COMP 4475 Topics in Artificial Intelligence

Instructor: Dr. Hongli Lyu

Project Report

Name: Parth Joshi

Lakehead ID: 1126914

Team Member's Name: Nikolas Brennan

Lakehead ID: 1109583

Contents

Contents 1

Acknowledgements 2

Abstract 3

1 Introduction

1.1 Research background 4

1.2 Research objective 4

1.3 Report structure 4

2 Theory Review

2.1 Neural Network model architectures 5

2.2 Broyden-Fletcher-Goldfarb-Shanno quasi-Newton Algorithm 5

3 Model Construction

3.1 Basic structure and variables 6

3.2 Star Dataset 7

3.3 More Models 7

4 Simulation Testing and Result Analysis

4.1 Model training process results 8

4.2 Model testing process results 11

4.3 Comparison results with other models 13

4.4 Result analysis 13

5 Conclusion……..15

References 15

Appendix code in MATLAB 16

Acknowledgements

With great appreciation I wish to express personal thanks to Dr Hongli Lyu, Graduate Assistant Ms. Lakshmi Preethi Kamak as well as my teammate Nikolas for giving their inputs in completion of this project.

Abstract

Artificial Neural Networks (ANN) are an important concept in the Artificial Intelligence field. They have been in use in video editing, detecting objects, detecting diseases and many more. ANNs usually have System Modelling, using supervised or unsupervised learning to train the AI and then predicting the results. This project uses ANN to predict the star type given the dataset of star types. This project can be useful for scientific purposes such as astronomy. Firstly, the seven input one output ANN Model is constructed, then the networks are trained by the given data, followed by being optimized. After that, it’s validated by the testing dataset, and then it shows the prediction of the star type accordingly. Moreover, the result is also compared with 4 other models for accuracy, efficiency, and performance purposes. This project has been purely written in MATLAB using MATLAB Toolboxes and Programming.

Key Words: Artificial Neural Networks (ANN), Neural Networks (NN), HR Diagram, MATLAB

1 Introduction

1.1 Research background

The prediction of star types is helpful for scientists to explore the universe, look for planets orbiting the star, study their life cycle, as well as to prove that they follow a certain graph in the space which is the HR Diagram (Hertzsprung-Russell Diagram). But, due to the complexity of predicting the star type, like dealing with huge numbers, keeping in mind so many star features (luminosity, temperate, etc.), it gets difficult for humans to forecast the star type and verify that it aligns with the HR Diagram.

Also, ANNs can be easily implemented, trained by the given dataset, optimized, comparable and they give accurate forecasts. Also, they can store information for future, deal with complex data/huge numbers as well as they can learn in a supervised and in unsupervised fashion. Thus, they are famous and have a widespread usage. Hence, NNs are used in this project to predict the star type based on a given dataset.

1.2 Research objective

Based on the dataset given [1], the objective of this project is to predict the star type as well as to prove that they follow the Hertzsprung-Russell Diagram, using ANN. First, a basic model is constructed, along with plotting the HR Diagram based on the dataset. Then the system is trained to predict the star type, followed by optimization (OptimizeHyperparameters, Bayesian). Finally, it forecasts the star type. A few more models are constructed to compare the results, performance, accuracy, along with verifying the output. Also, confusion charts, graphs and scatter plots have been used for a better understanding.

1.3 Report structure

This section lays out the introduction. The second part covers the theory. Section 3 describes the model construction. Section 4 shows the Simulation Testing and Result Analysis, and Section 5 gives the conclusion. In the end, there are references and MATLAB code.

2 Theory Review

2.1 Neural Network Model Architecture [2]

In this project, ANNs have been used with multiple inputs (7) with one output and the output is based on the dataset given. A general Neural Network has inputs, certain connected layers, an activation function (more about this in section 2.2), a softmax function and finally it consists of an output(s). Figs 1 and 2 show a general neural network.

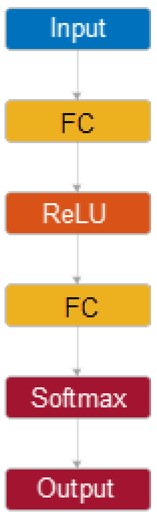


Fig 1 General Neural Network [2]

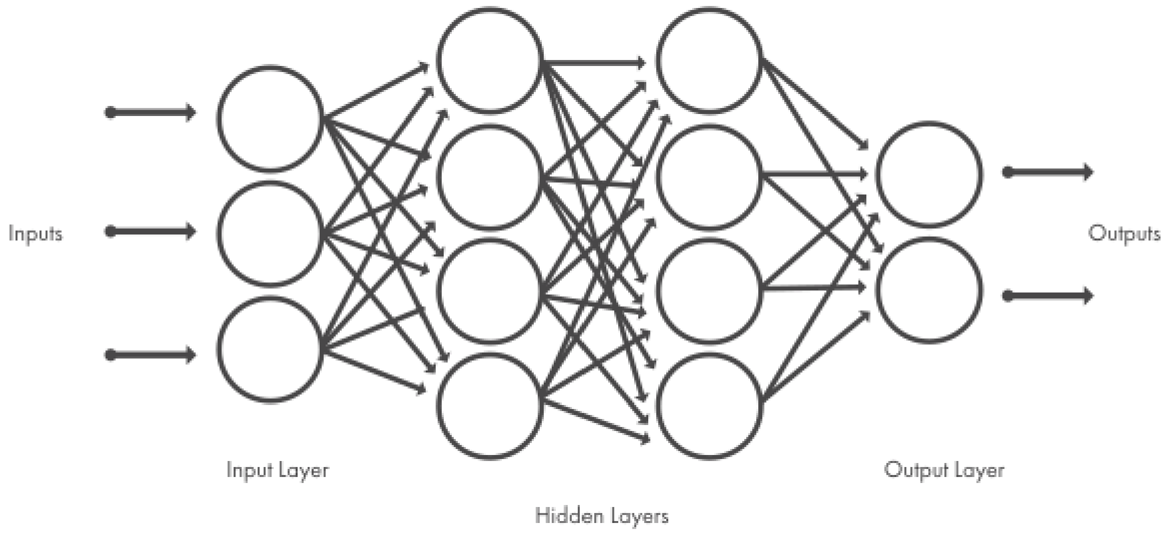


Fig 2 General Neural Network [3]

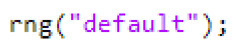
2.2 Broyden-Fletcher-Goldfarb-Shanno quasi-Newton Algorithm [2]

The Broyden-Fletcher-Goldfarb-Shanno quasi-Newton Algorithm (LBFGS) has been used in the project. This is an optimization algorithm and is a part of the activation function used in ANNs , which is “fitcnet”. Fitcnet function uses a limited memory LBFGS algorithm as its loss function minimization technique, where the software minimizes the cross-entropy loss.

3 Model Construction

3.1 Basic structure and variables [2]

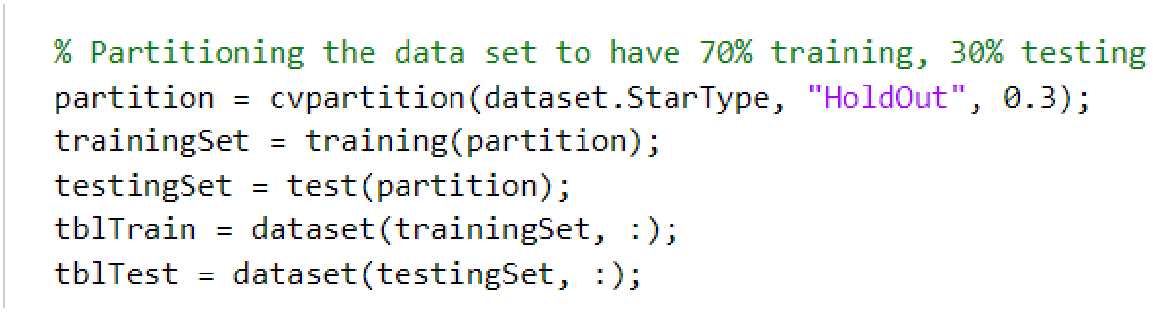
A basic Neural Network structure is constructed first a 7 input, 1 output model, based on the dataset with 10-layer neural network. First the dataset is initialized, loaded in the system and normalized.



```
rng("default");
```

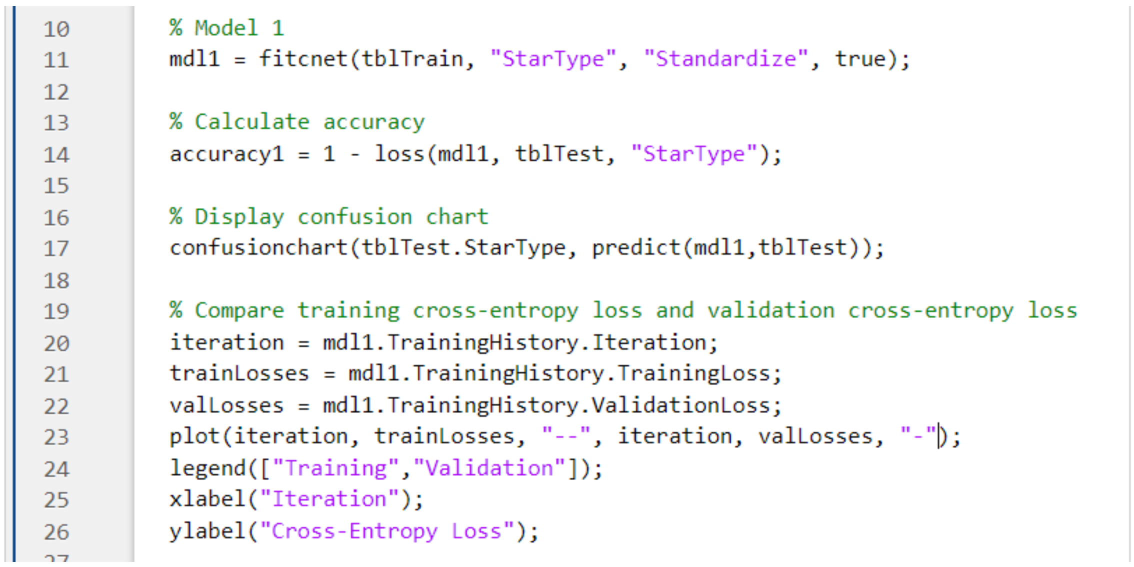
Then, we use rng(“default”) to set the random seed to the default value for reproducibility of the partition.

It’s also to ensure consistent simulation results, especially when it comes to the scenarios where there are random numbers being generated in MATLAB.



```
% Partitioning the data set to have 70% training, 30% testing
partition = cvpartition(dataset.StarType, "HoldOut", 0.3);
trainingSet = training(partition);
testingSet = test(partition);
tblTrain = dataset(trainingSet, :);
tblTest = dataset(testingSet, :);
```

The dataset is partitioned using the cvpartition function, which creates a random partition on a dataset, to have 70% training dataset and the rest 30% as a testing dataset. The system is trained using the training dataset and then tested using the testing dataset. The dataset function converts the variables into tables.



```
% Model 1
mdl1 = fitcnet(tblTrain, "StarType", "Standardize", true);

% Calculate accuracy
accuracy1 = 1 - loss(mdl1, tblTest, "StarType");

% Display confusion chart
confusionchart(tblTest.StarType, predict(mdl1,tblTest));

% Compare training cross-entropy loss and validation cross-entropy loss
iteration = mdl1.TrainingHistory.Iteration;
trainLosses = mdl1.TrainingHistory.TrainingLoss;
valLosses = mdl1.TrainingHistory.ValidationLoss;
plot(iteration, trainLosses, "--", iteration, valLosses, "-");
legend(["Training","Validation"]);
xlabel("Iteration");
ylabel("Cross-Entropy Loss");
```

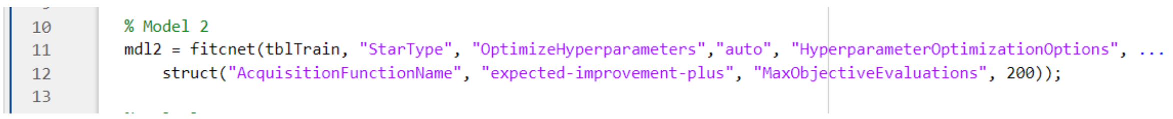
After that the first Model is obtained using the “fitcnet” function which was covered in Section 2.2 where we train the network using the training dataset to predict the star type. This is followed by calculating its accuracy, plotting a confusion chart along with using the predict function with the first model as well as the testing dataset as the parameters and finally comparing training cross-entropy loss and validation cross-entropy loss with a graph. This will be further discussed in section 3.3 and section 4. 3 more models have been constructed to compare the accuracy of the results as well as to compare the training cross-entropy loss and validation cross-entropy loss.

3.2 Star Dataset [1]

The Star Dataset consists of Temperature in Kelvin scale, Luminosity in Watts, Radius in metre, Absolute Magnitude, Star Type where Brown Dwarf -> Star Type = 0, Red Dwarf -> Star Type = 1, White Dwarf-> Star Type = 2, Main Sequence -> Star Type = 3, Supergiant -> Star Type = 4 and Hypergiant -> Star Type = 5. It also has Star Colour and the Spectral Class (O,B,A,F,G,K,M).

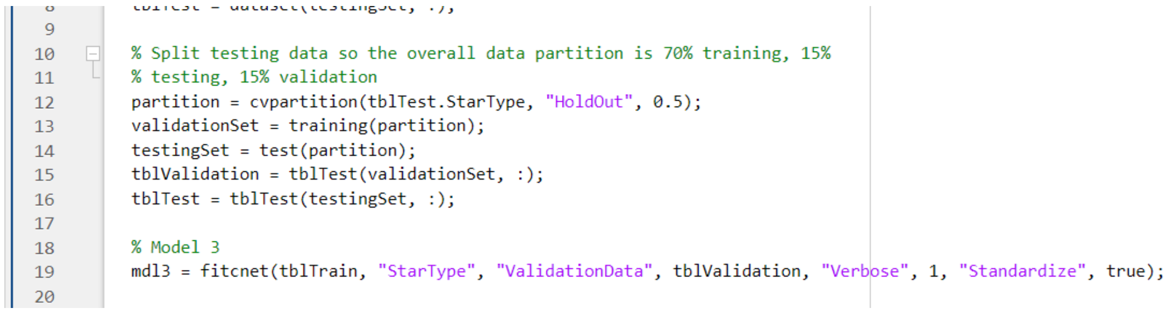
3.3 More Models [2]

As Mentioned in section 3.1, 3 other models were also constructed. They follow the same initial process of data partitioning, calculating the accuracy, plotting a confusion chart along with using the predict function with the respective model & the testing dataset as the parameters and comparing training cross-entropy loss and validation cross-entropy loss with a graph. However, the difference lies in obtaining the actual model using different methods. The results of these models along with confusion charts and graphs will be discussed in section 4.



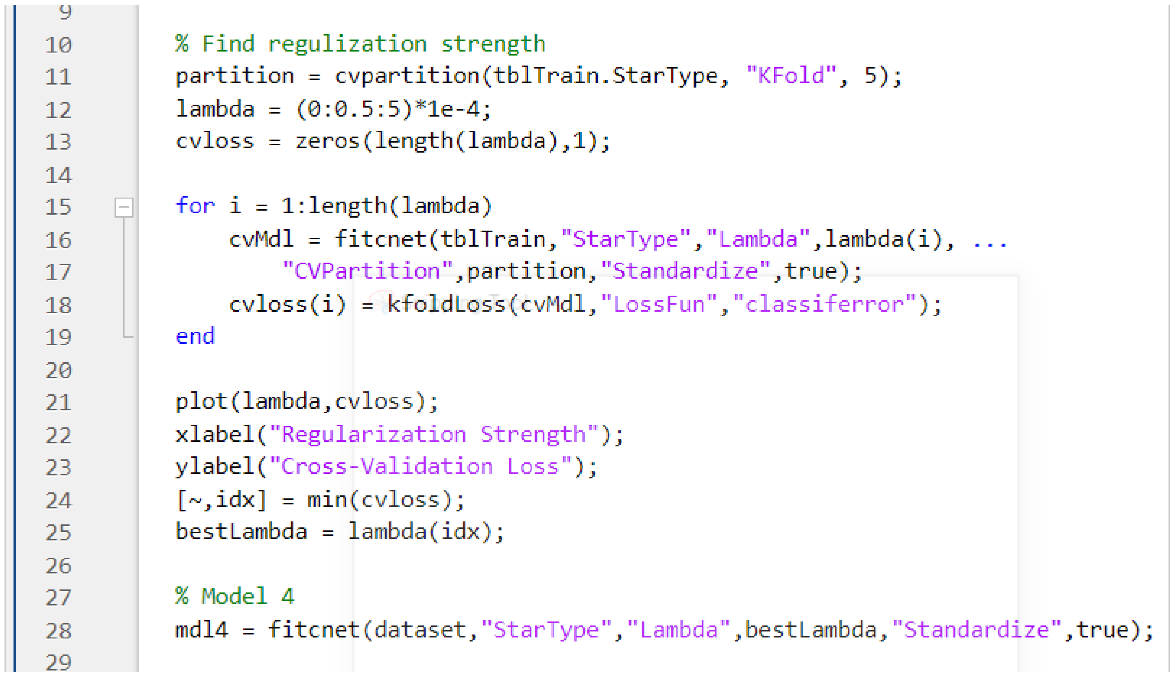
```
% Model 2
mdl2 = fitcnet(tblTrain, "StarType", "OptimizeHyperparameters","auto", "HyperparameterOptimizationOptions", ...
    struct("AcquisitionFunctionName", "expected-improvement-plus", "MaxObjectiveEvaluations", 200));
```

Model 2 also uses fitcnet method which is used to train neural network classifiers, with the training dataset to train it, response variable to be predicted (star type) and Optimize Hyperparameters being the important parameters. Optimize Hyperparameters is an argument available that improves the resulting classifier from fitcnet. In the end, 200 is the maximum training epoch.



```
tblTest = dataset(testingSet, :);

% Split testing data so the overall data partition is 70% training, 15%
% testing, 15% validation
partition = cvpartition(tblTest.StarType, "HoldOut", 0.5);
validationSet = training(partition);
testingSet = test(partition);
tblValidation = tblTest(validationSet, :);
tblTest = tblTest(testingSet, :);

% Model 3
mdl3 = fitcnet(tblTrain, "StarType", "ValidationData", tblValidation, "Verbose", 1, "Standardize", true);
```

Model 3 goes a bit different where the dataset is split into 70% training, 15% testing and 15% validation using the similar methods used in the other models. Then It uses fitcnet along with validating the data, as shown above. The purpose is to stop the training process early if the validation loss reaches a reasonable minimum.



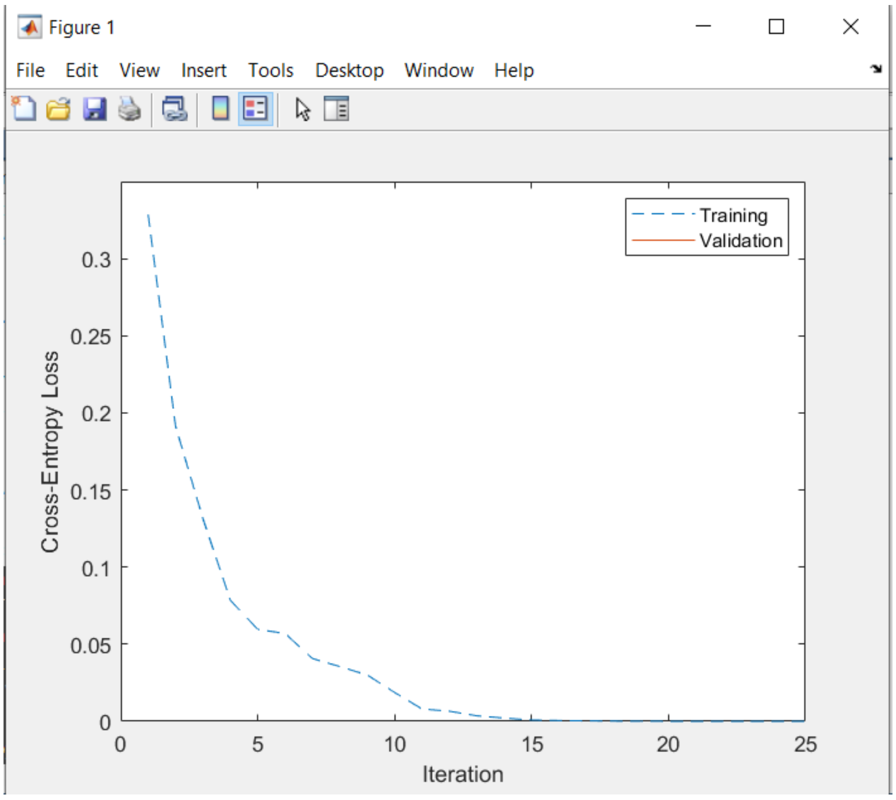
```
% Find regulization strength
partition = cvpartition(tblTrain.StarType, "KFold", 5);
lambda = (0:0.5:5)*1e-4;
cvloss = zeros(length(lambda),1);

for i = 1:length(lambda)
    cvMdl = fitcnet(tblTrain,"StarType","Lambda",lambda(i), ...
        "CVPartition",partition,"Standardize",true);
    cvloss(i) = kfoldLoss(cvMdl,"LossFun","classiferror");
end

plot(lambda,cvloss);
xlabel("Regularization Strength");
ylabel("Cross-Validation Loss");
[~,idx] = min(cvloss);
bestLambda = lambda(idx);

% Model 4
mdl4 = fitcnet(dataset,"StarType","Lambda",bestLambda,"Standardize",true);
```

Finally, in Model 4, after partitioning the dataset, regularization strength is calculated using Lambda and cvloss. A cvpartition object is created for stratified 5-fold cross-validation. cvp partitions the data into five folds, where each fold has roughly the same proportions of the star type. Fitcnet is used again while iterating it and standardizing the data. Then the results are plotted and then the final model is obtained by training the neural network classifier using the best lambda regularization strength.

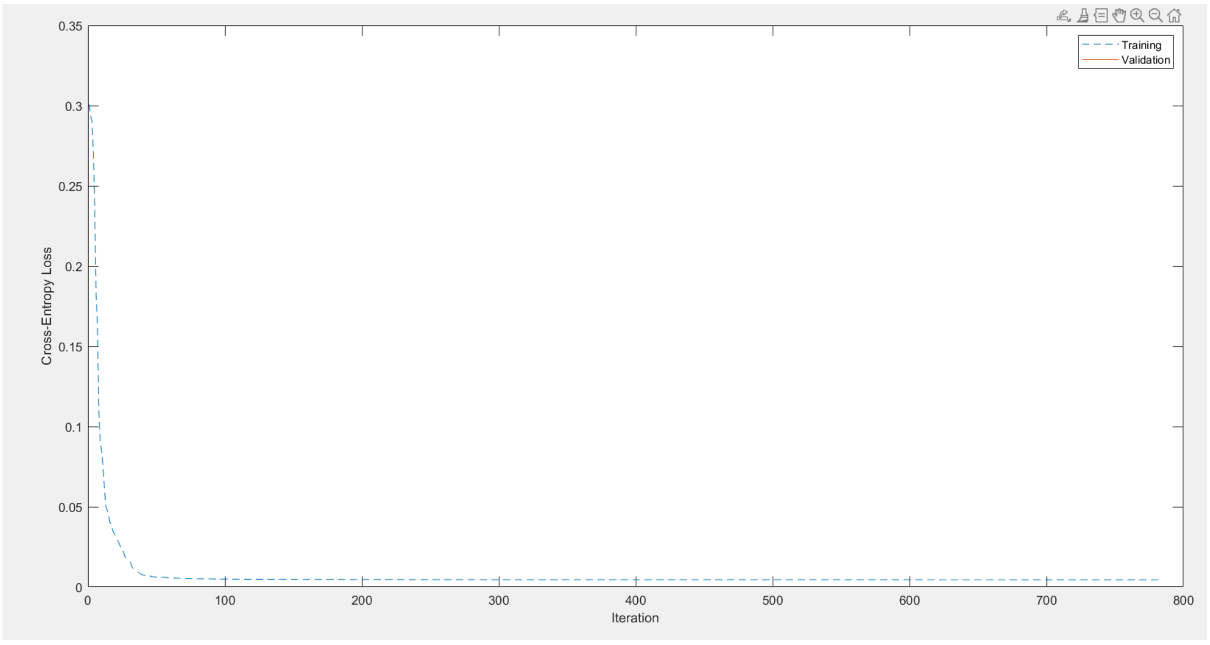
4 Simulation Testing and Result Analysis

4.1 Model training process results

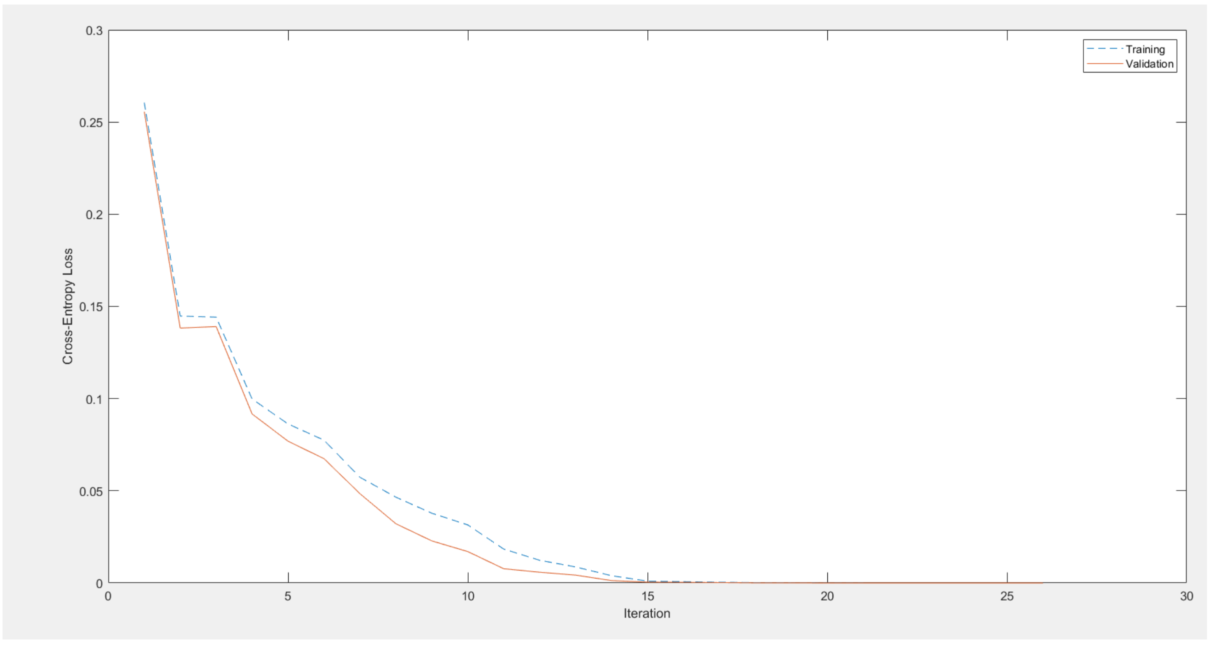
Please note that running the code files would take longer time (5-10 mins) to generate the results as it takes time to train the neural networks. For simplicity, FinalWorspace.mat has been also attached in the submission which has all the results stored. Few lines of code can be highlighted to plot the graphs and the confusion charts, then right clicking and then clicking on “evaluate selection in command window”. Model 1 is of 10 layer size, rectified linear unit (relu) activation function with 97.2% accuracy. The following figure shows its validation loss graph.



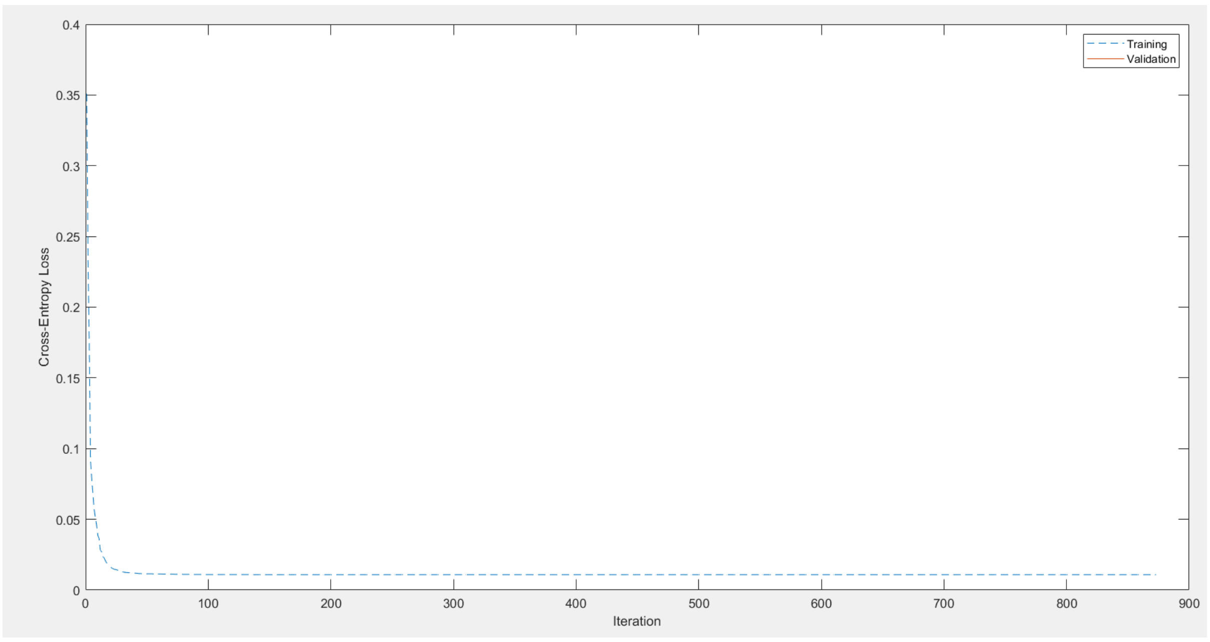
Model 2 is of 272 layer size, sigmoid activation function with 98.6% accuracy. The following figure shows its validation loss graph.



Model 3 is of 10 layer size, rectified linear unit (relu) activation function with 94.4% accuracy. The following figure shows its validation loss graph.

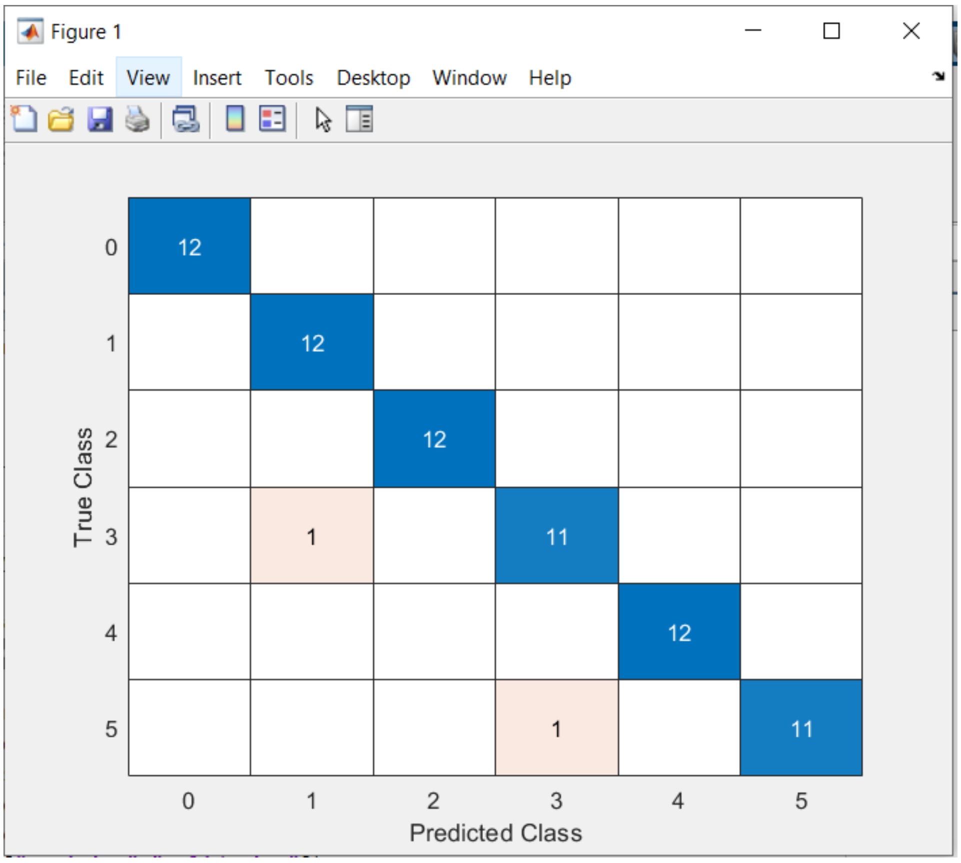


Model 4 is of 10 layer size, rectified linear unit (relu) activation function with 100% accuracy. The following figure shows its validation loss graph.



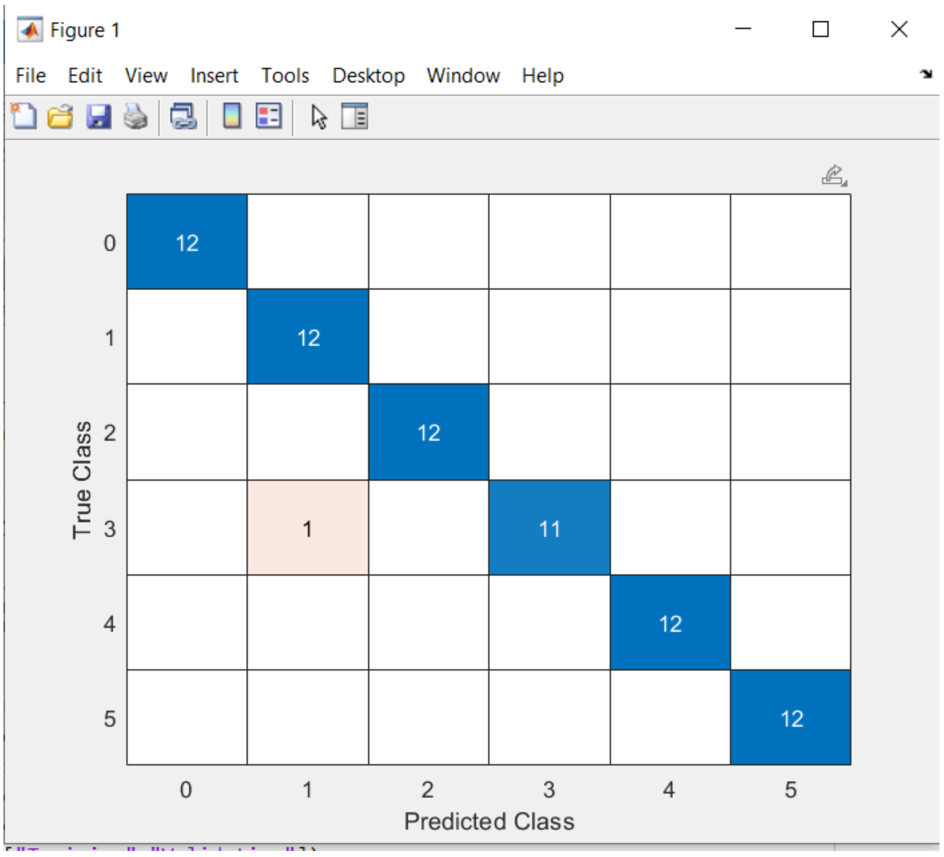
4.2 Model testing process results

We plot the confusion charts for each model. . The following figure shows confusion chart of model 1.



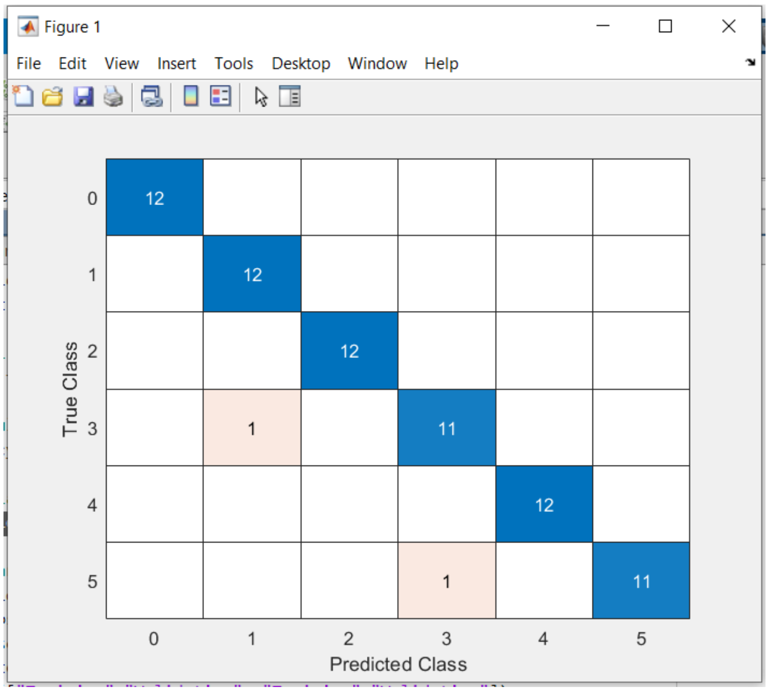
True Class \ Predicted Class	0	1	2	3	4	5
0	12					
1		12				
2			12			
3		1		11		
4					12	
5				1		11

The following figure shows confusion chart of model 2.



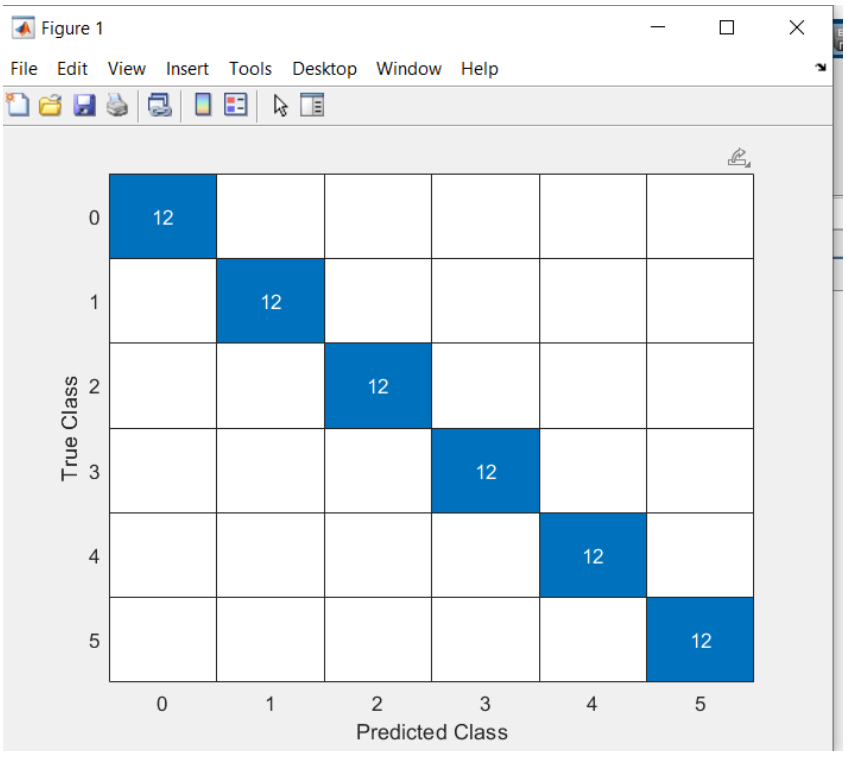
True Class \ Predicted Class	0	1	2	3	4	5
0	12					
1		12				
2			12			
3		1		11		
4					12	
5						12

The following figure shows confusion chart of model 3.



True Class \ Predicted Class	0	1	2	3	4	5
0	12					
1		12				
2			12			
3		1		11		
4					12	
5				1		11

The following figure shows confusion chart of model 4.

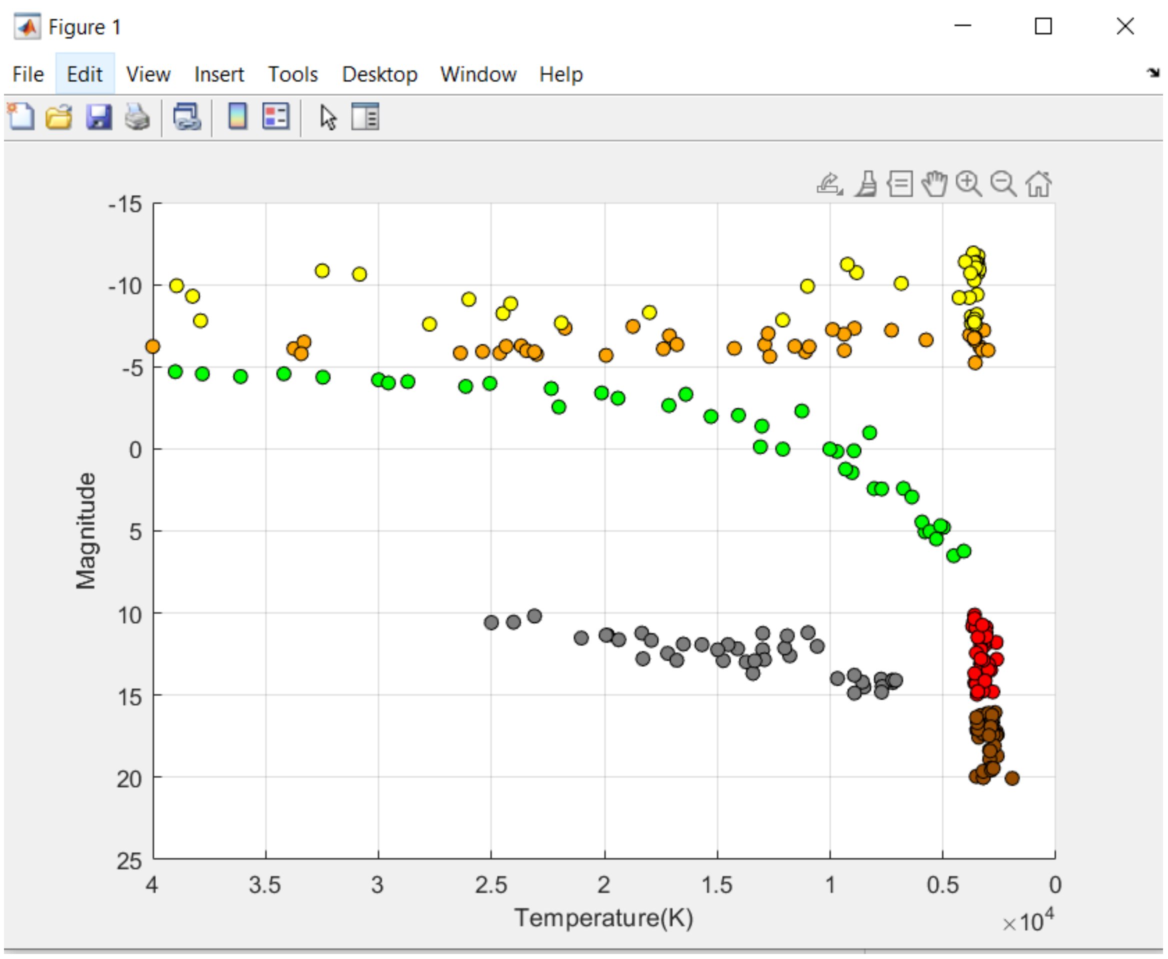


4.3 Comparison results with other models

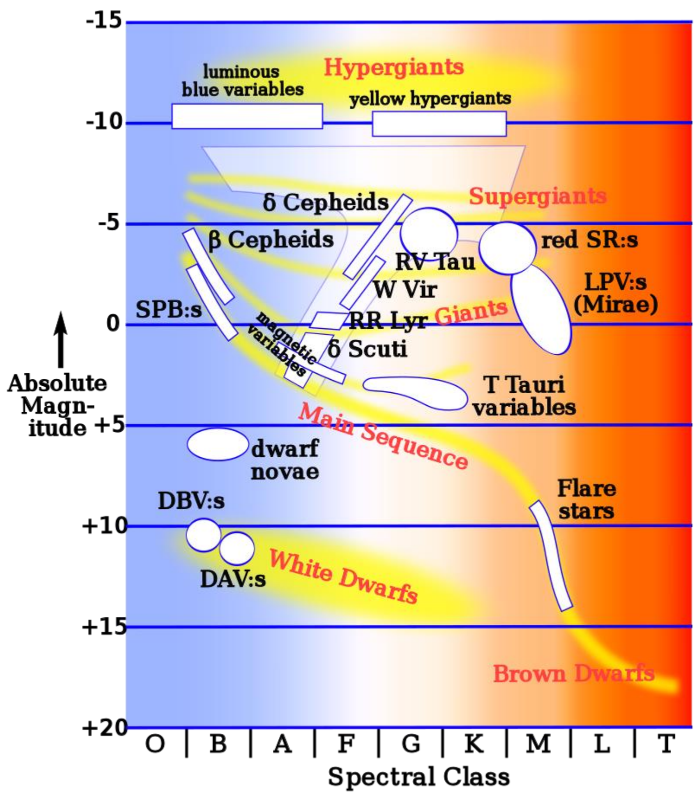
As we compare the results with the other models, we see that model 4 has 100% accuracy with an ideal confusion chart predicting the star type based on the given dataset. Also the validation loss graphs give a better insight on comparing the training cross-entropy loss and validation cross-entropy loss for each model.

4.4 Result analysis

We see that the star type forecasted by the 4 models align with the HR Diagram. This diagram has been plotted using MATLAB and verified using the actual diagram from [4]. The code for the HR diagram has been given in the end. The following is the HR Diagram (scatter plot) made using the same dataset used for this project, in MATLAB. Yellow and orange are supergiants, green is the main sequence, grey dots are the white dwarfs and brown dots are brown dwarfs.



The following is the actual HR Diagram, Source: [4]



5 Conclusion

In conclusion, we see that the results are 94-100% accurate in predicting the star type which is ideal for research purposes. We also conclude that ANNs can be a powerful and useful tool to predict complex results in a simpler way and is easy to understand and implement. Moreover, the results can also be verified using the HR Diagram.

References

1. Dataset taken from https://www.kaggle.com/datasets/deepu1109/star-dataset
2. https://www.mathworks.com/help/stats/fitcnet.html#mw_84c7b45a-3206-4135-8c6d-5d844fe6514e
3. https://www.mathworks.com/discovery/neural-network.html
4. https://en.wikipedia.org/wiki/Yellow_hypergiant#/media/File:HR-vartype.svg

Appendix

Code in MATLAB

Before running the code, make sure the "machine learning toolbox" add-on has been installed in MATLAB. Additionally, all the code files and the dataset are also attached with the submission. These files must be executed according to the sequence given for the desired results.

File: datainit.m

```
dataset = stars;
dataset.StarColor = grp2idx(dataset.StarColor);
dataset.SpectralClass = grp2idx(dataset.SpectralClass);
```

File: Model1.m

```
rng("default");

% Partitioning the data set to have 70% training, 30% testing
partition = cvpartition(dataset.StarType, "HoldOut", 0.3);
trainingSet = training(partition);
testingSet = test(partition);
tblTrain = dataset(trainingSet, :);
tblTest = dataset(testingSet, :);

% Model 1
mdl1 = fitcnet(tblTrain, "StarType", "Standardize", true);

% Calculate accuracy
accuracy1 = 1 - loss(mdl1, tblTest, "StarType");

% Display confusion chart
confusionchart(tblTest.StarType, predict(mdl1,tblTest));

% Compare training cross-entropy loss and validation cross-entropy loss
iteration = mdl1.TrainingHistory.Iteration;
trainLosses = mdl1.TrainingHistory.TrainingLoss;
valLosses = mdl1.TrainingHistory.ValidationLoss;
plot(iteration, trainLosses, "--", iteration, valLosses, "-");
legend(["Training","Validation"]);
xlabel("Iteration");
ylabel("Cross-Entropy Loss");
```

File: Model2.m

```
rng("default");

% Partitioning the data set to have 70% training, 30% testing
partition = cvpartition(dataset.StarType, "HoldOut", 0.3);
trainingSet = training(partition);
testingSet = test(partition);
tblTrain = dataset(trainingSet, :);
tblTest = dataset(testingSet, :);
```

```
% Model 2
mdl2 = fitcnet(tblTrain, "StarType", "OptimizeHyperparameters","auto",
"HyperparameterOptimizationOptions", ...
    struct("AcquisitionFunctionName", "expected-improvement-plus",
"MaxObjectiveEvaluations", 200));

% Calculate accuracy
accuracy2 = 1 - loss(mdl2, tblTest, "StarType");

% Display confusion chart
confusionchart(tblTest.StarType, predict(mdl2,tblTest));


% Compare training cross-entropy loss and validation cross-entropy loss
iteration = mdl2.TrainingHistory.Iteration;
trainLosses = mdl2.TrainingHistory.TrainingLoss;
valLosses = mdl2.TrainingHistory.ValidationLoss;
plot(iteration, trainLosses, "--", iteration, valLosses, "-");
legend(["Training","Validation"]);
xlabel("Iteration");
ylabel("Cross-Entropy Loss");
```

File: Model3.m

```
rng("default");

% Partitioning the data set to have 70% training, 30% testing
partition = cvpartition(dataset.StarType, "HoldOut", 0.3);
trainingSet = training(partition);
testingSet = test(partition);
tblTrain = dataset(trainingSet, :);
tblTest = dataset(testingSet, :);

% Split testing data so the overall data partition is 70% training, 15%
% testing, 15% validation
partition = cvpartition(tblTest.StarType, "HoldOut", 0.5);
validationSet = training(partition);
testingSet = test(partition);
tblValidation = tblTest(validationSet, :);
tblTest = tblTest(testingSet, :);

% Model 3
mdl3 = fitcnet(tblTrain, "StarType", "ValidationData", tblValidation, "Verbose",
1, "Standardize", true);

% Calculate accuracy
accuracy3 = 1 - loss(mdl3, tblTest, "StarType");

% Display confusion chart
confusionchart(tblTest.StarType, predict(mdl3,tblTest));

% Compare training cross-entropy loss and validation cross-entropy loss
iteration = mdl3.TrainingHistory.Iteration;
trainLosses = mdl3.TrainingHistory.TrainingLoss;
valLosses = mdl3.TrainingHistory.ValidationLoss;
plot(iteration, trainLosses, "--", iteration, valLosses, "-");
legend(["Training","Validation", "Training","Validation"]);
xlabel("Iteration");
```

```
ylabel("Cross-Entropy Loss");
```

File: Model4.m

```
rng("default");

% Partitioning the data set to have 70% training, 30% testing
partition = cvpartition(dataset.StarType, "HoldOut", 0.3);
trainingSet = training(partition);
testingSet = test(partition);
tblTrain = dataset(trainingSet, :);
tblTest = dataset(testingSet, :);

% Find regulization strength
partition = cvpartition(tblTrain.StarType, "KFold", 5);
lambda = (0:0.5:5)*1e-4;
cvloss = zeros(length(lambda),1);

for i = 1:length(lambda)
    cvMdl = fitcnet(tblTrain,"StarType","Lambda",lambda(i), ...
        "CVPartition",partition,"Standardize",true);
    cvloss(i) = kfoldLoss(cvMdl,"LossFun","classiferror");
end

plot(lambda,cvloss);
xlabel("Regularization Strength");
ylabel("Cross-Validation Loss");
[~,idx] = min(cvloss);
bestLambda = lambda(idx);

% Model 4
mdl4 = fitcnet(dataset,"StarType","Lambda",bestLambda,"Standardize",true);

% Calculate accuracy
accuracy4 = 1 - loss(mdl4, tblTest, "StarType", "LossFun", "classiferror");

% Display confusion char
confusionchart(tblTest.StarType, predict(mdl4,tblTest));

% Compare training cross-entropy loss and validation cross-entropy loss
iteration = mdl4.TrainingHistory.Iteration;
trainLosses = mdl4.TrainingHistory.TrainingLoss;
valLosses = mdl4.TrainingHistory.ValidationLoss;
plot(iteration, trainLosses, "--", iteration, valLosses, "-");
legend(["Training","Validation"]);
xlabel("Iteration");
ylabel("Cross-Entropy Loss");
```

Optionally, HR Diagram has been plotted using MATLAB to verify that the dataset aligns with the Diagram. This file runs only after running datainit.m.

File: StarDataScatterPlot.m

```
rows = height(dataset);
brown = [150 75 0]/256;
orange = [255 165 0]/256;
```

```
grey = [126 126 126]/256;
for x = 1:rows
    if dataset(x,:).StarType==0
        scatter(dataset(x,:).TemperatureK, dataset(x,:).AbsoluteMagnitudeMv, [],
brown, "filled", "MarkerEdgeColor", "k", "LineWidth", 0.5);
    elseif dataset(x,:).StarType==1
        scatter(dataset(x,:).TemperatureK, dataset(x,:).AbsoluteMagnitudeMv,
"red", "filled", "MarkerEdgeColor", "k", "LineWidth", 0.5);
    elseif dataset(x,:).StarType==2
        scatter(dataset(x,:).TemperatureK, dataset(x,:).AbsoluteMagnitudeMv, [],
grey, "filled", "MarkerEdgeColor", "k", "LineWidth", 0.5);
    elseif dataset(x,:).StarType==3
        scatter(dataset(x,:).TemperatureK, dataset(x,:).AbsoluteMagnitudeMv,
"green", "filled", "MarkerEdgeColor", "k", "LineWidth", 0.5);
    elseif dataset(x,:).StarType==4
        scatter(dataset(x,:).TemperatureK, dataset(x,:).AbsoluteMagnitudeMv, [],
orange, "filled", "MarkerEdgeColor", "k", "LineWidth", 0.5);
    elseif dataset(x,:).StarType==5
        scatter(dataset(x,:).TemperatureK, dataset(x,:).AbsoluteMagnitudeMv,
"yellow", "filled", "MarkerEdgeColor", "k", "LineWidth", 0.5);
    end
    hold on
end
hold off
set(gca, "YDir", "reverse");
set(gca, "Xdir", "reverse");
xlabel('Temperature(K)');
ylabel('Magnitude');
set(gca, "YGrid", 'on');
set(gca, "XGrid", 'on');
```