (class 1) has a unimodal distribution about the centre (as is evident from the nearly unvarying value of the likelihood function as the number of cluster componenets increases in figure 3), the outer class (class 2) presents a multimodal distribution with the modes located in a concentric ring about the centre that cannot be estimated using only a single Gaussian. Therefore, we have modelled the inner class using a single Gaussian and the outer class using a Gaussian mixture model having 4 components. Full covariance matrices are considered for each class. The determination of the choice of 4 clusters was done by plotting the likelihood function as the number of components increases and choosing the elbow point of the plot.

- Multi-layer feed forward neural network (MLFFNN) classifier with the following configuration parameters

  - 1 hidden layer consisting of 5 nodes
  - Hyperbolic tangent activation function at the hidden layer and logistic sigmoidal function at the output layer
  - Gradient descent with learning rate $\eta = 20$ and momentum factor $\alpha = 0.5$
  - Initialization of weights using random values in the range $[-1, 1]$
  - The training terminated after the gradient dropped below the threshold of 10^{-5}

  The decision region plots obtained from the two models are shown in figure 4. The confusion matrix and classification accuracy for both models is the same and is shown in figure 5.

Figure 3: Variation of the likelihood as the number of mixture components increases for the Bayes classifier

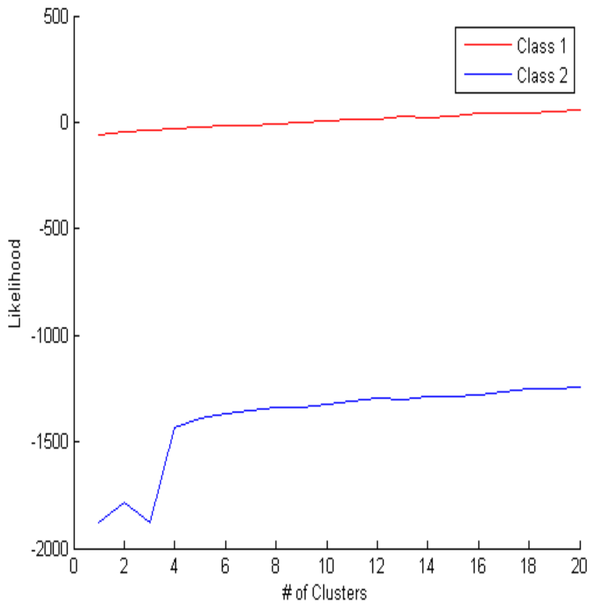
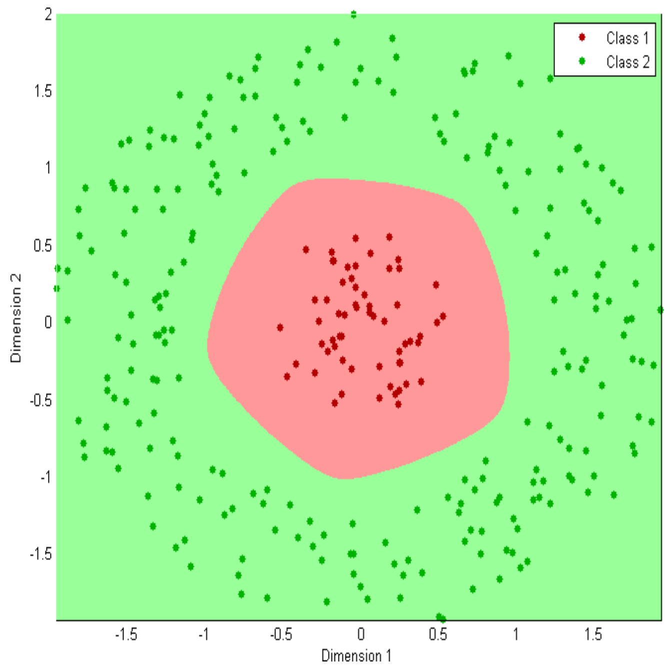


Figure 4: Decision region plots for different classifiers on the non-linearly separable dataset

(a) Bayes classifier



(b) MLFFNN classifier

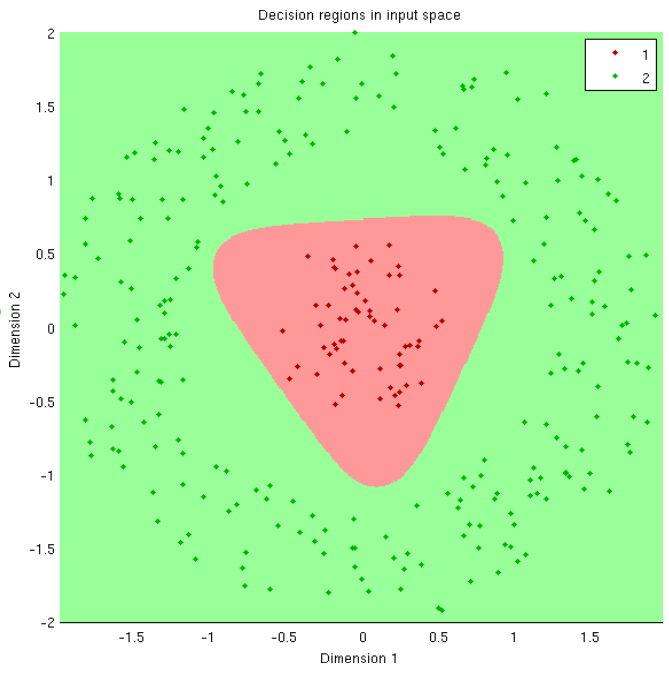
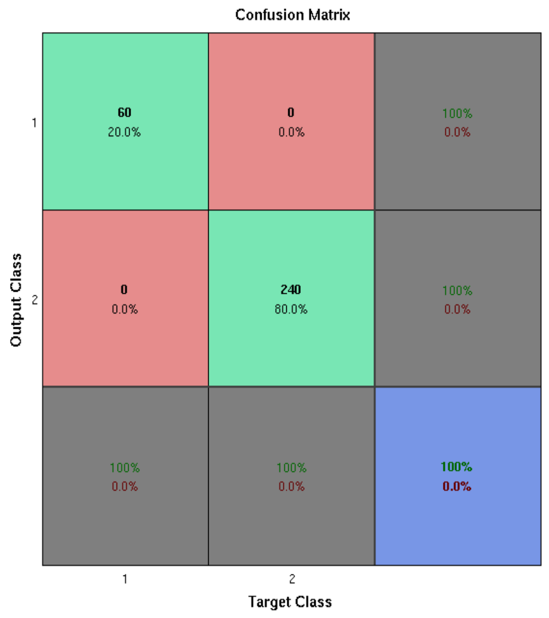


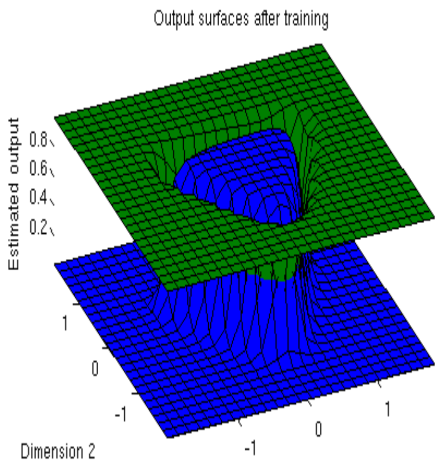
Figure 5: Confusion matrix and classification accuracy of MLFNN and Bayes classifier on non-linearly separable data


Confusion Matrix

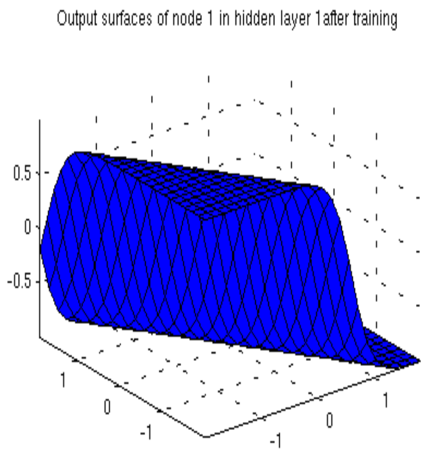
Output Class \ Target Class	1	2	
1	60 20.0%	0 0.0%	100% 0.0%
2	0 0.0%	240 80.0%	100% 0.0%
	100% 0.0%	100% 0.0%	100% 0.0%

Figure 6: Outputs of hidden and output layer nodes in MLFFNN

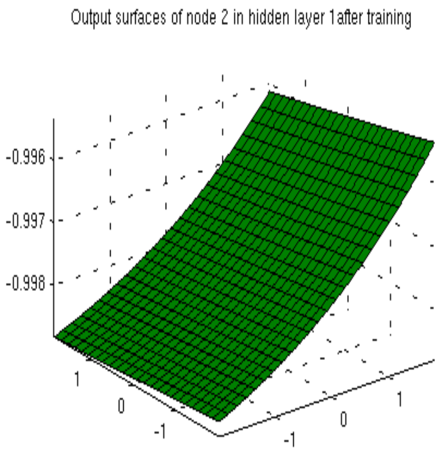
(a)



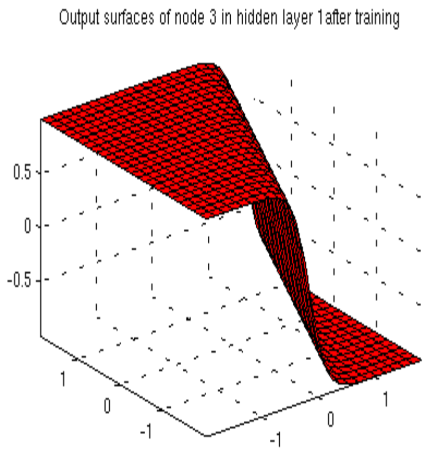
(b)



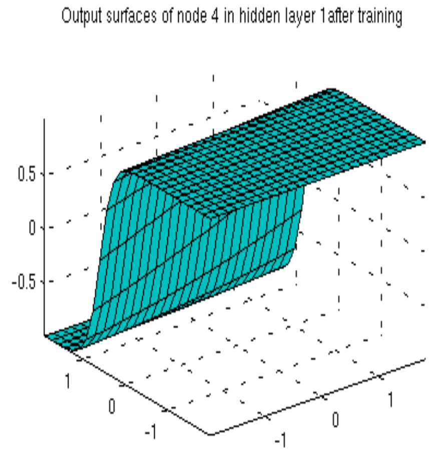
(c)



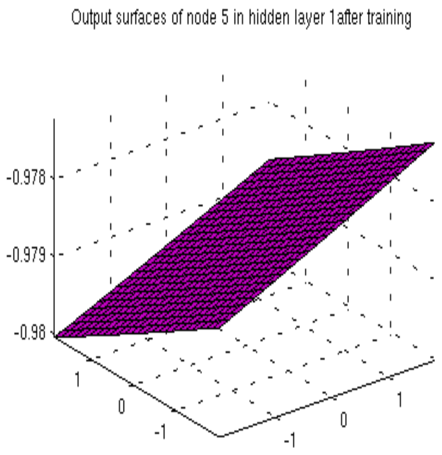
(d)



(e)



(f)



Observations

- The data is separable into 2 distinct regions and both models are able to classify the given data with 100% accuracy
- The decision boundaries formed by the GMM model are smooth curves since the boundaries are between combinations of Gaussians. The decision boundaries formed by the MLFNN classifier are sharper since its decision boundary is formed by the combination of several S-shaped sigmoidal functions. This is clearly evident in figure 6
- We can again choose to have a large value for the learning rate η for faster convergence since the data is completely separable.

Inferences

- Both models can be used for classifying non-linearly separable classes of the above kind. It should be noted that in case the data is non-linearly separable but its decision boundary is not a hyperquadric the above unimodal Bayes classifier will fail. In such a case, we will need to opt for Gaussian mixture models (GMMs). But the MLFFNN will work either way since it is a universal function approximator.
- It was seen that increasing the number of hidden layer nodes in MLFFNN can give a smoother decision boundary since there will be more sigmoids which will get combined to form the edges.

2.3 Overlapping classes

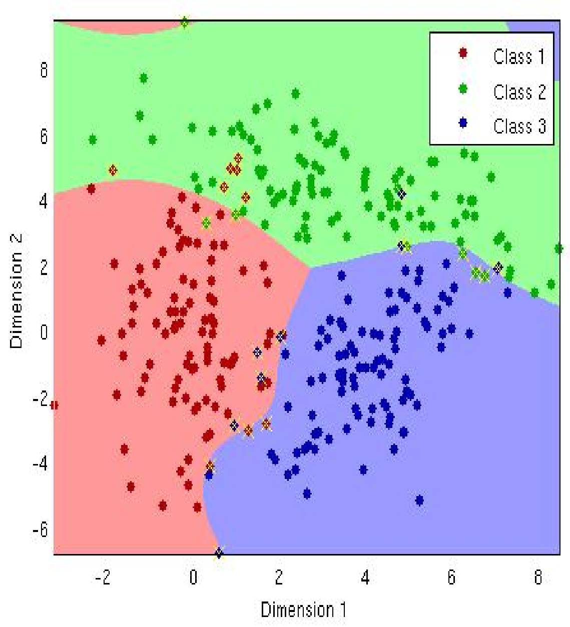
The given dataset consists of three classes. The following models were built for this dataset

- Bayes classifier using Gaussian mixture models. Each class is modelled using 3 Gaussian clusters. Full covariance matrices are considered for each class.
- Multi-layer feed forward neural network (MLFFNN) classifier with the following configuration parameters
  - 1 hidden layer consisting of 10 nodes
  - Hyperbolic tangent activation function at the hidden layer and logistic sigmoidal function at the output layer
  - Gradient descent with learning rate $\eta = 5$ and momentum factor $\alpha = 0.5$
  - Initialization of weights using random values in the range $[-1, 1]$
  - The training terminated because due to the early stopping criterion when the mean squared error reached approximately 0.18

The decision region plots obtained from both the models are shown in figure 7. The confusion matrix and classification accuracy for two models is shown in figure 8.

Figure 7: Decision region plots for different classifiers on the overlapping dataset

(a) Bayes classifier



(b) MLFFNN classifier

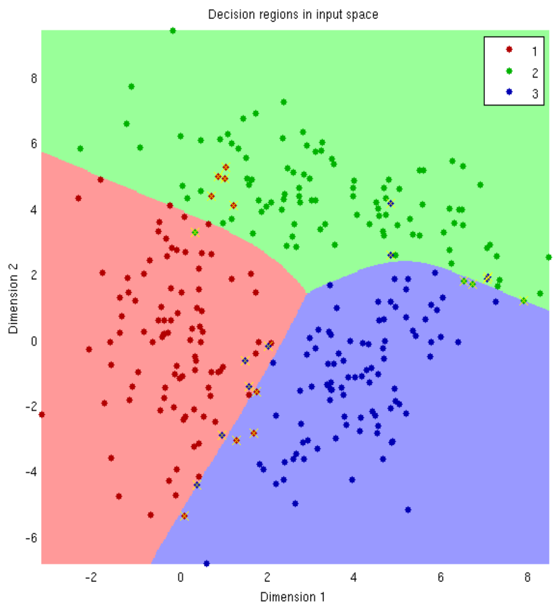
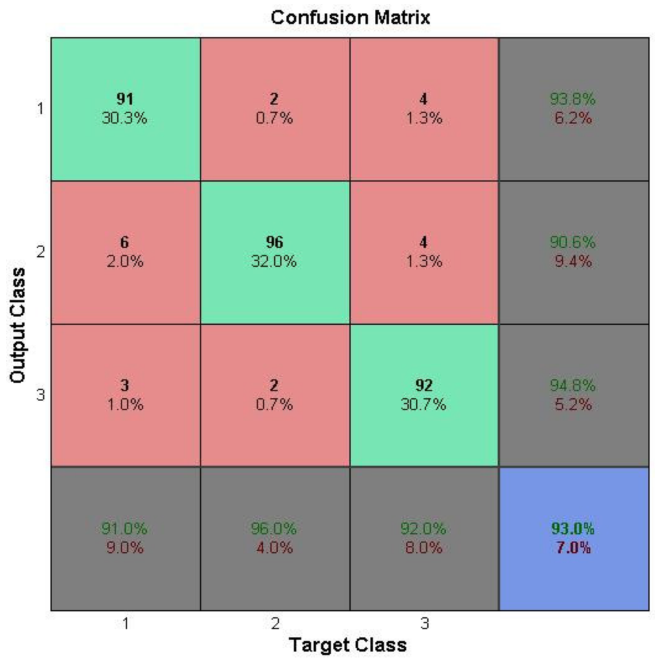
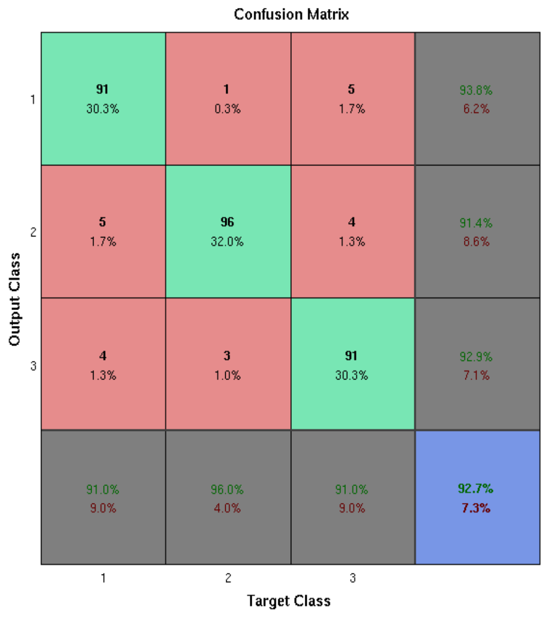


Figure 8: Confusion matrix and classification accuracy of MLFFNN and Bayes classifier on overlapping data

(a) Confusion matrix for Bayes classifier



(b) Confusion matrix for MLFNN classifier



Observations

- The classification accuracy of both the Gaussian mixture model classifier and the MLFFNN classifier are comparable. The marginal difference can be simply due to the choice of random initial weights used to initialize backpropagation for MLFFNN or the clustering in case of the GMM.
- The value of η must be reduced as compared to the earlier cases where the data was separable. Keeping a very high value of η here results in thrashing.

Inferences

- Both models are able to approximate arbitrarily complex decision boundaries. In case of MLFFNN, this is borne out by the fact that the validation check early stopping rule causes the algorithm to terminate. Thus the training is terminated before the MLFFNN starts overfitting the training examples.

3 Image Dataset

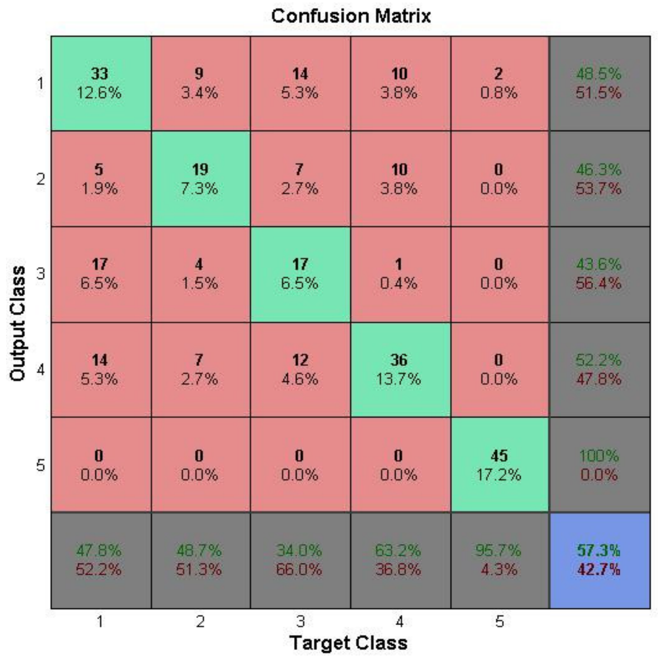
The given dataset consists of five classes — “billiards”, “hot-tub”, “mushroom”, “bath-tub” and “leopards”. The feature vectors are 48-dimensional colour histograms. The following models were built for this dataset

- Bayes classifier using Gaussian mixture models. Each class is modelled using different number of Gaussian clusters based on the number of points present in the class. The number of clusters used were 25, 18, 20, 20 and 20 for the classes respectively. Only diagonal covariance matrices were considered for each class since the amount of data is inadequate for estimation of full covariance matrices.
- Multi-layer feed forward neural network (MLFFNN) classifier with the following configuration parameters
  - 2 hidden layers consisting of 100 nodes in the first layer and 30 nodes in the second
  - Hyperbolic tangent activation function at the hidden layers and logistic sigmoidal function at the output layer
  - Gradient descent with learning rate $\eta = 0.3$ and momentum factor $\alpha = 0.92$
  - Initialization of weights using random values in the range $[-1, 1]$
  - The training terminated because due to the early stopping criterion when the mean squared error reached approximately 0.06

The confusion matrix and classification accuracy for the two models are shown in figure 9. The ROC curves are shown in figure 10.

Figure 9: Confusion matrix and classification accuracy of MLFFNN and Bayes classifier on image data

(a) Confusion matrix for Bayes classifier

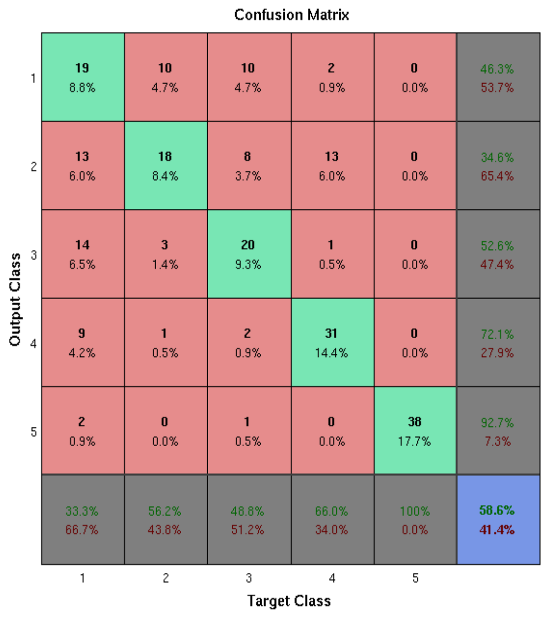


Confusion Matrix

Output Class	1	2	3	4	5	
1	33 12.6%	9 3.4%	14 5.3%	10 3.8%	2 0.8%	48.5% 51.5%
2	5 1.9%	19 7.3%	7 2.7%	10 3.8%	0 0.0%	46.3% 53.7%
3	17 6.5%	4 1.5%	17 6.5%	1 0.4%	0 0.0%	43.6% 56.4%
4	14 5.3%	7 2.7%	12 4.6%	36 13.7%	0 0.0%	52.2% 47.8%
5	0 0.0%	0 0.0%	0 0.0%	0 0.0%	45 17.2%	100% 0.0%
	47.8% 52.2%	48.7% 51.3%	34.0% 66.0%	63.2% 36.8%	95.7% 4.3%	57.3% 42.7%

Target Class

(b) Confusion matrix for MLFNN classifier



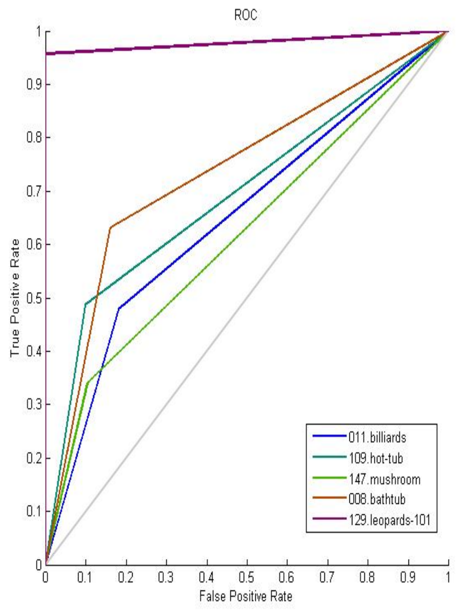
Confusion Matrix

Output Class	1	2	3	4	5	
1	19 8.8%	10 4.7%	10 4.7%	2 0.9%	0 0.0%	46.3% 53.7%
2	13 6.0%	18 8.4%	8 3.7%	13 6.0%	0 0.0%	34.6% 65.4%
3	14 6.5%	3 1.4%	20 9.3%	1 0.5%	0 0.0%	52.6% 47.4%
4	9 4.2%	1 0.5%	2 0.9%	31 14.4%	0 0.0%	72.1% 27.9%
5	2 0.9%	0 0.0%	1 0.5%	0 0.0%	38 17.7%	92.7% 7.3%
	33.3% 66.7%	56.2% 43.8%	48.8% 51.2%	66.0% 34.0%	100% 0.0%	58.6% 41.4%

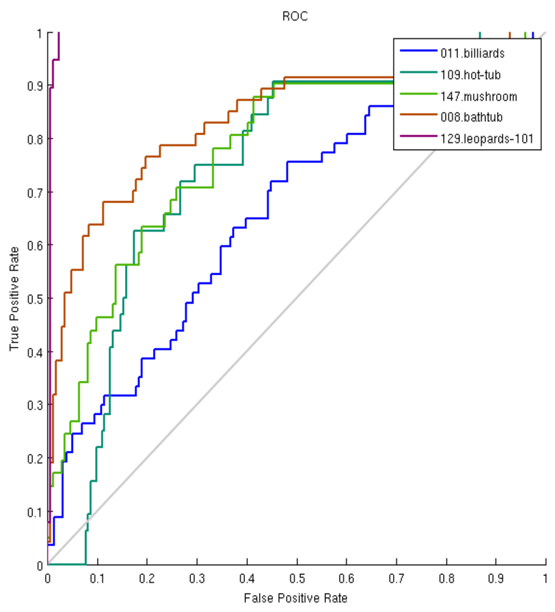
Target Class

Figure 10: Receiver operating characterisitcs for MLFFNN and Bayes classifier on image data

(a) ROC for Bayes classifier



(b) Confusion matrix for MLFNN classifier



Observations

- The MLFFNN model and the GMM Bayes classifier that were considered perfromed on par with each other on test data. The MLFFNN model gives just 1% improvement in accuracy over the GMM model.
- We found that with a single hidden layer model, the best classification accuracies over a wide range of parameter values were only $\approx 55\%$ over the test data. However, with the 2-layered model the best classification accuracy that could be achieved on the test dataset without overfitting was $\approx 59\%$.
- It can be seen that the "leopards" class is classified well by both the models but the predictions for the rest of the classes are much more confused.

Inferences

- The images in the "leopard" class contain primarily dark shades of yellow and green, which are not present in the same amount in the images of other classes whereas the colour histograms for the other classes are similar. This causes this class to be distinguished more easily.
- Using a multiple hidden layered model allows us to learn complex discriminant functions faster and more easliy as compared to the standard single hidden layer model.
- The low classification accuracies indicate that colour histograms alone might not provide enough discriminatory information to be able to clearly distinguish between classes of images.

Part II

Regression

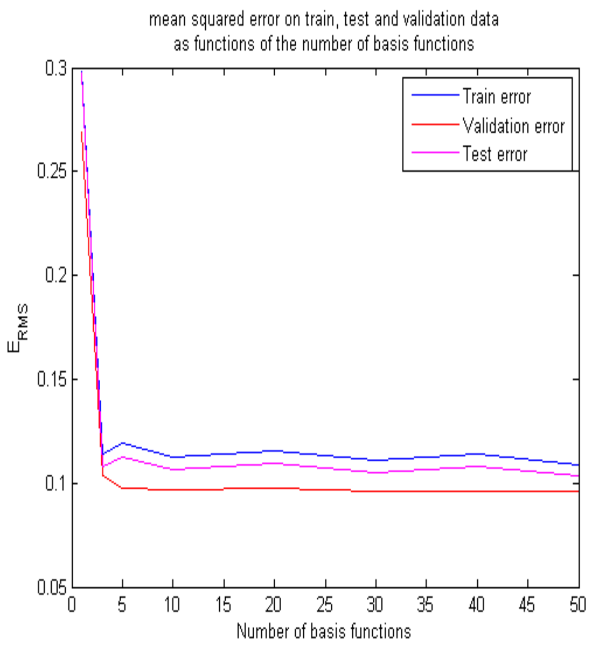
4 Univariate Data

The following models were built to perform regression on the univariate data

- Multi-layer feed forward neural network (MLFFNN) classifier with the following configuration parameters
  - 1 hidden layer consisting of $m = 10$ nodes.
  - Hyperbolic tangent activation function at the hidden layer and linear actiation function at the output layer
  - Gradient descent with learning rate $\eta = 0.01$ and momentum factor $\alpha = 0.9$
  - Initialization of weights using random values in the range $[-1, 1]$
  - The training terminated due to the early stopping criterion when the mean squared error reached approximately 0.095
- Radial basis function neural network (RBFNN) regression model with the following parameters
  - $m = 9$ basis functions with the centres computed using the K-means algorithm.
  - Regularization parameter $\lambda = \exp(-4)$

Figure 11: MSE as the model complexity m increases

(a) MLFFNN model



(b) RBFNN model

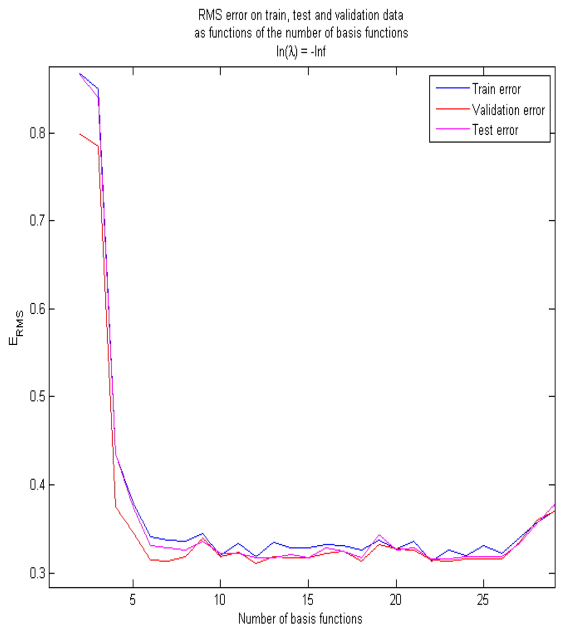


Figure 12: MSE with varying λ for RBFNN model

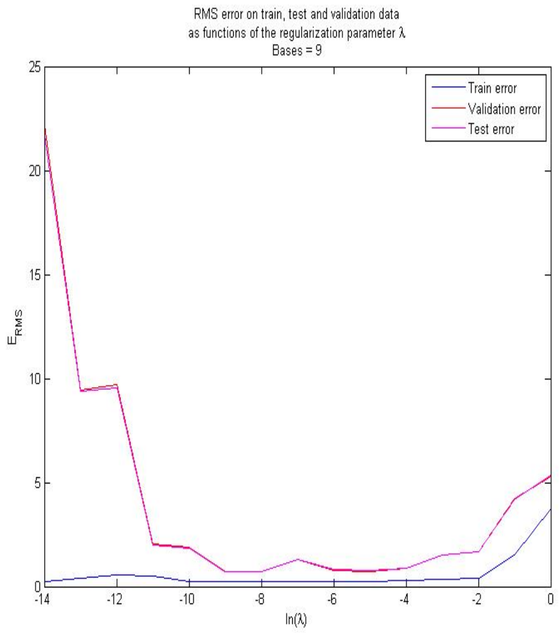
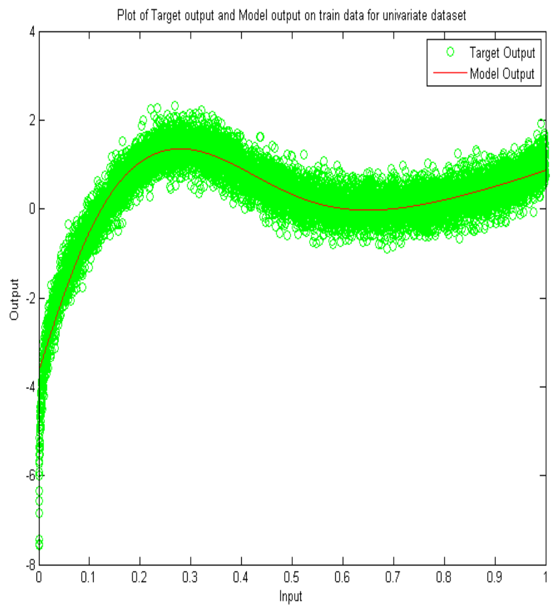
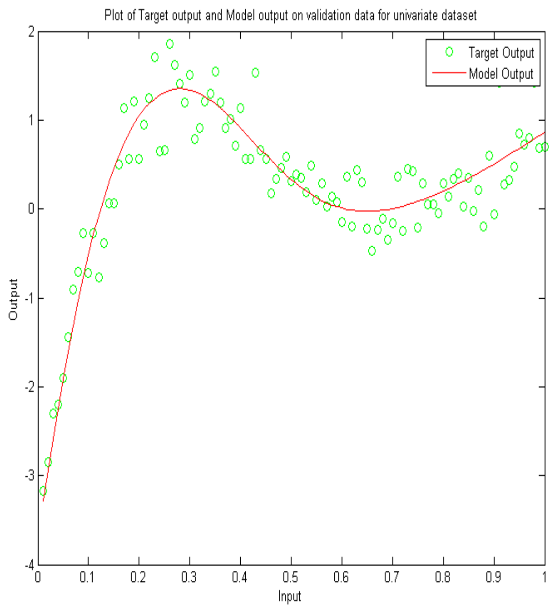


Figure 13: Model output and target output for training, test and validaton data (MLFFNN)

(a) Training data



(b) Validation data



(c) Test data

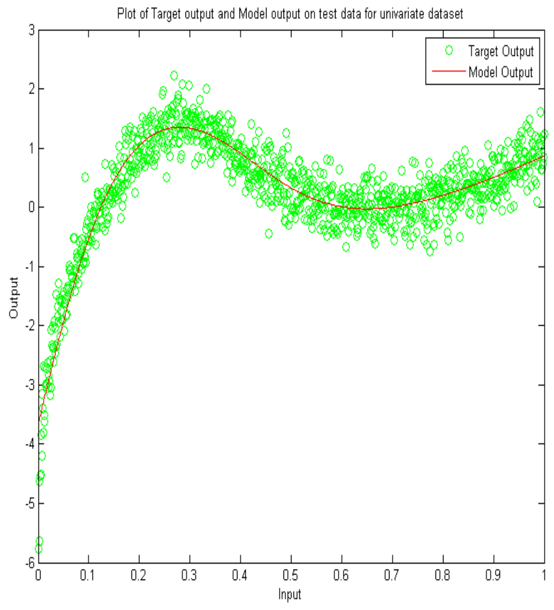
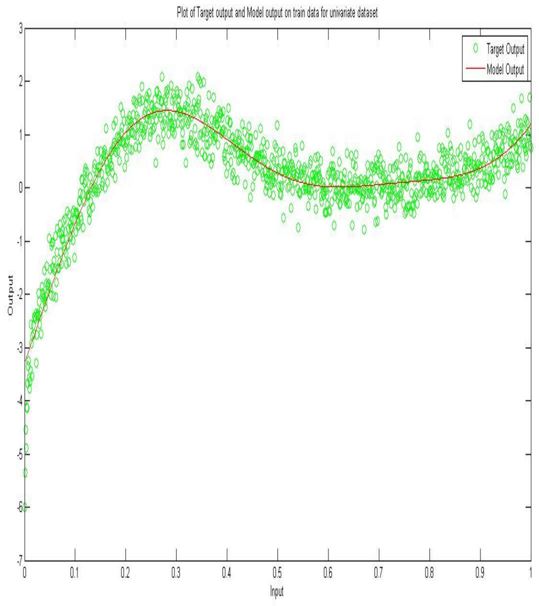
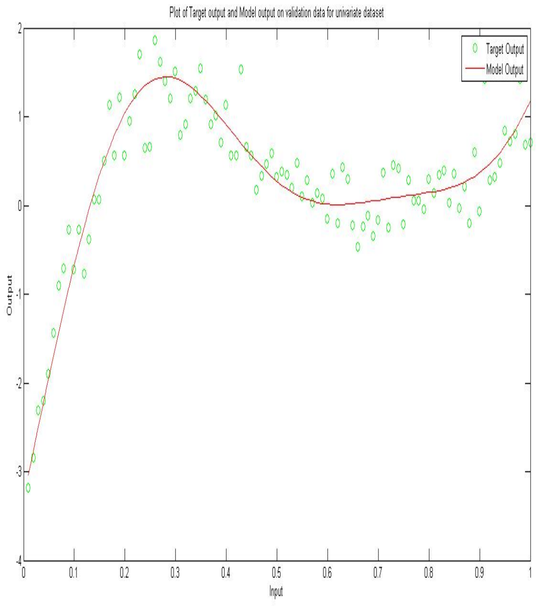


Figure 14: Model output and target output for training, test and validaton data (RBFNN)

(a) Training data



(b) Validation data



(c) Test data

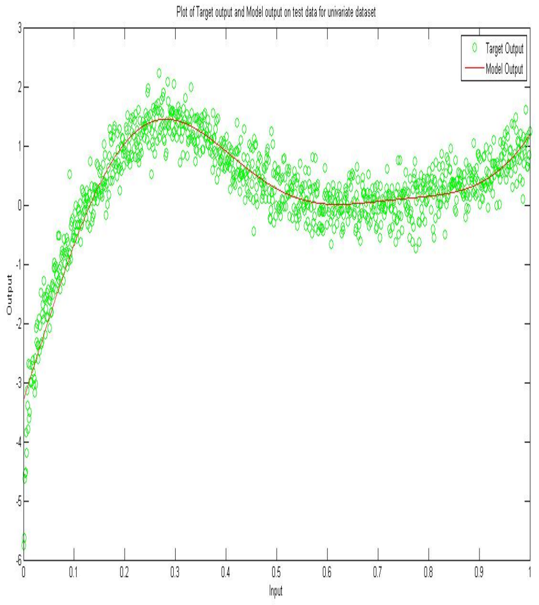
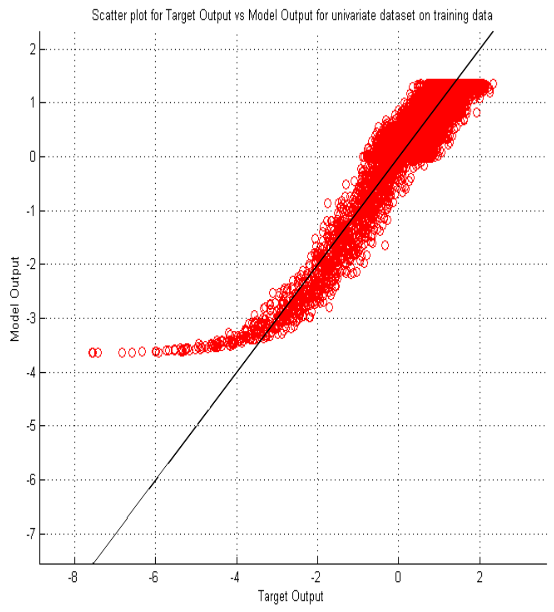
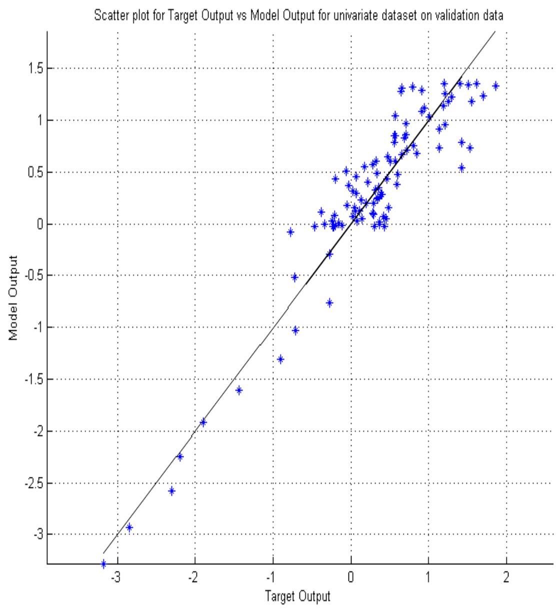


Figure 15: Scatter plot of model output vs target output for training, test and validaton data (MLFFNN)

(a) Training data



(b) Validation data



(c) Test data

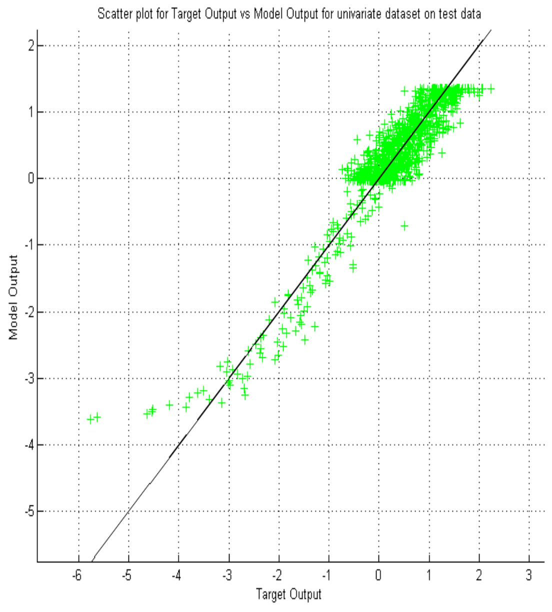
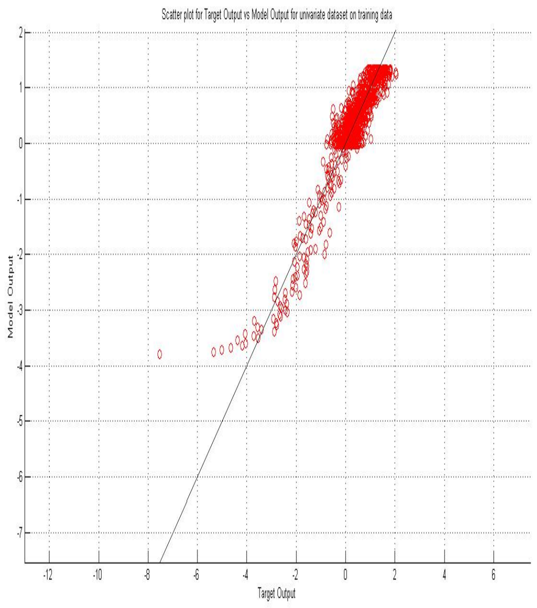
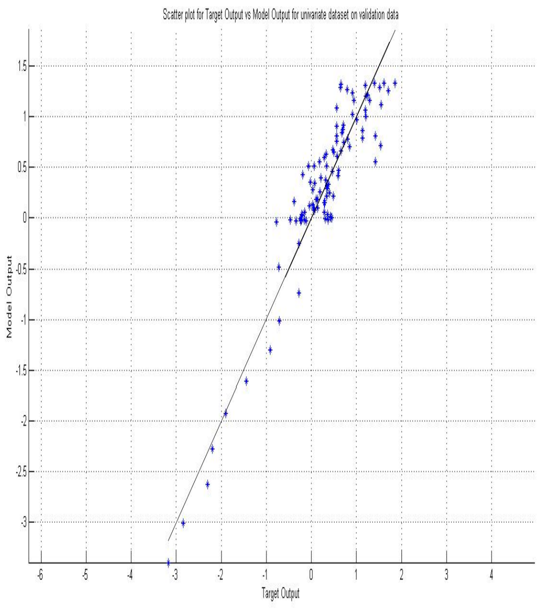


Figure 16: Scatter plot of model output vs target output for training, test and validaton data (RBFNN)

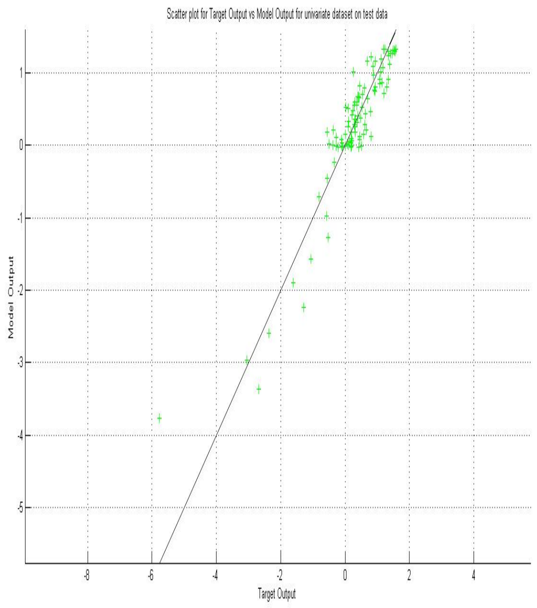
(a) Training data



(b) Validation data



(c) Test data



Observations

- For the MLFFNN model, the mean squared error (MSE) vs model complexity curve in figure 11a indicates that choosing around 10 nodes would give a good approximation as well as good generalization. Hence the given model was chosen.
- For the RBFNN model, the MSE vs model complexity curve in figure 11b indicates that indicates

that the number of basis functions chosen should be 9. After having chosen our model complexity, we determined that the best value of the regularization parameter $\lambda = \exp(-4)$ for the given m should be -4 since it gives minimum error on validation data figure 12.

- It can be seen from the scatter plots (figure 15) that most of the points lie in the range $[-0.5, 1.5]$. This can easliy be explained by looking at figure 13 where it is clear that the y-coordinate values of a large majority of the points lie in that range.

Inferences

- The plots of the model and target outputs in figures 13 and 14 show that both models performed quite well on the unseen test data. This is also borne out by the scatter plot in figures 15 and 16.
- In general, m tends to be larger for RBFNNs than for MLFFNNs since RBFNNs require that the chosen basis centres be well-distributed over the input space [WP]. However, in this case, the input space is small covering only a small range over only 1 dimension and hence there is no appreciable difference in the value of m.

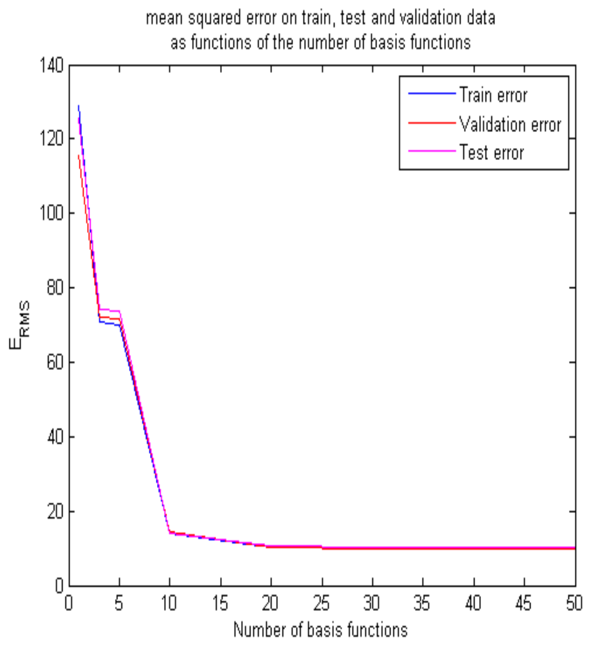
5 Bivariate Data

The following models were built to perform regression on the bivariate data

- Multi-layer feed forward neural network (MLFFNN) classifier with the following configuration parameters
  - 1 hidden layer consisting of 30 nodes
  - Hyperbolic tangent activation function at the hidden layer and linear actiation function at the output layer
  - Gradient descent with learning rate $\eta = 0.01$ and momentum factor $\alpha = 0.9$.
  - Initialization of weights using random values in the range $[-1, 1]$
  - The training terminated due to the early stopping criterion when the mean squared error reached approximately 10
- Radial basis function neural network (RBFNN) regression model with the following parameters
  - $m = 25$ basis functions with the centres computed using the K-means algorithm
  - Regularization parameter $\lambda = 0$

Figure 17: MSE as the model complexity m increases

(a) MLFFNN model



(b) RBFNN model

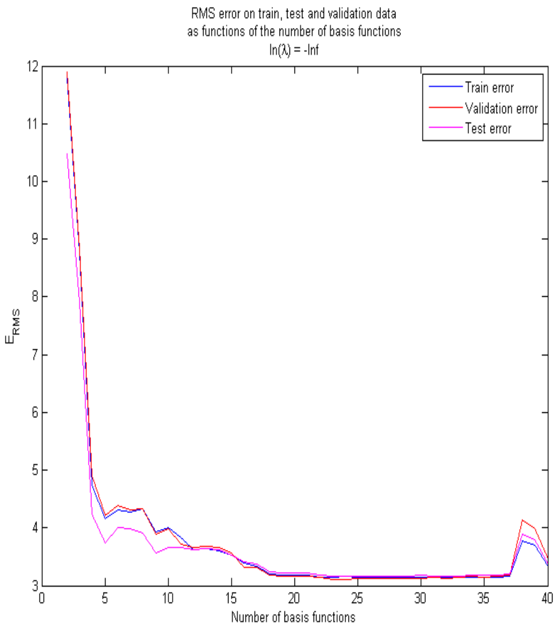


Figure 18: MSE with varying λ for RBFNN model

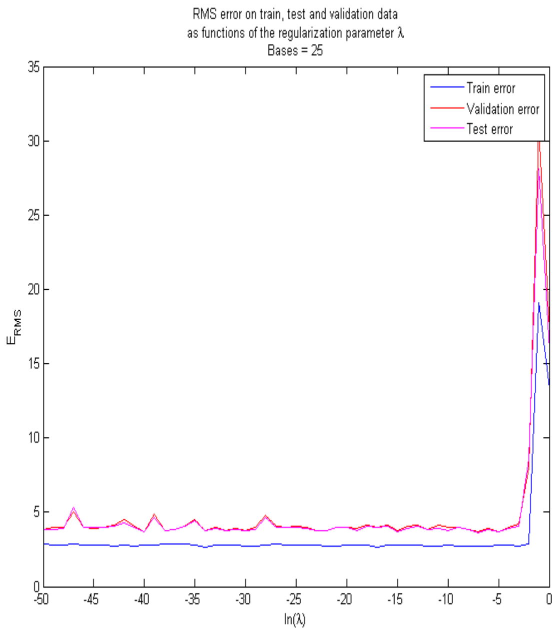
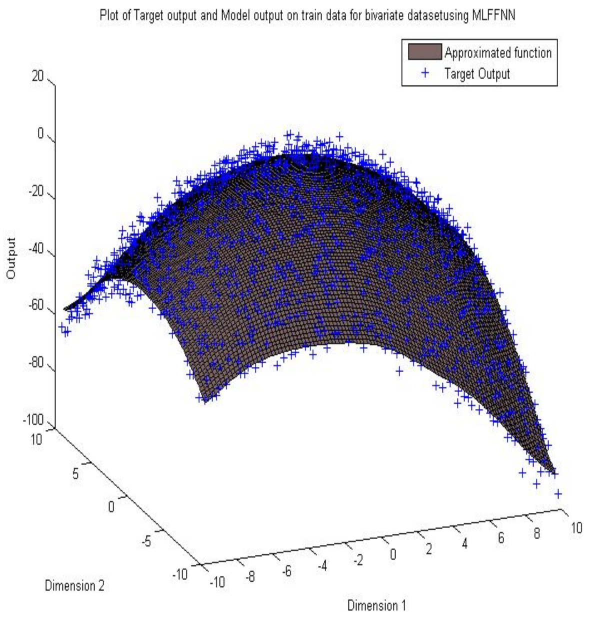
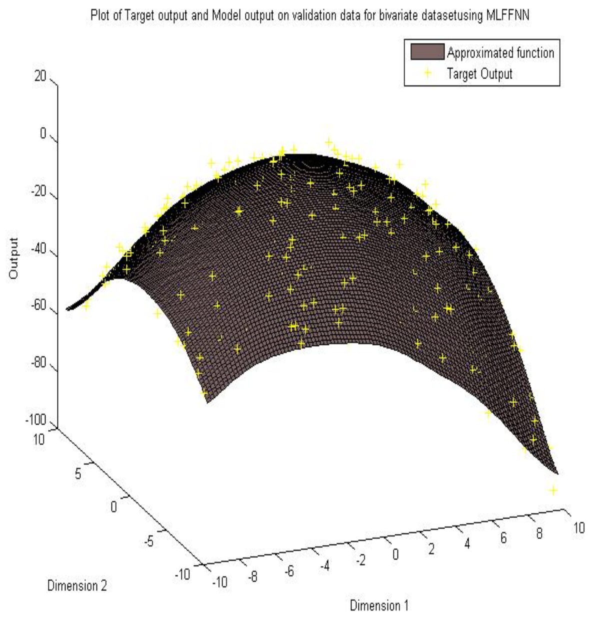


Figure 19: Model output and target output for training, test and validaton data (MLFFNN)

(a) Training data



(b) Validation data



(c) Test data

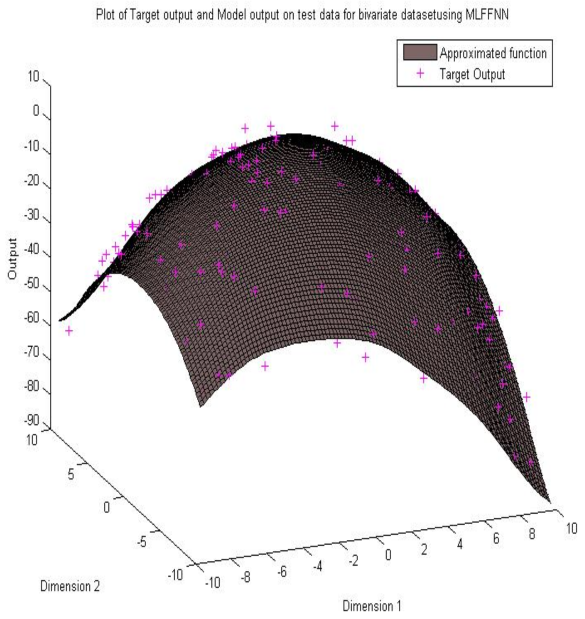
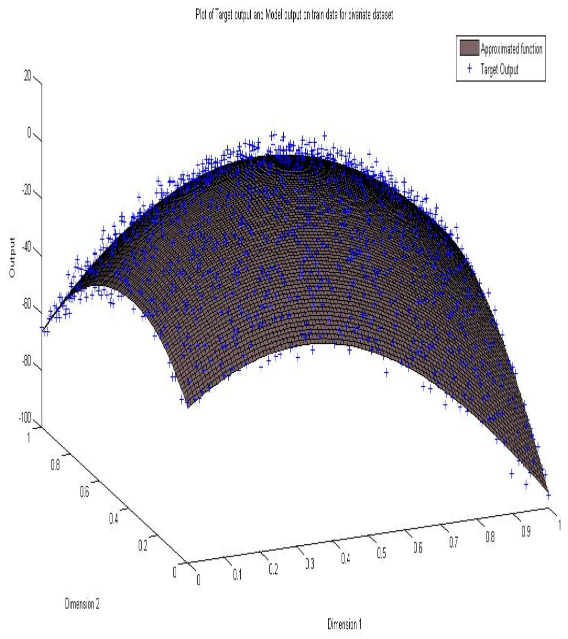
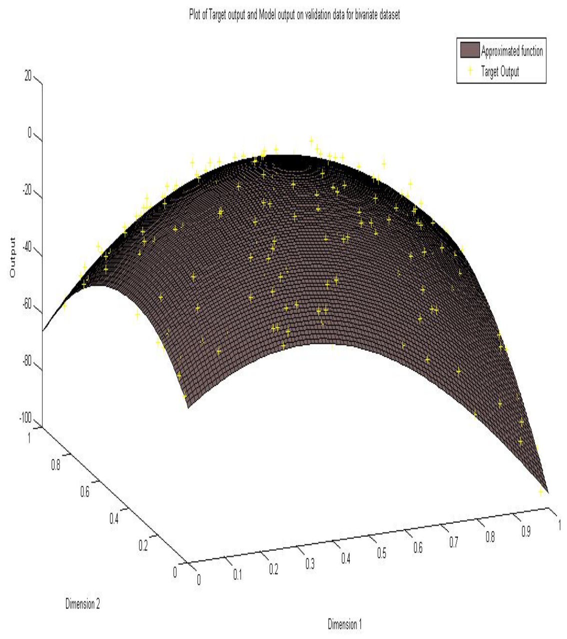


Figure 20: Model output and target output for training, test and validaton data (RBFNN)

(a) Training data



(b) Validation data



(c) Test data

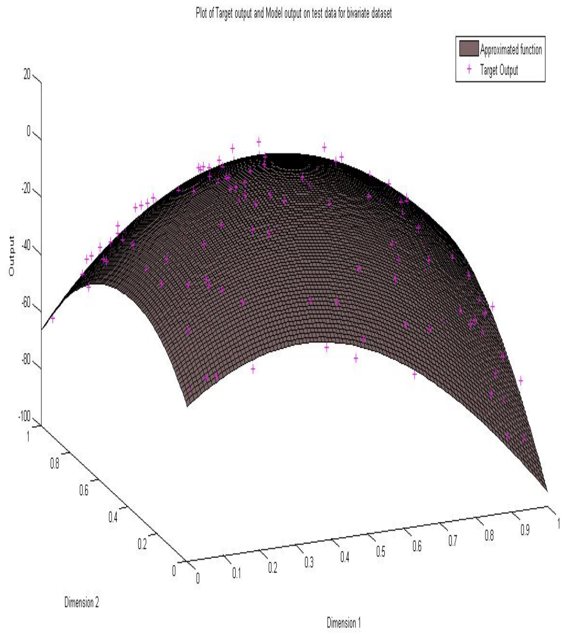
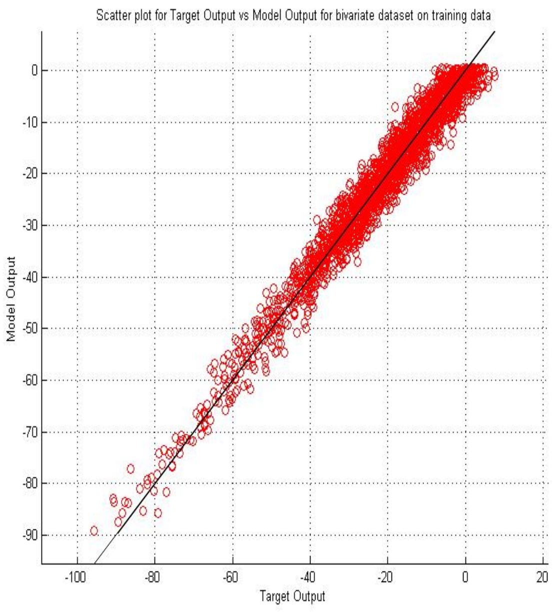
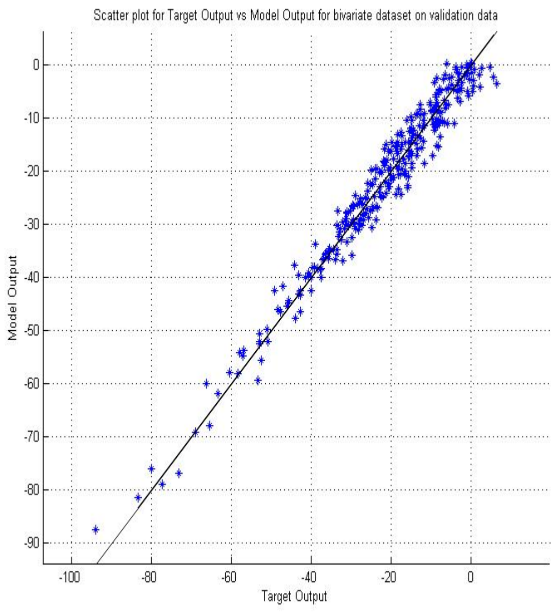


Figure 21: Scatter plot of model output vs target output for training, test and validaton data (MLFFNN)

(a) Training data



(b) Validation data



(c) Test data

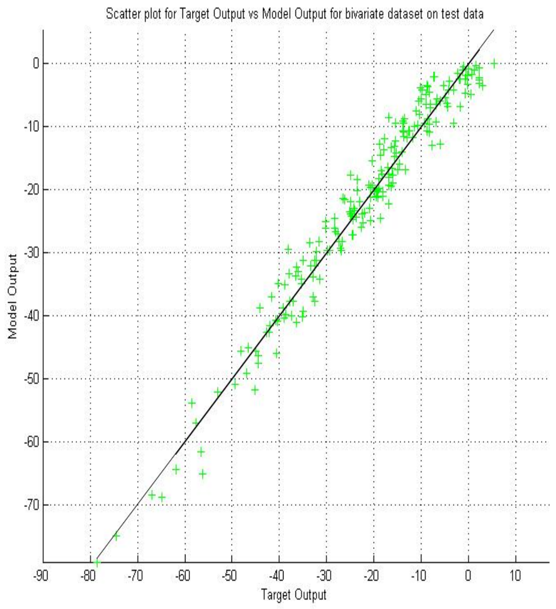
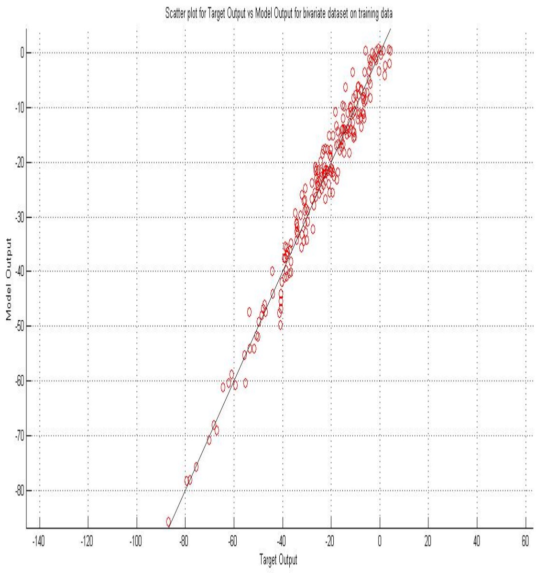
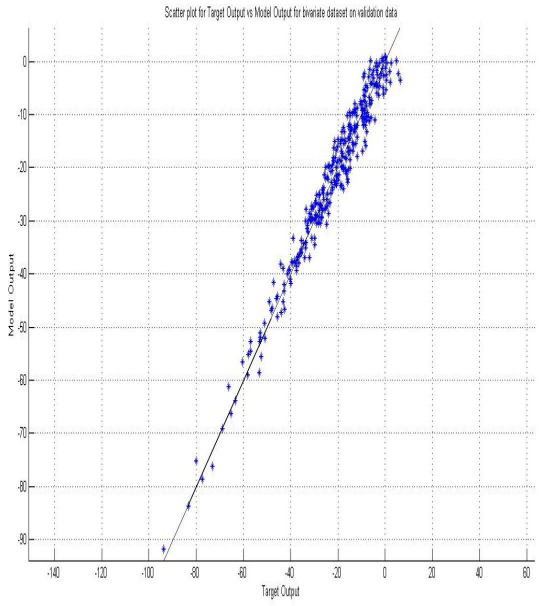


Figure 22: Scatter plot of model output vs target output for training, test and validaton data (RBFNN)

(a) Training data



(b) Validation data



(c) Test data

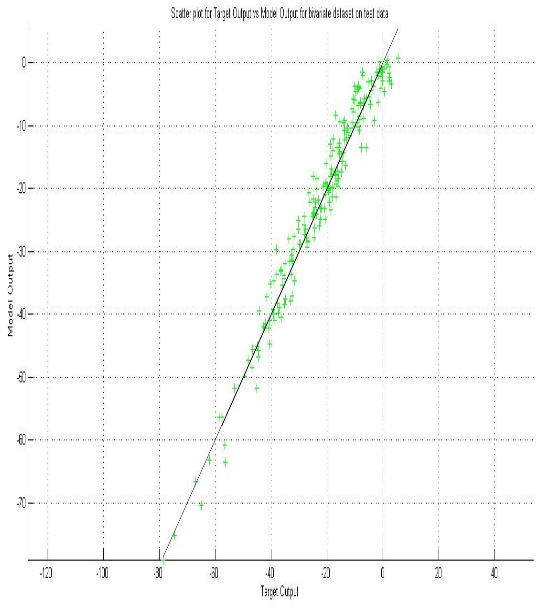
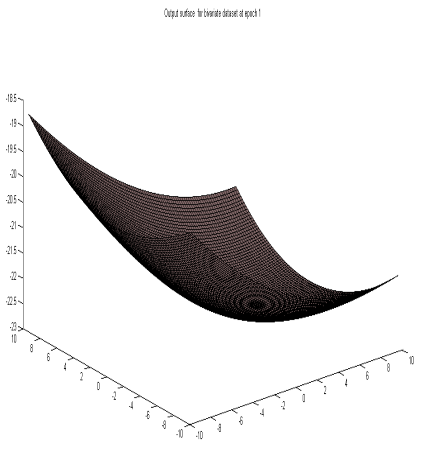
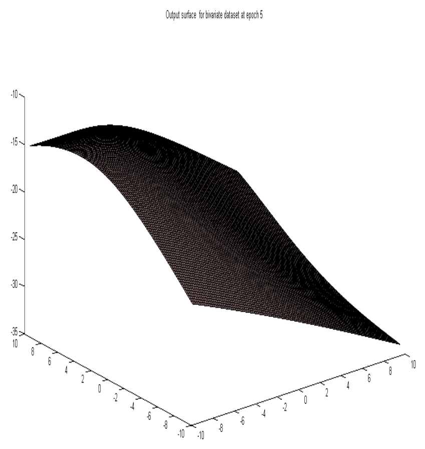


Figure 23: Output at the output node in MLFFNN at different epochs during training

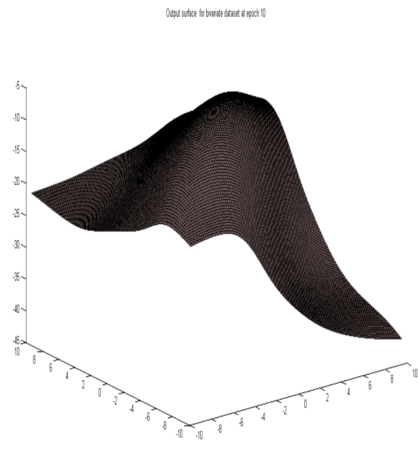
(a) After 1 epoch



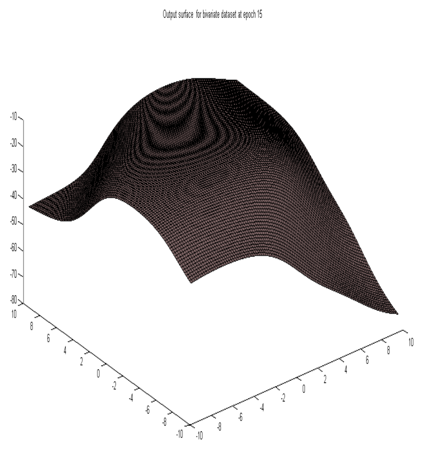
(b) After 5 epochs



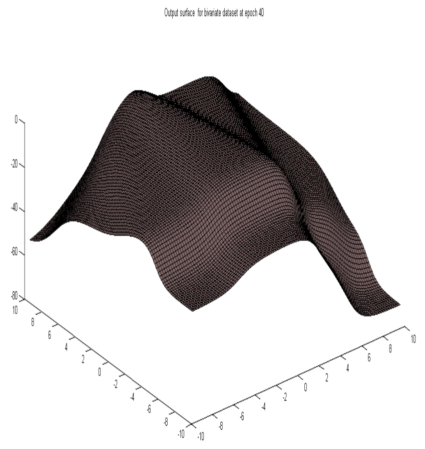
(c) After 10 epochs



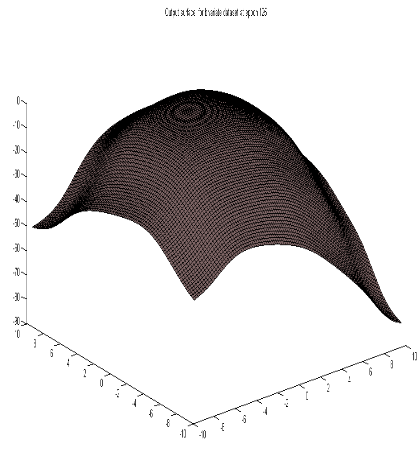
(d) After 15 epochs



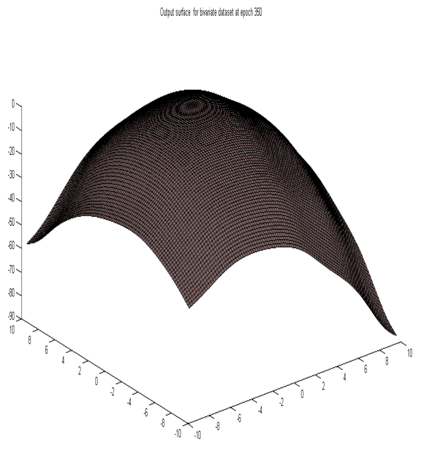
(e) After 40 epochs



(f) After 125 epochs



(g) After 350 epochs



(h) After 600 epochs

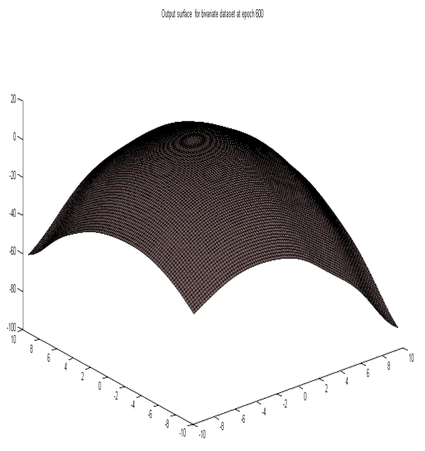
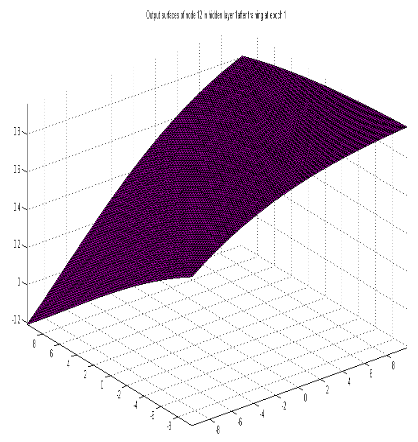
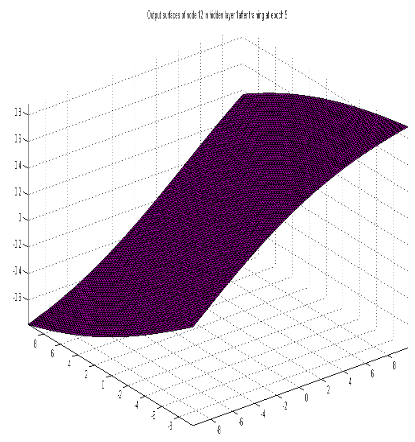


Figure 24: Output at a hidden layer node in MLFFNN at different epochs during training

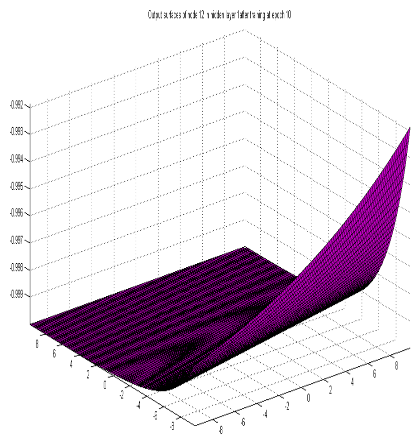
(a) After 1 epoch



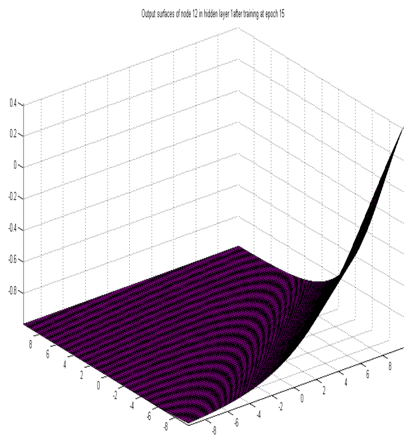
(b) After 5 epochs



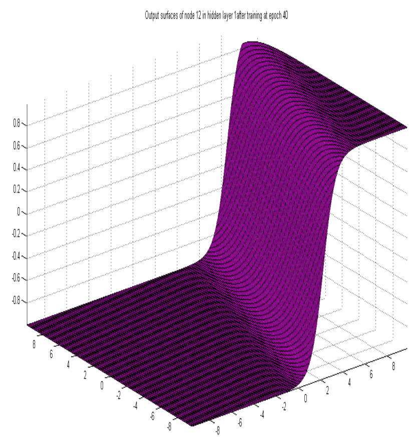
(c) After 10 epochs



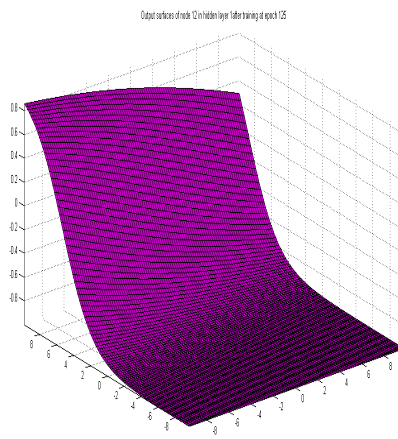
(d) After 15 epochs



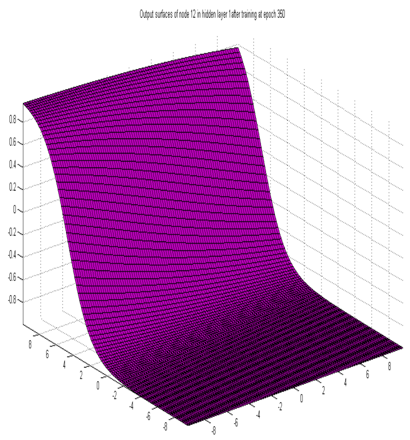
(e) After 40 epochs



(f) After 125 epochs



(g) After 350 epochs



(h) After 600 epochs

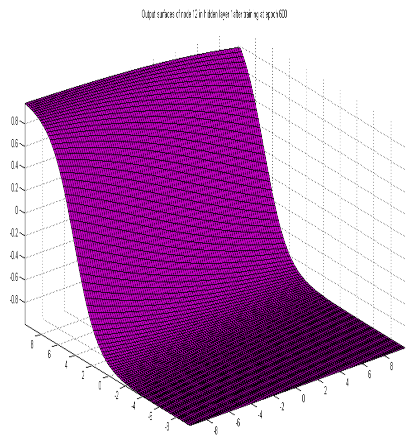
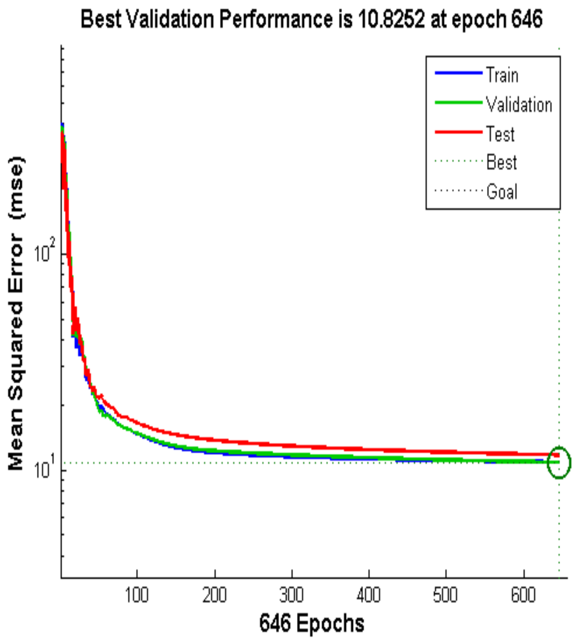


Figure 25: Decrease in MSE with number of epochs during training



Observations

- For the MLFFNN model, the MSE vs model complexity curve in figure 17a indicates that choosing around 30 nodes would give a good approximation as well as good generalization. Hence the given model was chosen.
- For the RBFNN model, the MSE vs model complexity curve in figure 17b indicates that indicates that the number of basis functions chosen should be 30. After having chosen our model complexity, we determined that the best value of the regularization parameter λ for the given m should be 0 since minimum error on validation data seems not to decrease with higher λ (figure 18).
- figure 23 shows the output surface of the MLFFNN network over the input space. As the training progresses from epoch to epoch, the surface begins to approximate the target output more and more closely.
- figure 24shows the output of 1 of the hidden layer nodes as the number of epochs increases. It is interesting to note how at between epochs 40 and 125 the surface completely changes direction. Other than that, in general, we can see that as the number of epochs increase the S-shaped hidden layer surface becomes sharper as the network learns to approximate the given function

Inferences

- The plots of the model and target outputs in figures 19 and20 show that both models performed quite well on the unseen test data. This is also borne out by the scatter plots in figures 19 and22.
- The variation in the output surface from epoch to epoch shows how the model learns the target function. However, it should be noted that the hidden layer outputs are sigmoidal (S-shaped) while the target output is Gaussian and hence, radially symmetrical. Therefore, we need quite a large number of hidden layer nodes to get a good approximation. This is one case when RBFNN needs much fewer nodes than the MLFFNN model to approximate the given function well.

- The shift in the hidden layer output surface from epoch 40 to 125 in figure 24 can be understood if we look at figure 25 which shows the change in the performance of the network with training epochs. In the initial portion of the curve, the error is steeply declining implying that the network is doing a lot of its learning at this point. We can expect that in this region, the change in weights will be much larger as compared to the later epochs and therefore, there will be variation in the output surface of the hidden layer. Once we are close to the error minima, the amount of change in the weights is less and there are only small changes in the hidden layer output surfaces.

- In general, m tends to be larger for RBFNNs than for MLFFNNs since RBFNNs require that the chosen basis centres be well-distributed over the input space [WP]. However, in this case, the input space is small covering only a small range over only 1 dimension and hence there is no appreciable difference in the value of m.

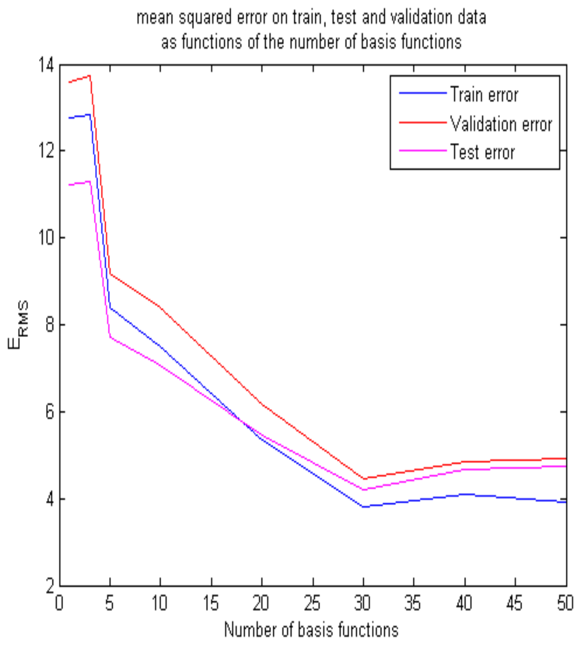
6 Multivariate Data

The multivariate dataset contained 19-dimensional feature vectors. The following models were built to perform regression on this data

- Multi-layer feed forward neural network (MLFFNN) classifier with the following configuration parameters
  - 1 hidden layer consisting of 30 nodes
  - Hyperbolic tangent activation function at the hidden layer and linear actiation function at the output layer
  - Gradient descent with learning rate $\eta = 0.01$ and momentum factor $\alpha = 0.9$
  - Initialization of weights using random values in the range $[-1, 1]$
  - The training terminated due to the early stopping criterion when the mean squared error reached approximately 4.
- Radial basis function neural network (RBFNN) regression model with the following parameters
  - 125 basis functions with the centres computed using the K-means algorithm
  - $m = 125$ basis functions with the centres computed using the K-means algorithm
  - Regularization parameter $\lambda = 0$

Figure 26: MSE as the model complexity m increases

(a) MLFFNN model



(b) RBFNN model

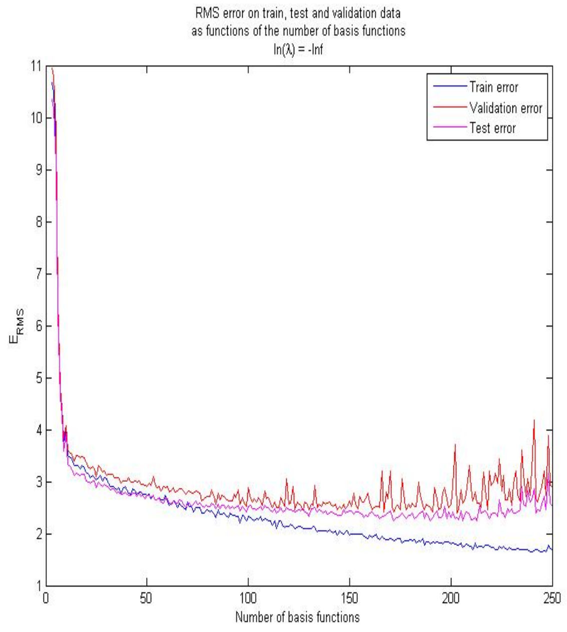


Figure 27: MSE with varying λ for RBFNN model

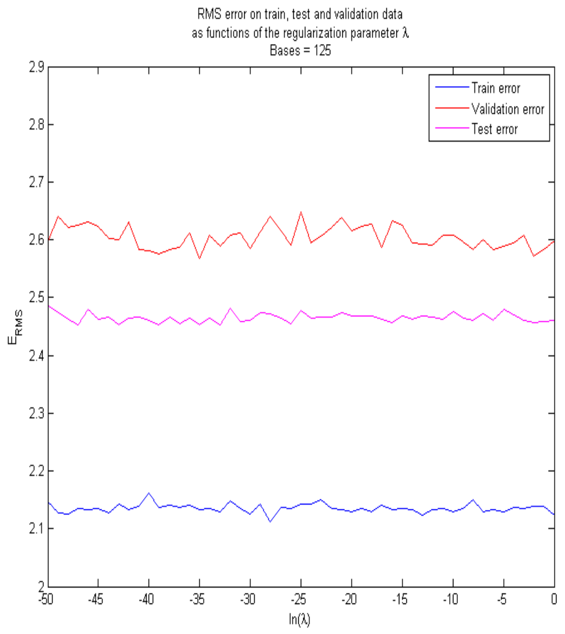
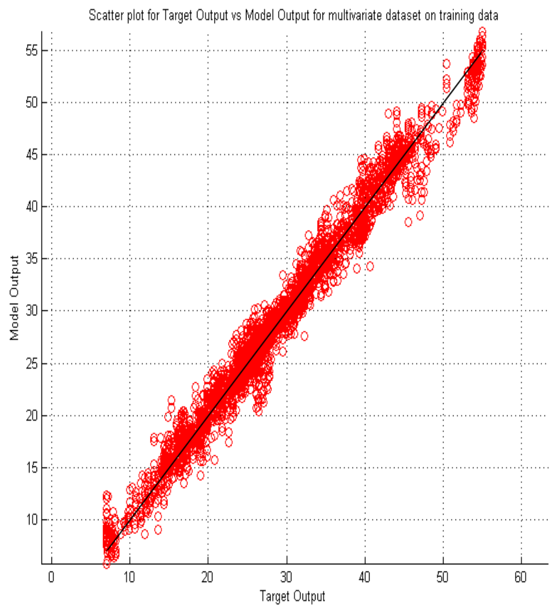
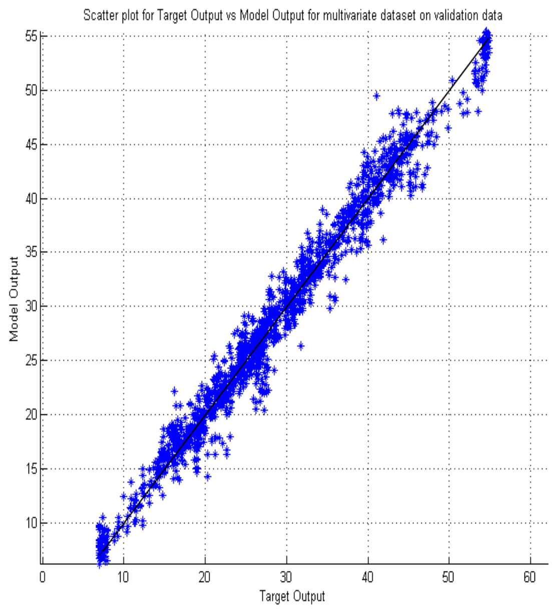


Figure 28: Scatter plot of model output vs target output for training, test and validaton data (MLFFNN)

(a) Training data



(b) Validation data



(c) Test data

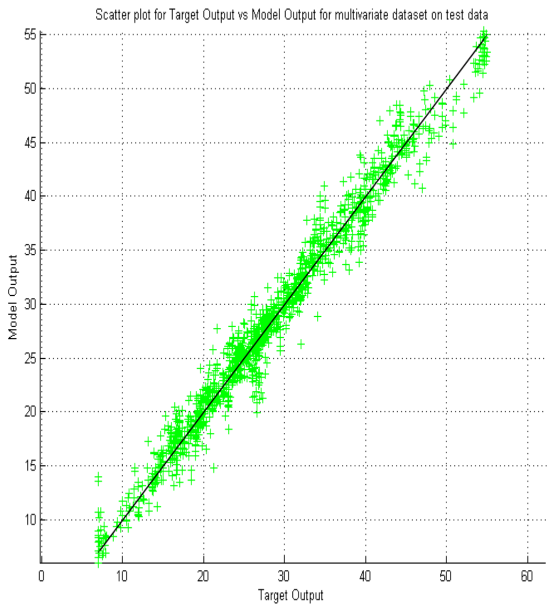
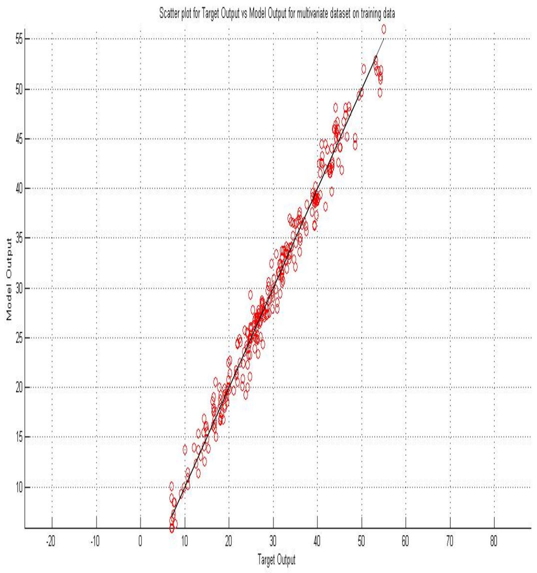
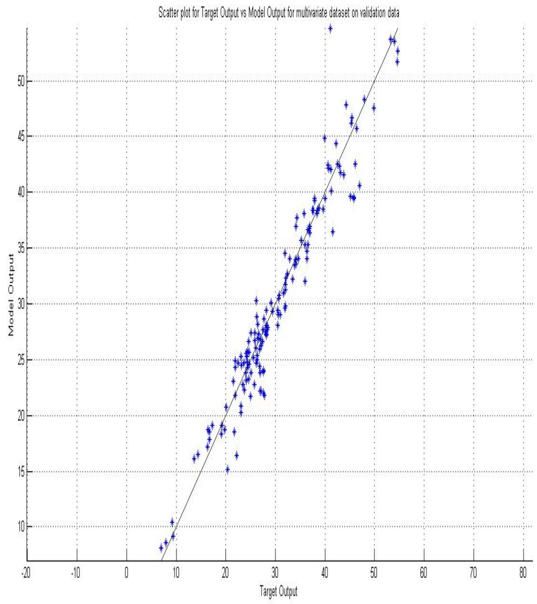


Figure 29: Scatter plot of model output vs target output for training, test and validaton data (RBFNN)

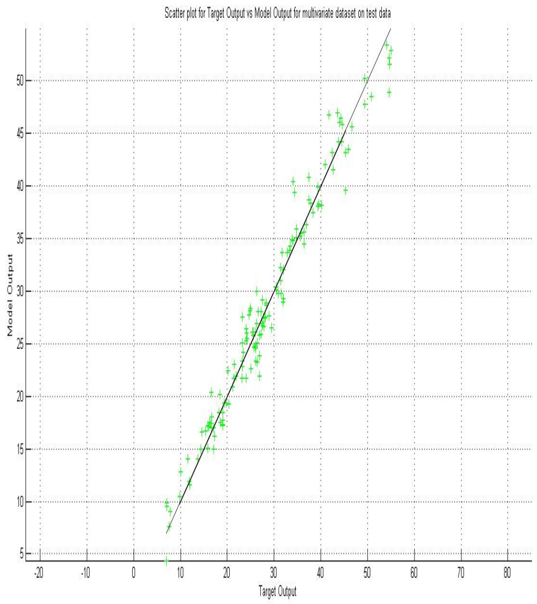
(a) Training data



(b) Validation data



(c) Test data



Observations

- For the MLFFNN model, the MSE vs model complexity curve in figure 26a indicates that choosing around 30 nodes would give a good approximation as well as good generalization. Hence the given model was chosen.
- For the RBFNN model, the MSE vs model complexity curve in figure 26b indicates that indicates that

the number of basis functions chosen should be 125. After having chosen our model complexity, we determined that the best value of the regularization parameter λ for the given m should be 0 since it gives minimum error on validation data seems not increase for higher value of λ(figure 27).

Inferences

- The scatter plots of the model and target outputs in figures figure 28 and figure 29 show that both models performed quite well on the unseen test data.
- Since the data is high-dimensional, the width of the radial basis functions in case of RBFNN needs to be large in order to cover the entire input space. Also, a large input space implies that the number of basis functions must be large. In this case, we can see that the value of m for the case of RBFNN is larger than that required to approximate the same function using MLFFNN. Although this is usually the case for high-dimensional data, RBFNN might often be preferred because the optimal weights can be found using a closed form expression and we do not need to perform an iterative procedure for this purpose. Further, unlike for MLFFNN, the error surface is quadratic (hence the single solution) and so the model does not suffer from local minima. Thus, training is much faster.

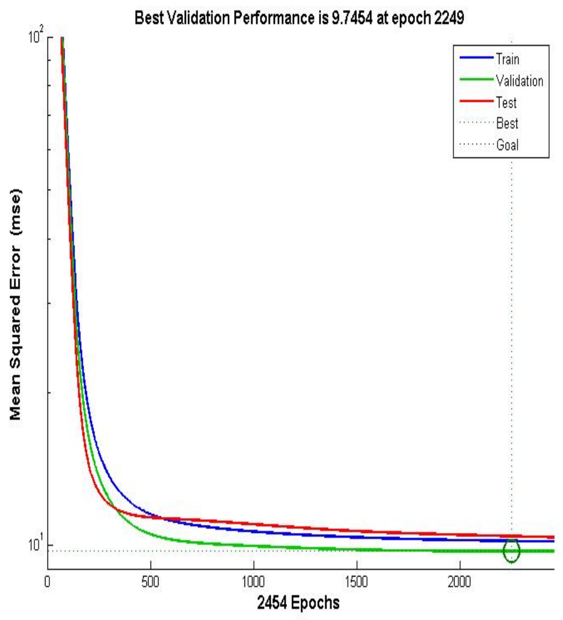
7 Additional experiments and observations

7.1 Gradient descent with momentum vs scaled conjugate gradient

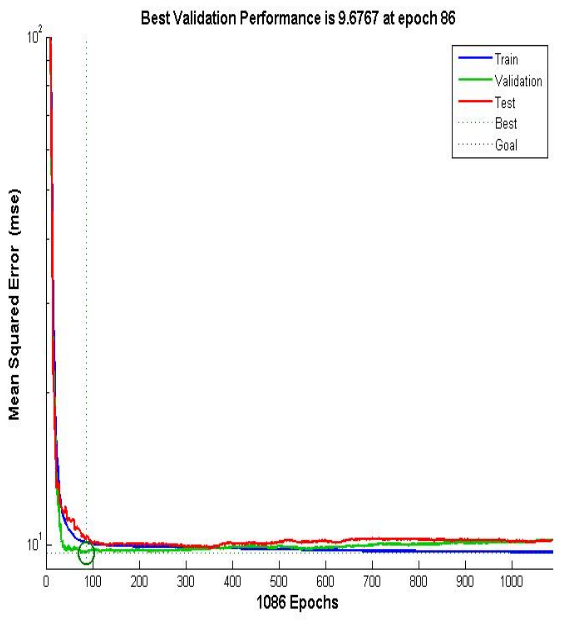
The gradient descent with momentum is a first-order optimization procedure whereas the scaled conjugate gradient takes second derivatives into account and is hence a second order procedure. As such, the rate of convergence of the scaled conjugate gradient method is in most cases much faster than that of the gradient descent method. A graphical comparison between the two is shown in fig. The results are for the function approximation task in case of the bivariate dataset. The parameter values are $\eta = 0.01$ and $\alpha = 0.3$ for the gradient descent with momentum procedure while the scaled conjugate gradient procedure uses $\sigma = 10^{-5}$ and $\lambda = 10^{-7}$.

Figure 30: Comparison of the gradient descent (with momentum) and sclaed conjugate gradient methods for bivariate data.

(a) Using gradient descent



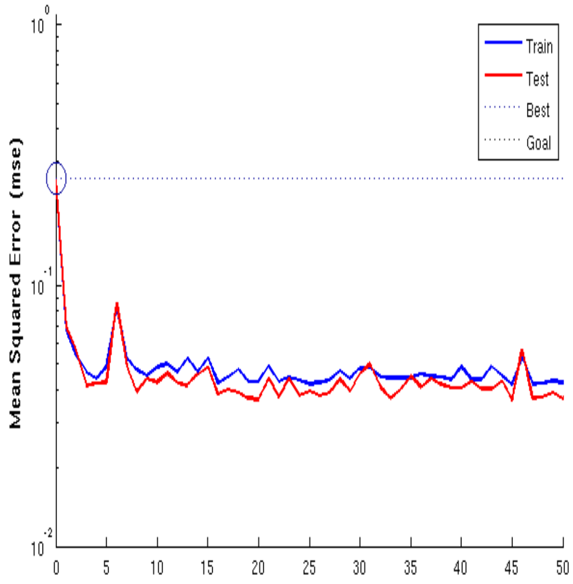
(b) Using scaled conjugate gradient



7.2 Pattern mode vs batch mode

MATLAB makes heavy use of vectorization and parallelization and hence, in the MATLAB enviroment batch mode training is much faster that training in pattern mode. To actually compare the results obtained using the two, we chose the overlapping dataset. Training for both the modes was started using the same random initial weights. The decision surfaces obtained show that the solution obtained using both batch mode and pattern mode of training converge to nearly the same solution when started using the same random initial weights. We also observed that the number of epochs required for pattern mode of training to reach close to the minima was much lesser than for batch mode. However, pattern mode is much more prone to weight fluctuations with higher learning rates than batch mode (figure 31).

Figure 31: Fluctuations observed in batch mode training with 10 hidden nodes, $\eta = 1$ and $\alpha = 0.5$. The training was stopped after 4 iterations and continued with lower learning rate till convergence.

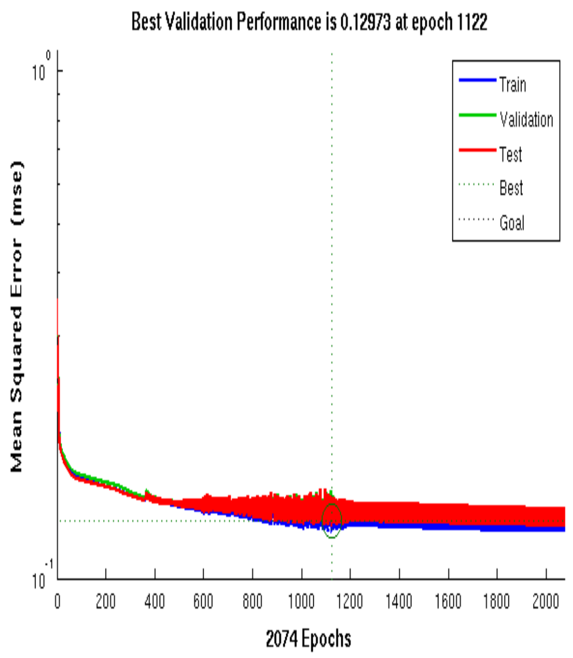


7.3 Fluctuations in the error as we approach the minima

We observed that while with inappropriate values of the η and α parameters, the mean squared error tends to fluctuate as the solution converges to the error minima, with an appropriate choice of these parameters, these flucutations could be removed *entirely* in all cases. A couple of illustrative examples are given for the case of image data in case of classification (figure 32) and bivariate data in case of regression (figure 33). We believe that this is because the training was done using batch mode rather than pattern mode where thrashing is reduced. Another possibility is the nature of the Rumelhart-McClelland momentum-based weight update that causes the weight change at the next iteration be an exponentially decaying weighted-average of previous weight change.

Figure 32: Plot of MSE vs number of epochs for image data classification with different parameter values

(a) $\eta = 0.3$, $\alpha = 0$



(b) $\eta = 0.3$, $\alpha = 0.92$

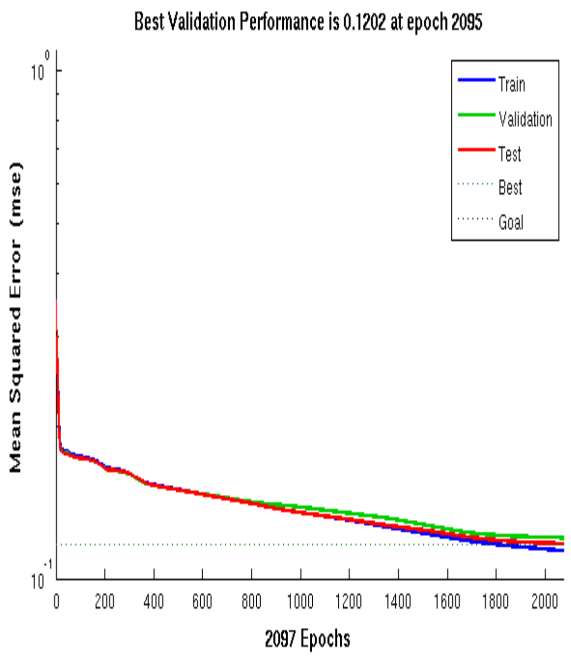
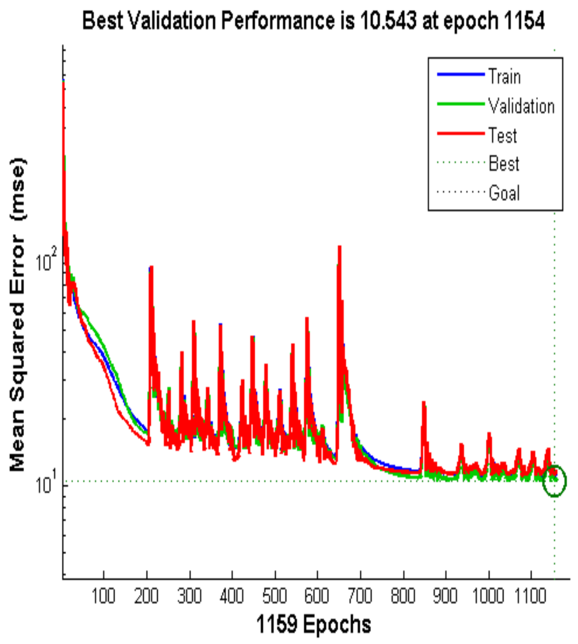
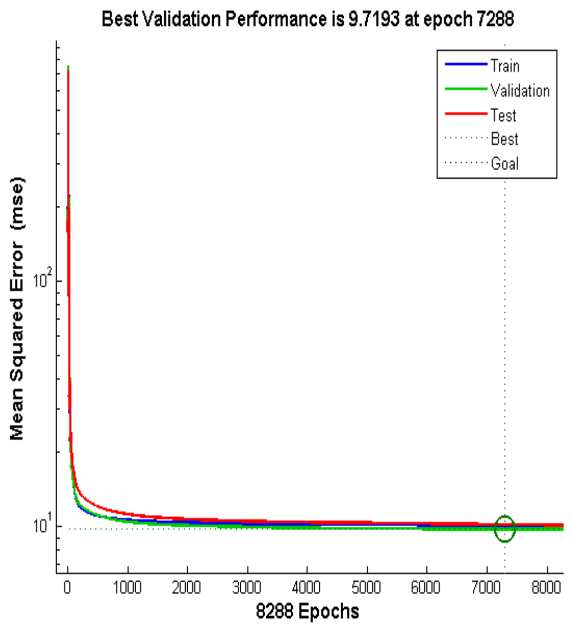


Figure 33: Plot of MSE vs number of epochs for bivariate data regression with different parameter values

(a) $\eta = 0.1$, $\alpha = 0.9$



(b) $\eta = 0.05$, $\alpha = 0.9$



7.4 Classification vs regression

It can be seen that we can choose our learning rate parameter to be fairly high in the case of classification while for regression, the learning rate should in general be low. This can be explained by the fact that the task of function approximation is inherently more complex than a classification task for an MLFFNN model. In case of classification, the function(s) the network tries to approximate are constant (either 1 or 0) over large regions of the input space. The network only needs to learn to approximate the discontinuities i.e. the decision boundaries to be able to perform the task. In regression, the network needs to learn the precise values of the desired output function at all points of the input space. This results in the error surface being much rougher in case of regression and a high learning rate causes the weights to be highly unstable. This same notion can be used to understand why even among the classification or regression tasks themselves, the ideal learning rate value decreases as the complexity of the task increases. For example, the classification of the image dataset requires a much lower learning rate for proper converges than the linearly separable or non-linearly separable datasets. Another rule-of-thumb that we have used is that if we decrease the learning rate, then we should generally increase the momentum factor to ensure faster convergence, particularly in the initial learning period since the effective learning rate in low curvature regions is given by $\hat{\eta} = {}^{\eta}/_{(1-\alpha)}$.

A Few Notes on the Implementation

It should be noted here that although maximum use has been made of the MATLAB toolbox for the MLFFNN portions of the assignment, our original choice was to write our own code. However, the final decision to go with the available toolbox was made because our implementation was very slow compared to the toolbox, as was to be expected, and this hampered our efforts not only to experiment with it but also debug it in case of errors. Also, the toolbox provided easy access to algorithms like the conjugate gradient method.

While every effort has been made to customize the implementation according to our requirements, there were some issues that we faced while doing so. While the toolbox API permits customization of most features, it provides scant documentation on how to write "custom" functions. So, we sometimes had to make do by choosing one of the several built-in functions provided. MATLAB provides a wide variety of choices for most settings and thus, this was not an issue most of the time. With regards to customization, we were even able to write our own custom weight initialization function based on the fan-in of a node, as taught in class. But we faced a lot of problems while writing functions that could accept parameters as inputs. For example, the built-in sigmoidal activation functions do not accept a slope parameter and all our efforts to write our own such function that could interface with the API were unsuccessful. Some issues related to the normalization of data by the toolbox also came up when we wrote the programs to simulate the neural networks and get the hidden layer outputs.

The RBFNN and perceptron models use our own code, rather than the existing implementations in MATLAB.

References

[SK] Satish Kumar, *Neural Networks: A Classroom Approach,* 2ed, McGraw-Hill

[DH] Richard O. Duda, Peter E. Hart, David G. Stork, *Pattern Classification,* 2ed, Wiley Student Edition

[SH] Simon Haykin, *Neural Networks and Learning Machines,* 3ed, Prentice-Hall of India

[SCG] http://www.ra.cs.uni-tuebingen.de/SNNS/UserManual/node241.html

[WP] http://en.wikipedia.org/wiki/Types_of_artificial_neural_networks