Computational Intelligence Assignment

A Report for Regression and Classification Using Neural Networks

By

Pedapudi Venkata Sai Vijay Kumar (A0178190Y)

Maddi Kamal Manisha (A0178259M)

Monisha Prasad (A0178265U)

Akshaya Balamurugan (A0178458L)

TABLE OF CONTENTS

PART - A: REGRESSION

1. INTRODUCTION

1.1. Domain Understanding

The California Cooperative Oceanic Fisheries Investigations (CalCOFI) is a unique partnership of the California Department of Fish & Wildlife and NOAA Fisheries Service and Scripps Institution of Oceanography. The organization was formed in 1949 to study the ecological aspects of the sardine population collapse off California. Today our focus has shifted to the study of the marine environment off the coast of California, the management of its living resources, and monitoring the indicators of El Nino and climate change. CalCOFI conducts quarterly cruises off southern & central California, collecting a suite of hydrographic and biological data on station and underway. Data collected at depths down to 500m include: temperature, salinity, oxygen, phosphate, silicate, nitrate and nitrite, chlorophyll, transmissometer, PAR, C14 primary productivity, phytoplankton biodiversity, zooplankton biomass, and zooplankton biodiversity.

1.2. Task Definition

This project aims to predict the temperature of the ocean and to be able to understand the biodiversity and patterns of life under the sea with least number of variables to reduce the cost using CRISP-DM approach. This prediction will also help in estimating the Global warming and melting of ice near pacific region.

2. DATA UNDERSTANDING

2.1. Structure of data set

The dataset is a time series data over a time span of 60 years collected at more than 50,000 sampling stations, out of which the latest available data is taken into consideration. The dataset contains a set of 74 features like ocean's temperature, height, depth, etc. with more than 8 lakh observations. Hence, the latest available data has been taken, i.e., the data collected for the year 2016. The attributes and their description is as below.

Attributes	Description
Cst_Cnt	Auto-numbered Cast Count

Attributes	Description
Btl_Cnt	Auto-numbered Bottle count
Sta_ID	CalCOFI Line and Station
Depth_ID	[Century]-[YY][MM][ShipCode]-[CastType][Julian Day]-[CastTime]-[Line][Sta][Depth][Bottle]-[Rec_Ind]
Depthm	Depth in meters
T_degC	Temperature of Water
Salnty	Salinity of water
O2ml_L	Milliliters of dissolved oxygen per Liter seawater
STheta	Potential Density of Water
02Sat	Oxygen Saturation
Oxy_umol/kg	Oxygen in micro moles per kilogram of seawater
BtlNum	Bottle Number
Reclnd	Record Indicator
T_prec	Temperature Units of Precision
T_qual	Temperature Quality Code
S_prec	Salinity Units of Precision
S_qual	Salinity Quality Code
P_qual	Pressure Quality Code
O_qual	Oxygen Quality Code
SThatq	Sigma Theta Quality Code
O2Satq	Oxygen Saturation Quality Code
ChlorA	Acetone extracted chlorophyll-a measured fluorometrically
Chlqua	Chlorophyll-a Quality Code
Phaeop	Phaeophytin concentration measured fluorometrically
Phaqua	Phaeophytin Quality Code
PO4µM	Phosphate concentration
PO4q	Phosphate Quality Code
SiO3µM	Silicate concentration
SiO3qu	Silicate Quality Code
NO2µM	Nitrite concentration
NO2q	Nitrite Quality Code
NO3µM	Nitrate concentration
NO3q	Nitrate Quality Code
NH3µM	Ammonium concentration
NH3q	Ammonium Quality Code
C14As1	14C Assimilation of replicate 1
C14A1p	14C Assimilation of replicate 1 precision

Attributes	Description
C14A1q	14C As1 Quality Code
C14As2	14C Assimilation of replicate 2
C14A2p	14C Assimilation of replicate 2 precision
C14A2q	14C As2 Quality Code
DarkAs	14C Assimilation Dark Bottle
DarkAp	14C Assimilation Dark Bottle precision
DarkAq	14C Assimilation Dark Bottle Quality Code
MeanAs	Mean 14C Assimilation of Bottle 1 and 2
MeanAp	Mean 14C Assimilation of Bottle 1 and 2 precisions
MeanAq	Mean 14C Assimilation Quality Code
IncTim	Incubation time
LightP	Light intensities expressed as a percentage
R_Depth	Reported Depth in Meters
R_TEMP	Reported Temperature
R_POTEMP	Reported Potential Temperature
R_SALINITY	Reported Salinity
R_SIGMA	Reported Potential Density of water
R_SVA	Reported Specific Volume Anomaly
R_DYNHT	Reported Dynamic Height
R_02	Reported milliliters of oxygen per liter of seawater
R_O2Sat	Reported Oxygen Saturation
R_SIO3	Reported Silicate Concentration
R_PO4	Reported Phosphate Concentration
R_NO3	Reported Nitrate Concentration
R_NO2	Reported Nitrite Concentration
R_NH4	Reported Ammonium Concentration
R_CHLA	Reported Chlorophyll-a
R_PHAEO	Reported Phaeophytin
R_PRES	Pressure in decibars
R_SAMP	Sample number
R_Oxy_µmol/kg	Reported Oxygen in micro moles per kilogram of seawater
DIC1	Replicate 1 - Dissolved Inorganic Carbon in micro moles per kilogram of seawater
DIC2	Replicate 2 - Dissolved Inorganic Carbon in micro moles per kilogram of seawater
TA1	Replicate 1 - Total Alkalinity in micro moles per kilogram of seawater

Attributes	Description
TA2	Replicate 2 - Total Alkalinity in micro moles per kilogram of seawater
DIC Quality (Comments)	Quality Comments associated with DIC sampling, drawing and analysis

Table 2.1: Attribute Description

2.2. Exploratory Data Analysis

It's been observed that with increasing depth, the temperature of the sea water falls due to in-accessibility of sunlight. Cold water is denser, so at higher depths the salinity would be higher. Hence an inverse proportionality exits between Temperature and Salinity.

Oxygen saturation is a ratio of the concentration of dissolved oxygen (O_2) in water to the maximum amount of oxygen that will dissolve in the water at that temperature and pressure under stable equilibrium. It's been observed that O_2 saturation is directly related to temperature.

Also, the density of deeper waters would be more than shallow waters. This makes the potential density of seawater inversely related to temperature. Also, temperature of the seawater and pressure are inversely proportional. These can be observed from the below plots.

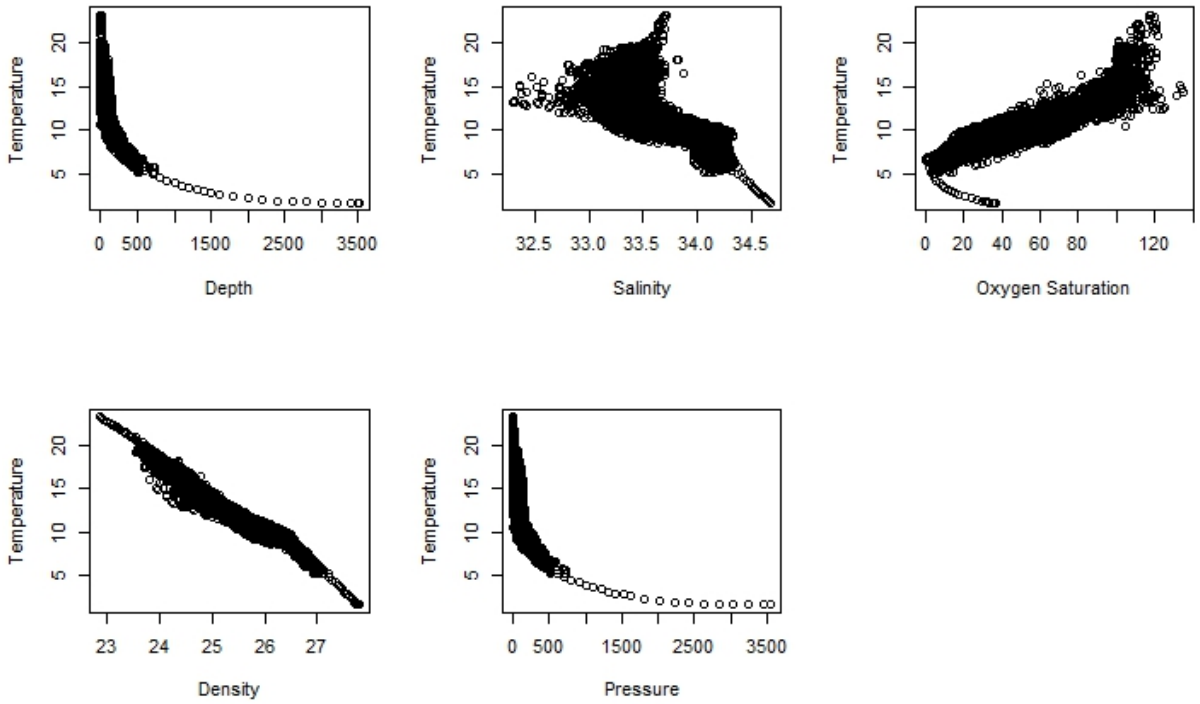


Fig 2.2: Attribute Relationship

3. DATA PREPARATION

3.1. Missing Value Analysis

The windowed data is analyzed for missing values. It has been observed that some of the attributes have 95% to 100% of missing data. These are removed as it has no effect on the prediction. The below plot shows the overall view of the missing values in the complete data:

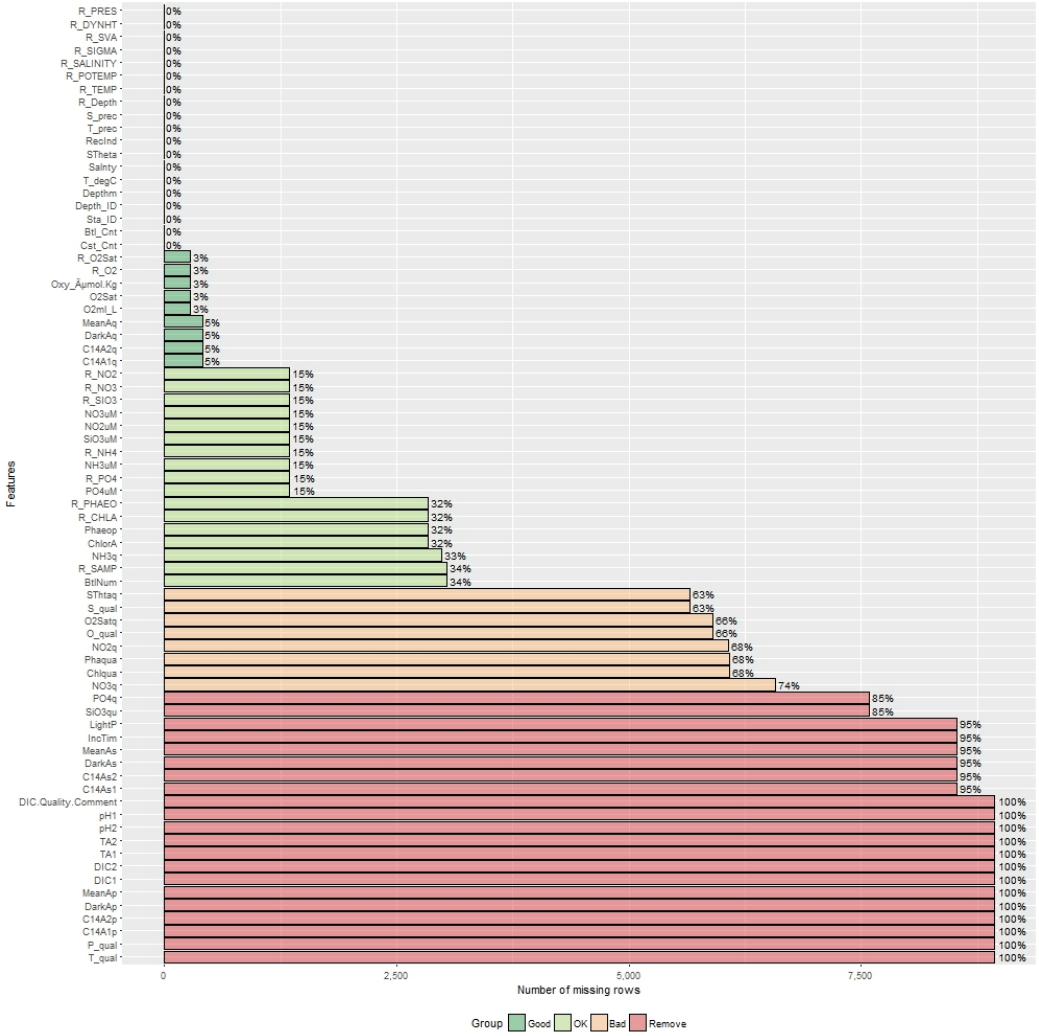


Fig 3.1: Missing Value Analysis

3.2 Imputation

The remaining attributes are now imputed using HMISC package and Linear model. The columns BtlNum and R_SAMP will not have any effect on the temperature and has been imputed with zero. As mentioned earlier, the columns describing the quality are imputed with value 1 as blanks represents good data quality and has no relationship with the temperature. Owing to the

linearity existing among few attributes, 3 linear models have been introduced to impute the following: O2ml_L based on Salinity, O2Sat based on O2ml_L and Oxy_Âµmol.Kg based on O2ml_L.

Furthermore, to impute the remaining missing values in the predictors, we have used aregImpute() which allows mean imputation using additive regression, bootstrapping and predictive mean matching. In bootstrapping, different bootstrap resamples are used for each of multiple imputations. Then, a flexible additive model (non-parametric regression method) is fitted on samples taken with replacements from original data and missing values are predicted using non-missing values. Then, it uses predictive mean matching (default) to impute missing values. Predictive mean matching works well for continuous and categorical (binary & multi-level).

3.3 Outlier and Inconsistent Value Detection

The target variable R_TEMP has a perfect normal distribution with no outliers. The same can be visualized below,

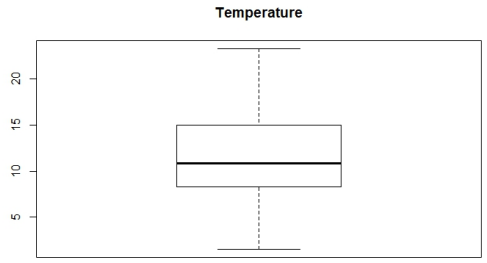


Fig 3.3: Outlier Analysis

All the predictor values are checked for inconsistent data i.e., data having negative values and values that are very high when it's supposed to vary only in a range. It's been found that all values are consistent and justifiable.

3.4 Feature Selection

As part of feature engineering, the attributes pertaining to quality codes (T_qual, P_qual, O4q, iO3qu), precision attributes (DarkAs, DarkAp, MeanAs, MeanAp) along with concentrations measured for different chemicals (C14As1, C14A1p, C14As2, C14A2p, DIC1, DIC2, TA1, TA2, pH2, pH1, LightP) are ignored as they don't influence the temperature. At the end of this step, we have 15 predictors along with 1 target variable which will be considered for the model building. The same can be referred from the table below:

Important Attributes	Comments
R_DEPTH, R_SALINITY, R_SIGMA, R_SVA, R_DYNHT	Depth, salinity, density have an influence on the target variable
R_O2, R_O2Sat, R_CHLA R_PHAEO, R_PRES, oxy_mol_kg	These are measures of oxygen levels and chlorophyll which gives us the information on life sustainability
R_SIO3, R_PO4, R_NO3, R_NO2, R_NH4, NH3Um	These are the major chemical compositions that influence the water temperature.

Table 3.4: Feature Selection

3.5 Relationship Study

To understand the relationship between the dependent variables and the target variable, the correlation matrix is generated.

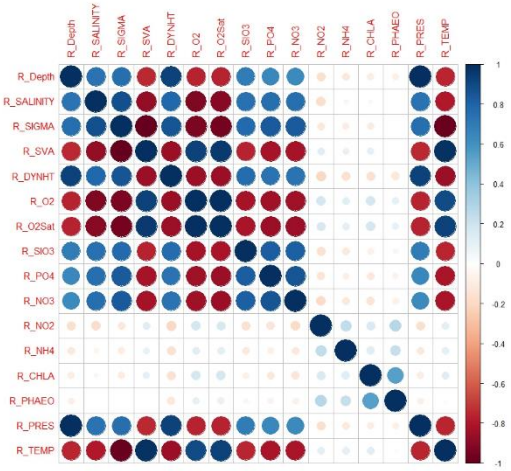


Fig 3.5: Correlation Plot

A regression using different neural network architectures is applied on this data and predictions are done. This is discussed in detail in the following section.

3.6 Normalizing Data

Once the important features are extracted, the data is now normalized using min- max normalization technique so that neural network converges easily for the entire training and testing data.

3.7 Training and Testing Data Split

The total records 7200 are split in a ratio of 80:20 for training and testing the models.

4. DATA MODELING

4.1 Regression with Neural Network

A fully connected neural network with hidden layers and linear activation function is trained for adjusted weights by defining a loss function which is least square error by gradient descent learning. In this case we have chosen one hidden layer, as we don't have any activation function.

$$\text{No of hidden nodes} = \sqrt{(nin * nop)} = 4$$

Also, intuitively we have chosen our starting point as 4 nodes in the input layer and extended it up to 6. Model has been tuned for an optimal value which is discussed in the next section.

4.1.1. Hyperparameter Tuning:

The weight decay which inhibits the overfit and the number of hidden nodes are obtained after multiple iterations using caret package in R. Once the optimal values are obtained, the model is trained using this. Predictions on the testing data are done using the trained model.

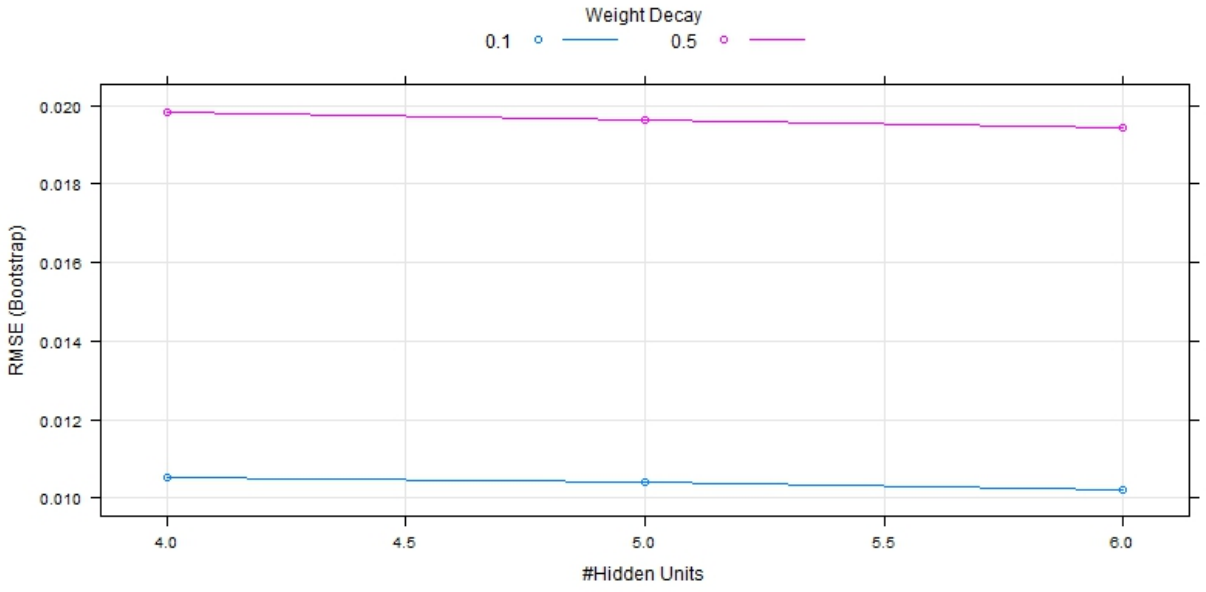


Fig 4.1.1.1: Tuning of NeuralNet

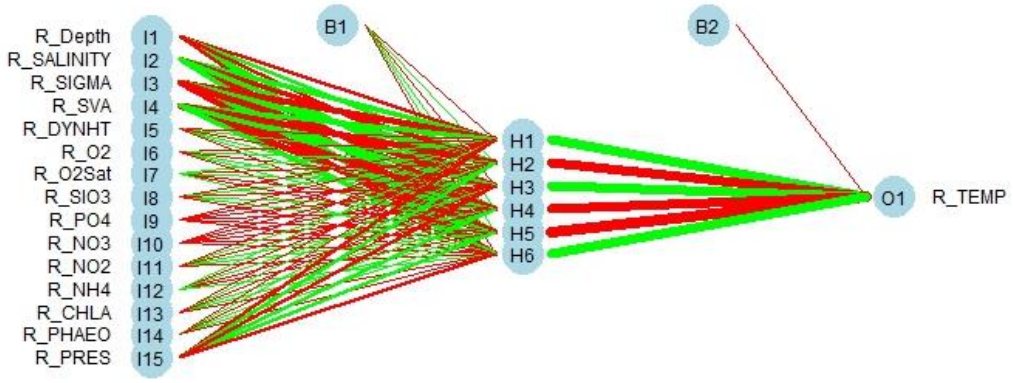


Fig 4.1.1.2: Neural Network

In this case, the model is built with 6 hidden nodes and a weight decay of 0.1 for 500 iterations**.** This resulted in a RMSE of **0.01011.** A plot of actual vs predicted temperature values is shown below.

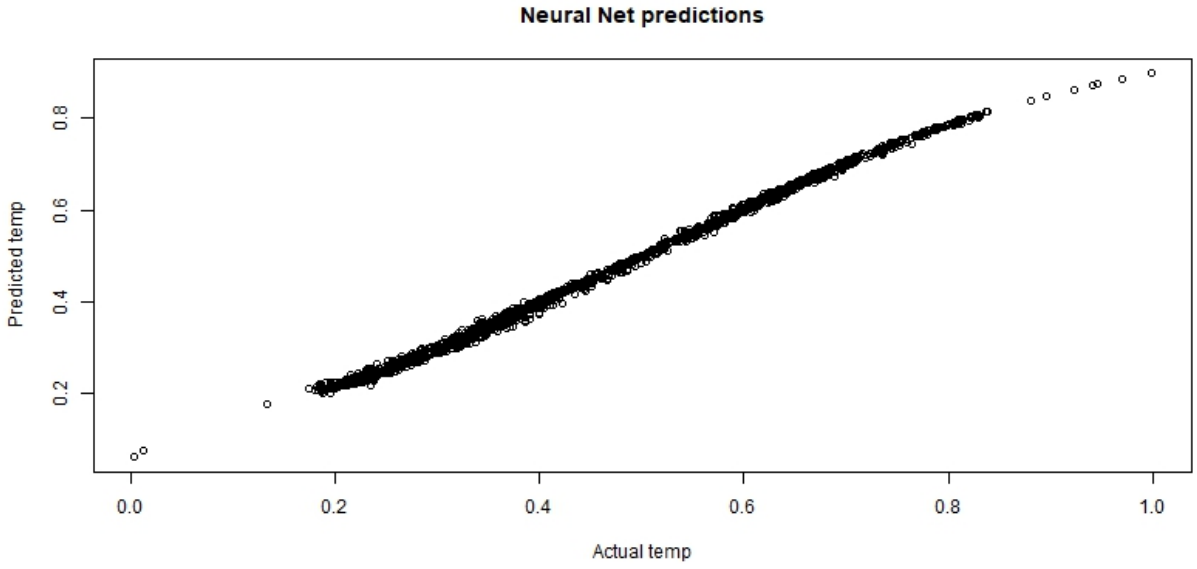


Fig 4.1.1.3: Predicted vs. Actual

4.2 Regression with RBF

It is a semi supervised learning neural net with three layers. The input layer is fully connected to the hidden layer that has values at cluster centers learned by K-means and calculates the Gaussian distance between fed input variables and cluster centers. The output layer is a

weighted sum of the gaussian activations which would learn based on the error function as and when it progresses through iterations.

4.2.1. Hyperparameter Tuning:

The RBF function in RSNNS package is used for this. The optimal number of hidden nodes has been chosen as 10 i.e. number of cluster centers and the output node would be one. The training has been performed using this network.

From the learning curve given below its observed that after 100 iterations there is no change in the error i.e. the weights are such that it has reached the minima of loss function. The RMSE of the finalized model is **0.03555** and the graph between predicted and actual temperature values is plotted.

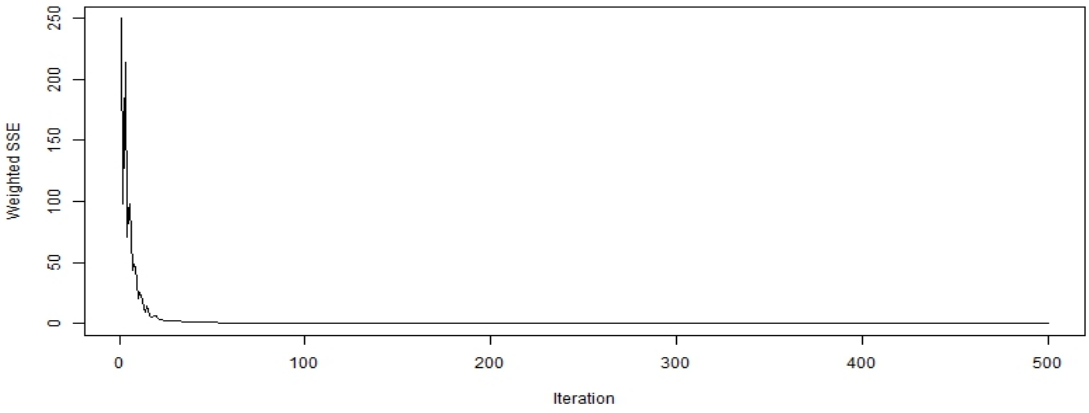


Fig 4.2.1.1: Tuning the RBF

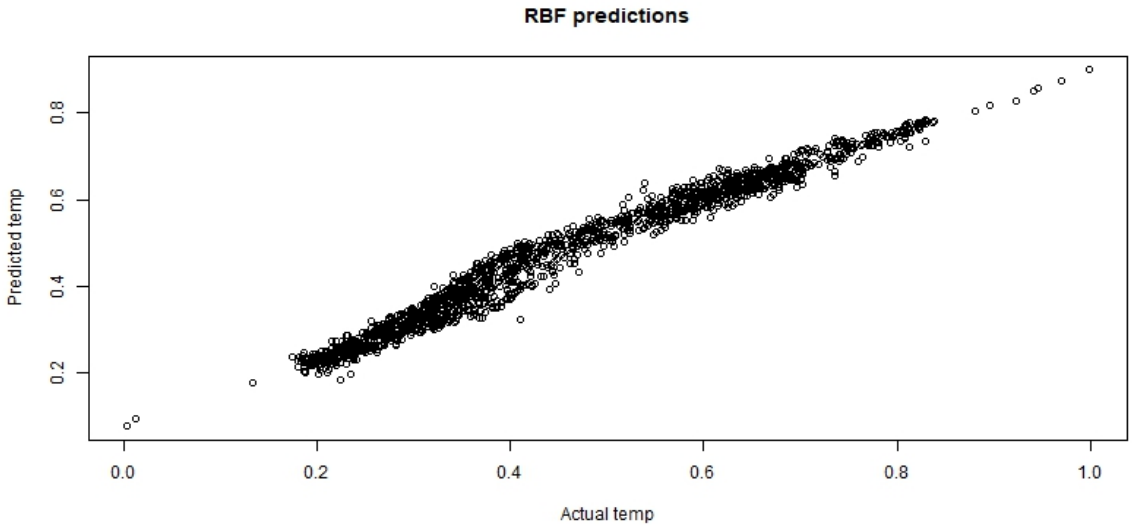


Fig 4.2.1.2: Predicted vs. Actual

4.3 Regression with GRNN

GRNN is a one pass learning and self-growing neural network, which performs linear regression by fitting the probability distribution using Parzen Window approach. It has an input layer and three computational layers: the first one being the Pattern Layer, which contains all the training predictor variables and calculates the distance between the input variables and Gaussian activation function. The next layer is a summation layer with 2 nodes, which adds up all the weighted and unweighted outputs of activation function. This is connected to the output layer with single output node which divides the input fed to it.

4.3.1. Hyperparameter Tuning:

The radial Gaussian activation function with sigma value in pattern units is changed to get the optimal value. Caret package in R with a grid of various sigma values is used to tune the model. The below is the graph of the variance of error w.r.t changing sigma values. It's been observed that sigma of 0.25 yielded the least error. The model is built with this tuned parameter.

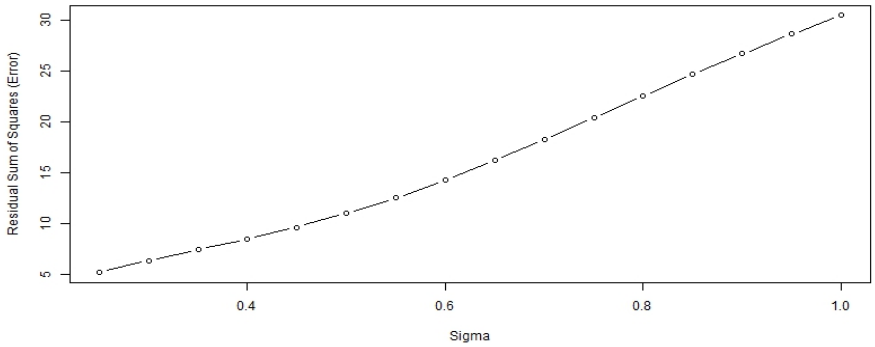


Fig 4.3.1.1: Tuning GRNN

The model using this optimal sigma value is now taken for prediction. Below is the actual versus predicted temperature values plot for the GRNN model.

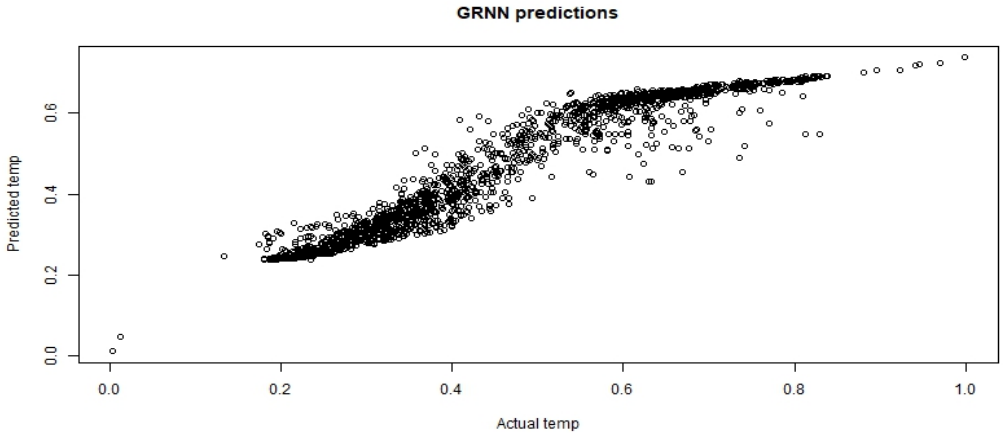


Fig 4.3.1.2: Predicted vs. Actual

4.4. Improving model performance using Ensemble

The ensemble is a statistical modeling strategy to combine the models generated on the data which can perform better than each individual model. The ensemble technique used in this data set is discussed in detail in the following section.

4.4.1. Ensemble using average:

The ensemble using average is created by averaging the prediction of a set of expert models. Generally, a set of expert models with varying parameters such as initial synaptic weights are created to improve the performance. The risk of overfitting can be reduced significantly with this ensemble. The marginal improvement in the results observed is shown as below:

rmse_grnn	0.0540946967533819
rmse_nn	0.0101108379491473
rmse_rbf	0.0355517792313695
rmse_ensemble_avg	0.0295484423137597

Fig 4.4.1.1: RMSE Comparison of each Model

After comparing the RMSE values of the existing models with the given ensemble, it didn't give a significant improvement over all the three models. Hence, the model is omitted from further fine tuning and a more robust ensemble approach is taken into consideration.

4.4.2. Ensemble using Gradient Boosting Model

A basic model is fit and the subsequent models learn on the error/residuals. We have created a stacking model with Gradient Boosting as the top layer and the 3 base models belong to the bottom layer. It's a sequential learning in which a model is fit and the subsequent model learns on the gradient of the loss function.

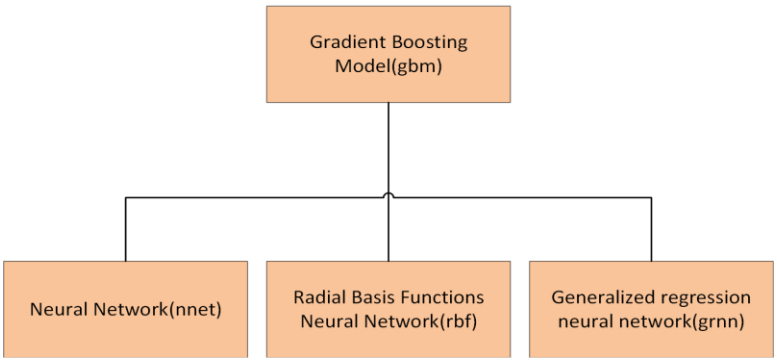


Fig 4.4.2.1: Ensemble Architecture

4.4.2.1. Hyperparameter Tuning

The GBM model is trained using the training predictions of the three base models. So, the prediction of three models will be the input for the ensemble model. Here we are allowing the system to tune the algorithm automatically by setting the tuneLength value, which indicates the number of different values to try for each algorithm parameter. By setting tuneLength = 3, the final values selected for the model were n.trees = 150, interaction.depth = 3, shrinkage = 0.1 and n.minobsinnode = 10. The RMSE observed is **0.00646** which is least of all models.

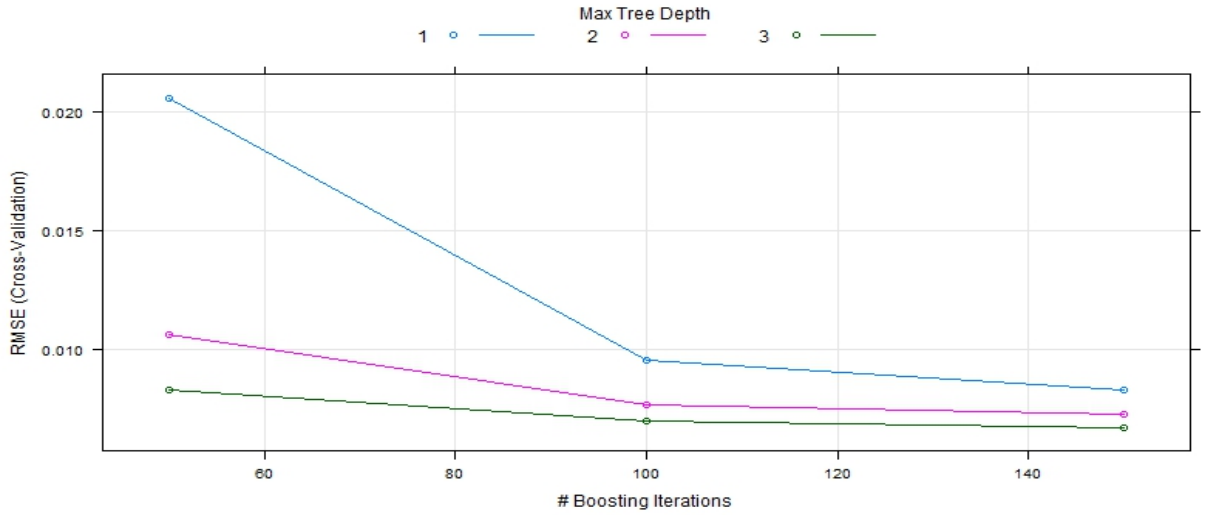


Fig 4.4.2.1.1.: Tuning the Ensemble

The below is the plot for actual versus predicted temperature values.

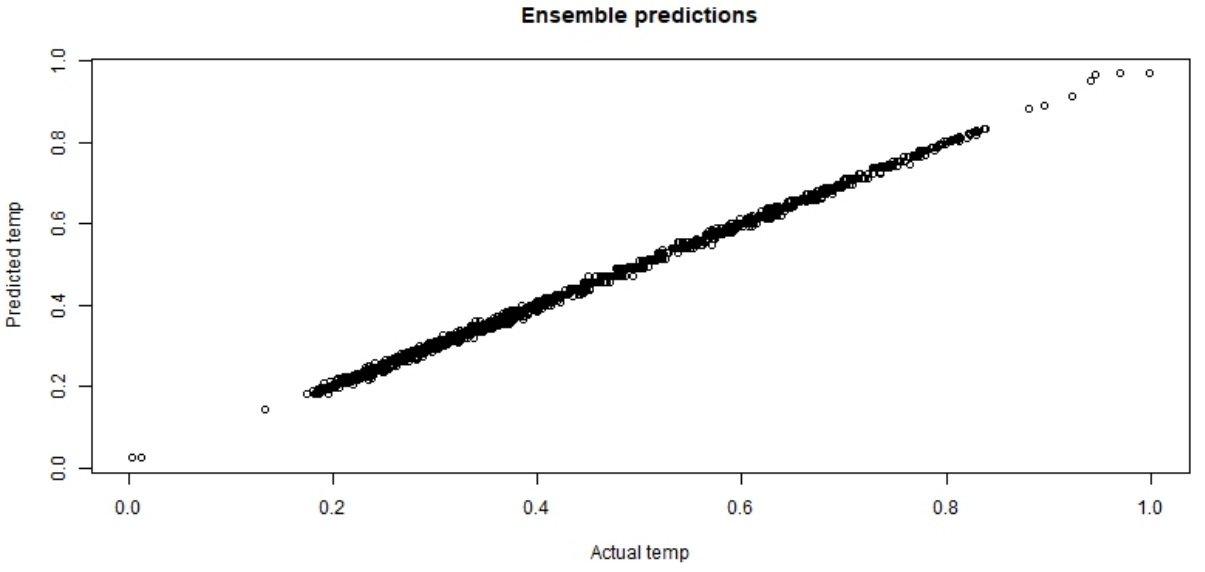


Fig 4.4.2.1.2: Predicted vs. Actual

The ensemble has given the least error rate with a rmse of **0.00646** and the model is cross validated with 10- fold cross validation.

rmse_ensemble_avg	0.0295484423137597
rmse_ensemble_gbm	0.00646095682706123
rmse_grnn	0.0540946967533819
rmse_nn	0.0101108379491473
rmse_rbf	0.0355517792313695

Fig 4.4.2.1.3: RMSE Value Comparison

5. CONCLUSION

The data has been checked for inconsistent values and missing values have been imputed using Hmisc package that fills missing values using Predictive Mean Matching. Feature Engineering is performed on the cleaned data and 15 important features have been selected that are crucial for predicting the Temperature i.e. the Target Variable. Then Radial Basis Neural Network, Generalized Neural network along with Back Propagation Neural Network are built and their hyper parameters are tuned for minimizing the errors of prediction. We have come up with two ensemble models one is Averaging the prediction and a Gradient Boosting Model stacked on top of base model's predictions. It has been observed that Ensemble Gradient boosting Model which learns on the predictions of the base models outperforms other models, as it learns from the error of the base models.

REFERENCE

[1]https://www.analyticsvidhya.com/blog/2017/02/introduction-to-ensembling-along-with-implementation-in-r/

[2]http://www.calcofi.org/new.data/index.php/reporteddata/2013-09-30-23-23-27/database-tables-description

PART - B: CLASSIFICATION

1. INTRODUCTION

1.1. Executive Summary

Fashion MNIST is a dataset created by researchers at Zalando, in the hopes of replacing the most commonly used MNIST dataset. Fashion MNIST dataset emerged to be a potential one by replacing its parent MNIST dataset as it's not as overused and easy as MNIST. It is a challenging dataset that has always got an extra scope of improving in terms of tuning the hyperparameters, generalization. Apparently, this dataset is computationally very light and can be executed in CPU in reasonable amount of time. Having that as an advantage, it sets a benchmark among many other datasets which are considered for future enhancements. The main objective here is to classify images in Fashion MNIST dataset and determine to which among the 10 classes an input image belongs to. A detailed description of the dataset is provided in the next section.

2. DATA UNDERSTANDING

Fashion MNIST is an image dataset of various articles of clothing and accessories such as shirts, trousers, coats, sandals, and other fashion items. It consists of 70,000 images in total, out of which 60,000 images belong to the training set and 10,000 images belong to the test set. Each example is a grayscale image with a height of 28 pixels and width of 28 pixels. All images are associated with a label from 10 classes as mentioned below.

Labels	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Table 2.1: Attribute Description

The data is in the form of ubyte files which can be converted to csv and viewed as rows and columns, where each row represents an image, column 1 represents the class label and remaining columns are pixel numbers. Since the images are of size 28×28, the number of columns will be 784. For our classification purpose, we just load the ubyte files and open it in python using a function called 'with open ()'.

3. DATA PREPROCESSING

Data preprocessing is crucial to any problem on hand and this must be done right for image data as well since this serves in building any effective neural network during modeling phase. Images are pre-processed before feeding to the network to have quicker convergence in terms of speed and to promote stability. Here are some of the techniques employed in pre-processing of both training and validation images.

3.1.1. Reshaping

As for any image processing is concerned, the training and validation images are first converted into NumPy arrays. The arrays are then reshaped before its fed to the original network. The dimensions of the input data would be of the form [60,000, 28, 28,1] indicating the number of training images, image width, image height and the depth. A depth value of 1 represents Grayscale channel. The same process has been repeated for the validation set by altering only the first parameter to the number of images from validation test. The same input shape is preserved for all the models. Resizing images as such is not needed in this case as they are already of the optimal size for the model building.

3.1.2. Augmentation

Data augmentation in general is performed to introduce variety in information to the model for learning. The training set consisting of 60,000 images in total with approximately 6000 images for each class and validation set consisting of 10,000 images with approximately 1000 images per class covers most of the possible variations that can be done to augment the data. Hence no augmentation techniques like shearing, rotation is needed further to avoid repetitions.

4. DATA MODELING

Once the raw data is processed and is transformed to the state wherein it can be used for further processing, models are constructed. With the aim of proving that ensemble approach is an efficient one, three different models have been trained and they are combined to get a final ensemble model. To bring in uniformness, each of the model with batch size as 32.

4.1. Transfer Learning Model

First and foremost, model is built using the concept of 'Transfer Learning'. Supposedly, transfer learning is not a machine learning model by itself. In neural network context, transfer learning is using a set of weights trained on one model for a relevant task. The newly trained model then makes use of these weights while training. Since Fashion MNIST dataset is like MNIST dataset and the image size being 28*28 for both, the weights from MNIST dataset is transferred to Fashion MNIST dataset. Other specifications include the following:

Architecture for MNIST

Initially, the MNIST dataset is trained using two hidden layers of CNN, each having the following set of functions defined for learning.

Layer 1

Convolutional Layer 1

A sequential model with a stack of layers are implemented. First, a Conv2D layer having the number of filters set to 64, a kernel of size (3,3) with the processed input shape is added to give out the activation maps. The activation function used here is 'ReLu' which is non-linear by nature.

Pooling Layer 1

A pool layer with pool size set to (2,2) is added having the information from the previous layer. A dropout of 0.25 is introduced in this to avoid overfitting.

Layer 2

Convolutional Layer 2

A sequential model with a stack of layers are implemented. First, a Conv2D layer having the number of filters set to 32, a kernel of size (3,3) with the processed input shape is added to give out the activation maps. The activation function used here is 'ReLu' too.

Pooling Layer 2

Another pool layer with pool size set to (2,2) is added having the information from previous layer. A dropout of 0.25 is introduced here also to avoid overfitting.

Dense Layers

Furthermore, a layer to flatten the input and a dense layer to implement the output operation is used along with the information from the previous layers. The dense layer is rather described as a fully connected layer which performs the classification based on the features extracted from the Convolutional layers and downsampled using pooling layers. This dense layer uses 'softmax' as the activation function to perform multiclass classification by assigning a determined probability in the range 0-1 for each node.

Model Compilation for MNIST

To compile the model, two key arguments are needed. The following are the ones with respect to the MNIST model.

Optimizers

RMSProp with a learning rate of 0.0001 and a decay rate of 1e-6 is used as the optimizer.

Loss Function

The loss function 'categorical_crossentropy' is used here since the targets are categorical in nature representing 10 classes.

Model Training for MNIST

Once the model is compiled, the next step is to train the model multiple times which is usually the number of epochs and that is set to 10 here. The compiled model is now trained using the NumPy arrays along with class labels. Once the training is completed, the corresponding weights are saved for transfer learning.

Once the MNIST dataset training is completed, model is developed for Fashion MNIST dataset using the weights obtained. The weights are transferred from MNIST to Fashion MNIST layer by layer and the same is achieved by specifying the parameter by_name=TRUE, while loading the weights. One important thing to be noted is that the architecture of this transfer learning model should be the same as the architecture of the MNIST model.

Architecture for Fashion MNIST

Fashion MNIST dataset is trained using two hidden layers of CNN, each having the following set of functions defined for learning.

Layer 1

Convolutional Layer 1

A sequential model with a stack of layers are implemented. First, a Conv2D layer having the number of filters set to 64, a kernel of size (3,3) with the processed input shape is added to give out the activation maps. The activation function used here is 'ReLu' which is non-linear by nature.

Pooling Layer 1

A pool layer with pool size set to (2,2) is added having the information from the previous layer. A dropout of 0.25 is introduced in this to avoid overfitting.

Layer 2

Convolutional Layer 2

A sequential model with a stack of layers are implemented. First, a Conv2D layer having the number of filters set to 32, a kernel of size (3,3) with the processed input shape is added to give out the activation maps. The activation function used here is 'ReLu' too.

Pooling Layer 2

Another pool layer with pool size set to (2,2) is added having the information from previous layer. A dropout of 0.25 is introduced here also to avoid overfitting.

Dense Layers

Furthermore, a layer to flatten the input and a dense layer to implement the output operation is used along with the information from the previous layers. The dense layer is rather described as a fully connected layer which performs the classification based on the features extracted from the Convolutional layers and down sampled using pooling layers. This dense layer uses 'softmax' as the activation function to perform multiclass classification by assigning a determined probability in the range 0-1 for each node.

Model Compilation for Transfer Learning Model

To compile the model, two key arguments are needed. The following are the ones with respect to the Transfer Learning model.

Optimizers

RMSProp with a learning rate 0.0001 and a decay rate of 1e-6 is used as the optimizer.

Loss Function

The loss function 'categorical_crossentropy' is used here since the targets are categorical in nature representing 10 classes.

Model Training for Fashion MNIST

Once the model is compiled, the next step is to train the model multiple times which is usually the number of epochs and that is set to 10 here. The compiled model is now trained using the NumPy arrays along with class labels and the validation set to perform validation at the end of each epoch. Once the training is completed, the corresponding weights are saved for future purpose.

Model Prediction & Evaluation

Now a prediction is run on the validation test data using the trained model obtained from transfer learning. The accuracies are saved and the evaluation scores are computed. Various metrics like accuracy & loss plots, ROC curves, confusion matrix are obtained to validate the trained model which resulted as part of transfer learning. Below are the plots obtained for this model.

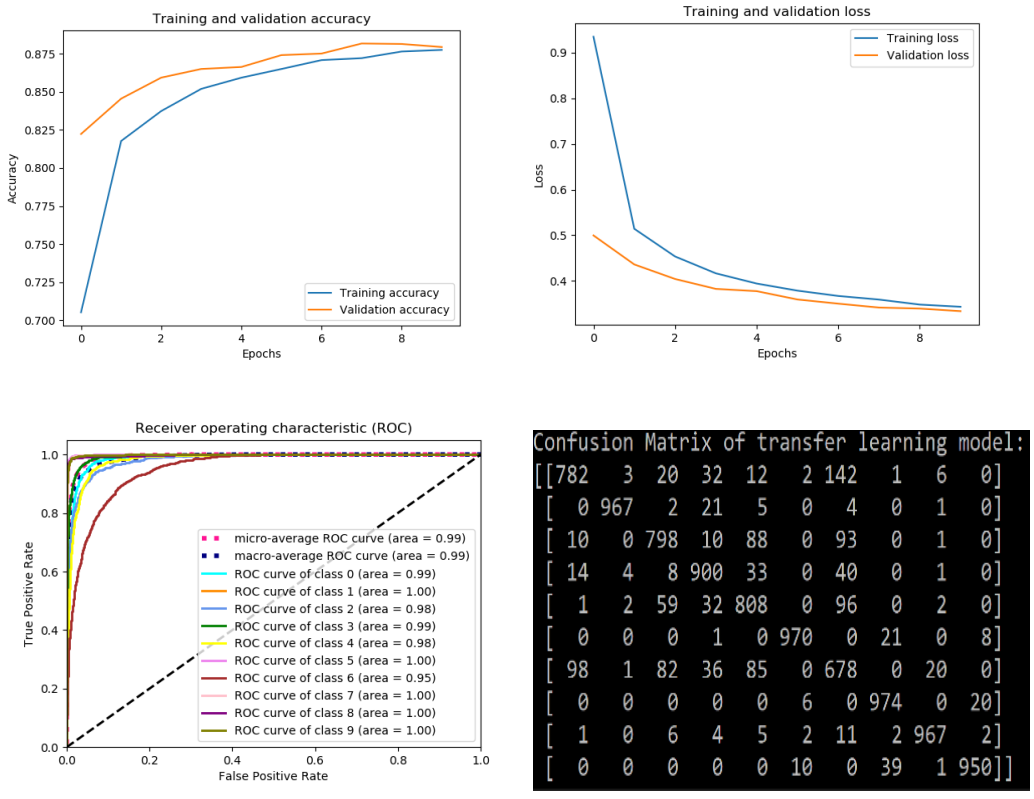


Fig 4.1: Various Performance Metrics

4.2. Simple Convolutional Neural Network

The second model picked to train Fashion MNIST dataset is quite a straightforward Convolutional neural network. This model is again packed with Convolutional layers and dense layers.

Layers Architecture

In this CNN, one hidden layer is defined to process the 28*28 grayscale training images as is:

Layer 1

Convolutional Layer 1

A sequential model with a stack of layers is again implemented. Initially, a Conv2D layer having the number of filters set to 64, a kernel of size (3,3) with the processed input shape is added to give out the activation maps. The activation function used here is tanh. It's a better version of logistic sigmoid and this is also used for classification problem.

Pooling Layer 1

A pool layer with pool size set to (3,3) is added having the information from previous layer. The same dropout of 0.25 is introduced here to avoid overfitting.

Dense Layers

Furthermore, a layer to flatten the input and a dense layer to implement the output operation is used along with the data from the previous layers. The dense layer here is again described as a fully connected layer which performs the classification based on the features extracted from the Convolutional layer and down sampled using pooling layer. This dense layer uses 'softmax' as the activation function to perform multiclass classification.

Model Compilation

To compile the model, two key arguments are required. The following are the ones with respect to the simple CNN model.

Optimizers

RMSProp with a learning rate 0.0001 and a decay rate of 1e-6 is used here as the optimizer.

Loss Function

The same loss function 'categorical_crossentropy' is used here since the targets are categorical in nature representing 10 classes.

Model Training for Fashion MNIST

Once the model is compiled, the next step is to train the model multiple times which is usually the number of epochs and that is set to 6 here. The compiled model is now trained using the NumPy arrays along with class labels and the validation set to perform validation at the end of each epoch. Once the training is completed, the corresponding weights are saved for further processing.

Model Prediction & Evaluation

A prediction is run on the validation test data using the trained model obtained from this simple CNN. The accuracies are saved and the evaluation scores are computed. Various metrics like accuracy & loss plots, ROC curves, confusion matrix are obtained to validate the trained CNN model. Below are the plots obtained for this model.

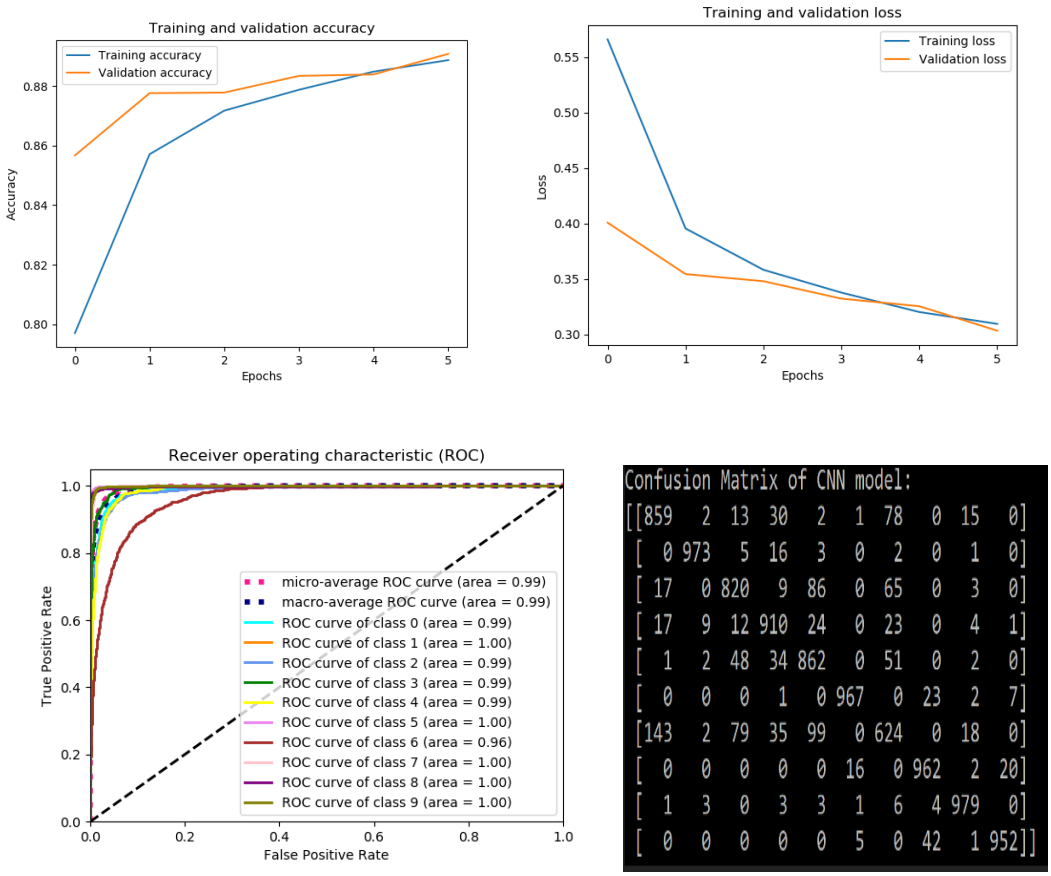


Fig 4.2: Various Performance Metrics

4.3. Long Short Term-Memory RNN

The third model chosen to train Fashion MNIST dataset is LSTM recurrent neural network.

Layers Architecture

In the LSTM approach there is no concept of Convolutional layers, pooling layers. The hidden layer is only responsible to replace the operations used in CNNs and make the difference. Instead of neurons, LSTM has got blocks of memory connected through the layers.

Layer 1

A layer is introduced to embed the dimensions. To process these training images, the row and column dimensions is specified as 128*128. Then data in rows is first encoded using a time distributed wrapper and finally the columns data is encoded by passing the encoded rows.

Dense Layers

As far any NN, there is one final dense layer that uses ‘softmax’ as the activation function to perform multiclass classification with the encoded columns and number of classes as the main arguments.

Model Compilation

To compile the model, two key arguments are required. The following are the ones with respect to the LSTM model.

Optimizers

RMSProp with a learning rate 0.0001 and a decay rate of 1e-6 is used here as the optimizer.

Loss Function

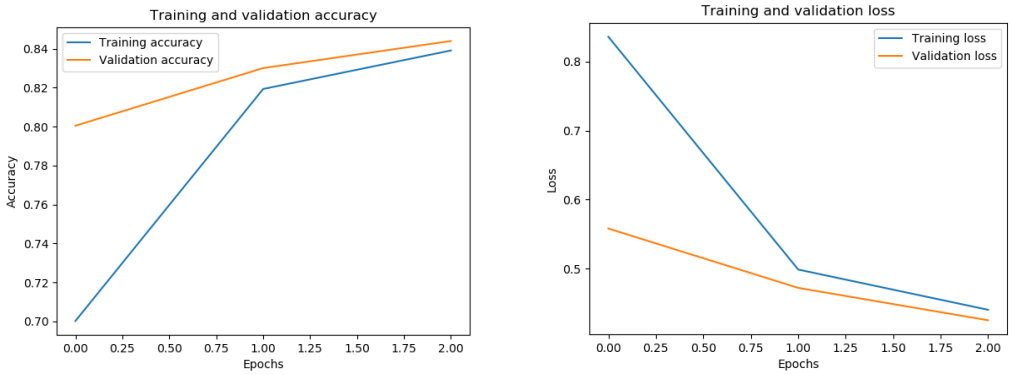
The same loss function ‘categorical_crossentropy’ is used here since the targets are categorical in nature representing 10 classes.

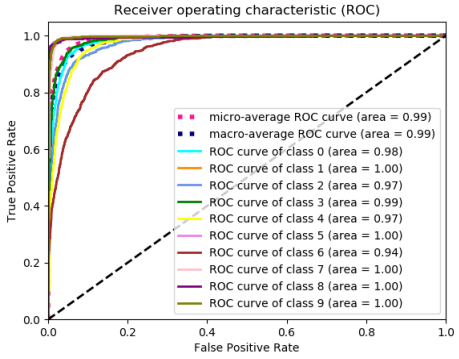
Model Training

Once the model is compiled, the next step is to train the model multiple times which is usually the number of epochs and that is set to 3 here. The compiled model is now trained using the NumPy arrays along with class labels and the validation set to perform validation at the end of each epoch. Once the training is completed, the corresponding weights are saved for further processing.

Model Prediction & Evaluation

A prediction is run on the validation test data using the trained model obtained from LSTM RNN. The accuracies are saved and the evaluation scores are computed. Various metrics like accuracy & loss plots, ROC curves, confusion matrix are obtained to validate the LSTM RNN model. Below are the plots obtained for this model.






```
Confusion Matrix of LSTM model:
[[802   6   5  75   7   1  91   0  11   2]
 [  3 949   4  32   7   0   4   0   1   0]
 [ 18   2 707  16 154   0  98   0   5   0]
 [ 17  19   4 872  46   0  40   0   2   0]
 [  0   0  87  51 749   1 108   0   4   0]
 [  0   1   0   0   0 929   0  58   0  12]
 [161   5  98  51  73   2 586   0  24   0]
 [  0   0   0   0   0  14   0 942   0  44]
 [  5   1   2   1   3   3  12   4 963   6]
 [  0   0   0   1   0   9   1  48   1 940]]
```

Fig 4.3: Various Performance Metrics

5. ENSEMBLE APPROACH

Once the models are trained and saved, the next step is to form an ensemble model to improve the overall accuracy and to have a robust model for classifying the images. The predictions from the transfer learning model, simple CNN model and LSTM RNN models are extracted and combined to form an ensemble. The weights saved at each stage are now reused to form the ensemble.

Layers Architecture

First the trained weights are loaded. The same input layer is shared between all the three models. The main change comes in the top layer. Here is where the averaging of three models' outputs are computed. The Average () merge layer computes the average of three models which is used to boost the performance of the ensemble.

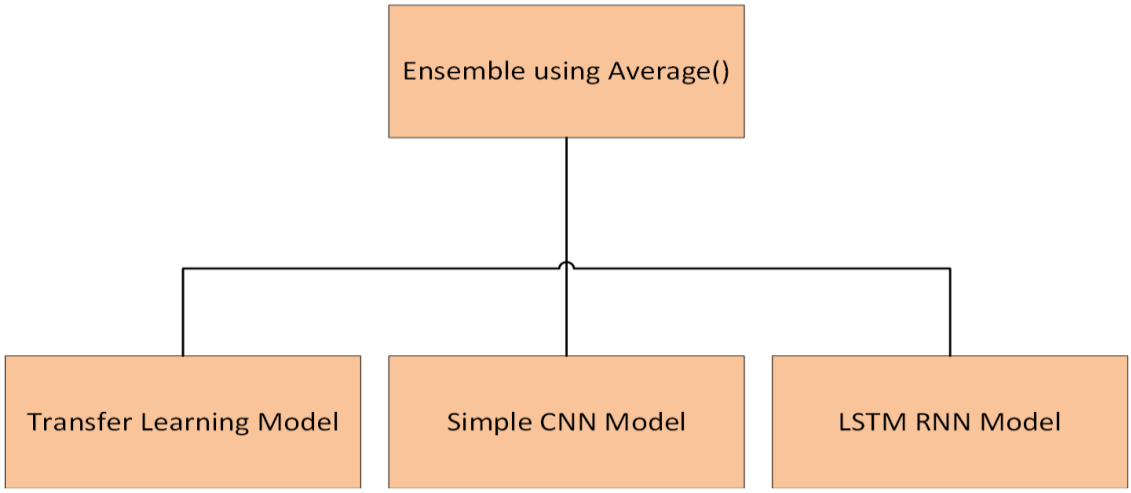


Fig 5.1.1: Architecture of Ensemble

Model Compilation

To compile this ensemble model, two key arguments are again required.

Optimizers

RMSProp with a learning rate 0.0001 and a decay rate of 1e-6 is used here as the optimizer.

Loss Function

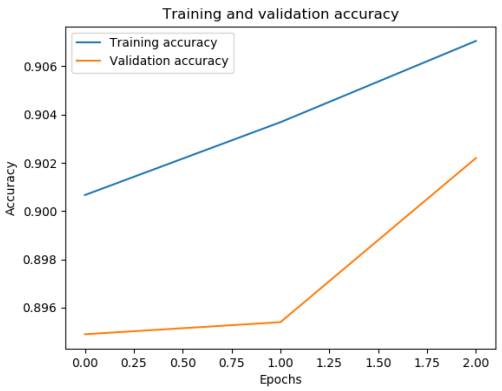
The same loss function 'categorical_crossentropy' is used here since the targets are categorical in nature representing 10 classes.

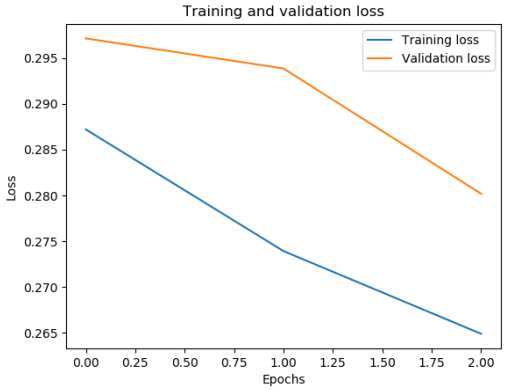
Model Training

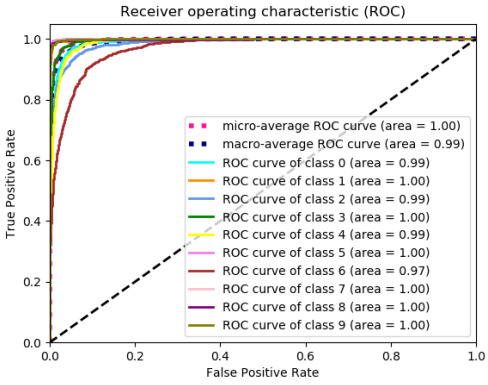
Once the model is compiled, the next step is to train the model multiple times which is usually the number of epochs and that is set to 3 here. The compiled model is now trained using the NumPy arrays along with class labels and the validation set to perform validation at the end of each epoch. As soon as the training is done, the corresponding weights are saved.

Model Prediction & Evaluation

Finally, a prediction is run on the validation data using the ensemble model obtained from CNN, transfer learning and LSTM RNN. The accuracies are saved and the evaluation scores are computed. Various metrics like accuracy & loss plots, ROC curves, confusion matrix are obtained to validate the ensemble model. Below are the plots obtained for this model.








```
Confusion Matrix of ensemble model:
[[871   1   9  16   3   1  93   0   6   0]
 [  1 974   1  17   3   0   3   0   1   0]
 [ 14   0 823  12  67   0  83   0   1   0]
 [ 21   3   8 922  19   0  25   0   2   0]
 [  1   1  57  37 818   0  85   0   1   0]
 [  0   0   0   1   0 981   0  14   0   4]
 [115   0  59  28  57   0 726   0  15   0]
 [  0   0   0   0   0   8   0 972   0  20]
 [  2   1   1   2   2   2   8   2 979   1]
 [  0   0   0   0   0   7   0  36   1 956]]
```

Fig 5.1: Various Performance Metrics

6. CONCLUSION

On comparing the individual models and the ensemble model in the aspects of metrics like validation accuracy, ROC plots, loss incurred and confusion matrix, it is much evident that the ensemble approach has topped.

```
custom_fashion_mnist 0.8794
own_cnn_fashion_mnist 0.8908
LSTM_fashion_mnist 0.8439
ensemble_fashion_mnist 0.9022
```

Fig 6.1: Various Performance Metrics

As a reference to the accuracies, ensemble model has achieved a high of 90% in this case outperforming all the other individual neural network architectures. This in turn gets aligned well with the objective considered.

REFERENCES

[1] https://towardsdatascience.com/ensembling-convnets-using-keras-237d429157eb

[2]https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/