Matlab simulation for Understanding how to solve pattern recognition problems using backpropagation

Sreelalan Subramanian (6720695)

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

The Matlab simulation assignment is focused on multi-layer perceptron and its workings, and how solving pattern recognition problems is done using back propagation in mlp. The most frequent approach for updating weights in a multi-layer perceptron is back propagation. The cancer dataset is used in this assignment. Hidden patterns in the dataset are studied, and how a mlp works on this dataset is understood, using the provided dataset. The assignment's first experiment looks at how the number of hidden nodes in a single hidden layer affects the mlp's learning capacity. The number of nodes is varied between 2,8 and 32. Similarly, the experiment focuses on how epochs affect the robustness of the learning of the models. The number of epochs that are specified for to experiment are 1,2,4,8,16,32,64. Overall, 21 combinations of epochs and hidden nodes are used to train the mlps. Each model is trained in 30 different train–test splits and the error rate is identified. The best hidden layer-epoch combination was found to be 32 hidden layers and 16 epochs among the 21 combinations. The best combination is found when the test error rate is lowest and then starts to rise owing to overfitting. In experiments 2 and 3, the found combination is employed as a basic classifier. Experiment 2 builds on the findings of experiment 1. In experiment 2, we look at how mlp performs in a group setting i.e. as an ensemble. Initialize an ensemble of basic classifiers with varying weights. They are considered as an ensemble and individual classifiers in isolation. The testing is carried out with a range of ensemble sizes ranging from three to twenty-five. The majority voting method is used to evaluate the ensemble models. The accuracy score is the metric utilised here. The experiment is repeated with different epochs and the same hidden layers. Ensembles with higher base classifiers did well in the entire experiment. Similarly, in an ensemble scenario, ensemble models with the same number of hidden layers and epochs perform better. Ensemble models also outperform individual classifiers in terms of total performance. Experiment 3 investigates how optimizers affect mlp learning. Experiment 3 is a rerun of experiment 2 using two alternative optimisers: When the same tests as in Experiment 2 are repeated, it is discovered that trainrp outperforms trainlm and trainscg. Trainlm outperforms trainrp in training data, with higher accuracy, but it underperforms trainrp in testing data, indicating that trainlm has overfitting problems. Through trainrp doesn't perform that well in train data like trainlm, it compensates in test data and minimises the gap between train and test accuracies. Experiment 4 investigates the topic of nonlinear decision boundaries. The data points are produced at random from overlapping bivariate classes. Training is done on 300 data points and testing is done in 3000 data points. Once the 3000 data points are predicted, we can see that the decision boundary is plotted at **[-0.1713, -0.0215]** which is nearer to the optimal decision boundary [-2/3,0], and the Euclidean distance is 0.46 between the optimal and plotted centre of decision boundary.

Introduction

The analysis and understanding of patterns and correlations among the data to extrapolate new information is called as Pattern recognition. Pattern recognition is one of the problems machine learning is trying to solve. Neural networks are excellent tools to solve pattern recognition problems. This can be proved by **Universal approximation theorem** that neural networks with a single hidden layer can learn any non linear mathematical function. These neural networks work on the principle of feed forward and back propagation. The data is fed to the neural nets in feed forward fashion. At the final layer, the loss is calculated with the output and the loss calculated is back propagated to update the weights. In the given exercise, we are tasked to understand how neural networks solve pattern recognition problem using back propagation in cancer dataset. The exercise has 4 parts. The first part involves creating an optimal neural network with 1 layer by experimenting with n number of epochs and m number of hidden nodes. The second part involves using the found optimal network to further experiment on ensemble techniques. The third involves repeating the second experiment with different optimisers. The final part involves distinguishing two equi-probable overlapping classes in two dimensions.

1. Experiment 1:

1.1 Overview

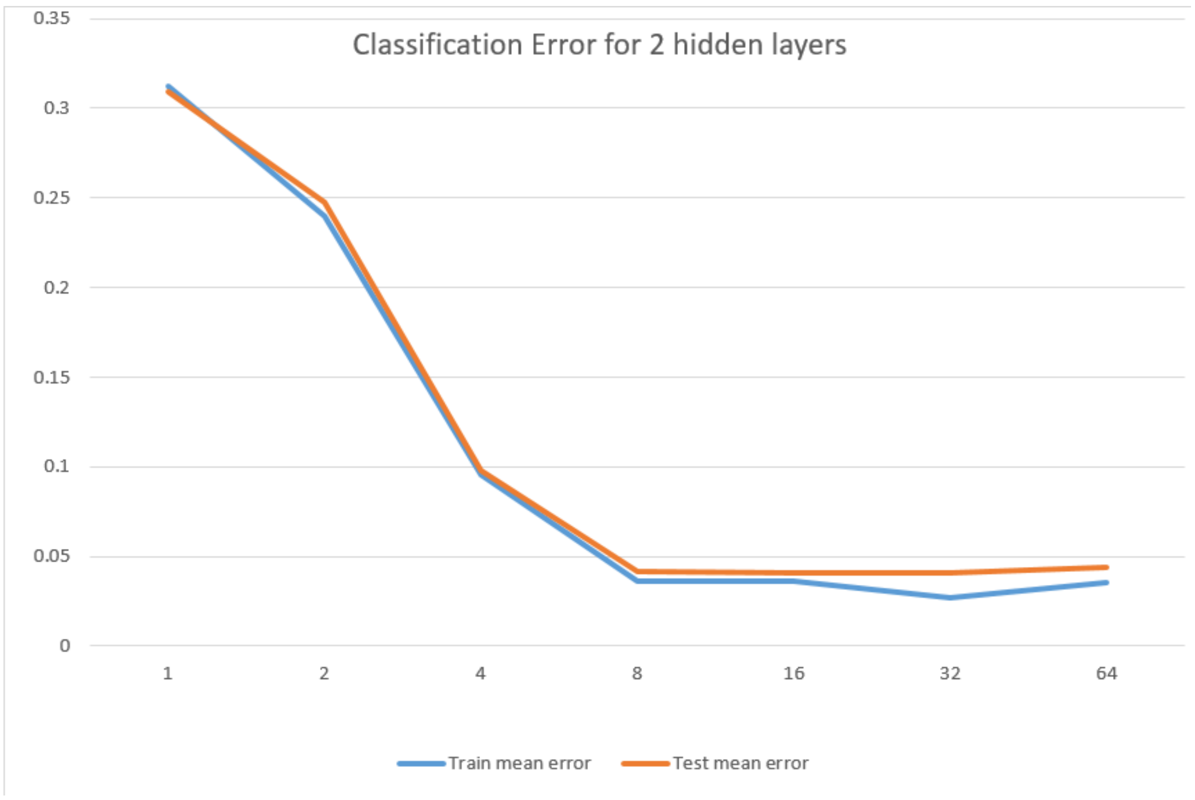
The first experiment involves training neural networks using cancer dataset available in the matlab repository. The neural network used for the experiment is a single hidden layered multi-layer perceptron. The data is split into 50/50 train – test data randomly. The model should be run for different epochs-hidden layer combinations. The above said network is built using the nntools library available in the matlab. The nntools is a customisable tool to create neural networks easily without any much of coding. Once the required parameters are selected, the script for the neural networks can be generated using the script generation function. The function generates the script directly into the matlab workspace. The rest of the optimization can be done in the matlab workspace.

1.2 The procedure

The experiment should be done for 3 different hidden layer neurons. The counts are 2,8,32. The epochs considered for the neural network are 1,2,4,8,16,32,64. The combination of the neuron count and epochs give 21 different combinations. Early stopping should be set to 0 so that the model could run for the specified amount of epochs. The model should be run on these combinations. And each combination should be run 30 times with different train-test split generating randomly. After running 30 times for a single combination, mean and standard deviation of classification error should be taken to evaluate the model. The same step should be repeated for all the combinations. Finally, we will get 21 values for mean of classification error and 21 values for standard deviation. After the values are generated. The mean values are plotted against the number of epochs.

1.3 Results and Analysis

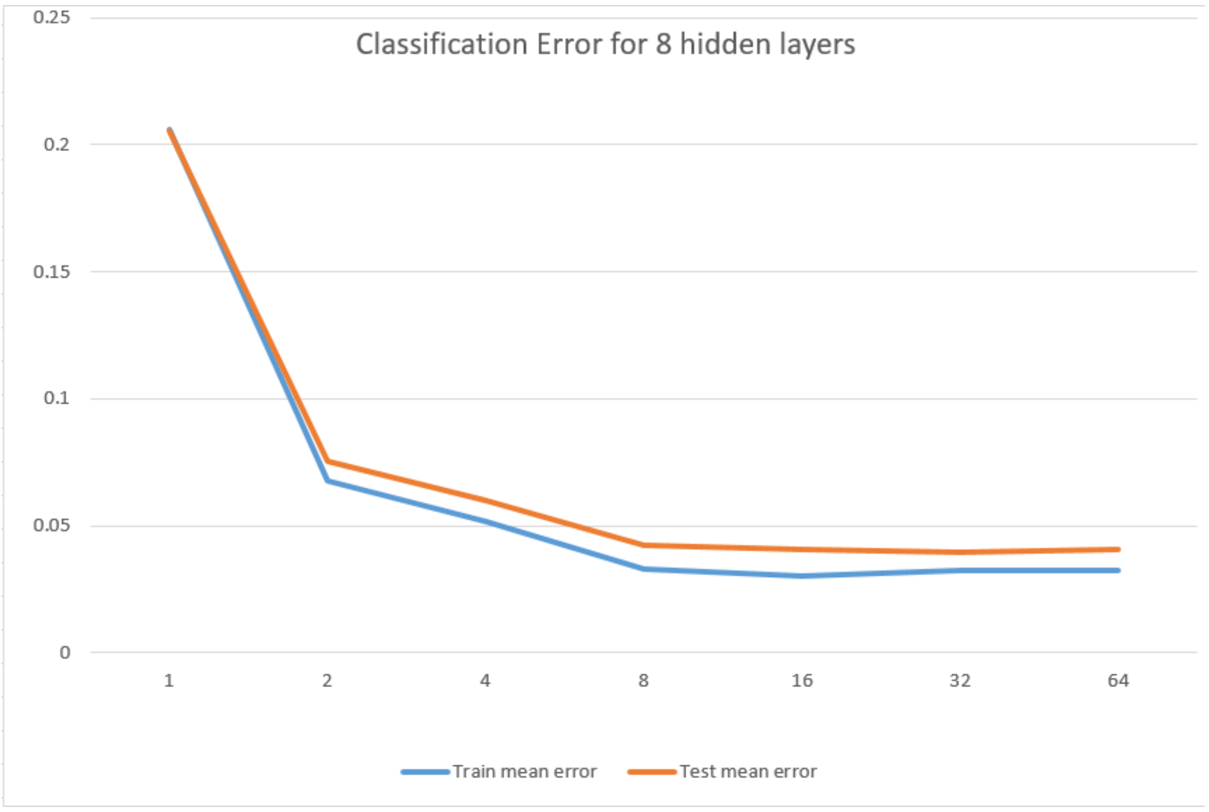
As per the results, with model with 2 hidden layers after the first epoch has a train classification error of 0.3122. After the 4th epoch the train error plateaus. The test error is constantly above the train error. After 16 epochs, the train error slightly dips showing signs of overfitting. But, overall, the



Note: The chart is plotted as number of epochs as x axis and error rate as y axis

The model could learn with a lowest train classification error of 0.0267 and test classification error of 0.407 at 16 epochs after which the test classification error slowly rises up.

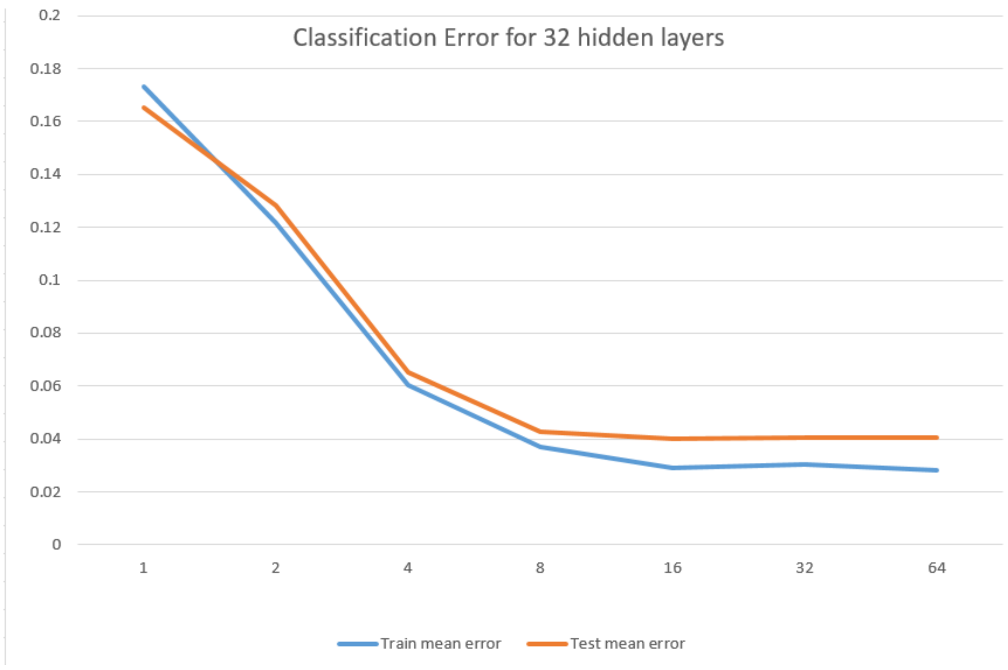
But after increasing number of hidden units to 8, the model has started performing even better. The model's initial classification error has reduced to 0.206. At the 2nd epoch, the training error shows a sharper dip to the error indicates that the model is learning quicker and better than the model with hidden layer of 2.



Note: The chart is plotted as number of epochs as x axis and error rate as y axis

As the epochs progress, the train and test classification errors have started diverging. At the 8th epoch, there is a clear difference between the train and test errors. As the training error keeps lowering, the test error is going up after the 32nd epoch showing the signs of overfitting. But overall, the model has learnt far better. The difference in testing error acrthe oss 2nd epoch and 8th epoch shows a clear difference between how the model has learned with the increased number of neurons.

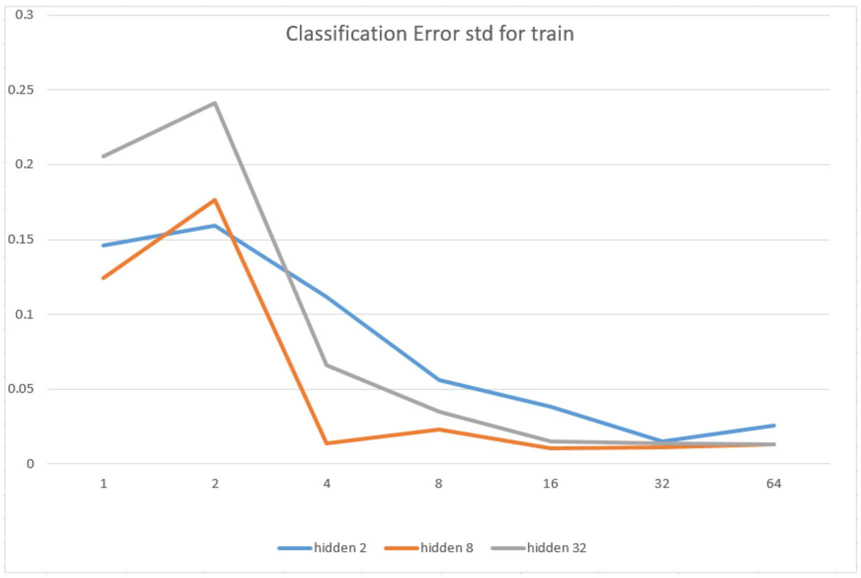
The model has learned even further with 32 nodes. The initial error rate is 0.1734. Though the



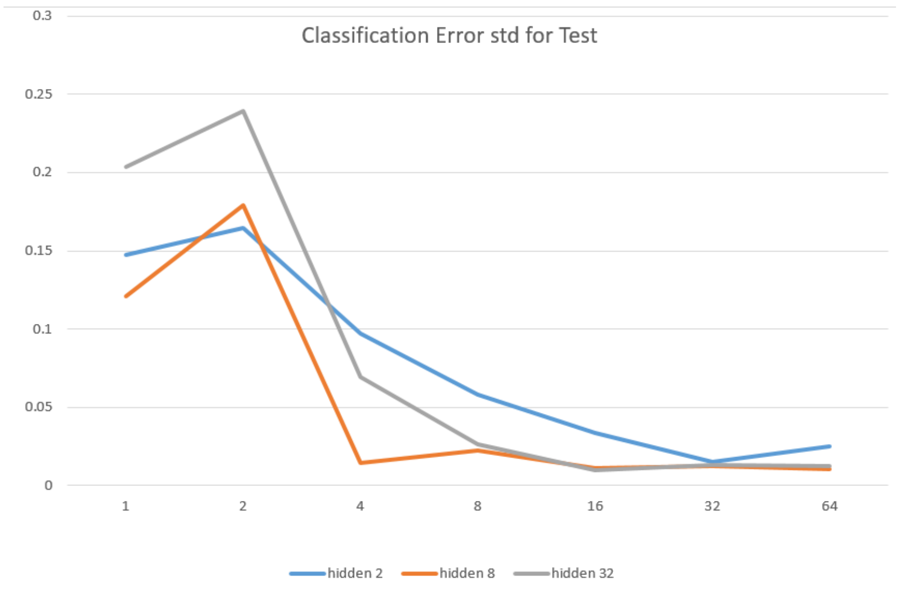
Note: The chart is plotted as number of epochs as x axis and error rate as y axis

The model learned quicker, the loss does not fall immediately but rather slowly, but the overall loss comes down to the global minimum of 0.028 across 21 iterations at 64th epoch. But the same can't be said with the test data, as the test error starts diverging after 16th epoch from the minimum of 0.0395. At the 64th epoch, the loss is 0.404 pointing to the overfitting of the model.

From the above experiments, though the training loss does go down, the same can't be said with the test data and they start diverging because of the overfitting of the model. Overfitting is a phenomenon in machine learning when, the model learns and performs the best from the training data but does not perform well in the test data. In overfitting, the model starts to learn the noise from the training data. Because of this, the model would show high variance. This can be seen in the standard deviation of the data points. They start to diverge after a point. By taking the mean of error from different combinations of the data sets, the possibility of extreme overfitting was minimised.



Note: The chart is plotted as number of epochs as x axis and sd of error rate as y axis



Note: The chart is plotted as the number of epochs as x axis and sd of error rate as y axis

The above graphs point to the standard deviation from 30 runs for each and every combination of the data. Both train and test data follow a similar pattern.

From the above experiments, the optimum hidden layer combination is found to be 32-16. This is because the training error is around 0.29 and the test error is around 0.399. Using the above-identified combination, experiment 2 is carried on.

2. Experiment 2:

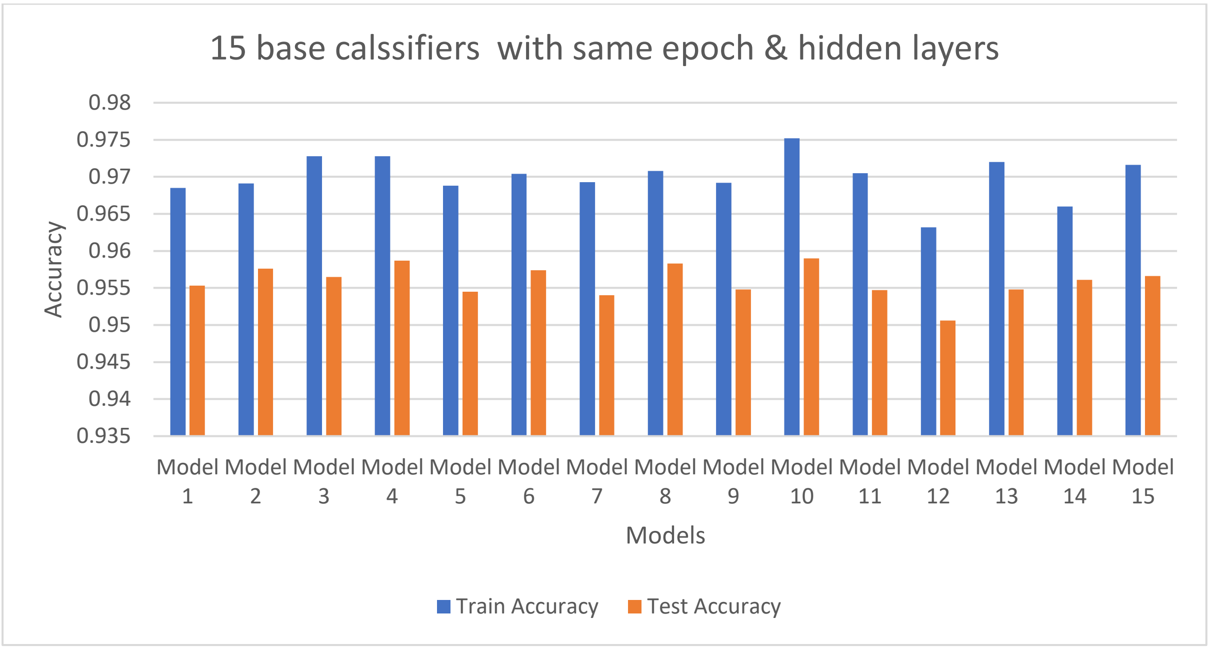
2.1 Overview

From the optimum epoch and hidden layer parameters identified from experiment 1, the base classifier is defined using those parameters. Then, an ensemble of 15 models with the same base classifier and different random weights are initialised. They are treated as both individual classifiers and then as ensembles. The ensemble is a technique used in machine learning where multiple individual models are aggregated and the output of the aggregated model is taken to predict the output. Usually voting is done where the predicted class is taken from what the majority of the classifiers have predicted. Later, the number of base classifiers in the ensemble is varied to see the variations in the accuracies. The experiment is repeated with different ensemble of different epoch/node count with variable complexities to see how the accuracies change.

2.2 The procedure

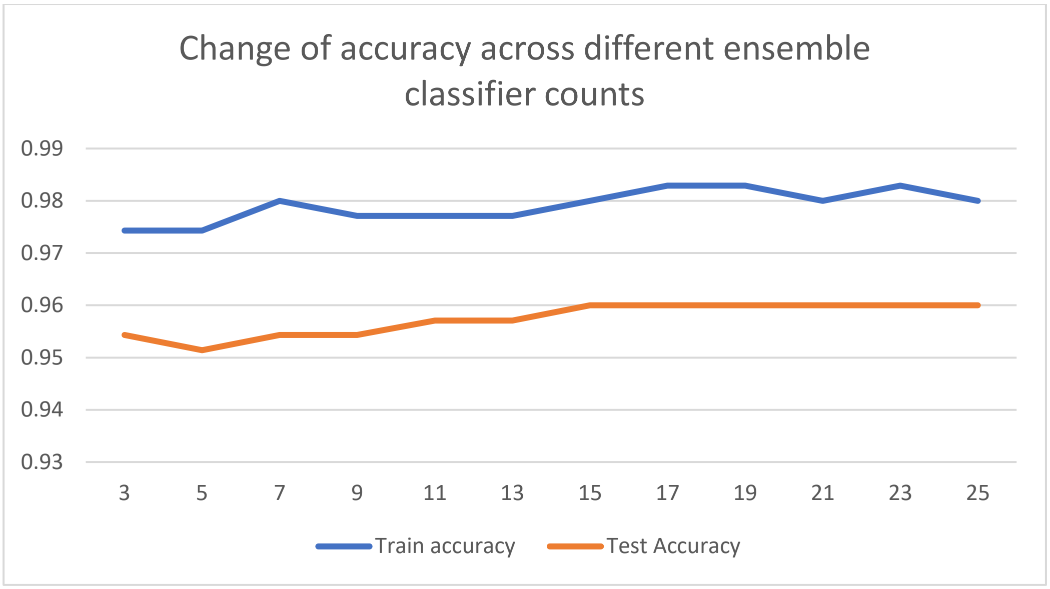
The ensemble models are trained separately and their training accuracy is identified. Then using the ensemble technique, the cumulative accuracy of the ensemble models is identified. Then, different count of base classifiers from 3 to 25 are taken and their individual ensemble accuracies are calculated. The ensemble models are further trained using random hidden layers and epochs lower than optimum. Similarly, a different set of ensemble models are trained with random hidden layers and epochs greater than optimum. Finally, keeping the number of the hidden nodes constant, random epochs are chosen and the model is trained further. All these trainings are carried out initially for individual classifiers, and then later, they are combined as an ensemble and the results are analysed. 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

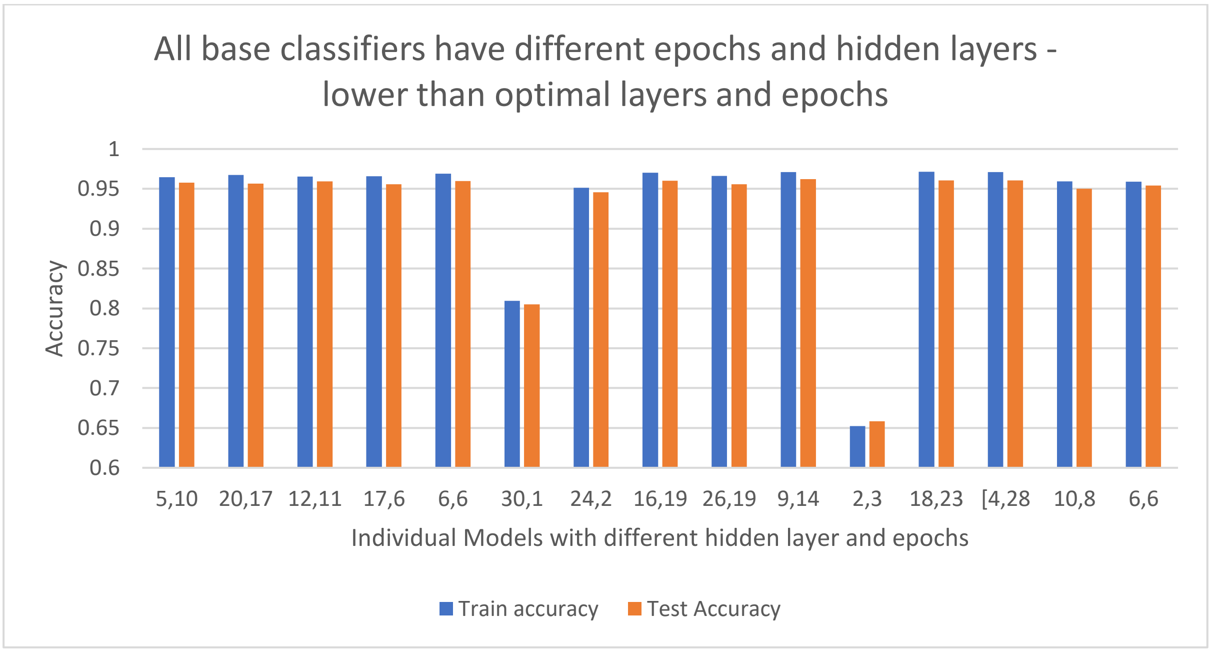


The model accuracies for individual models are tabulated using a bar graph. All models with the exception of model 3 show similar train accuracy scores. The performance of these models is solely dependent on the weights that are initialised. The standard deviation of train loss is 0.002 and the standard deviation of test loss is 0.002, pointing to how close the models have performed. The average mean train and test accuracies across the models are 0.97% and 0.95% respectively. This proves that, in spite of the different weights assigned, the models have learned the same way. The randomization of weights hasn't caused any substantial difference in the model's learning capacity. This point can be further proved by changing the number of nodes and epochs below and above the optimum that was identified in the 1st experiment.

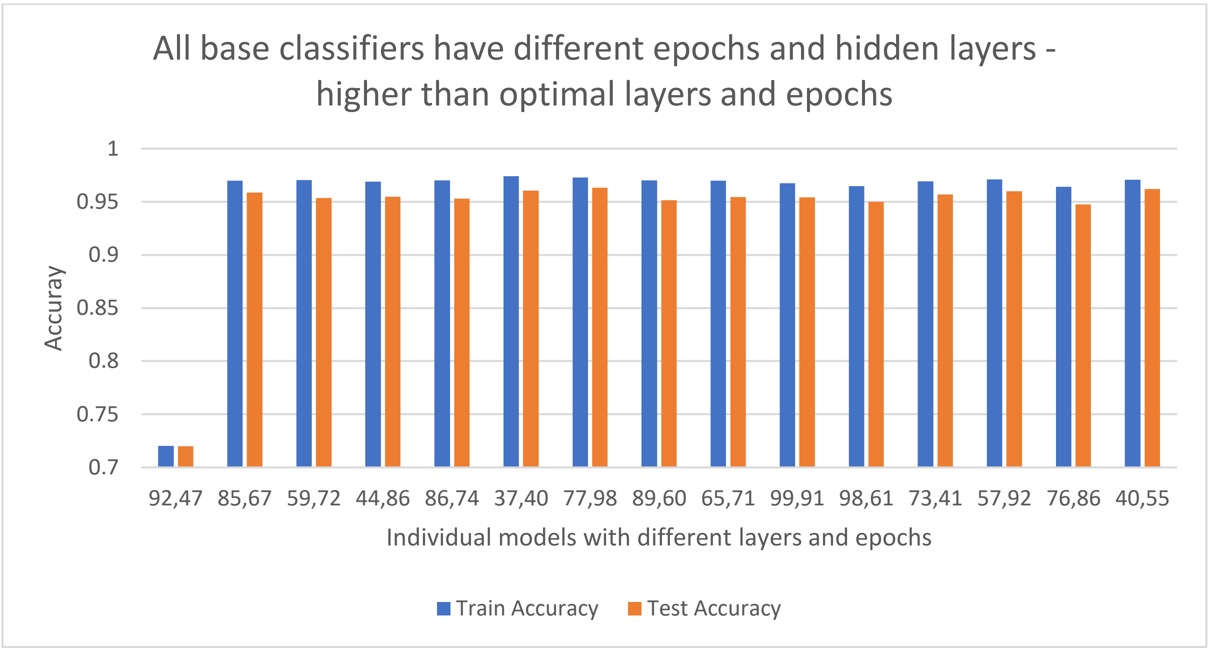
By varying the number of counts of base classifiers in the ensemble, we get different levels of accuracy. The number of base classifiers is varied from 3 to 25. This can be further understood from the graphs which were generated.



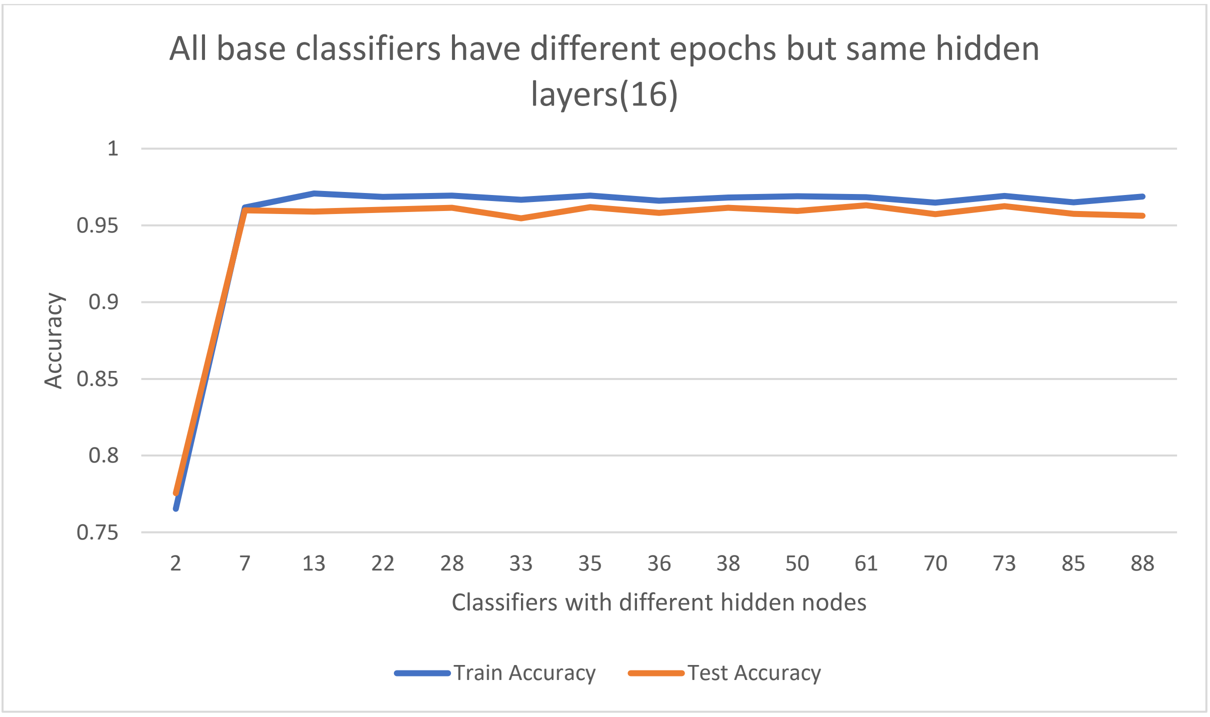
The above line graph shows how the accuracies vary as the number of base classifiers in the ensemble varies. With a lower number of ensemble classifiers, the accuracy is low which is closer to the base classifiers. As the number of classifiers increases, we can see a steady rise in the accuracies in both test and train data. Though train accuracies keep increasing after 15 classifier counts, the test accuracies after 15 don't show any improvements showing overfitting characteristics.



The third graph shows how each base classifier is modified using different hidden units and epochs combinations. The graph shows poor learning when both hidden layer and epochs are at the lowest (2,3). The data shows a higher degree of variation across the models.

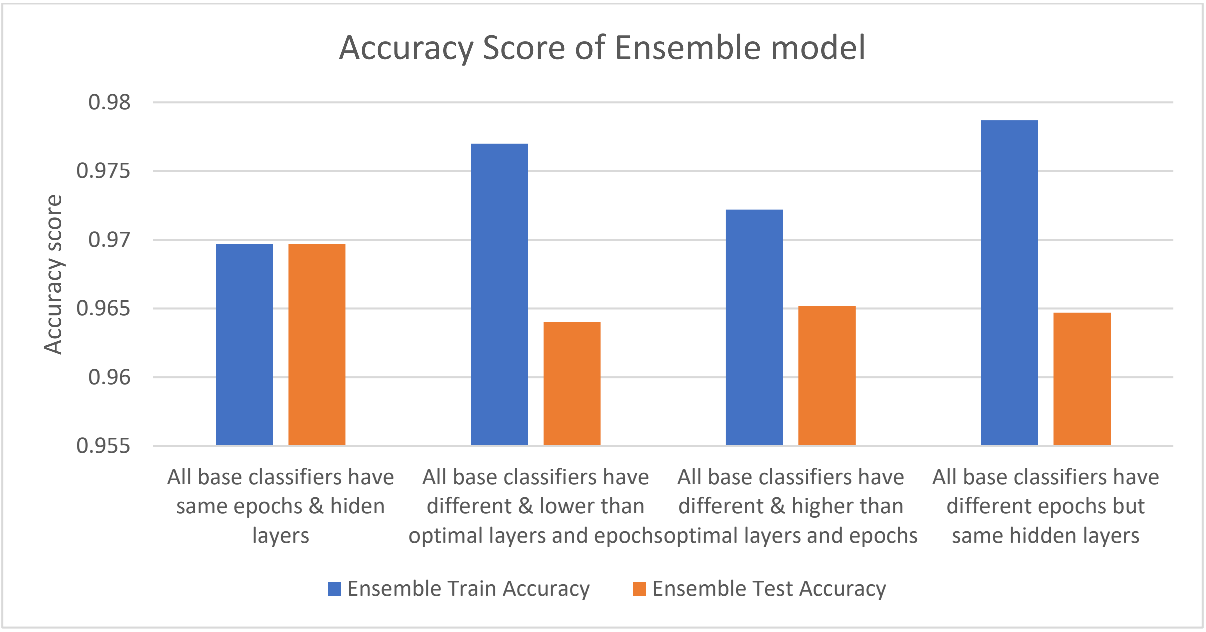


By increasing the number of hidden layer-epochs combination, the performance of the models has improved, but not everywhere. The model with 92 hidden neurons and 47 epochs have performed worse. From the above two graphs, a pattern emerges where the models with similar number of hidden neuron-epochs combinations have performed really well. The model with 77 hidden neurons and 98 epochs combination, where the combinations has performed even better than the optimal which was found in the first experiment. The rest of the models have underperformed, pointing to the overfitting of the data.



After changing both hidden neurons and epochs in previous runs, now, the optimal hidden layer count of 16 is fixed and the epoch counts are taken over the range of 1 to 100. Looking at the graph, the model struggled to learn with 2 epochs. The high error rate or lower accuracies points to how poor the model was after the second epoch. But after running the model for different epochs in increasing order, it is evident that model has learnt well. But after the optimal epoch that was identified, the model has started overfitting as it has started learning the noises. Each model was trained with each individual epoch number, so the variation of train loss across is the epochs is because of the variability of the random weights.

Finally, the ensemble of the above models is taken and they are used to calculate the accuracies of train and test data.



The ensemble with the base classifiers has the best test classification error with 0.0330. The train error is 0.0258. The ensemble classification error is far better than the optimum which was found in the experiment 1. Other ensemble classifiers has performed better than the individual classifiers in their respective categories. Ensemble technique proves that the method can be used to boost the accuracy and reduce the classification error.

3. Experiment 3

3.1 Overview

Experiment 3 follows the same methodology of experiment 2, but here, the optimizers are changed. The optimiser that was used previously was ‘trainscg’. ‘trainscg’ optimiser is changed to ‘trainlm’ and ‘trainrp’ and the training is repeated.

3.2 Procedure

The weights of neural networks are updated using an algorithm called as back propagation. But, in order for back propagation to update the weights, it should select the optimum weight. Picking inappropriate weights won’t let the model learn. The weights are selected in a way to minimise the loss. This step is done by the optimisers. Optimisers are the algorithms that are used to minimise the loss by changing the parameters of the neural networks like learning rate [1]. There are various optimisers. Some are

- Gradient Descent

- Stochastic Gradient Descent
- Adam
- Levenberg–Marquardt
- RProp
- RMS Prop
- Conjugate gradient

Gradient Descent is one of the important optimisers algorithm. The algorithm is powerful and is considered as a backbone for machine learning and deep learning. It is a first order The intuition behind gradient decent can be understood using a good analogy of getting lost in the hills. Consider a person is stuck in a hill and the person wants to get down. The person would use the local features like direction and steep and decide on which direction to go [2]. Similarly, gradient descent uses local features to identify on which direction it should go. The main aim of the gradient descent is to reach global minima of the loss function. Based on several parameters like current weights, learning rate and loss function. It

The gradient descent decides how the weights should be updated using the equation

$$\theta(n+1) = \theta(n) - \eta \cdot \nabla J(\theta) \text{ [3]}$$

where $\theta(n+1)$ is the new weight, $\theta(n)$ is the current weight, η is the learning rate and $J(\theta)$ is the loss function. Based on where the current weight is, the algorithm would decide whether the next step has to be larger or smaller. The – sign mentioned in the equation represents how much the loss function is subtracted from the current weights so that the algorithm could go against the gradient, that is, towards the local minima. If the gradients are steeper, the steps are larger, if the gradients are flatter, the algorithm takes smaller steps. That is taken care by the learning rate. Current learning rate is 0.01.

The optimiser that was used till now is Conjugate gradient. The optimizer is used in cases Cholesky decomposition, energy minimisation problems [4]. Unlike gradient descent, SCG algorithm does not perform line search. The algorithm is versatile and can be used in both linear and non linear functions.

The functions that are used in this experiment are

- **Levenberg–Marquardt algorithm - 'trainlm'**
- **RProp - 'trainrp'**

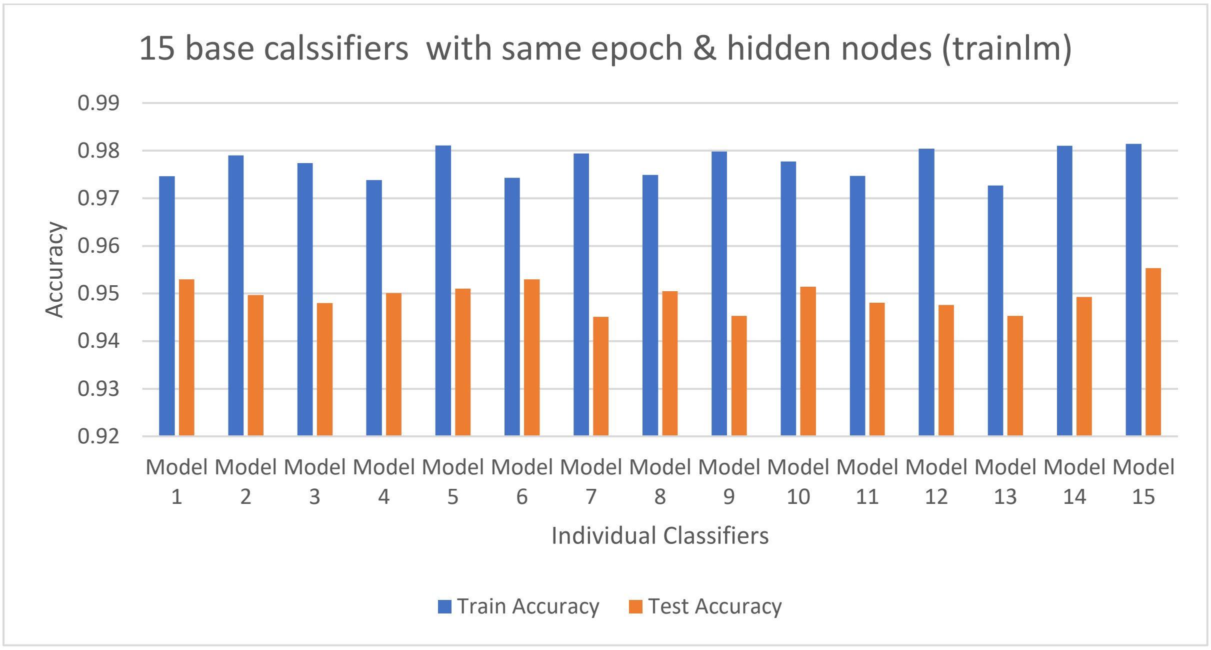
Levenberg–Marquardt algorithm is usually used to solve least square problems where the systems are non linear. The algorithm is an iterative algorithm that can find global minima if there are less number of local minima [5].

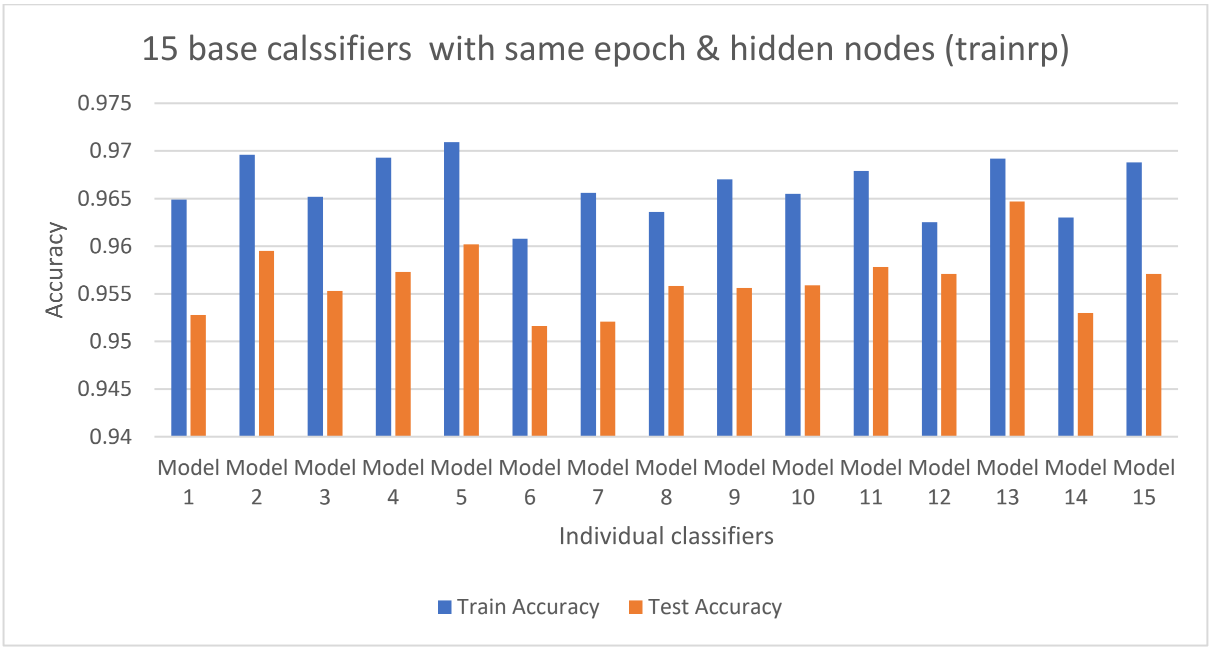
RProp algorithm is a popular algorithm which uses sign and magnitude of gradients to update the gradients. It can adapt well to each step and it handles weights individually [6].

By using these algorithms, the learning pattern of the base classifiers changes which can be analysed to see how the optimisers have performed.

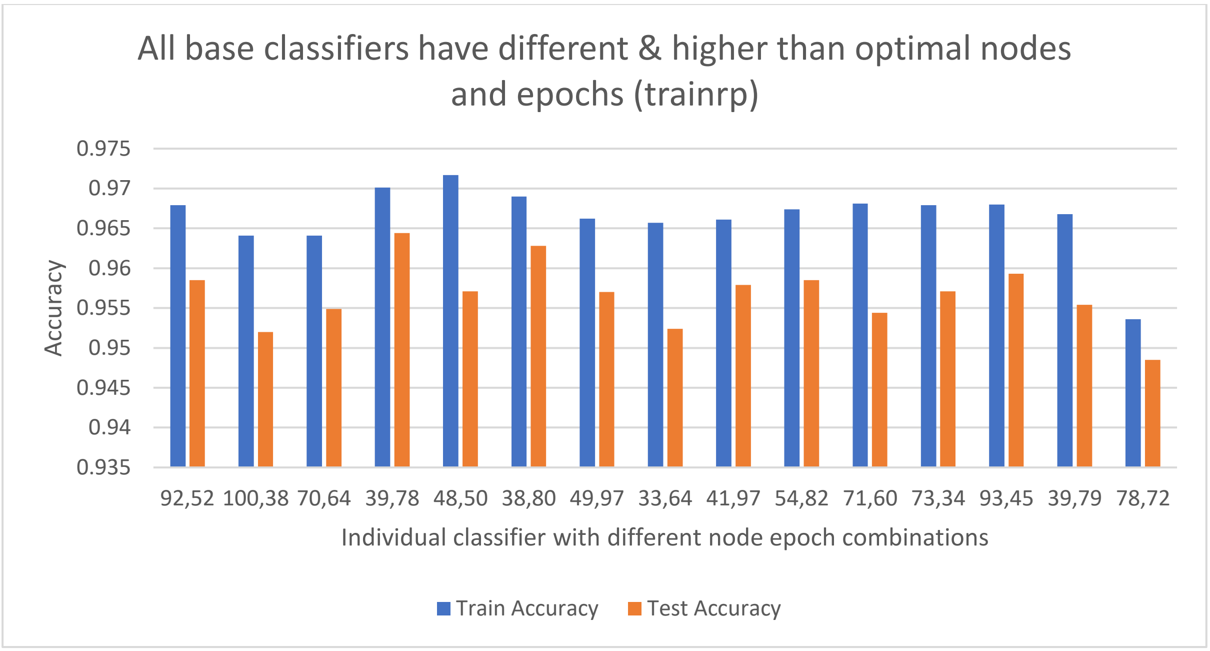
3.3 Results and Analysis

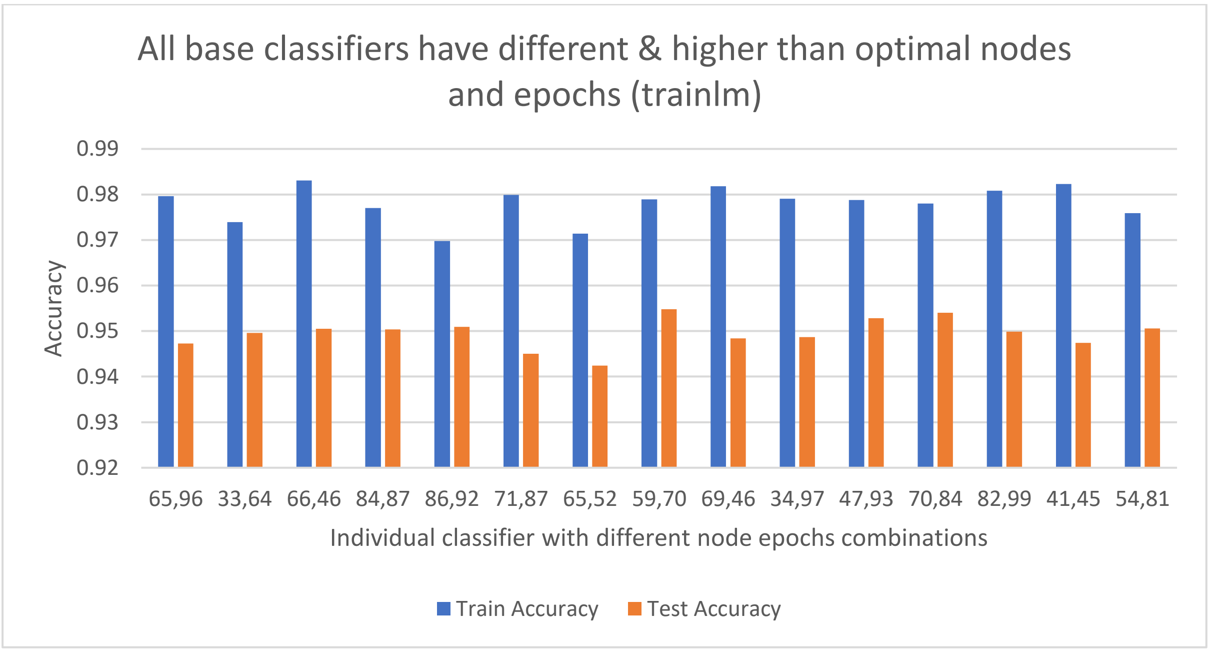
Initially the ensemble of models is trained with trainlm. Based on the train and test error, we could see that the mean train error being lower than what we got for trainscg. But the test error is higher to what we got for train scg. The similar pattern is followed across all the ensemble models with different parameters. All base classifiers which have different epochs and hidden layers and have lower than optimal layers and epochs has performed better than all the base classifiers with trainscg optimiser.





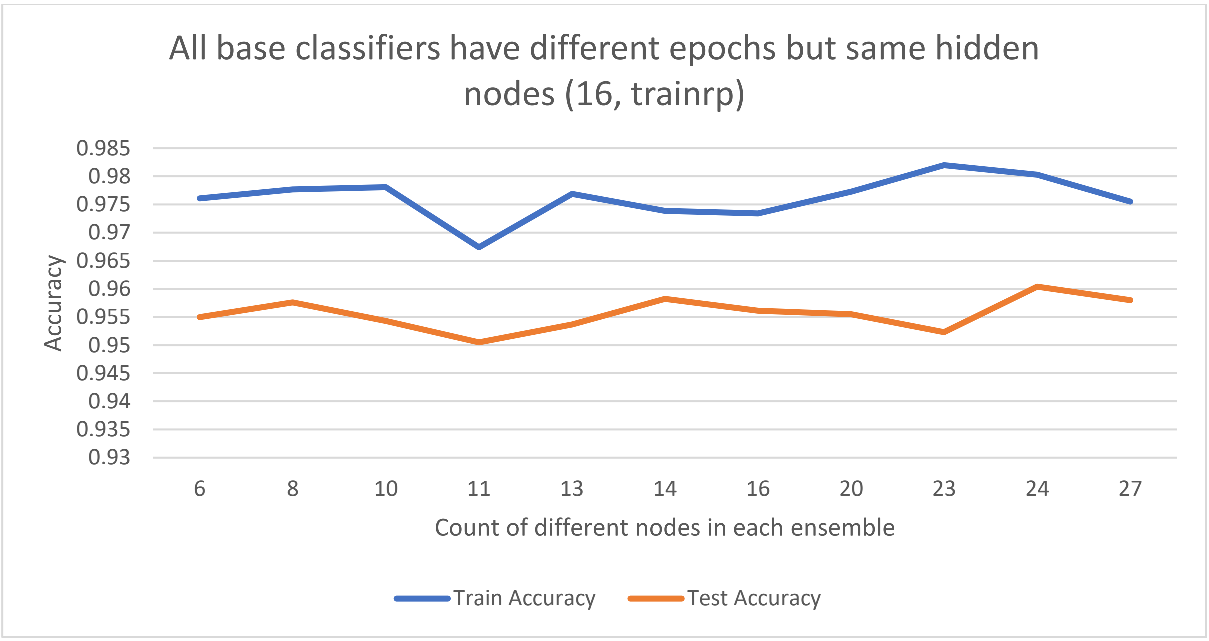
Comparing the graphs with 15 classifiers with same epoch and hidden layers among trainlm and trainrp, we can notice a difference. Trainlm has performed better in terms of train accuracy but trainrp has performed better in terms of test accuracy.

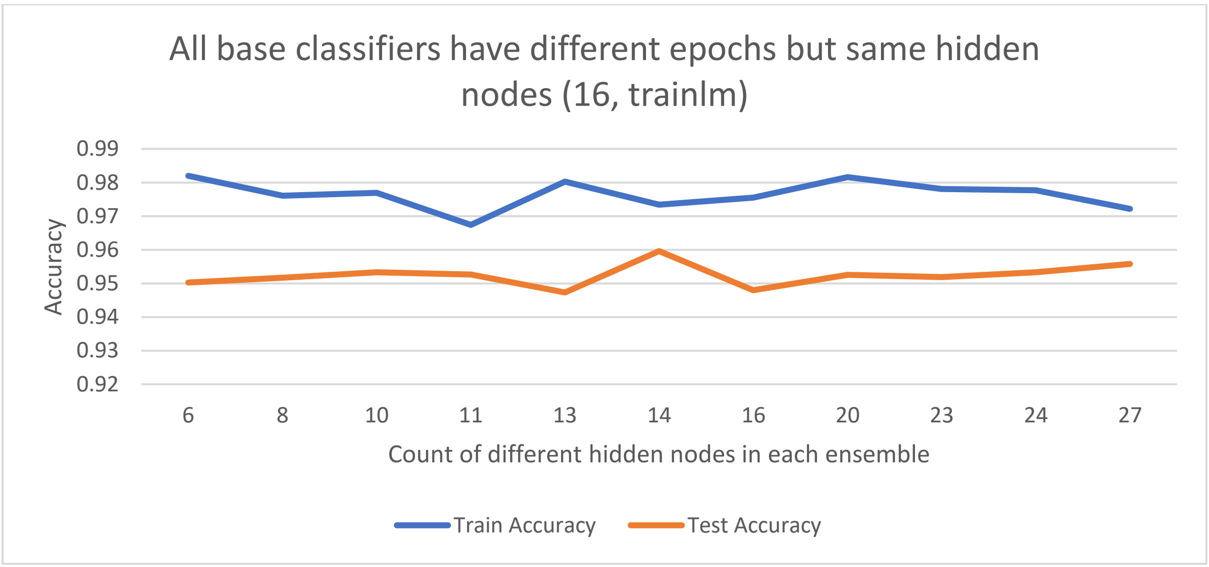




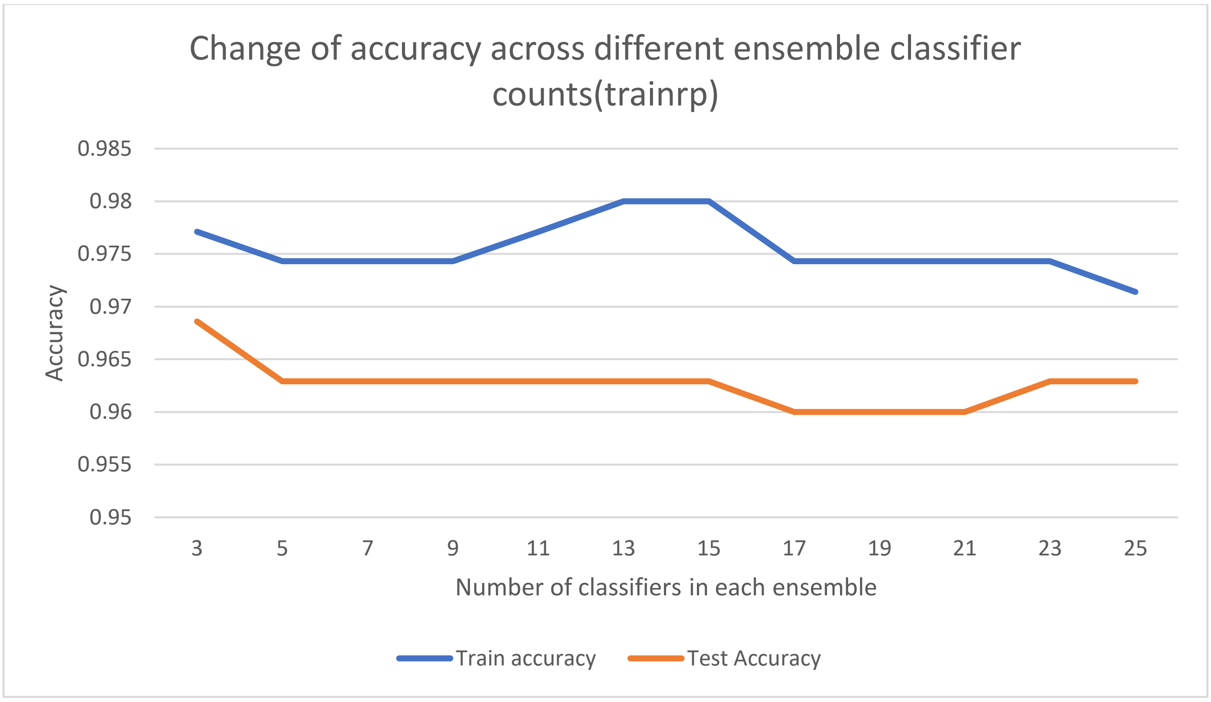
In the above graphs, when the number of layers – epoch count are taken in random, we see a pattern continuing where the trinlm is performing better overall than trainrp in train and vice versa in test hinting to a slight overfitting.

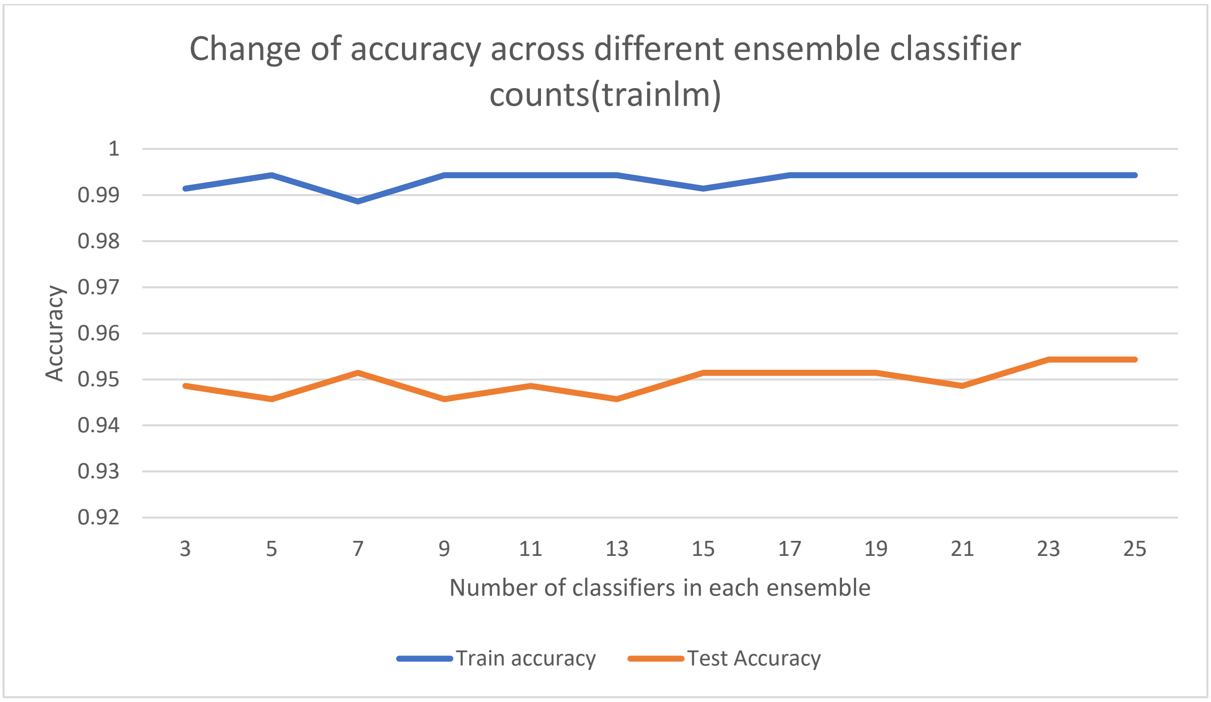
The below graphs show the performance of models by keeping number of hidden nodes constant, we can see that the trainlp optimiser constantly performs better than trainlm, which seems to be the trend throughout the experiments. There are few points where the trainrp has outperformed trainlm but those points are isolated points. The gap between train and test data in model trained using trainrp is smaller than the model trained using trainlm. This shows that the model trained using trinlm has more overfitting issues.





In the graph below where we have taken different odd number of counts for different classifiers, we can see the same pattern. Overall the best performances comes where the number of ensemble models are higher. Both in trinlm and trainrp we can see a performance jump in test accuracy around 21-25 range. Regarding the learning, the same pattern follows, where trainrp shows sign of overfitting while trainlm has train and test accuracies lie closer to each other.





From the above experiment, we can conclude that trainrp optimiser does a better job than trainlm and trainscg in minimising error and maximising accuracy in test data without much overfitting. This can be considered a proofs that the trainrp can adapt based on the training data and help converging better than other optimisers for this given dataset.

4. Experiment 4

4.1 Overview

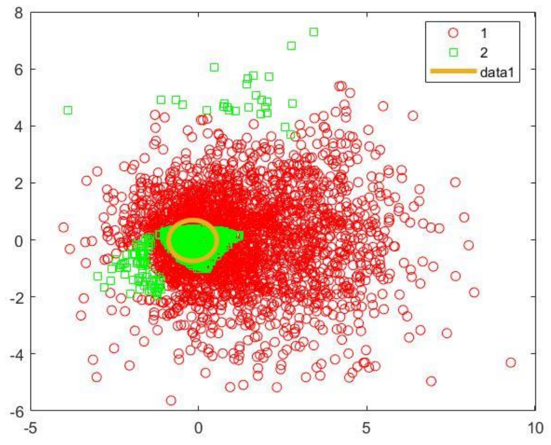
The requirement of experiment is to differentiatetwo equi-probable classes of two-dimensional Gaussians' which are overlapping in nature. Their parameters include mean $\mu 1$ of [0,0] with variance $\sigma 1$ as class 1. For class 2, it has a mean $\mu 2$ of [2,0] with variance $\sigma 2$ of 4. It can be proved that the Bayes (optimum) boundary is a circle whose centre is located at [− 2/3,0]and radius 2.34. Based on the problem statement, random data of each class is generated and neural networks are used to train the model.

4.2 Procedure

In order to generate random variables of two different classes with two different mean and variance, mvnrnd function is used. Each class has 1650 2 dimensional array so that while concatenating we will get a dataset set of 3300 records. After concatenating, we need to generate 2 different labels. They are concatenated too, and they are one hot encoded. Finally, the data is rearranged randomly to have normally distributed. The train/test split is done as 300/3000 respectively. Here 10% of the data is considered for training and 90% of the data is considered for testing. Using the patternnet, we can train for various epochs and hidden units combinations. It has been identified that 60 hidden units with 38 epochs give the optimum solution. After identifying hidden layer epoch pair, we can use the pair to generate ensemble of base classifier like we did in experiment 2.

4.3 Results & Analysis

After the predicting the data, we are calculating the mean and variance of any of the predicted class, here the mean and variance of class 2 is calculated. The calculated mean for the class 2 **is [-0.1713, -0.0215]**. The variance is calculated as **0.7162**. Using the calculated mean and variance, a circle is plotted on the scatter plot of the predicted values. The below graphs point to the predicted classes. Red represents the class 1 and green represents the class 2. The small circle represents the decision boundary calculated using the mean and variance. The optimal decision boundary is **[-2/3,0]** The Euclidean distance between the optimal origin and obtained origin is **0.4671.** From the Euclidean distance, it is clearly evident that the values within the decision boundaries are of similar class proving the theory that the decision boundary exists between [-2/3,0].



Note: The graph shows different classes and the decision boundary calculated

Conclusion

After completing 4 experiments, and with the data that we have analysed, the inference we have made based on the observations are

1. Lower hidden units- epoch pair takes time to learn
2. Higher hidden units- epoch pair can learn well
3. Training a model beyond a point causes model to over fit.
4. Same base classifier with different weights initialised will end up converge after certain effects showing that the initial weights doesn't contribute much to the performance of the models
5. Ensemble techniques can lower the classification error in both testing and training data
6. Using appropriate optimiser for the dataset can improve the learning of the model
7. We have proved that the decision boundary of 2 equi-probable classes with mean [0,0] and [2,0] with variances 1 and 4 respectively has a decision boundary closer to [-2/3,0] with a Euclidean distance of **0.4671**

These inferences show how neural networks can adapt to the given problem statement and has the capacity to learn non-linear boundaries. These experiments show that the universal approximation theorem holds good.

Citations

1. https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/#:~:text=An%20optimizer%20is%20a%20function,loss%20and%20improve%20the%20accuracy.
2. https://en.wikipedia.org/wiki/Gradient_descent
3. https://ruder.io/optimizing-gradient-descent/index.html#gradientdescentvariants
4. https://en.wikipedia.org/wiki/Conjugate_gradient_method
5. https://en.wikipedia.org/wiki/Levenberg%E2%80%93Marquardt_algorithm
6. https://florian.github.io/rprop/#:~:text=RProp%20is%20a%20popular%20gradient,dynamically%20for%20each%20weight%20independently.

Appendix

Code of Experiment 1:

```
% Script generated by Neural Pattern Recognition app
% Created 10-Mar-2022 01:02:40
%
% This script assumes these variables are defined:
%
%   x - input data.
%   t - target data.
[x,t]=cancer_dataset;
x = x;
t = t;
```

```
trainFcn = 'trainscg';  % Scaled conjugate gradient backpropagation.

% Create a Pattern Recognition Network
for hidden= [2,8,32]
    final_train_eror_mean=[];
    final_test_eror_mean=[];
    final_sd_train=[];
    final_sd_test=[];
    final_train_loss=[];
    hiddenLayerSize = hidden;
    net2 = patternnet(hiddenLayerSize, trainFcn);

    for epoches=  [ 1, 2, 4, 8, 16, 32,64]
        % Setup Division of Data for Training, Validation, Testing
        %net.divideParam.trainRatio = 50/100;
        %net.divideParam.valRatio = 0/100;
        %net.divideParam.testRatio = 50/100;
        ls_train=[];
        ls_test=[];
        train_loss=[];
        for iters=1:30
            net=net2;
            net.trainParam.epochs=epoches;
            net.trainParam.lr = 0.01;
            random=randperm(length(x));
            xtrain=x(1:9,random(1:round(length(x)*0.5)));
            ytrain=t(1:2,random(1:round(length(x)*0.5)));
            xtest=x(1:9,random(round(length(x)*0.5:end)));
            ytest=t(1:2,random(round(length(x)*0.5:end)));

            % Train the Network
            [net,tr] = train(net,xtrain,ytrain);

            % Test the Network
            y = net(xtest);
            y1=net(xtrain);
            e = gsubtract(ytest,y);
            e1 = gsubtract(ytrain,y1);
            performance = perform(net,ytest,y);
            performance1 = perform(net,ytrain,y1);
            tind = vec2ind(ytest);
            yind = vec2ind(y);
            tind1 = vec2ind(ytrain);
            yind1 = vec2ind(y1);
            percentErrors = sum(tind ~= yind)/numel(tind);
            percentErrors1 = sum(tind1 ~= yind1)/numel(tind1);
            %accuracy = sum(y == ytest)/numel(ytest)
            %accuracy1 = sum(y1 == ytrain)/numel(ytrain)
            ls_train=[ls_train,(percentErrors1)];
            ls_test=[ls_test,(percentErrors)];
            train_loss=[train_loss,performance1];
            %l=loss(net,xtrain,y1)
            % View the Network
            %view(net)
        end
    final_train_eror_mean=[final_train_eror_mean,mean(ls_train)];
    final_test_eror_mean=[final_test_eror_mean,mean(ls_test)];
    final_sd_train=[final_sd_train,std(ls_train)]
    final_sd_test=[final_sd_test,std(ls_test)]
    final_train_loss=[final_train_loss,mean(train_loss)];
    %length(ls_train)
    clear ls_train;
    clear ls_test;
    clear train_loss;
    end
subplot(3,1,1),
plot([1,2,4,8,16,32,64],final_train_eror_mean)
title('Train error for hidden layer: ', hidden)
xlabel("No of epoches")
ylabel("classification error")
subplot(3,1,2),
plot([1,2,4,8,16,32,64],final_test_eror_mean)
title('Test error for hidden layer: ', hidden)
xlabel("No of epoches")
ylabel("classification error")
subplot(3,1,3),
plot([1,2,4,8,16,32,64],final_train_loss)
title('Train loss: ', hidden)
xlabel("No of epoches")
ylabel("loss")
final_train_eror_mean;
final_test_eror_mean;
clear final_train_eror_mean;
clear final_test_eror_mean;
end
view(net)
```

Code of Experiment 2a:

```
[x,t]=cancer_dataset;
for a=1:25
    train_error_mod{a}=[];
    test_error_mod{a}=[];

    mod{a}=model_function(x,t,32,16);
end
train_error_ensemble=[];
test_error_ensemble=[];
train_error_ensemble_mean=[];
test_error_ensemble_mean=[];

for iter=1:30
    random=randperm(length(x));
    xtrain=x(1:9,random(1:round(length(x)*0.5)));
    ytrain=t(1:2,random(1:round(length(x)*0.5)));
    xtest=x(1:9,random(round(length(x)*0.5:end)));
    ytest=t(1:2,random(round(length(x)*0.5):end));

    for a=1:25

        [train_error_mod_temp,test_error_mod_temp,mod1{a}] = calculate_performance(mod{a},xtrain,ytrain,xtest,ytest);
        train_error_mod{a}=[train_error_mod{a},train_error_mod_temp];
        test_error_mod{a}=[test_error_mod{a},test_error_mod_temp];
    end
    [train_error,test_error]=majority_voting(mod1,xtrain,xtest,ytrain,ytest);
    train_error_ensemble=[train_error_ensemble,train_error];
    test_error_ensemble=[test_error_ensemble,test_error];
end
final_train_mean=[];
final_test_mean=[];
for a=1:25
    final_train_mean=[final_train_mean,mean(train_error_mod{a})];
    final_test_mean=[final_test_mean,mean(test_error_mod{a})];
end
cum_train_error_var=[];
cum_test_error_var=[];
for bla=3:25
    temp_var=rem(bla,2);
    if temp_var ~= 0
        disp(bla)
        [train_error_var,test_error_var]=majority_voting(mod1(1:bla),xtrain,xtest,ytrain,ytest);
        cum_train_error_var=[cum_train_error_var,1-train_error_var];
        cum_test_error_var=[cum_test_error_var,1-test_error_var];
    end
end
```

Code of experiment 2b

```
[x,t]=cancer_dataset;
for a=1:15
    hl_ep=[];
    train_error_mod{a}=[];
    test_error_mod{a}=[];
    %hidden=randi([33 100],1,1);
    ep=randi([1 100],1,1)
    %[hidden,ep]
    mod{a}=model_function(x,t,16,ep);
end
train_error_ensemble=[];
test_error_ensemble=[];
```

```
train_error_ensemble_mean=[];
test_error_ensemble_mean=[];

for iter=1:30
    random=randperm(length(x));
    xtrain=x(1:9,random(1:round(length(x)*0.5)));
    ytrain=t(1:2,random(1:round(length(x)*0.5)));
    xtest=x(1:9,random(round(length(x)*0.5:end)));
    ytest=t(1:2,random(round(length(x)*0.5):end));

    for a=1:15

        [train_error_mod_temp,test_error_mod_temp,mod1{a}] = calculate_performance(mod{a},xtrain,ytrain,xtest,ytest);
        train_error_mod{a}=[train_error_mod{a},train_error_mod_temp];
        test_error_mod{a}=[test_error_mod{a},test_error_mod_temp];
    end
    [train_error,test_error]=majority_voting(mod1,xtrain,xtest,ytrain,ytest);
    train_error_ensemble=[train_error_ensemble,train_error];
    test_error_ensemble=[test_error_ensemble,test_error];
end
final_train_mean=[];
final_test_mean=[];
for a=1:15
    final_train_mean=[final_train_mean,mean(train_error_mod{a})];
    final_test_mean=[final_test_mean,mean(test_error_mod{a})];
end
```

code of experiment 3a

```
[x,t]=cancer_dataset;
for a=1:15
    hl_ep=[];
    train_error_mod{a}=[];
    test_error_mod{a}=[];
    hidden=randi([33 100],1,1);
    ep=randi([33 100],1,1);
    [hidden,ep]
    mod{a}=model_function_with_diff_optim(x,t,hidden,ep);
end
train_error_ensemble=[];
test_error_ensemble=[];
train_error_ensemble_mean=[];
test_error_ensemble_mean=[];

for iter=1:30
    random=randperm(length(x));
    xtrain=x(1:9,random(1:round(length(x)*0.5)));
    ytrain=t(1:2,random(1:round(length(x)*0.5)));
    xtest=x(1:9,random(round(length(x)*0.5:end)));
    ytest=t(1:2,random(round(length(x)*0.5):end));

    for a=1:15

        [train_error_mod_temp,test_error_mod_temp,mod1{a}] = calculate_performance(mod{a},xtrain,ytrain,xtest,ytest);
        train_error_mod{a}=[train_error_mod{a},train_error_mod_temp];
        test_error_mod{a}=[test_error_mod{a},test_error_mod_temp];
    end
    [train_error,test_error]=majority_voting(mod1,xtrain,xtest,ytrain,ytest);
    train_error_ensemble=[train_error_ensemble,train_error];
    test_error_ensemble=[test_error_ensemble,test_error];
end
final_train_mean=[];
final_test_mean=[];
for a=1:15
    final_train_mean=[final_train_mean,mean(train_error_mod{a})];
    final_test_mean=[final_test_mean,mean(test_error_mod{a})];
end
```

Code of experiment 3b

```
[x,t]=cancer_dataset;
for a=1:25
    train_error_mod{a}=[];
    test_error_mod{a}=[];
    mod{a}=model_function_with_diff_optim(x,t,32,16);
end
train_error_ensemble=[];
test_error_ensemble=[];
train_error_ensemble_mean=[];
test_error_ensemble_mean=[];

for iter=1:30
    random=randperm(length(x));
    xtrain=x(1:9,random(1:round(length(x)*0.5)));
    ytrain=t(1:2,random(1:round(length(x)*0.5)));
    xtest=x(1:9,random(round(length(x)*0.5:end)));
    ytest=t(1:2,random(round(length(x)*0.5):end));

    for a=1:25

        [train_error_mod_temp,test_error_mod_temp,mod1{a}] = calculate_performance(mod{a},xtrain,ytrain,xtest,ytest);
        train_error_mod{a}=[train_error_mod{a},train_error_mod_temp];
        test_error_mod{a}=[test_error_mod{a},test_error_mod_temp];
    end
    [train_error,test_error]=majority_voting(mod1,xtrain,xtest,ytrain,ytest);
    train_error_ensemble=[train_error_ensemble,train_error];
    test_error_ensemble=[test_error_ensemble,test_error];
end
final_train_mean=[];
final_test_mean=[];
for a=1:25
    final_train_mean=[final_train_mean,mean(train_error_mod{a})];
    final_test_mean=[final_test_mean,mean(test_error_mod{a})];
end
cum_train_error_var=zeros(30,12);
cum_test_error_var=zeros(30,12);
for iter=1:30
    random=randperm(length(x));
    xtrain=x(1:9,random(1:round(length(x)*0.5)));
    ytrain=t(1:2,random(1:round(length(x)*0.5)));
    xtest=x(1:9,random(round(length(x)*0.5:end)));
    ytest=t(1:2,random(round(length(x)*0.5):end));
    count=1;
    for bla=3:25

        temp_var=rem(bla,2);
        if temp_var ~= 0
            [train_error_var,test_error_var]=majority_voting(mod1(1:bla),xtrain,xtest,ytrain,ytest);
            cum_train_error_var(iter,count)=1-train_error_var;
            cum_test_error_var(iter,count)=1-test_error_var;
            count=count+1;
        end
    end
end
```

Code of experiment 4

```
[x1,y1] = mvnrnd([0,0],[1,1],1650);
y1=1*ones(length(x1),1);
[x2,y2] = mvnrnd([2,0],[2,2],1650);
y2=2*ones(length(x2),1);
x=vertcat(x2,x1);
y=vertcat(y2,y1);
labels = categorical(y);
y = onehotencode(labels,2);
[row_idx,cols]=size(x);
rowidx = randperm(row_idx);
x=transpose(x(rowidx, : ));
y=transpose(y(rowidx,:));
final_train_eror_mean=[];
final_test_eror_mean=[];
final_sd_train=[];
final_sd_test=[];
final_train_loss=[];



for a=1:15
    train_error_mod{a}=[];
    test_error_mod{a}=[];
    hidden=randi([1 50],1,1)
    ep=randi([1 50],1,1)
    mod{a}=model_function(x,y,hidden,ep);
end
train_error_ensemble=[];
test_error_ensemble=[];
train_error_ensemble_mean=[];
test_error_ensemble_mean=[];

for iter=1:30
    random=randperm(length(x));
    xtrain=x(1:2,random(1:round(length(x)*0.0909)));
    ytrain=y(1:2,random(1:round(length(x)*0.0909)));
    xtest=x(1:2,random(301:end));
    ytest=y(1:2,random(301:end));

    for a=1:15

        [train_error_mod_temp,test_error_mod_temp,mod1{a}] = calculate_performance(mod{a},xtrain,ytrain,xtest,ytest);
        train_error_mod{a}=[train_error_mod{a},train_error_mod_temp];
        test_error_mod{a}=[test_error_mod{a},test_error_mod_temp];
    end
    [train_error,test_error]=majority_voting(mod1,xtrain,xtest,ytrain,ytest);
    train_error_ensemble=[train_error_ensemble,train_error];
    test_error_ensemble=[test_error_ensemble,test_error];
end
final_train_mean=[];
final_test_mean=[];
for a=1:15
    final_train_mean=[final_train_mean,mean(train_error_mod{a})];
    final_test_mean=[final_test_mean,mean(test_error_mod{a})];
end

ensemble_y_test=majority_voting_testing(mod,xtest);
indeces_for_label=find(ensemble_y_test==1);
label1=xtest(1:2,indeces_for_label);
gscatter(xtest(1:1,:),xtest(2:2,:),ensemble_y_test,'rgb','osd')
hold on
r = var(label1(1:1,:));
x =  mean(label1(1:1,:));
y = mean(label1(2:2,:));
th = 0:pi/50:2*pi;
xunit = r * cos(th) + x;
yunit = r * sin(th) + y;
plot(xunit, yunit,'LineWidth',3)
hold off
```

Auxiliary functions

1. Majority voting

```
function [train_error,test_error]=majority_voting(models,xtrain,xtest,ytrain,ytest)
model_train_outs=[];
model_test_outs=[];
    for a=1:length(models)
        model=models{a};
        ypred_train=model(xtrain);
        ypred_test=model(xtest);
        yind_train = vec2ind(ypred_train);
        yind_test = vec2ind(ypred_test);
        model_train_outs=[model_train_outs,reshape(yind_train,length(yind_train),1)];
        model_test_outs=[model_test_outs,reshape(yind_test,length(yind_test),1)];
    end
    train_error=transpose(model_train_outs);
    test_error=transpose(model_test_outs);
    ensemble_y_train=mode(train_error);
    ensemble_y_test=mode(test_error);
    tind_train=vec2ind(ytrain);
    tind_test=vec2ind(ytest);
    train_error = sum(tind_train ~= ensemble_y_train)/numel(tind_train);
    test_error = sum(tind_test ~= ensemble_y_test)/numel(tind_test);
    %train_error=vec2ind(ypred_train);
    %test_error=vec2ind(ypred_test);
```

```
24.
```

2. Calculate performance

```
function [percentErrors_train,percentErrors_test,mod] = calculate_performance(mod,xtrain,ytrain,xtest,ytest)
    mod=train(mod,xtrain,ytrain);
    ypred_train = mod(xtrain);
    ypred_test = mod(xtest);
    tind_train = vec2ind(ytrain);
    yind_train = vec2ind(ypred_train);
    percentErrors_train = sum(tind_train ~= yind_train)/numel(tind_train);
    tind_test = vec2ind(ytest);
    yind_test = vec2ind(ypred_test);
    percentErrors_test = sum(tind_test ~= yind_test)/numel(tind_test);
```

3. Model function

```
function mod = model_function(xtrain,ytrain,hidden,epochs)
    trainFcn = 'trainscg';
    hiddenLayerSize = hidden;
    net = patternnet(hiddenLayerSize, trainFcn);
    net=configure(net,xtrain,ytrain);
    net.trainParam.epochs=epochs;

    %[net,tr] = train(net,xtrain,ytrain);
    mod=net;
end
```

4. Model function with diff optim

```
function mod = model_function_with_diff_optim(xtrain,ytrain,hidden,epochs)
    %trainFcn = 'trainlm';
    trainFcn='trainrp';
    hiddenLayerSize = hidden;
    net = patternnet(hiddenLayerSize, trainFcn);
    net=configure(net,xtrain,ytrain);
    net.trainParam.epochs=epochs;

    %[net,tr] = train(net,xtrain,ytrain);
    mod=net;nd
```