NEURAL NETWORK SIMULATION USING MATLAB FOR UNDERSTANDING HOW TO SOLVE PATTERN RECOGNITION PROBLEMS USING BACKPROPAGATION

VAISHALI SHIVAKUMAR (6738581)

Assignment Report for AI and AI Programming (EEEM005)

Computer Vision, Robotics and Machine Learning

Department of Electronic Engineering

Faculty of Engineering and Physical Sciences

University of Surrey

Guildford, Surrey, GU2 7XH, UK

Table of Contents

EXECUTIVE SUMMARY

This report focuses on using a multi-layer perceptron (MLP) to solve pattern recognition problems, specifically using the cancer dataset. The experiments in this report vary the number of hidden nodes and epochs to determine their effect on the MLP's learning capacity and robustness. After testing 21 different combinations, it was found that the best combination was 32 hidden layers and 16 epochs. This combination was then used as a basic classifier in experiments 2 and 3, which investigated ensemble learning and the effect of optimizers on MLP learning, respectively. It was found that ensembles with higher base classifiers and the trainrp optimizer performed better. Experiment 4 explored the use of an MLP to produce nonlinear decision boundaries for overlapping bivariate classes. The results showed that the plotted decision boundary was close to the optimal decision boundary with a small Euclidean distance. Overall, this report highlights the importance of varying the number of hidden nodes and epochs in an MLP and the effectiveness of ensemble learning and different optimizers in improving MLP performance.

INTRODUCTION

The process of pattern recognition is essential to machine learning, which involves analyzing and understanding patterns and correlations among the data to extract new information. Neural networks are powerful tools for pattern recognition, capable of learning complex mathematical functions and mapping input data to output values. Neural networks use a single hidden layer to learn these functions, and the universal approximation theorem proves that these networks can learn any non-linear function. The backpropagation algorithm is used to train these neural networks, which involves feeding data through the network in a forward direction, calculating the loss, and then backpropagating the error to update the weights and biases of the network.

In this assignment, we focus on how neural networks can solve pattern recognition problems in a cancer dataset. The first part of the assignment involves creating an optimal neural network with one layer by experimenting with the number of epochs and hidden nodes. The second part builds upon the first by using the optimal network to experiment with ensemble techniques. The third part of the assignment repeats the experiments with different optimizers. Finally, the fourth part focuses on distinguishing between two equiprobable, overlapping classes in two dimensions using neural networks. Throughout the assignment, MATLAB is used to implement and analyze the neural networks, experimenting with different parameters to optimize the performance of the networks. Overall, this assignment provides a comprehensive overview of how neural networks can be used for pattern recognition and the importance of experimentation to optimize the performance of these networks.

1. Experiment 1:

1.1 Overview

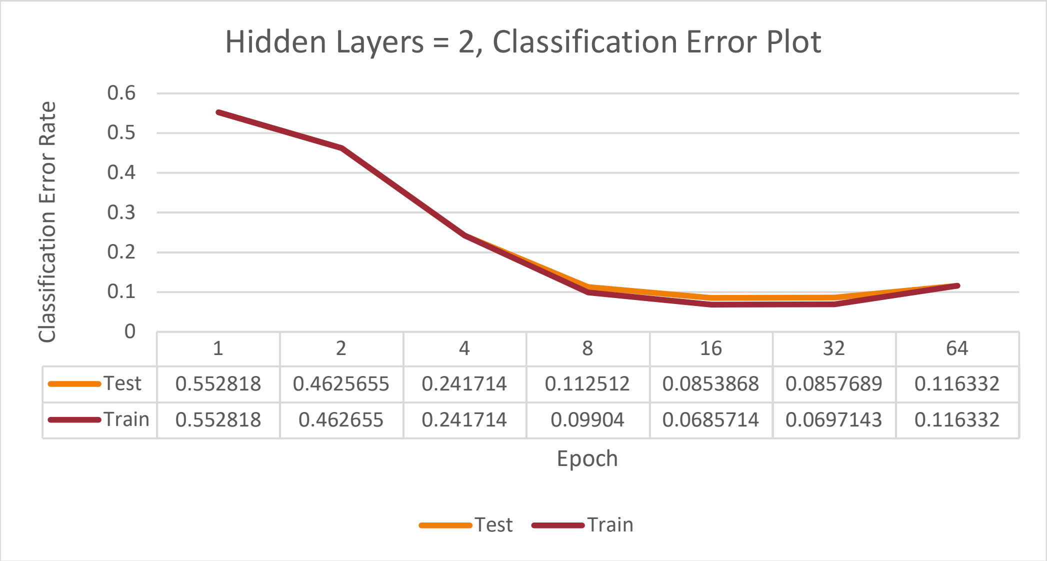
The experiment is about training neural networks with a cancer dataset available in the MATLAB repository. This network is built using the nntools library of MATLAB, which is a multi-layer perceptron with one hidden layer. The nntools is a flexible platform to build and train neural networks with less coding. The nntools app enables to experiment with different parameters. After satisfactory parameters are obtained, the script for the neural network can be obtained using the script generation function in the tool and later optimized in the MATLAB workspace. The dataset is randomly split into 50/50 ratios for training and testing the neural network. The model experiments against different epochs-hidden layer combinations.

1.2 The procedure

The experiment is done for a combination of 3 different hidden layers with neurons count of 2, 8 and 32 and with different epochs values as 1, 2, 4, 8, 16, 32 and 64. Thus a total of 21 combinations of results can be obtained. The early stopping is set to 0 so that the model can run for the specified number of epochs. Each of these combinations is run 30 times with different values of train-test split ratios. After running 30 times for each of these combinations the mean and standard deviation of the classification error are computed which is then used to evaluate the performance of the model. This procedure is repeated for all the other combinations as well and finally, 21 values of the mean and standard deviation of the classification error are obtained. Finally, the mean values are plotted against the number of epochs for visualization and analysis.

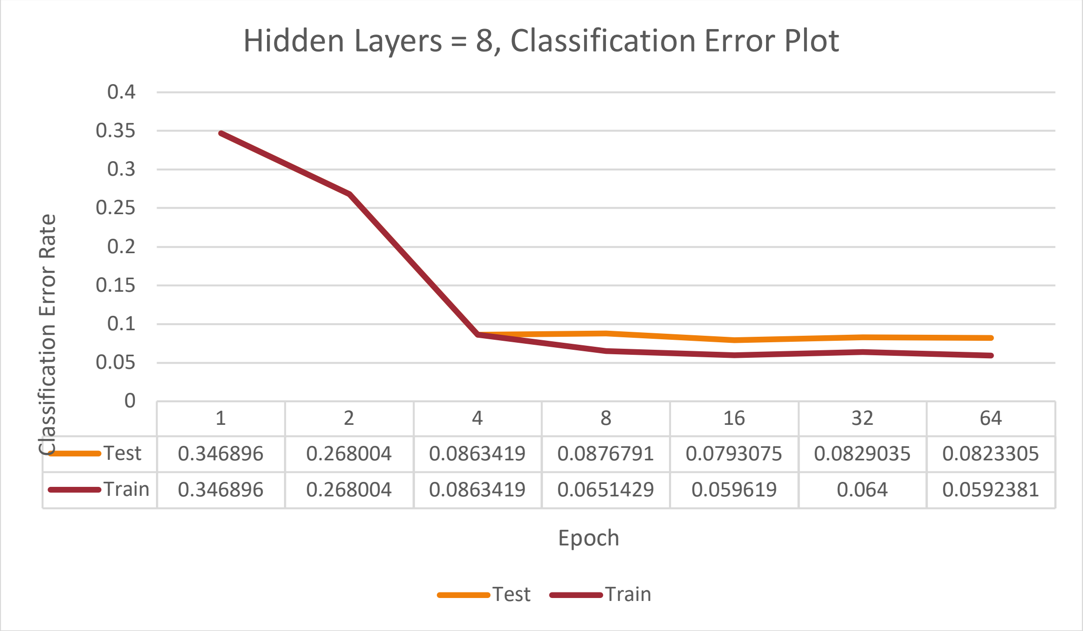
1.3 Results and Analysis

The first model with 2 hidden layers starts with the same test and train classification error rates of 0.552818 and however the train and test error rates start to deviate from the 8th epoch.



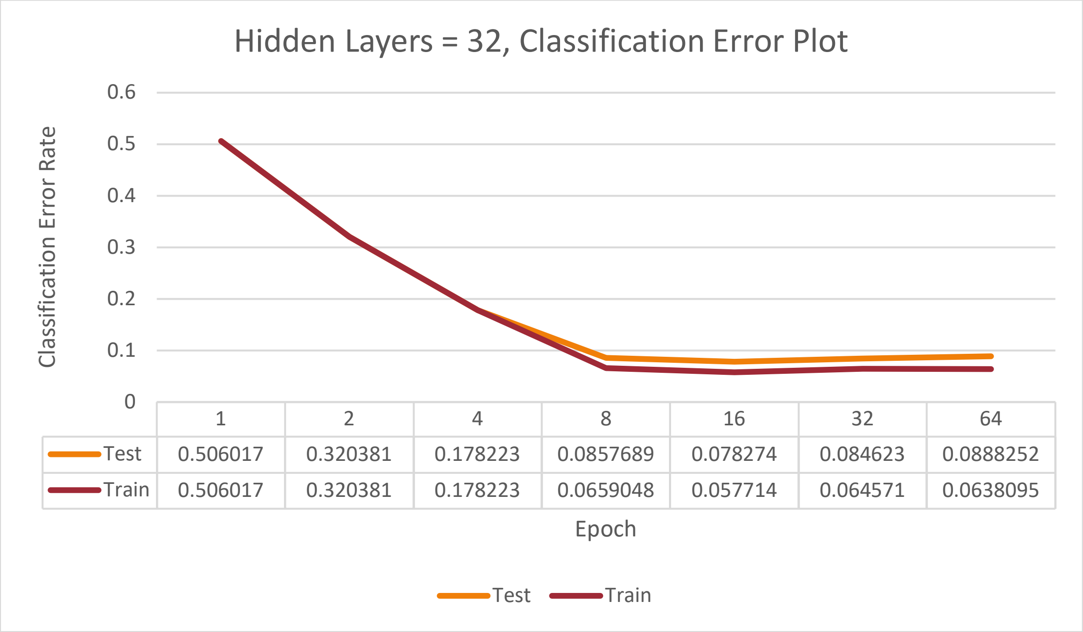
	1	2	4	8	16	32	64
Test	0.552818	0.4625655	0.241714	0.112512	0.0853868	0.0857689	0.116332
Train	0.552818	0.462655	0.241714	0.09904	0.0685714	0.0697143	0.116332

Then from the 16th epoch the error rates become stable, but the test error rate is found to be above the train error rate. Overall, the model has the least test error rate of 0.08538 at the 16th Epoch and least train Classification error rate of 0.06857 at the 16th Epoch.



	1	2	4	8	16	32	64
Test	0.346896	0.268004	0.0863419	0.0876791	0.0793075	0.0829035	0.0823305
Train	0.346896	0.268004	0.0863419	0.0651429	0.059619	0.064	0.0592381

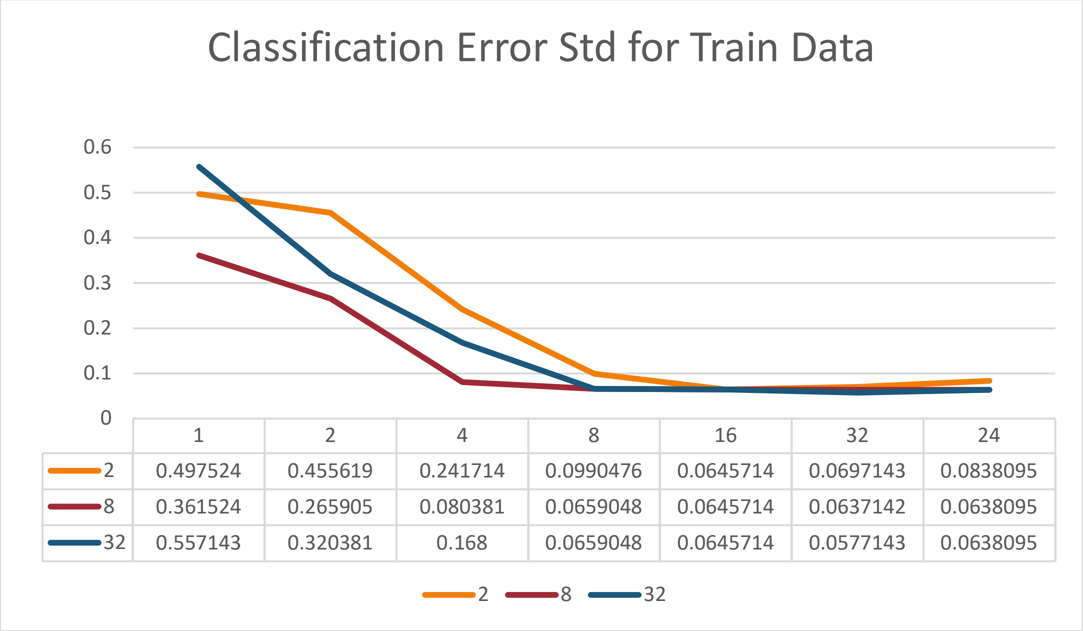
The model with 8 hidden layers, it starts off with classification error rate of 0.346896 however the both the train Classification error rate and the test classification error rates dips sharply at the 4th Epoch indicating the fact that model can learn quickly as the nodes are increased. From the graph it can be ascertained that the error rates become stable from the 4th epoch and the test classification error rate beings to deviate from the tarin classification error rate and lies above the train classification error rate indicating overfitting of the model. As the epoch increases there is a clear difference in train and test classification error rates.



	1	2	4	8	16	32	64
Test	0.506017	0.320381	0.178223	0.0857689	0.078274	0.084623	0.0888252
Train	0.506017	0.320381	0.178223	0.0659048	0.057714	0.064571	0.0638095

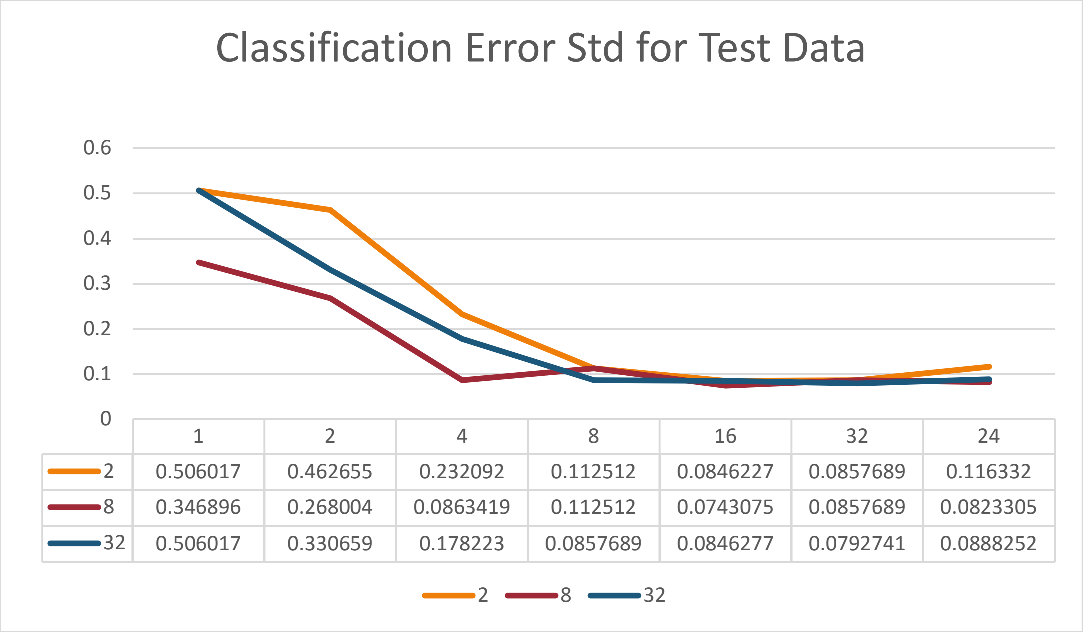
In the model with 32 hidden layers the dip in error rate is gradual and no sharp dip is found. However, from the 8th epoch the difference between the classification error rates of the train and test data can be clearly seen and the test data lies well above the train data indicating overfitting of the model from the 8th epoch. But the model reached the global minimum value of 0.0577143 for the train classification error rate at the 16th epoch and 0.0782741 for the test classification error rate at the 16th epoch.

Overfitting occurs when the model learns too much from the training data which causes poor performance on the test data. This happens because the model learns the noise present in the training data. Thus, the model begins to show high variance when overfitting occurs.



	1	2	4	8	16	32	24
2	0.497524	0.455619	0.241714	0.0990476	0.0645714	0.0697143	0.0838095
8	0.361524	0.265905	0.080381	0.0659048	0.0645714	0.0637142	0.0638095
32	0.557143	0.320381	0.168	0.0659048	0.0645714	0.0577143	0.0638095

The graph above and the one below shows the standard deviations for each epoch node combination for 30 iterations. Both the train data and the test data follow a similar pattern.



	1	2	4	8	16	32	24
2	0.506017	0.462655	0.232092	0.112512	0.0846227	0.0857689	0.116332
8	0.346896	0.268004	0.0863419	0.112512	0.0743075	0.0857689	0.0823305
32	0.506017	0.330659	0.178223	0.0857689	0.0846277	0.0792741	0.0888252

From the first experiment, the optimum hidden layer epoch combination is 32-16. This is because the training classification error rate has the global minimum value of 0.0577 and the test classification error rate has a minimum value of 0.0782. Using the above-identified combination. Experiment 2 is carried on.

2. Experiment 2:

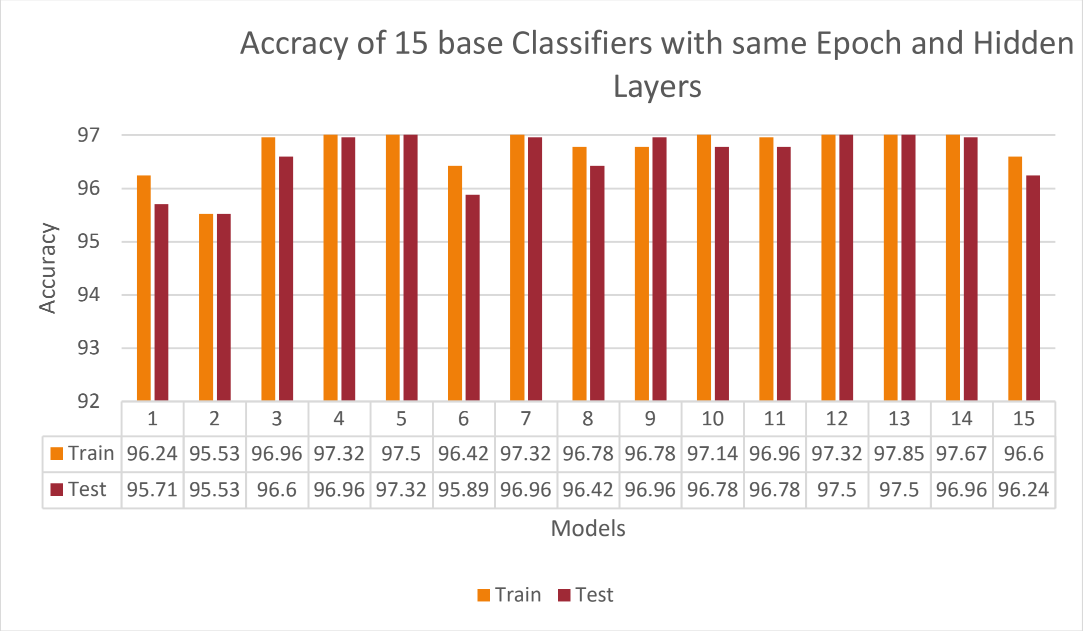
2.1 Overview

This experiment is about using ensemble technique to study the variations in accuracy of the prediction. In ensemble learning multiple individual models are aggregated and the aggregated model output is used for prediction. The prediction is done based on majority voting. The majority voting is a technique where the predicted output will be the output class which majority of the classifiers in the ensemble had predicted. A base classifier is defined with the optimal values of epoch and hidden layers obtained from experiment 1. Then the same base classifier is used to create an ensemble of 15 models with random initial weights. They are treated as both individual classifiers and then as ensembles. Further, the experiment is carried out by varying the count of base classifiers, then with a different ensemble of different node epoch combinations to study the variations in the accuracy of the model.

2.2 The procedure

This experiment is done in different stages. In the first stage, the individual classifiers are trained, and their individual accuracies are identified. Then the cumulative accuracy of the ensemble model is found using the ensemble technique. In the second stage, the number of base classifiers count is varied, the experiment is repeated, and the accuracy of individual ensemble models is calculated. In the third stage, the number of base classifiers count is kept constant, but the experiment is repeated for different node epoch combinations below the optimal combination and the accuracy of the ensemble model is computed. In the fourth stage, the number of base classifiers count is kept constant, but the experiment is repeated for different node epoch combinations above the optimal combination and the accuracy of the ensemble model is computed. In the final stage, the number of hidden layers and the number of base classifier count is kept constant, but the epoch is varied and then the accuracy of the ensemble model is computed. Later the accuracy of the ensemble model from the second stage to the final stage is plotted for visualization. In order to ascertain that any training data biases are alleviated, each model is trained 30 different times with 30 different training and test data combinations.

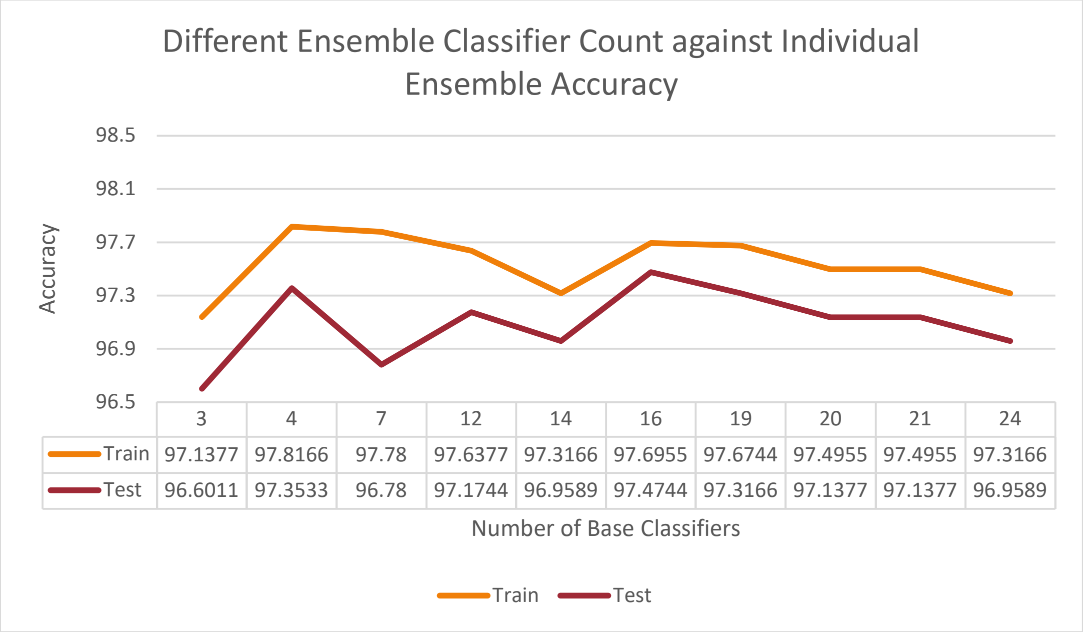
2.3 Results and analysis



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
■ Train	96.24	95.53	96.96	97.32	97.5	96.42	97.32	96.78	96.78	97.14	96.96	97.32	97.85	97.67	96.6
■ Test	95.71	95.53	96.6	96.96	97.32	95.89	96.96	96.42	96.96	96.78	96.78	97.5	97.5	96.96	96.24

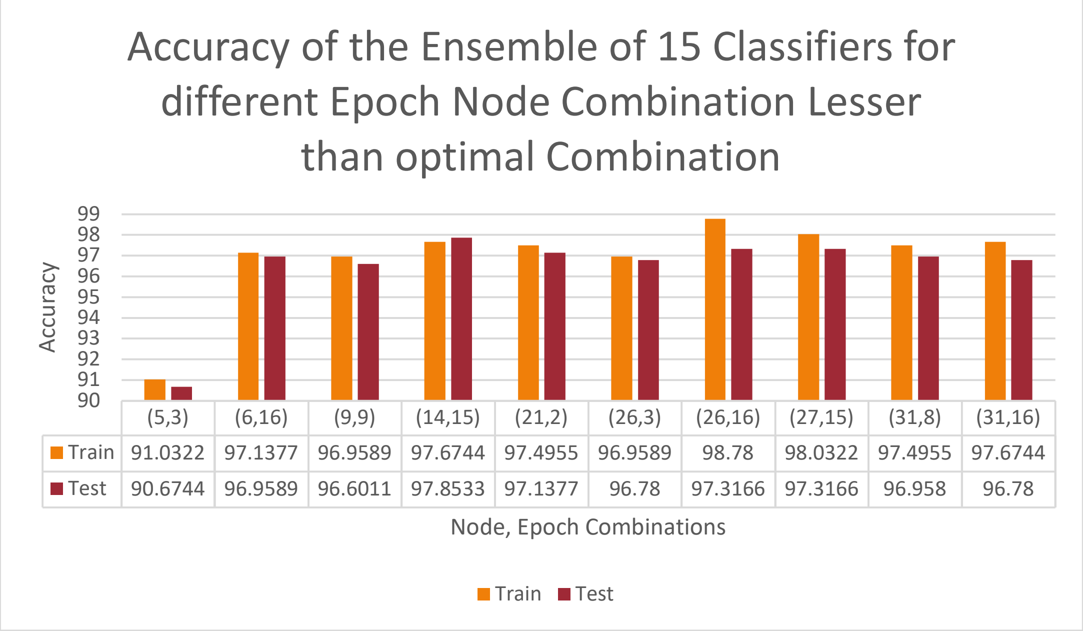
The above Bar graph depicts the accuracy of individual models in an ensemble to Train and test accuracies. Most of the models other than 1, 2 and 6 have similar train accuracy scores because the weights of the models are initialized randomly, and the random value seed each time depends on the iteration value. The average Mean train accuracy and Average Mean Test accuracy for this ensemble are 96.959% and 96.674% respectively. So, from the accuracy score it can be confirmed that despite the weights of the models being initialized randomly the ensemble has learned in a similar way. The learning rate is not affected by initializing the weights randomly.

In the second stage of the experiment, the epoch and node values are kept constant but now the experiment is conducted with an ensemble of classifiers. An ensemble of 10 classifiers in a range of 3-25 is randomly used for training the data and thereafter the accuracies of the individual ensemble are plotted, and the overall mean accuracy of all the ensembles is calculated. Here each of the ensembles is considered an individual model.



	3	4	7	12	14	16	19	20	21	24
Train	97.1377	97.8166	97.78	97.6377	97.3166	97.6955	97.6744	97.4955	97.4955	97.3166
Test	96.6011	97.3533	96.78	97.1744	96.9589	97.4744	97.3166	97.1377	97.1377	96.9589

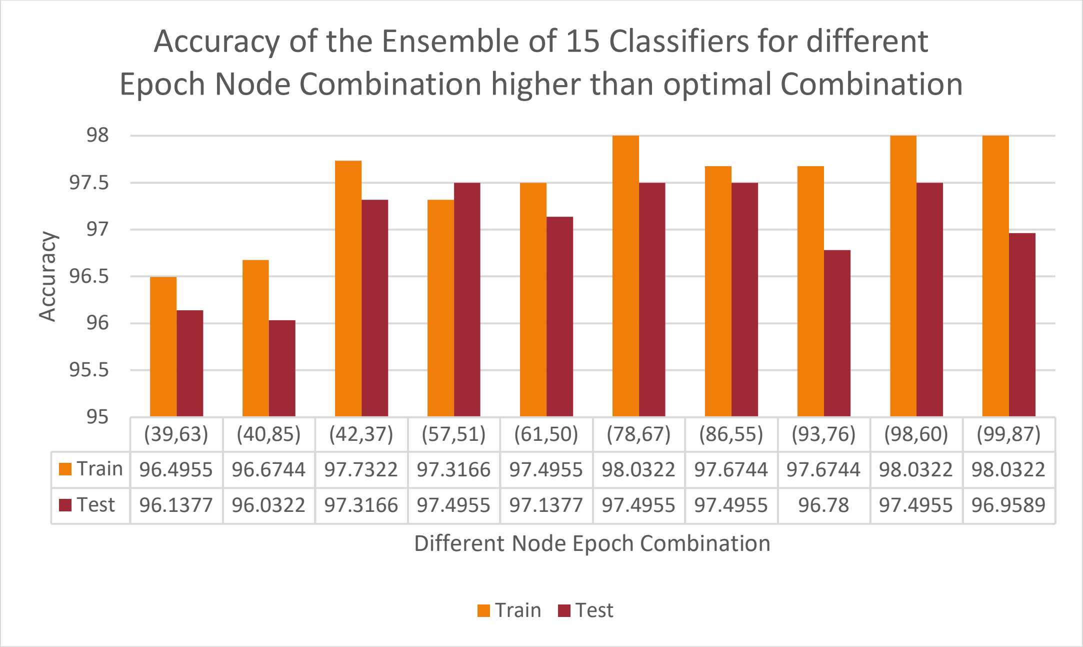
The accuracy varies as the number of counts of base classifiers in the ensemble. The ensemble performed well for higher count values and performed same as the base classifier at lower values for the test data. Beyond 15, the ensemble shows overfitting as the test accuracy value starts decreasing.



	(5,3)	(6,16)	(9,9)	(14,15)	(21,2)	(26,3)	(26,16)	(27,15)	(31,8)	(31,16)
Train	91.0322	97.1377	96.9589	97.6744	97.4955	96.9589	98.78	98.0322	97.4955	97.6744
Test	90.6744	96.9589	96.6011	97.8533	97.1377	96.78	97.3166	97.3166	96.958	96.78

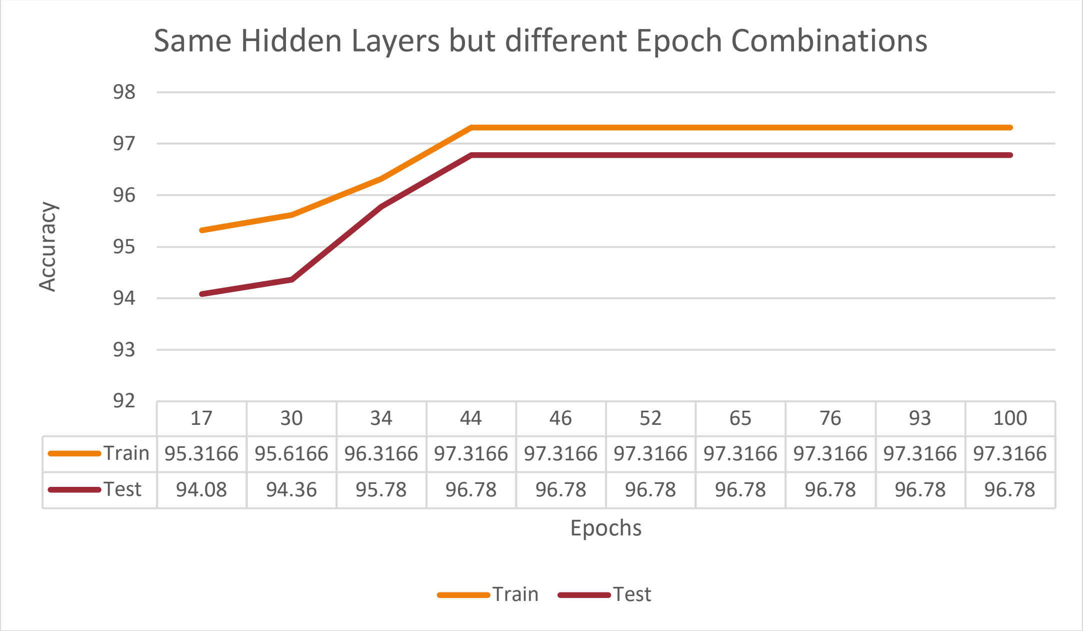
In the third stage, the number of base Classifier count is kept constant, but the experiment is repeated for different nodes, epoch combinations which are below optimum node-epoch combination of 32-16. The graph above represents the accuracy value of the ensemble of 15 Classifiers for 10 different Node Epoch Combinations which are lower than the optimum 32-16 combination. It can be seen from the graph that the ensemble model.

shows high variation. The lower values show poor performance. Thus, the ensemble has poor performance if both the hidden layers and epoch values are less. Node values 26 and beyond have shown much better performance complemented with similar epoch values. Also, most of the combinations where node epoch values were closer had similar train and test accuracies.


Accuracy of the Ensemble of 15 Classifiers for different Epoch Node Combination higher than optimal Combination

	(39,63)	(40,85)	(42,37)	(57,51)	(61,50)	(78,67)	(86,55)	(93,76)	(98,60)	(99,87)
🟧 Train	96.4955	96.6744	97.7322	97.3166	97.4955	98.0322	97.6744	97.6744	98.0322	98.0322
🟥 Test	96.1377	96.0322	97.3166	97.4955	97.1377	97.4955	97.4955	96.78	97.4955	96.9589

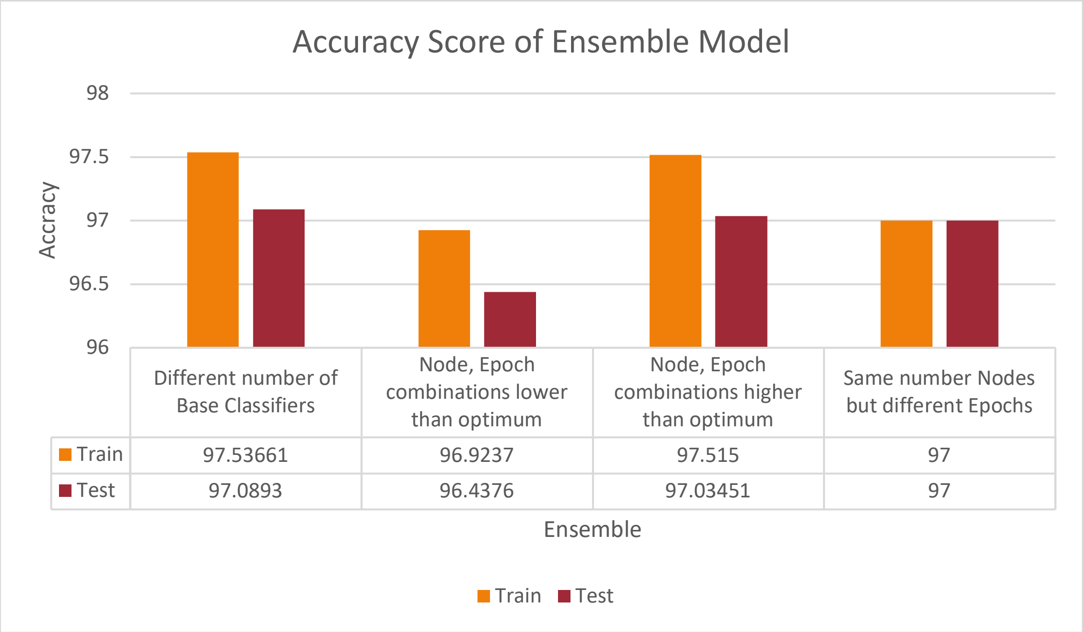
In the fourth stage, the experiment was repeated with different Epoch, Node combinations higher than optimal value with same number of base classifiers in the ensemble. The pairs with different epoch and hidden layer combinations exhibit the lowest performance, while the rest of the other combinations performed better. The combination with 78 nodes and 67 epochs has performed well, but the rest of the combinations show overfitting of the data.



	17	30	34	44	46	52	65	76	93	100
Train	95.3166	95.6166	96.3166	97.3166	97.3166	97.3166	97.3166	97.3166	97.3166	97.3166
Test	94.08	94.36	95.78	96.78	96.78	96.78	96.78	96.78	96.78	96.78

In the fifth stage the experiment is repeated with constant number of nodes but different epoch values for this ensemble. The number of hidden neurons was fixed at 32, and the range of epochs was set between 1 and 100. It was observed that the model struggled to learn with only 2 epochs, leading to high error rates and lower accuracy. However, as the number of epochs increased, the model began to learn better and showed signs of overfitting.

Nevertheless, it was noted that after the optimal epoch was reached, the model began to overfit by learning the noise in the data. Each model was trained individually with a different epoch number, so the variation in train loss across epochs was due to random weight variability. Finally, an ensemble of these models was used to calculate accuracies for both train and test data.



	Different number of Base Classifiers	Node, Epoch combinations lower than optimum	Node, Epoch combinations higher than optimum	Same number Nodes but different Epochs
🟧 Train	97.53661	96.9237	97.515	97
🟥 Test	97.0893	96.4376	97.03451	97

In the experiment, an ensemble was created with multiple base classifiers, outperformed other ensemble. Furthermore, other ensembles also outperformed their respective individual classifiers. This highlights the effectiveness of ensemble techniques in improving accuracy and reducing classification errors.

3. Experiment 3

3.1 Overview

Experiment 3 follows the same methodology as Experiment 2 but involves a change in optimizers. The optimizer 'trainscg' used in Experiment 2 is replaced with 'trainlm' and 'trainrp', and the training process is repeated.

3.2 Procedure

The backpropagation algorithm updates the weights of neural networks, but it needs to choose the optimal weights to enable learning. Inappropriate weights can prevent the model from learning. The weights are chosen to minimize the loss, and this is accomplished using optimizers. Optimizers are algorithms that change the parameters of neural networks, such as learning rate, to minimize loss [1]. There are various optimizers available, including Gradient Descent, Stochastic Gradient Descent, Adam, Levenberg–Marquardt, RProp, RMS Prop, and Conjugate gradient.

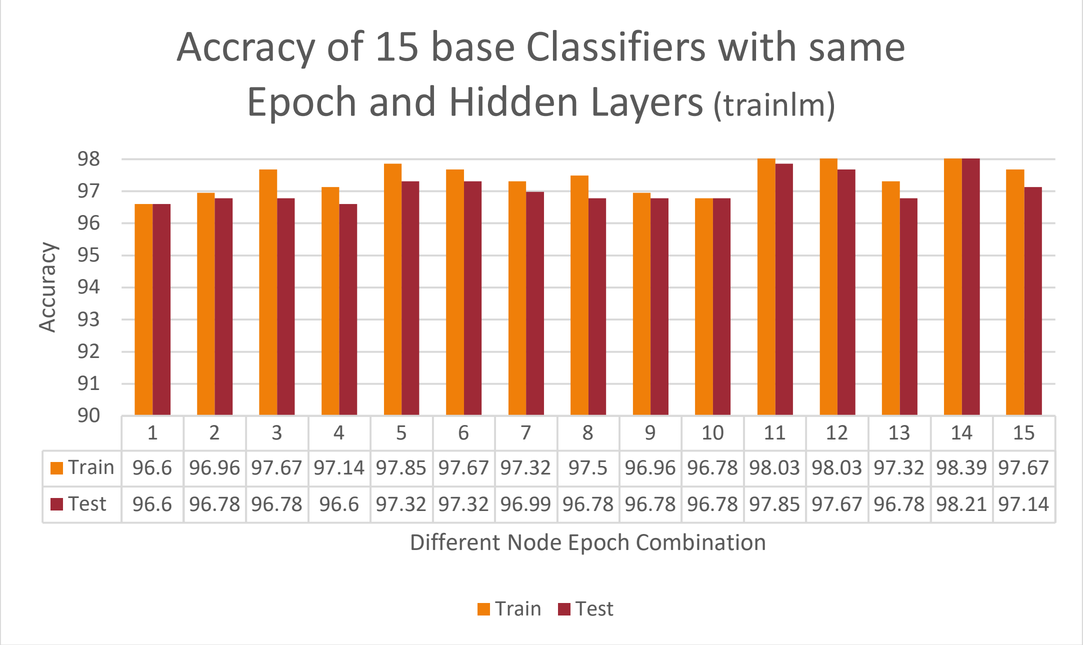
Gradient Descent is one of the most important optimization algorithms and is considered the backbone of machine learning and deep learning. It is a first-order optimization algorithm that finds the direction of the steepest descent to minimize the loss function. In gradient descent, the goal is to reach the global minimum of the loss function based on several parameters, such as current weights, learning rate, and loss function. The algorithm decides how weights should be updated using the equation[2]

$$A(n+1) = A(n) - \eta \cdot \nabla L(A)$$

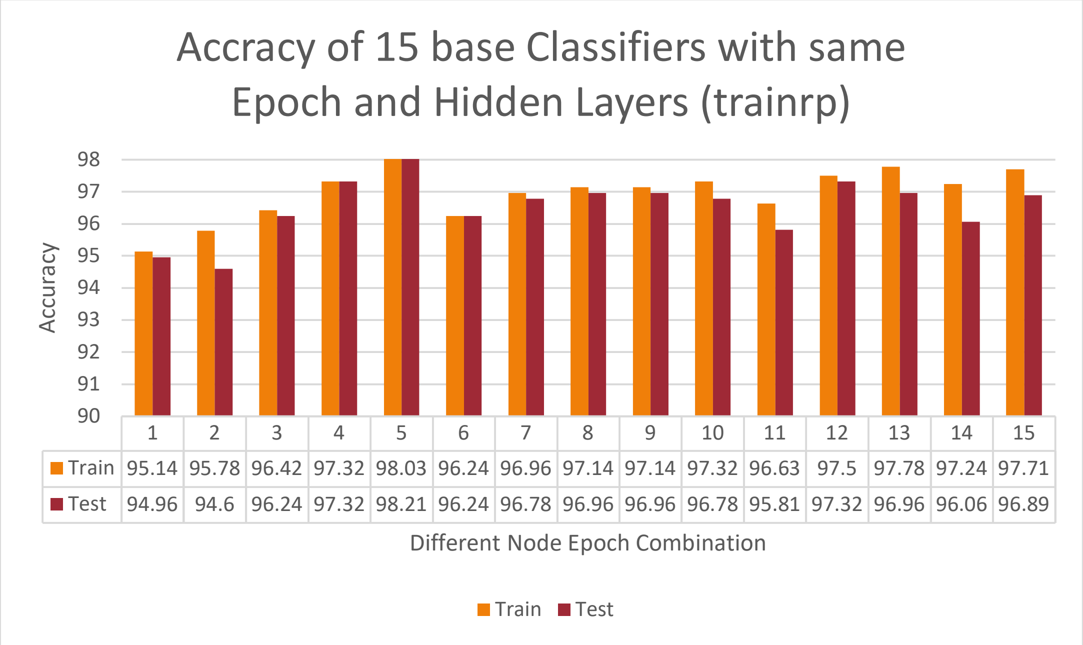
where A(n+1) is a new weight, A(n) is the current weight, η is the learning rate, and L(A) is the loss function. The negative sign in the equation represents how much the loss function is subtracted from the current weights so that the algorithm could go against the gradient, towards the local minima. If the gradients are steeper, the steps are larger, and if the gradients are flatter, the algorithm takes smaller steps, which is taken care of by the learning rate. In this experiment, the current learning rate is 0.01.

In Experiment 3, the optimizers used in Experiment 2 (trainscg) are replaced with Levenberg–Marquardt (trainlm) and RProp (trainrp) algorithms. The Levenberg–Marquardt algorithm is an iterative algorithm used to solve least-square problems where the systems are non-linear. It can find global minima if there are fewer local minima [3]. On the other hand, RProp algorithm is a popular algorithm that uses the sign and magnitude of gradients to update the weights. It can adapt well to each step and handle weights individually [4]. Using these algorithms changes the learning pattern of the base classifiers, which can be analyzed to evaluate the performance of the optimizers.

3.3 RESULTS & ANALYSIS



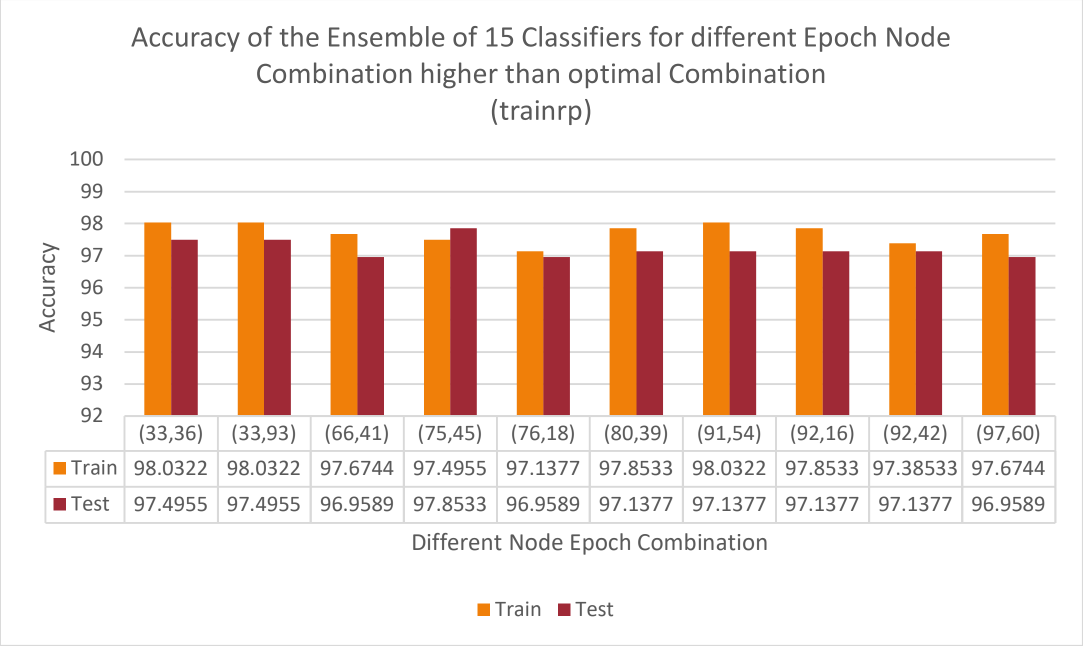
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
■ Train	96.6	96.96	97.67	97.14	97.85	97.67	97.32	97.5	96.96	96.78	98.03	98.03	97.32	98.39	97.67
■ Test	96.6	96.78	96.78	96.6	97.32	97.32	96.99	96.78	96.78	96.78	97.85	97.67	96.78	98.21	97.14



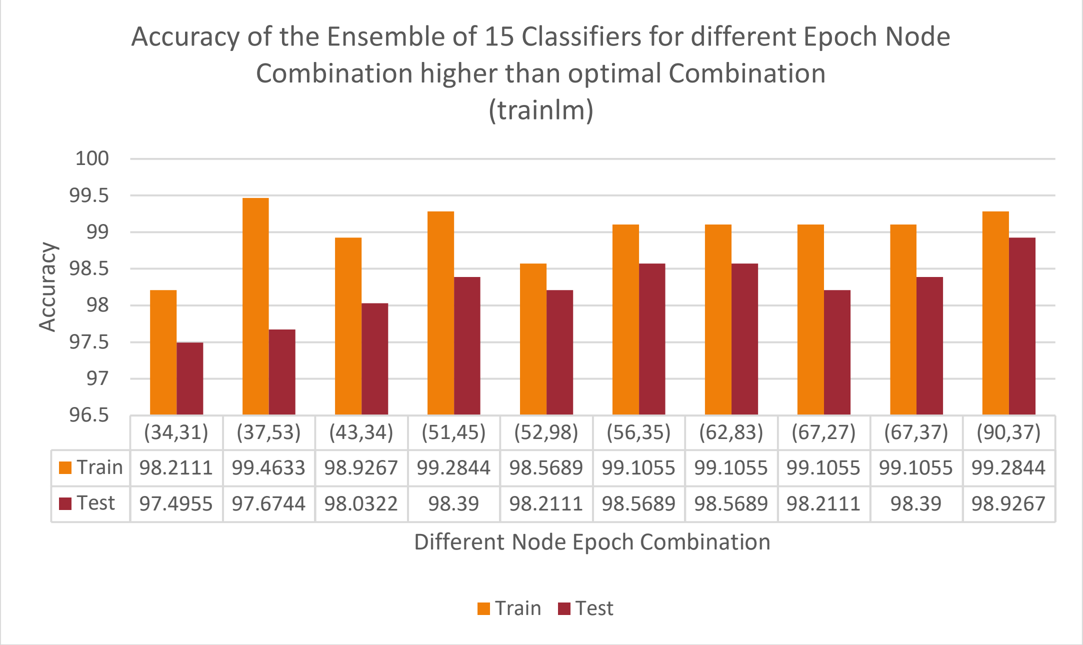
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
■ Train	95.14	95.78	96.42	97.32	98.03	96.24	96.96	97.14	97.14	97.32	96.63	97.5	97.78	97.24	97.71
■ Test	94.96	94.6	96.24	97.32	98.21	96.24	96.78	96.96	96.96	96.78	95.81	97.32	96.96	96.06	96.89

In the beginning, the ensemble of models is trained using the trainlm optimizer. Upon analyzing the train and test errors, it is observed that the accuracy is much better than what was obtained using the trainscg optimizer. This trend is consistent across all the ensemble models with varying parameters. In particular, all base classifiers that have different epochs and hidden layers and lower than optimal layers and epochs have performed better than all the base classifiers with trainscg optimizer.

To compare the performance of the trainlm and trainrp optimizers on the ensemble of 15 classifiers with the same epoch and hidden layers, their graphs were analyzed. It was observed that trainlm resulted in higher train accuracy, but trainrp showed better test accuracy.



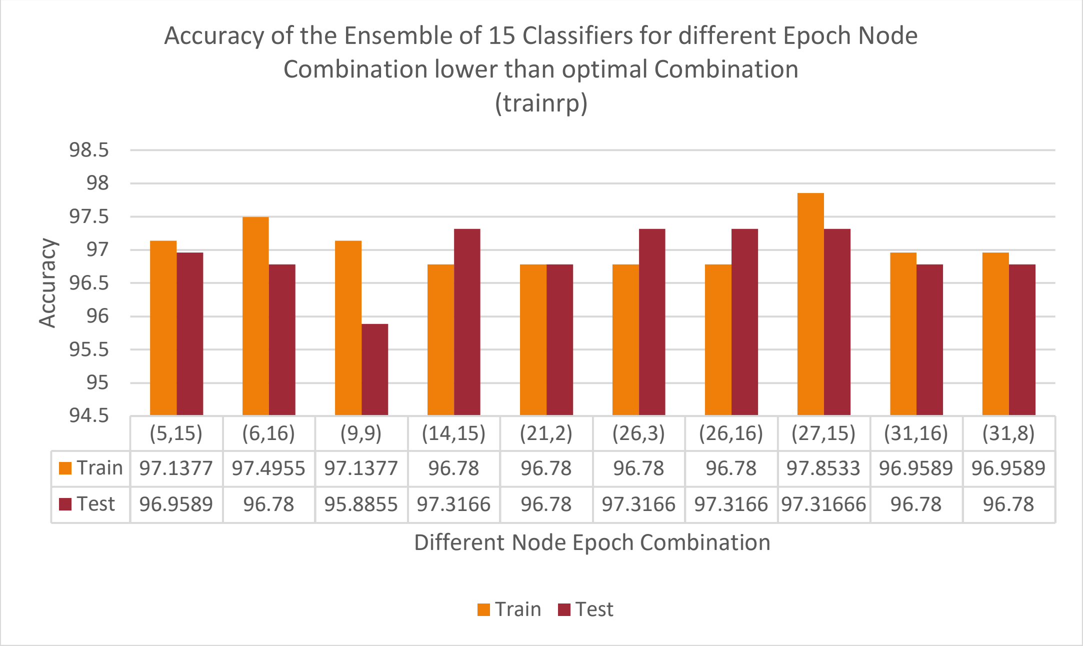
	(33,36)	(33,93)	(66,41)	(75,45)	(76,18)	(80,39)	(91,54)	(92,16)	(92,42)	(97,60)
Train	98.0322	98.0322	97.6744	97.4955	97.1377	97.8533	98.0322	97.8533	97.38533	97.6744
Test	97.4955	97.4955	96.9589	97.8533	96.9589	97.1377	97.1377	97.1377	97.1377	96.9589



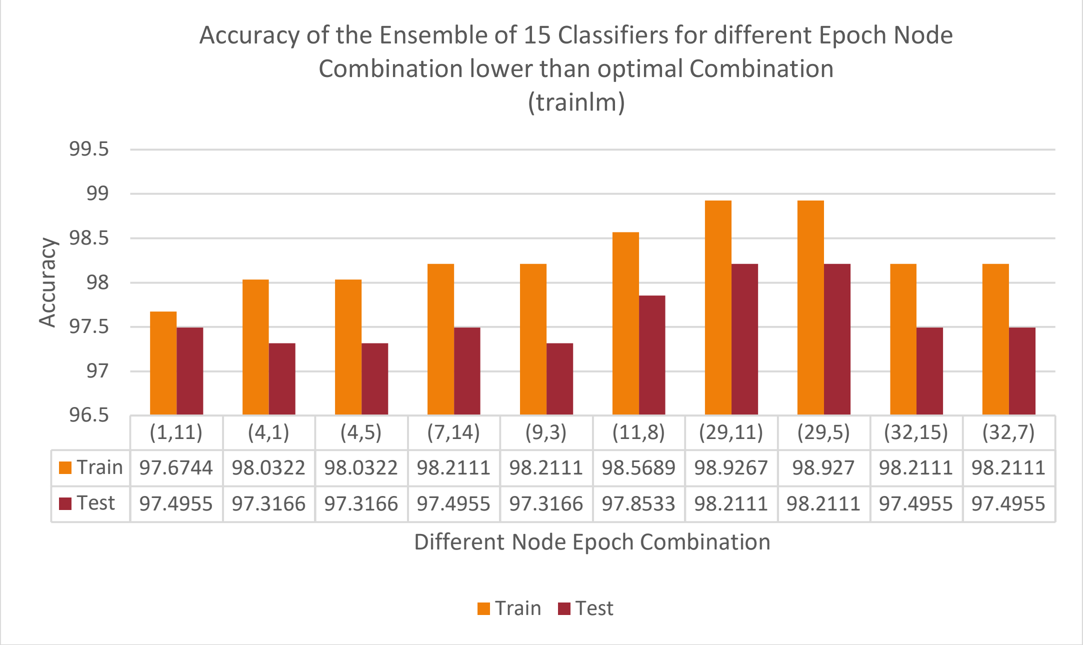
	(34,31)	(37,53)	(43,34)	(51,45)	(52,98)	(56,35)	(62,83)	(67,27)	(67,37)	(90,37)
Train	98.2111	99.4633	98.9267	99.2844	98.5689	99.1055	99.1055	99.1055	99.1055	99.2844
Test	97.4955	97.6744	98.0322	98.39	98.2111	98.5689	98.5689	98.2111	98.39	98.9267

The previous set of graphs demonstrated a consistent pattern where trainlm performed better in terms of train accuracy, while trainrp performed better in terms of test accuracy, implying a slight overfitting. In contrast, the following graphs, which maintain a constant number of hidden nodes, show that trainrp consistently outperforms trainlm. However, there are a few instances where trainlm has a slight edge over trainrp, but they are isolated. Additionally, the gap between train and test data is smaller in the model trained using trainrp than in the model

trained using trainlm, indicating that the model trained with trainlm has more overfitting issues.

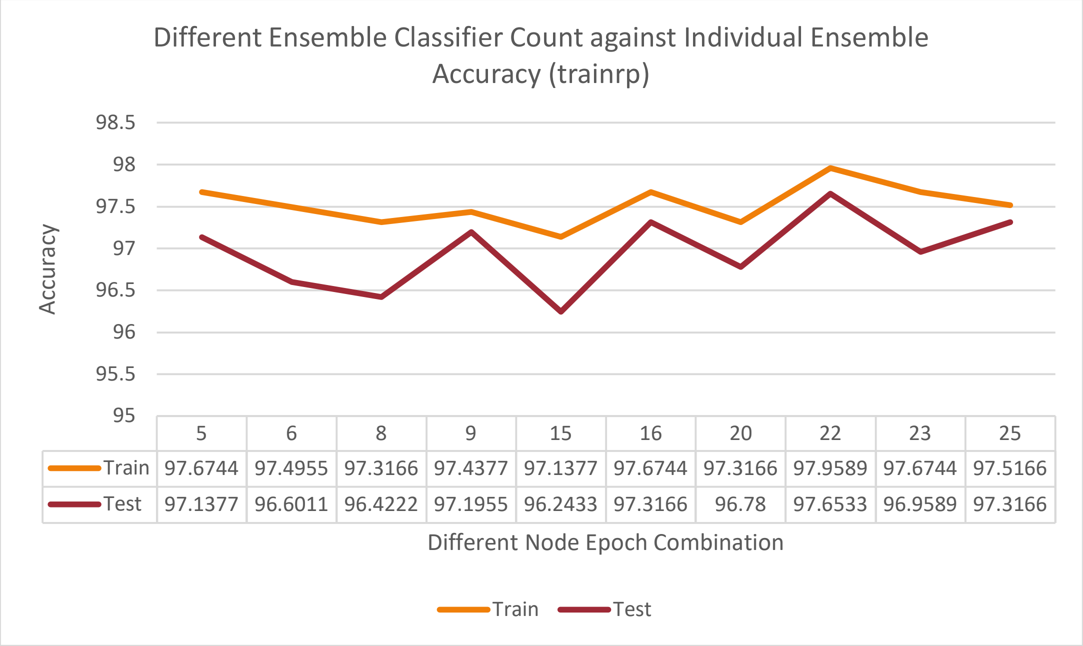

Accuracy of the Ensemble of 15 Classifiers for different Epoch Node Combination lower than optimal Combination (trainrp)

	(5,15)	(6,16)	(9,9)	(14,15)	(21,2)	(26,3)	(26,16)	(27,15)	(31,16)	(31,8)
■ Train	97.1377	97.4955	97.1377	96.78	96.78	96.78	96.78	97.8533	96.9589	96.9589
■ Test	96.9589	96.78	95.8855	97.3166	96.78	97.3166	97.3166	97.31666	96.78	96.78

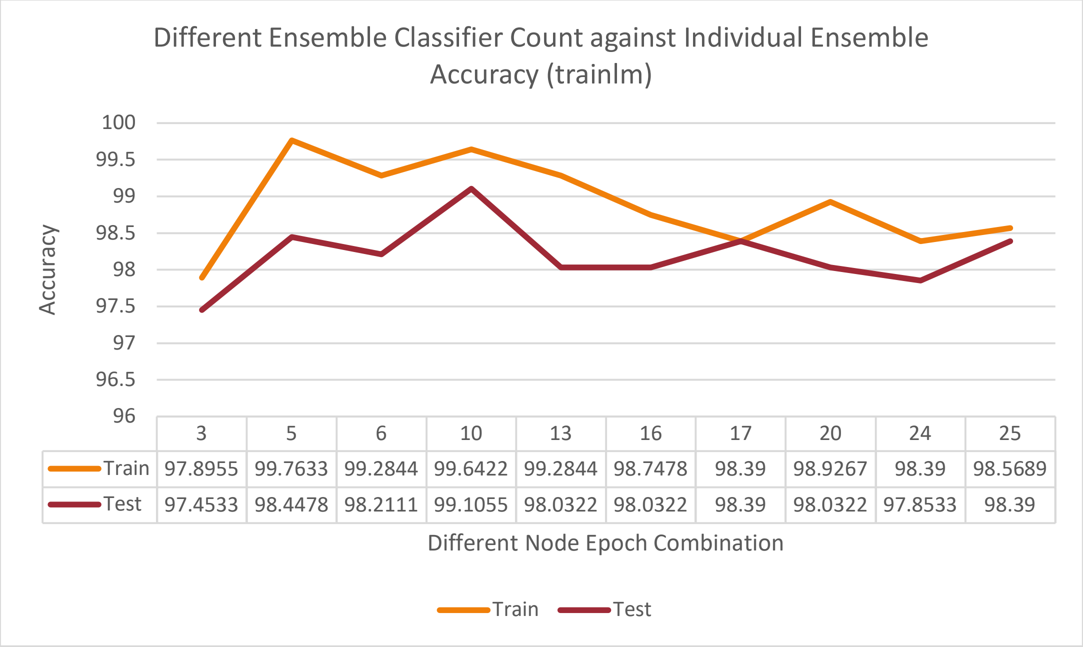

Accuracy of the Ensemble of 15 Classifiers for different Epoch Node Combination lower than optimal Combination (trainlm)

	(1,11)	(4,1)	(4,5)	(7,14)	(9,3)	(11,8)	(29,11)	(29,5)	(32,15)	(32,7)
■ Train	97.6744	98.0322	98.0322	98.2111	98.2111	98.5689	98.9267	98.927	98.2111	98.2111
■ Test	97.4955	97.3166	97.3166	97.4955	97.3166	97.8533	98.2111	98.2111	97.4955	97.4955

In the graph presented below, where an odd number of classifiers were used for different models, we see the same pattern as before. The overall best performance occurs when the number of ensemble models is higher. In both trainlm and trainrp, there is a performance improvement in test accuracy around the 21-25 range. In terms of learning, the same pattern continues, where trainrp shows signs of overfitting while trainlm has train and test accuracies closer to each other.



	5	6	8	9	15	16	20	22	23	25
Train	97.6744	97.4955	97.3166	97.4377	97.1377	97.6744	97.3166	97.9589	97.6744	97.5166
Test	97.1377	96.6011	96.4222	97.1955	96.2433	97.3166	96.78	97.6533	96.9589	97.3166



	3	5	6	10	13	16	17	20	24	25
Train	97.8955	99.7633	99.2844	99.6422	99.2844	98.7478	98.39	98.9267	98.39	98.5689
Test	97.4533	98.4478	98.2111	99.1055	98.0322	98.0322	98.39	98.0322	97.8533	98.39

The above experiment suggests that trainrp optimizer outperforms trainlm and trainscg in terms of minimizing error and maximizing accuracy in test data with less overfitting. It can be considered as evidence that trainrp can adjust to the training data and converge better than other optimizers for this dataset.

4. Experiment 4

4.1 Overview

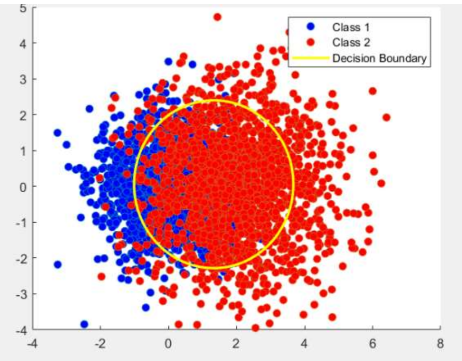
The experiment involves differentiating between two equi-probable classes of overlapping two-dimensional Gaussians. Class 1 has a mean μ1 of [0,0] and variance σ1, while Class 2 has a mean μ2 of [2,0] and variance σ2 of 4. It is known that the optimum boundary between the two classes is a circle centered at [−2/3,0] with a radius of 2.34. To solve the problem, random data points are generated for each class, and neural networks are utilized for model training.

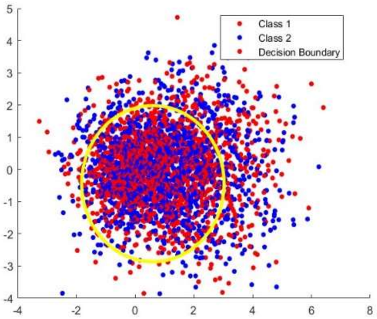
4.2 Procedure

To generate a dataset with two different classes and different mean and variance, the 'mvnrnd' function is used. Each class has 1650 two-dimensional arrays, and when concatenated, it creates a dataset of 3300 records. Two labels are generated for each class, concatenated, and one-hot encoded. The data is then randomly rearranged to ensure a normal distribution. The dataset is split into a 10% training set and a 90% testing set. The 'patternnet' algorithm is used to train the model with various epoch and hidden unit combinations. After experimentation, it is found that the best combination is 100 hidden units and 85 epochs. This combination is then used to generate an ensemble of base classifiers as in experiment 2.

4.3 Results & Analysis

After predicting the data, we calculated the mean and variance of class 2. The calculated mean for class 2 is [1.9773, 0.0800], with a variance of 1.8916 and -0.0139. Using the obtained mean and variance, a circle is plotted on the scatter plot of the predicted values. The below graphs show the predicted classes. Red represents class 1, and blue represents class 2. The small circle represents the decision boundary calculated using the mean and variance. The radius of the circle is 2.4164, with its center at [0.3952, 0.5560]. The Euclidean distance between the optimal origin and obtained origin is 0.635. From the Euclidean distance, it is clear that the values within the decision boundaries are of a similar class, proving the theory that the decision boundary exists between [-2/3,0].





Conclusion

From the four experiments we conducted and the data we analyzed, we have come to several conclusions:

1. Using a lower pair of hidden units and epochs takes more time to learn.

2. A higher pair of hidden units and epochs can achieve better learning.

3. Training a model beyond a certain point can lead to overfitting.

4. Using the same base classifier with different initial weights results in convergence after some time, indicating that the initial weights do not significantly impact model performance.

5. Ensemble techniques can improve the accuracy of both training and testing data.

6. Choosing an appropriate optimizer for the dataset can enhance the model's learning.

7. We were able to demonstrate that for a dataset with two equi-probable classes having mean [0,0] and [2,0] with variances of 1 and 4, respectively, the decision boundary is closer to [-2/3,0] with a Euclidean distance of 0.635.

These observations highlight the ability of neural networks to adapt to different problem statements and learn non-linear boundaries, which is supported by the Universal Approximation Theorem.

CITATIONS

1. https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/#:~:text=An%20optimizer%20is%20a%20function,overall%20loss%20and%20improving%20accuracy.
2. https://en.wikipedia.org/wiki/Gradient_descent
3. https://en.wikipedia.org/wiki/Levenberg%E2%80%93Marquardt_algorithm
4. https://en.wikipedia.org/wiki/Rprop
5. https://in.mathworks.com/help/deeplearning/ug/cancer-detection.html

APPENDIX

Code of Exercise:

```
% Solve a Pattern Recognition Problem with a Neural Network
% Script generated by Neural Pattern Recognition app
% Created 21-Mar-2023 14:50:09
%
% This script assumes these variables are defined:
%
%   cancerInputs - input data.
%   cancerTargets - target data.
[x,t]=cancer_dataset;

x = x;
t = t;

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainscg';  % Scaled conjugate gradient backpropagation.

% Create a Pattern Recognition Network
hiddenLayerSize = 10;
%hiddenLayerSize = 2;
%hiddenLayerSize = 8;
%hiddenLayerSize = 32;
net = patternnet(hiddenLayerSize, trainFcn);

% Choose Input and Output Pre/Post-Processing Functions
% For a list of all processing functions type: help nnprocess
net.input.processFcns = {'removeconstantrows','mapminmax'};

% Setup Division of Data for Training, Validation, Testing
% For a list of all data division functions type: help nndivision
net.divideFcn = 'dividerand';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 50/100;
net.divideParam.valRatio = 35/100;
net.divideParam.testRatio = 15/100;

% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
net.performFcn = 'crossentropy';  % Cross Entropy

% Choose Plot Functions
% For a list of all plot functions type: help nnplot
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
    'plotconfusion', 'plotroc'};


% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)
tind = vec2ind(t);
yind = vec2ind(y);
percentErrors = sum(tind ~= yind)/numel(tind);

% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y)
valPerformance = perform(net,valTargets,y)
testPerformance = perform(net,testTargets,y)

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various plots.
figure, plotperform(tr)
figure, plottrainstate(tr)
figure, ploterrhist(e)
figure, plotconfusion(t,y)
figure, plotroc(t,y)

% Deployment
% Change the (false) values to (true) to enable the following code blocks.
% See the help for each generation function for more information.
if (false)
    % Generate MATLAB function for neural network for application
    % deployment in MATLAB scripts or with MATLAB Compiler and Builder
    % tools, or simply to examine the calculations your trained neural
    % network performs.
    genFunction(net,'myNeuralNetworkFunction');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a matrix-only MATLAB function for neural network code
    % generation with MATLAB Coder tools.
    genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
    y = myNeuralNetworkFunction(x);
```

```
end
if (false)
    % Generate a Simulink diagram for simulation or deployment with.
    % Simulink Coder tools.
    gensim(net);
end

% Perform prediction on the training data
%Y = net(X);

% Convert predicted labels to binary values (0 or 1)
%Y_binary = round(Y);

% Calculate the confusion matrix
%C = confusionmat(T, Y_binary);

% Calculate the classification error rate

%classification_error_rate = sum(sum(C, 2) - diag(C)) / numel(T);
```

Code of Experiment 01:

```
% Load the cancer dataset
[x,t]=cancer_dataset;
x = x;
t = t;

% Define X_train as the training data
X_train = x;
% Definition of nodes and epochs
nodes = [2,8,32];
epochs = [ 1, 2, 4, 8, 16, 32,64];

% Set the number of runs
num_runs = 30;

% Initialize arrays to store results
final_train_error_mean = zeros(length(nodes), length(epochs));
final_test_error_mean = zeros(length(nodes), length(epochs));
final_sd_train = zeros(length(nodes), length(epochs));
final_sd_test = zeros(length(nodes), length(epochs));
final_train_loss = zeros(length(nodes), length(epochs));

% Loop over each node/epoch combination
for i = 1:length(nodes)
    for j = 1:length(epochs)
        train_error_rates = zeros(1, num_runs);
        test_error_rates = zeros(1, num_runs);
        train_losses = zeros(1, num_runs);

        % Loop over each run
        for k = 1:num_runs
            [X_train, y_train, X_test, y_test] = train_test_split(x, t);

            % Train the neural network on the training set
            net = patternnet(nodes(i), 'trainrp');
            net.trainParam.epochs = epochs(j);
            net.trainParam.lr = 0.01;
            net = train(net, X_train, y_train);
            % Perform prediction on the training and test sets
            y_train_pred = net(X_train);
            y_test_pred = net(X_test);

            % Calculate the classification error rate for train and test sets
            train_error_rates(k) = sum(round(y_train_pred) ~= y_train, 'all') /
numel(y_train);
            test_error_rates(k) = sum(round(y_test_pred) ~= y_test, 'all') /
numel(y_test);

            % Calculate the train loss
            train_losses(k) = perform(net, y_train, y_train_pred);
        end

        % Calculate average and standard deviation of train error rates, test error
rates, and train losses
        final_train_error_mean(i, j) = mean(train_error_rates);
        final_test_error_mean(i, j) = mean(test_error_rates);
        final_sd_train(i, j) = std(train_error_rates);
        final_sd_test(i, j) = std(test_error_rates);
        final_train_loss(i, j) = mean(train_losses);

    end
end

% Find the index of the minimum test error rate
[min_test_error, min_test_error_idx] = min(final_test_error_mean(:));
[optimal_node_idx, optimal_epoch_idx] = ind2sub(size(final_test_error_mean),
min_test_error_idx);

% Obtain the optimal node and epoch values
optimal_node = nodes(optimal_node_idx);
optimal_epoch = epochs(optimal_epoch_idx);

% Report the optimal test error rate and its associated node/epoch values
fprintf('Optimal Test Error Rate: %.4f\n', min_test_error);
fprintf('Optimal Node Value: %d\n', optimal_node);
fprintf('Optimal Epoch Value: %d\n', optimal_epoch);


% Plotting results
for i = 1:length(nodes)
    disp(nodes(i));
    disp("train");
    disp(final_train_error_mean(i, :));
    disp(final_sd_train(i, :));
    disp("test");
    disp(final_test_error_mean(i, :));
    disp(final_sd_test(i, :));
    disp("end");
    % Create a new figure with figure number set to i
    figure(i);

    % Plot the train and test error rates with error bars
    plot(epochs, final_train_error_mean(i, :), 'b-', 'LineWidth', 1, 'MarkerSize',
12);
    hold on;
    plot(epochs, final_test_error_mean(i, :), 'r-', 'LineWidth', 1, 'MarkerSize',
12);
```

```
    % Add labels and title
    xlabel('Epochs');
    ylabel('Classification Error Rate');
    title(['Nodes = ' num2str(nodes(i))]);

    % Add legend
    legend('Train', 'Test', 'Location', 'best');

    figure(i+3);

    % Plot the train and test error rates with error bars
    plot(epochs, final_sd_train(i, :), 'b-', 'LineWidth', 1, 'MarkerSize', 12);
    hold on;
    plot(epochs, final_sd_test(i, :), 'r-', 'LineWidth', 1, 'MarkerSize', 12);

    % Add labels and title
    xlabel('Epochs');
    ylabel('Classification Error Rate (std)');
    title(['Nodes = ' num2str(nodes(i))]);

    % Add legend
    legend('Train', 'Test', 'Location', 'best');
end



% Define colors for each node
colors = {'r', 'g', 'b'}; % Add more colors if needed
% Plotting final_train_error_mean with line graph for all nodes
figure(length(nodes)+6);
for i = 1:length(nodes)
    plot(epochs, final_train_error_mean(i, :), [colors{i} '-'], 'LineWidth', 1,
'MarkerSize', 12);
    hold on;
end
xlabel('Epochs');
ylabel('Classification Error Rate');
title('Train Classification Error Rate for All Nodes');
legend(cellstr(num2str(nodes')), 'Location', 'best');
% Plotting final_test_error_mean with line graph for all nodes
figure(length(nodes)+7);
for i = 1:length(nodes)
    plot(epochs, final_test_error_mean(i, :), [colors{i} '-'], 'LineWidth', 1,
'MarkerSize', 12);
    hold on;
end
xlabel('Epochs');
ylabel('Classification Error Rate');
title('Test Classification Error Rate for All Nodes');
legend(cellstr(num2str(nodes')), 'Location', 'best');

% Plotting final_sd_train with line graph for all nodes
figure(length(nodes)+8);
for i = 1:length(nodes)
    plot(epochs, final_sd_train(i, :), [colors{i} '-'], 'LineWidth', 1,
'MarkerSize', 12);
    hold on;
end
xlabel('Epochs');
ylabel('Standard Deviation');
```

```
title('Standard Deviation of Train Classification Error Rate for All Nodes');
legend(cellstr(num2str(nodes')), 'Location', 'best');

% Plotting final_sd_test with line graph for all nodes
figure(length(nodes)+9);
for i = 1:length(nodes)
    plot(epochs, final_sd_test(i, :), [colors{i} '-'], 'LineWidth', 1, 'MarkerSize',
12);
    hold on;
end
xlabel('Epochs');
ylabel('Standard Deviation');
title('Standard Deviation of Test Classification Error Rate for All Nodes');
legend(cellstr(num2str(nodes')), 'Location', 'best');
```

Code of Experiments 2 & 3 combined:

[a]

```
[x,t] = cancer_dataset;
x = x;
t = t;

numIterations = 30;

numBaseClassifiers = 15;
baseClassifiers = cell(1, numBaseClassifiers);



trainAccuracies = zeros(1, numBaseClassifiers);
testAccuracies = zeros(1, numBaseClassifiers);

for iter = 1:numIterations
    [X_train, y_train, X_test, y_test] = train_test_split(x, t);

    optimalEpoch = 16;
    optimalHiddenLayers = 32;

    for i = 1:numBaseClassifiers
        baseClassifier = build_model_function(X_train, y_train, optimalHiddenLayers,
optimalEpoch, i);
        [trainAccuracies(i), testAccuracies(i)] = calculate_accuracy(baseClassifier,
X_train, y_train, X_test, y_test);
        baseClassifiers{i} = baseClassifier;

        disp(['Base Classifier ', num2str(i), ' Accuracy:']);
        disp(['Train Accuracy: ', num2str(trainAccuracies(i)), '%']);
        disp(['Test Accuracy: ', num2str(testAccuracies(i)), '%']);
    end
    avgMeanTrainAccuracy = mean(trainAccuracies);
    avgMeanTestAccuracy = mean(testAccuracies);

    disp(['Iteration: ', num2str(iter)]);
    disp('Individual Model Accuracies:');
    disp(['Train Accuracies: ', num2str(trainAccuracies)]);
    disp(['Test Accuracies: ', num2str(testAccuracies)]);

    disp('Average Mean Train and Test Accuracies:');
    disp(['Average Mean Train Accuracy: ', num2str(avgMeanTrainAccuracy), '%']);
    disp(['Average Mean Test Accuracy: ', num2str(avgMeanTestAccuracy), '%']);
    disp(['Average Mean Train Accuracy: ', num2str(avgMeanTrainAccuracy), '%']);
    disp(['Average Mean Test Accuracy: ', num2str(avgMeanTestAccuracy), '%']);
end

% Bar graph for model accuracies
figure;
bar([trainAccuracies; testAccuracies]');
legend('Train Accuracy', 'Test Accuracy');
xlabel('Model Index');
ylabel('Accuracy (%)');
title('Model Accuracies for Individual Models');
xticks(1:numBaseClassifiers);
xticklabels({'Model 1', 'Model 2', 'Model 3', 'Model 4', 'Model 5', 'Model 6',
'Model 7', 'Model 8', 'Model 9', 'Model 10', 'Model 11', 'Model 12', 'Model 13',
'Model 14', 'Model 15'});
grid on;
```

[b]

```
[x, t] = cancer_dataset;
x = x;
t = t;

numIterations = 30;

numBaseClassifiers = randperm(23,10) + 2; % Generate 10 random numbers in the range
of 3 to 25
numBaseClassifiers = sort(numBaseClassifiers);

ensemblePrediction = cell(1, length(numBaseClassifiers));
ensembleTrainPrediction = cell(1, length(numBaseClassifiers));
ensembleTestAccuracy = zeros(1, length(numBaseClassifiers));

ensembleTrainAccuracy = zeros(1, length(numBaseClassifiers));

optimalEpoch = 16; % optimal epoch
optimalHiddenLayers = 32; % optimal hidden layers

for i = 1:length(numBaseClassifiers) % i.e 12
    baseClassifiers = cell(1, numBaseClassifiers(i));
    y_pred_ind = cell(1, numBaseClassifiers(i));
    y_pred_train_ind = cell(1, numBaseClassifiers(i));

    trainAccuracies_ind = zeros(1, numBaseClassifiers(i));
    testAccuracies_ind = zeros(1, numBaseClassifiers(i));

    for iter = 1:numIterations
        % Get Train Test split
        [X_train, y_train, X_test, y_test] = train_test_split(x, t);

        % Build Classifiers
        for j = 1:numBaseClassifiers(i)
            baseClassifier = build_model_function(X_train, y_train,
optimalHiddenLayers, optimalEpoch, j);
            baseClassifiers{j} = baseClassifier; % Store base classifier in cell
array
            [trainAccuracies_ind(j), testAccuracies_ind(j), y_pred_ind{j},
y_pred_train_ind{j}] = calculate_accuracy(baseClassifier, X_train, y_train, X_test,
y_test);
```

```
        end

        % Train Accuracy of the ensemble
        ensembleTrainAccuracy(i) = mean(trainAccuracies_ind);

        % Call Majority Voting Function
        [ensembleTrainPrediction{i}] = majority_vote(y_pred_train_ind, X_train);
        [ensemblePrediction{i}] = majority_vote(y_pred_ind, X_test);

        %Calculate accuracy of ensemble:
        [ensembleTrainAccuracy(i)] =
calculate_accuracy_ensemble(ensembleTrainPrediction{i}, y_train);
        [ensembleTestAccuracy(i)] =
calculate_accuracy_ensemble(ensemblePrediction{i}, y_test);
    end
end

% Overall Accuracy of ensemble:

TrainAccuracy = mean(ensembleTrainAccuracy);
TestAccuracy = mean(ensembleTestAccuracy);
disp(numBaseClassifiers);
disp("Train");
disp(ensembleTrainAccuracy);
disp("test");
disp(ensembleTestAccuracy);
disp(['Ensemble Train Accuracy: ', num2str(TrainAccuracy), '%']);
disp(['Ensemble Test Accuracy: ', num2str(TestAccuracy), '%']);

% Plot numBaseClassifiers count array vs. train and test accuracy
figure;
plot(numBaseClassifiers, ensembleTrainAccuracy, 'bo-', 'LineWidth', 2, 'MarkerSize',
8, 'DisplayName', 'Train Accuracy');
hold on;
plot(numBaseClassifiers, ensembleTestAccuracy, 'rx-', 'LineWidth', 2, 'MarkerSize',
8, 'DisplayName', 'Test  Accuracy');
xlabel('Number of Base Classifiers');

ylabel('Accuracy');
title('Ensemble Accuracy vs. Number of Base Classifiers');
legend('Train Accuracy', 'Test Accuracy');
grid on;
```

[c]

```
[x, t] = cancer_dataset;
x = x;
t = t;

optimalEpoch = 16; % optimal epoch
optimalHiddenLayers = 32; % optimal hidden layers
num_combinations = 10; % Number of random combinations to generate

% Initialize cell array to store the random combinations as pairs
random_combinations = cell(1, num_combinations);

% Generate random combinations of nodes and epochs
for i = 1:num_combinations
    % Generate random number for nodes
    random_nodes = randi([1, optimalHiddenLayers]);
    % Generate random number for epochs
    random_epochs = randi([1, optimalEpoch]);

    % Store the random combination as a pair in the cell array
    random_combinations{i} = [random_nodes, random_epochs];
end

disp(random_combinations);

numIterations = 30;
numBaseClassifiers = 15;

ensemblePrediction = cell(1, num_combinations);
ensembleTrainPrediction = cell(1, num_combinations);
ensembleTestAccuracy = zeros(1, num_combinations);
ensembleTrainAccuracy = zeros(1, num_combinations);

for i = 1:num_combinations
    baseClassifiers = cell(1, numBaseClassifiers);
    y_pred_ind = cell(1, numBaseClassifiers);
    y_pred_train_ind = cell(1, numBaseClassifiers);
    trainAccuracies_ind = zeros(1, numBaseClassifiers);
    testAccuracies_ind = zeros(1, numBaseClassifiers);

    nodes = random_combinations{i}(1,1);
    epoch = random_combinations{i}(1,2);

    for iter = 1:numIterations
        % Get Train Test split
        [X_train, y_train, X_test, y_test] = train_test_split(x, t);

        % Build Classifiers
        for j = 1:numBaseClassifiers
            baseClassifier = build_model_function(X_train, y_train, nodes, epoch,
j);
            baseClassifiers{j} = baseClassifier; % Store base classifier in cell
array
            [trainAccuracies_ind(j), testAccuracies_ind(j), y_pred_ind{j},
y_pred_train_ind{j}] = calculate_accuracy(baseClassifier, X_train, y_train, X_test,
y_test);

        end

        % Train Accuracy of the ensemble
        ensembleTrainAccuracy(i) = mean(trainAccuracies_ind);

        % Call Majority Voting Function
        [ensembleTrainPrediction{i}] = majority_vote(y_pred_train_ind, X_train);
        [ensemblePrediction{i}] = majority_vote(y_pred_ind, X_test);

        %Calculate accuracy of ensemble:
        [ensembleTrainAccuracy(i)] =
calculate_accuracy_ensemble(ensembleTrainPrediction{i}, y_train);
        [ensembleTestAccuracy(i)] =
calculate_accuracy_ensemble(ensemblePrediction{i}, y_test);
    end
end

disp(random_combinations);
disp(ensembleTestAccuracy);
disp(ensembleTrainAccuracy);

TrainAccuracy = mean(ensembleTrainAccuracy)
TestAccuracy = mean(ensembleTestAccuracy)

% Convert cell array to numeric array for x-axis values
x_values = cellfun(@(x) x(1), random_combinations);

% Plot x_values vs. train and test accuracy
figure;
plot(x_values, ensembleTrainAccuracy, 'bo-', 'LineWidth', 2, 'MarkerSize', 8,
'DisplayName', 'Train Accuracy');
hold on;
plot(x_values, ensembleTestAccuracy, 'rx-', 'LineWidth', 2, 'MarkerSize', 8,
'DisplayName', 'Test Accuracy');
xlabel('Number of Base Classifiers');
ylabel('Accuracy');
title('Ensemble Accuracy vs. Number of Base Classifiers');
legend('Train Accuracy', 'Test Accuracy');
grid on;
```

[d]

```
[x, t] = cancer_dataset;
x = x;
t = t;

optimalEpoch = 16; % optimal epoch
optimalHiddenLayers = 32; % optimal hidden layers
num_combinations = 10; % Number of random combinations to generate

% Initialize cell array to store the random combinations as pairs
random_combinations = cell(1, num_combinations);

% Generate random combinations of nodes and epochs
for i = 1:num_combinations
    % Generate random number for nodes
    random_nodes = randi([optimalHiddenLayers, 100]);

    % Generate random number for epochs
    random_epochs = randi([optimalEpoch, 100]);



    % Store the random combination as a pair in the cell array

    random_combinations{i} = [random_nodes, random_epochs];
end

disp(random_combinations);

numIterations = 30;
numBaseClassifiers = 15;

ensemblePrediction = cell(1, num_combinations);
ensembleTrainPrediction = cell(1, num_combinations);
ensembleTestAccuracy = zeros(1, num_combinations);
ensembleTrainAccuracy = zeros(1, num_combinations);
```

```
for i = 1:num_combinations % i.e 12
    baseClassifiers = cell(1, numBaseClassifiers);
    y_pred_ind = cell(1, numBaseClassifiers);
    y_pred_train_ind = cell(1, numBaseClassifiers);
    trainAccuracies_ind = zeros(1, numBaseClassifiers);
    testAccuracies_ind = zeros(1, numBaseClassifiers);

    nodes = random_combinations{i}(1,1);
    epoch = random_combinations{i}(1,2);

    for iter = 1:numIterations
        % Get Train Test split
        [X_train, y_train, X_test, y_test] = train_test_split(x, t);

        % Build Classifiers
        for j = 1:numBaseClassifiers
            baseClassifier = build_model_function(X_train, y_train, nodes, epoch, 
j);
            baseClassifiers{j} = baseClassifier; % Store base classifier in cell 
array
            [trainAccuracies_ind(j), testAccuracies_ind(j), y_pred_ind{j}, 
y_pred_train_ind{j}] = calculate_accuracy(baseClassifier, X_train, y_train, X_test, 
y_test);
        end

        % Train Accuracy of the ensemble
        ensembleTrainAccuracy(i) = mean(trainAccuracies_ind);

        % Call Majority Voting Function
        [ensembleTrainPrediction{i}] = majority_vote(y_pred_train_ind, X_train);
        [ensemblePrediction{i}] = majority_vote(y_pred_ind, X_test);

        %Calculate accuracy of ensemble:
        [ensembleTrainAccuracy(i)] = 
calculate_accuracy_ensemble(ensembleTrainPrediction{i}, y_train);
        [ensembleTestAccuracy(i)] = 
calculate_accuracy_ensemble(ensemblePrediction{i}, y_test);
    end
end

% Overall Accuracy of ensemble:
disp(random_combinations);
disp(ensembleTestAccuracy);
disp(ensembleTrainAccuracy);

TrainAccuracy = mean(ensembleTrainAccuracy)
TestAccuracy = mean(ensembleTestAccuracy)


% Convert cell array to numeric array for x-axis values
x_values = cellfun(@(x) x(1), random_combinations);

% Plot x_values vs. train and test accuracy
figure;
plot(x_values, ensembleTrainAccuracy, 'bo-', 'LineWidth', 2, 'MarkerSize', 8, 
'DisplayName', 'Train Accuracy');
hold on;
plot(x_values, ensembleTestAccuracy, 'rx-', 'LineWidth', 2, 'MarkerSize', 8, 
'DisplayName', 'Test Accuracy');
```

```
xlabel('Number of Base Classifiers');
ylabel('Accuracy');
title('Ensemble Accuracy vs. Number of Base Classifiers');
legend('Train Accuracy', 'Test Accuracy');
grid on;
```

[e]

```
[x, t] = cancer_dataset;
x = x;
t = t;

optimalEpoch = randi([1, 100], 1, 10); % optimal epoch
optimalEpoch = sort(optimalEpoch);
optimalHiddenLayers = 32; % optimal hidden layers

disp(optimalEpoch);
disp(length(optimalEpoch));

numIterations = 30;
numBaseClassifiers = 15;

ensemblePrediction = cell(1, length(optimalEpoch));
ensembleTrainPrediction = cell(1,length(optimalEpoch));
ensembleTestAccuracy = zeros(1, length(optimalEpoch));
ensembleTrainAccuracy = zeros(1, length(optimalEpoch));

for i = 1:length(optimalEpoch) % i.e 12
    baseClassifiers = cell(1, numBaseClassifiers);
    y_pred_ind = cell(1, numBaseClassifiers);
    y_pred_train_ind = cell(1, numBaseClassifiers);
    trainAccuracies_ind = zeros(1, numBaseClassifiers);
    testAccuracies_ind = zeros(1, numBaseClassifiers);

    nodes = optimalHiddenLayers;
    epoch = optimalEpoch(i);

    for iter = 1:numIterations
        % Get Train Test split
        [X_train, y_train, X_test, y_test] = train_test_split(x, t);

        % Build Classifiers
        for j = 1:numBaseClassifiers
            baseClassifier = build_model_function(X_train, y_train, nodes, epoch,
j);
            baseClassifiers{j} = baseClassifier; % Store base classifier in cell
array
            [trainAccuracies_ind(j), testAccuracies_ind(j), y_pred_ind{j},
y_pred_train_ind{j}] = calculate_accuracy(baseClassifier, X_train, y_train, X_test,
y_test);
        end



        % Train Accuracy of the ensemble
        ensembleTrainAccuracy(i) = mean(trainAccuracies_ind);

        % Call Majority Voting Function
        [ensembleTrainPrediction{i}] = majority_vote(y_pred_train_ind, X_train);
        [ensemblePrediction{i}] = majority_vote(y_pred_ind, X_test);
        %Calculate accuracy of ensemble:
        [ensembleTrainAccuracy(i)] =
calculate_accuracy_ensemble(ensembleTrainPrediction{i}, y_train);
        [ensembleTestAccuracy(i)] =
calculate_accuracy_ensemble(ensemblePrediction{i}, y_test);
    end

end

% Overall Accuracy of ensemble:
disp(optimalEpoch);
disp(ensembleTestAccuracy);
disp(ensembleTrainAccuracy);

TrainAccuracy = mean(ensembleTrainAccuracy)
TestAccuracy = mean(ensembleTestAccuracy)

% Plot x_values vs. train and test accuracy
figure;
plot(optimalEpoch, ensembleTrainAccuracy, 'bo-', 'LineWidth', 2, 'MarkerSize', 8,
'DisplayName', 'Train Accuracy');
hold on;
plot(optimalEpoch, ensembleTestAccuracy, 'rx-', 'LineWidth', 2, 'MarkerSize', 8,
'DisplayName', 'Test Accuracy');
xlabel('Number of Base Classifiers');
ylabel('Accuracy');
title('Ensemble Accuracy vs. Number of Base Classifiers');
legend('Train Accuracy', 'Test Accuracy');
grid on;
```

Code for Experiment 4:

```
%% STEP 01:

% Parameters for Class 1
mean1 = [0, 0];
variance1 = eye(2);

% Parameters for Class 2
mean2 = [2, 0];
variance2 = [2, 0; 0, 2];

% Generate random variables for Class 1 and Class 2
num_samples = 1650; % Number of samples for each class
split_idx = 0.1*2*num_samples;
start = split_idx + 1;
class1_samples = mvnrnd(mean1, variance1, num_samples);
class2_samples = mvnrnd(mean2, variance2, num_samples);

% Concatenate the samples from both classes
data = [class1_samples; class2_samples];

% Generate labels for the samples
labels = [repmat([1, 0], num_samples, 1); repmat([0, 1], num_samples, 1)];


% Randomly shuffle the data and labels
shuffled_indices = randperm(size(data, 1));
```

```
data = data(shuffled_indices, :);
labels = labels(shuffled_indices, :);

%% STEP 02:

train_data = data(1:split_idx, :);
train_labels = labels(1:split_idx, :);
test_data = data(start:end, :);
test_labels = labels(start:end, :);

% Set the number of hidden units and epochs for the neural network
[hidden_units, epochs, train_accuracy, test_accuracy] =
estimate_node_epoch(train_data', train_labels', test_data', test_labels');

fprintf('Train Accuracy: %.2f%%\n', train_accuracy);
fprintf('Test Accuracy: %.2f%%\n', test_accuracy);

%% STEP 03:

[ensemble_train_accuracy, ensemble_test_accuracy, y_pred_train, y_pred_test] =
get_ensemble_accuracy(train_data', train_labels', test_data', test_labels',
hidden_units, epochs);

fprintf('Ensemble Train Accuracy: %.2f%%\n', ensemble_train_accuracy);
fprintf('Ensemble Test Accuracy: %.2f%%\n', ensemble_test_accuracy);

%% STEP 04 PLOT DECISION BOUNDARY (modified code)
% Find the indices for the label with value 1
idx = find(y_pred_test(:, 1) > 0);
% Get the features corresponding to the label with value 1
label1 = test_data(idx, :);
% Create a scatter plot with predicted labels as colors
figure(1);
hold on;
gscatter(test_data(:, 1), test_data(:, 2), y_pred_test', 'rb', '.', 15);
hold on;
% Calculate the center of the circle
x = mean(label1(:, 1))
y = mean(label1(:, 2))
x_var = cov(label1(:, 1))
y_var = cov(label1(:, 2))

% Calculate the radius of the circle
r = sqrt(var(label1(:, 1))) * 2.5
% Generate a range of angles for the circle
th = 0:pi/50:2*pi;

% Calculate the x-coordinates and y-coordinates of the circle
x_circle = r*cos(th) + x;
y_circle = r*sin(th) + y;

% Plot the decision boundary
plot(x_circle, y_circle, 'y', 'LineWidth', 3);

% Add a legend
legend('Class 1', 'Class 2', 'Decision Boundary');
hold off;
```

```
%% STEP 05 PLOT DECISION BOUNDARY

%Plot the samples
figure(2);
hold on;
scatter(class1_samples(:, 1), class1_samples(:, 2), 'MarkerFaceColor', 'blue');
scatter(class2_samples(:, 1), class2_samples(:, 2), 'MarkerFaceColor', 'red');

% Calculate the mean and variance of each class
mu1 = mean(class1_samples)
sigma1 = cov(class1_samples)
mu2 = mean(class2_samples)
sigma2 = cov(class2_samples)

% Calculate the decision boundary as a circle
center = (mu2 - mu1) * 2/3 + mu1; % center of the circle
radius = 2.34; % radius of the circle
theta = linspace(0, 2*pi, 100);
x_circle = center(1) + radius*cos(theta);
y_circle = center(2) + radius*sin(theta);

% Plot the decision boundary
plot(x_circle, y_circle, 'y', 'LineWidth', 2);

% Add a legend
legend('Class 1', 'Class 2', 'Decision Boundary');
```

Auxillary Functions:

[a] train_test_split.m

```
function [X_train, y_train, X_test, y_test] = train_test_split(x, t)
p = randperm(size(x, 2));
numTrain = round(0.5 * size(x, 2));
numTest = size(x, 2) - numTrain;
X_train = x(:, p(1:numTrain));
y_train = t(:, p(1:numTrain));
X_test = x(:, p(numTest+1:end));
y_test = t(:, p(numTest+1:end));
```

[b] majority_vote.m

```
function ensemblePrediction = majority_vote(Prediction, X_test)
    ensemblePrediction = zeros(2,size(X_test,2));
    all_results = [0,1]; %possible outcomes
    i = 1;
    for col = 1:2:size(X_test,2)*2
        election_array = zeros(1, length(all_results));
        label_zero = zeros(1, length(Prediction));
        label_ones = zeros(1, length(Prediction));
        for row = 1:length(Prediction)
            temp = Prediction{row};
            label_zero(row) = temp(col);
            label_ones(row) = temp(col+1);
            if max(temp(col), temp(col+1))==temp(col)
                index = 1;
            else
```

```
                index = 2;
            end
            election_array(index) = election_array(index) + 1;
        end


        [~,I] = max(election_array);
        avg_label_zero = mean(label_zero);
        avg_label_one = mean(label_ones);
        ensemblePrediction(1, i) = avg_label_zero;
        ensemblePrediction(2, i) = avg_label_one;
        i = i+1;
    end
end
```

[c] get_ensemble_accuracy.m

```
function [ensemble_train_accuracy, ensemble_test_accuracy, ensembleTrainPrediction,
ensembleTestPrediction] = get_ensemble_accuracy(X_train,y_train,X_test,y_test,
optimalEpoch, optimalHiddenLayers)
numIterations = 30;

numBaseClassifiers = 15;

y_pred_ind = cell(1, numBaseClassifiers);
y_pred_train_ind = cell(1, numBaseClassifiers);

trainAccuracies_ind = zeros(1, numBaseClassifiers);
testAccuracies_ind = zeros(1, numBaseClassifiers);

for iter = 1:numIterations
    % Build Classifiers
    for j = 1:numBaseClassifiers
        baseClassifier = build_model_function(X_train, y_train, optimalHiddenLayers,
optimalEpoch, j);
        [trainAccuracies_ind(j), testAccuracies_ind(j), y_pred_ind{j},
y_pred_train_ind{j}] = calculate_accuracy(baseClassifier, X_train, y_train, X_test,
y_test);
    end
    % Call Majority Voting Function
    [ensembleTrainPrediction] = majority_vote(y_pred_train_ind, X_train);
    [ensembleTestPrediction] = majority_vote(y_pred_ind, X_test);

    %Calculate accuracy of ensemble:
    [ensemble_train_accuracy] = calculate_accuracy_ensemble(ensembleTrainPrediction,
y_train);
    [ensemble_test_accuracy] = calculate_accuracy_ensemble(ensembleTestPrediction,
y_test);
end
```

[d] estimate_node_epoch.m

```
function [optimal_node, optimal_epoch, train_accuracy, test_accuracy] =
estimate_node_epoch(X_train,y_train,X_test,y_test)
% Definition of nodes and epochs
% nodes = randi([1 100],1,10);
% epochs = randi([1 100],1,10);
% nodes = sort(nodes)
% epochs = sort(epochs)
```

```
nodes = 100;
epochs = 85;
% Set the number of runs
num_runs = 30;

% Initialize arrays to store results
final_train_error_mean = zeros(length(nodes), length(epochs));
final_test_error_mean = zeros(length(nodes), length(epochs));


final_sd_train = zeros(length(nodes), length(epochs));
final_sd_test = zeros(length(nodes), length(epochs));
final_train_loss = zeros(length(nodes), length(epochs));

% Loop over each node/epoch combination
for i = 1:length(nodes)
    for j = 1:length(epochs)
        train_error_rates = zeros(1, num_runs);
        test_error_rates = zeros(1, num_runs);
        train_losses = zeros(1, num_runs);

        % Loop over each run
        for k = 1:num_runs
            % Train the neural network on the training set
            net = patternnet(nodes(i), 'trainrp');
            net.trainParam.epochs = epochs(j);
            net.trainParam.lr = 0.01;
            net = train(net, X_train, y_train);

            % Perform prediction on the training and test sets
            y_train_pred = net(X_train);
            y_test_pred = net(X_test);

            % Calculate the classification error rate for train and test sets
            train_error_rates(k) = sum(round(y_train_pred) ~= y_train, 'all') /
numel(y_train); % Update here
            test_error_rates(k) = sum(round(y_test_pred) ~= y_test, 'all') /
numel(y_test); % Update here

            % Calculate the train loss
            train_losses(k) = perform(net, y_train, y_train_pred);
        end

        % Calculate average and standard deviation of train error rates, test error
rates, and train losses
        final_train_error_mean(i, j) = mean(train_error_rates);
        final_test_error_mean(i, j) = mean(test_error_rates);
        final_sd_train(i, j) = std(train_error_rates);
        final_sd_test(i, j) = std(test_error_rates);
        final_train_loss(i, j) = mean(train_losses);
    end
end

% Find the index of the minimum test error rate
[min_test_error, min_test_error_idx] = min(final_test_error_mean(:));
[optimal_node_idx, optimal_epoch_idx] = ind2sub(size(final_test_error_mean),
min_test_error_idx);

% Obtain the optimal node and epoch values
optimal_node = nodes(optimal_node_idx);
```

```
optimal_epoch = epochs(optimal_epoch_idx);

train_accuracy = 100 * (1-final_train_error_mean);
test_accuracy = 100 * (1-final_test_error_mean);
```

[e] encode_data.m

```
function [y_test_en] = encode_data(y_test)
y_test_en = zeros(1,size(y_test,2));
all_results = [0,1];
z = 1;
for col = 1:2:size(y_test,2)*2

    if max(y_test(col), y_test(col+1))==y_test(col)
        index = 1;
    else
        index = 2;
    end
    y_test_en(z) = all_results(index);
    z = z + 1;
end
```

[f] calculate_accuracy_ensemble.m

```
function [ensembleAccuracy] = calculate_accuracy_ensemble(ensemblePrediction,
y_test)
ensembleAccuracy = 100 * sum(~round(ensemblePrediction) ~= y_test, 'all') /
numel(y_test);
```

[g] calculate_accuracy.m

```
function [train_acc,test_acc, y_pred, y_train_pred] =
calculate_accuracy(mod,x_train,y_train,x_test,y_test)
[mod, tr]=train(mod,x_train,y_train);
y_train_pred = mod(x_train);
y_pred = mod(x_test);
train_acc = 100 * sum(~round(y_train_pred) ~= y_train, 'all') / numel(y_train);
test_acc = 100 * sum(~round(y_pred) ~= y_test, 'all') / numel(y_test);
```

[h] build_model_function.m

```
function baseClassifier = build_model_function(x_train,y_train,nodes,epochs, j)
baseClassifier = patternnet(nodes);
rng(j);
baseClassifier = init(baseClassifier);
baseClassifier.trainParam.epochs = epochs;
% baseClassifier.trainFcn = 'trainscg';
% baseClassifier.trainFcn = 'trainlm';
baseClassifier.trainFcn = 'trainrp';
baseClassifier=configure(baseClassifier,x_train,y_train);
```