Machine Learning for Data Science

M. H. Rahman [1]

[1]Associate Professor
Department of Statistics and Data Science
Jahangirnagar University
Bangladesh
E-mail: habib.drj@juniv.edu

Twenty Fifty

Book References

Texts

1. DeLurgio, S. A. (1998). Forecasting principles and applications. McGraw-Hill/Irwin.
2. Dunham, M. H. (2003): Data Mining: Introductory and Advanced Topics, 1st edition, Pearson.
3. Han J., Kamber M., Pei J (2011): Data Mining: Concepts and Techniques, 3rd edition, Elsevier.

References

1. Larose, D. T. (2006): Data Mining: Methods and Models, Wiley-Interscience, India.
2. Schalkoff, R. (2005): Pattern Recognition Statistical, Structural and Neural Approaches, John Wiley and Sons, New York.

Machine Learning

Artificial Neural Network

Neural Network

Artificial Neural Network

Neural networks (NN) often referred to as Artificial Neural Network (ANN) to distinguish from the Biological Neural Networks. Artificial Neural Network (ANN) is a computer intensive, algorithmic procedure for transforming inputs into desired outputs using highly connected networks of relatively simple processing units (neurons or nodes). Neural networks are modeled after the neural activity of human brain.

The three essential features of ANN are -

- ✓ the basic computing units (neurons or nodes)
- ✓ the network architecture describing the connections between the computing units
- ✓ the training algorithm used to find the value of network parameters (weights) for performing a particular task.

Neural Network

Artificial Neural Network ..

The computing units are connected to one another in the sense that the output from one units can serve as part of the input to another unit. Each computing units transforms an input to an output using some prespecified function.

In statistical applications, the computing units are arranged in a series of layers with connections between nodes in different layers, but not between nodes in the same layer.

The layer receiving the initial inputs is called the input layer. The final layer is called the output layer. Any layers between the input and output layers are called hidden layers.

Neural Network

A simple schematic representation of multi-layer ANN is shown in the following figure.

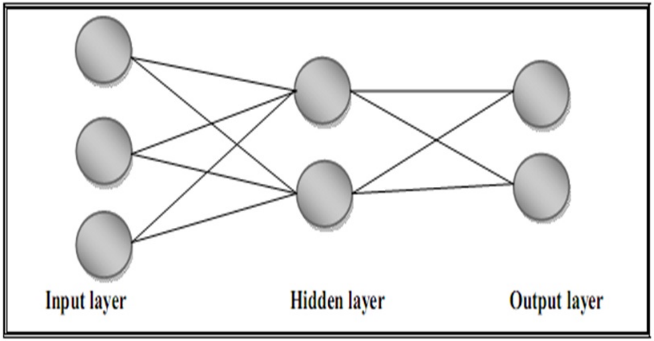


Figure: An ANN with one Hidden Layer

Neural Network

ANN Applications

The several different types of application of Artificial Neural Networks include the following-

- ✓ **Classification problems** consist of credit approval processes (i.e., acceptance or rejection) and defect identification (i.e., good or defective).
- ✓ **Mathematical modeling** occurs when the ANN is used to identify or emulate a mathematical function.
- ✓ **Forecasting and prediction** includes univariate or multivariate forecasting application.

Neural Network

ANN Applications..

- ✓ **Reconstruction or Recognition** involves identifying a pattern in noisy or unclear signal. For example, character recognition applications are indicative of reconstruction. The character recognition methods in computer scanners and fax board use ANNs.
- ✓ **Clustering** involves finding logical groupings of values in data- for example in marketing, when trying to group types of consumer and their responses to different types of promotions.
- ✓ **Routing** consists of determining the best route through a number of destinations, the traveling salesman problem being an example.

As mentioned, the ANN approach that is most relevant to forecasting is a back propagation method.

Neural Network

Activation Function

The output of each node in ANN is based on a function is known as Activation Function. An Activation Function is sometimes called a processing element function or squashing function. Activation Function may be unipolar, with values in [0, 1] or bipolar, with values in [-1, 1].

Linear: $f(x) = cs$, Here c is a constant positive values.

Threshold or Step: $f(x) = \begin{cases} 1 & if \;\; x > T \\ 0 & otherwise \end{cases}$

Ramp Function: $f(x) = \begin{cases} -1 & ; \; x \leq -1 \\ 0 & ; \; -1 < x < +1 \\ +1 & ; \; x \geq +1 \end{cases}$

Sigmoid Function: $f(x) = \dfrac{1}{1+e^{-x}}$

Hyperbolic Tangent: $f(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$, **Gaussian:** $f(x) = e^{-x^2}$

Neural Network

Steps in Developing an ANN

We discuss the development of ANN using following Figure (Considering one output). We present the 16 steps in the development of ANNs. These 16 steps are part of the following four larger activities, which are very smaller to those of other foresting methods:

Identification. Design the ANN by selecting the input and output variables, its architecture and validation experiment, while trying not to violate the principal of parsimony.

Estimation. Train the ANN to minimize the RSE or RMS paying attention to whether to network is too simplistic or too complex.

Diagnosis. Test the ANN on out of sample data and compare the out of sample RSE or RMS to that of the training data.

Forecasting. Use the ANN to forecast, always monitoring its goodness of fit.

Neural Network

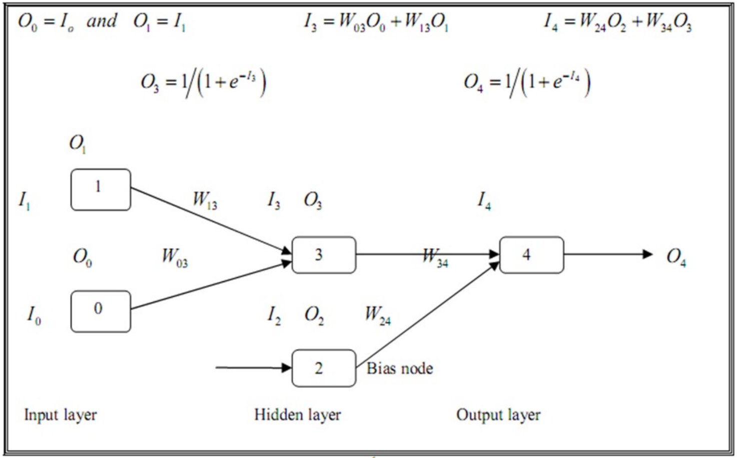


Figure: An ANN 2x1x1

Neural Network

The following is a brief introduction to each step of training and validating an ANN.

1. **Determining the structure of ANN** based on some underlying theory about what influences the dependent variable. In this case, the structure is given in figure. In general, this involves choosing input variables, the number of input nodes, the number of hidden layers and nodes, the transfer function type and the number of output nodes.
2. **Divide the input data and output data into two groups,** the first to be used to train (i. e. fit) the network, the second to be used to validate the network in an out of sample experiment or forecast.
3. **Scale the input variables** and the desired output variable to the range 0 to 1. Because of its mathematical structure, it is necessary to rescale the values of the variables in an ANN to between 0 to 1.

$$Scale\ Value = \frac{Actual - Minimum}{Maximum - Minimum}$$

Neural Network

4. **Set initial weights and start a training epoach** using the training data set by repeating steps (5) to (13): An epoach is calculation of errors and the adjustment of weights by processing all observations in the training set. Also in this step, the initial values are given to all of the weights in the ANN. These initial values can influences the speed and RMS that result from the training set. Most program allows weights be initialized with all zeros or random numbers from -1 to 1.
5. **Input scale variable.** The scaled inputs of an observation are received at the input nodes 0 to 1 of above Figure.
6. **Distribute the scale input** to each hidden node. Each hidden node receives all scaled input variables, which result in a parallel processing of all inputs at multiple nodes. The output O_j of an input node equals to its input I_j .

Neural Network

7. **Weight and Sum inputs to receiving nodes.** At each hidden node (i.e., nod 3), the outputs of the inputs node (i.e. node 0 and 1) are weighted and summed. Thus the input to node 3 is: $I_3 = W_{03}O_0 + W_{13}O_1$.
8. **Transform hidden-node inputs to outputs.** At each hidden node, the weighted inputs are transformed into an output in the range 0 to 1. That is, $O_3 = \dfrac{1}{1 + e^{-I_3}}$
9. **Weight and sum hidden node outputs as inputs to the output nodes.** The output of the hidden node O_3 and O_4 are weighted and summed. That is, $I_4 = W_{24}O_2 + W_{34}O_3$
10. **Transform inputs at the output nodes.** At the output node 4, the weighted input I_4 is transformed into the output O_4 in the range 0 to 1. This is the final output of the ANN. That is, $O_4 = \dfrac{1}{1 + e^{-I_4}}$

Neural Network

11. **Calculate Output Errors**. The scaled output value O_4 is compared to the scaled desired output D_4 and error is calculated:$e_i = D_4 - O_4$ where i is the observation in the training set.
12. **Backpropagate errors to adjust weights**. Based on the errors of step (11), the weights throughout the network are modified so as to move toward minimization of the RMS value.
13. **Continue the epoch**. Repeat step (5) to (12) for all observations in the input data set. Each pass through all observations is called an epoch.

Neural Network

14. **Calculate the epoch RMS** value of the errors. If RMS is low enough, stop and go to step (15). If RMS is not low enough, repeat steps (5) to (15) until the stopping condition is reached.
$$RMS = \sqrt{\frac{\sum e_i^2}{nt}}$$
where, $RMS = Residual \ \ Standard \ \ Error$
$e_i = Errors \ \ during \ \ each \ \ observation \ \ in \ \ latest \ \ epoach$
$nt = Number \ \ of \ \ observation \ \ in \ \ training \ \ set$
15. **Judge out-of-sample validity**. Having trained the ANN using one set of input data, now it should be validated using out-of-sample data.
16. **Use the model in forecasting**. Using the model in an actual forecasting setting, being continuous to monitor it using tracing signals and other devices.

ANN

Referece

DeLurgio, S. A. (1998). Forecasting principles and applications (Vol. 49). New York: Irwin/McGraw-Hill.

ANN

Backpropagation and Training ANNs
Just as an estimation of regression coefficients using ordinary least squares, the weights in an ANN must be estimated. While there are several methods of training a network, the most successful and widely used is the feedforward, backpropagation method.

The method of propagating information about errors at the output layers back to hidden layers was discovered in the mid-1970s, but this method was either unknown or ignored. It was rediscovered in the 1980s and the method termed *backpropagation* was refined and publicized by Devid Rumelhart and James McClelland (1986).

ANN

Backpropagation
If the neurode output functions are differentiable, then the adjustment of weights has a simple solution- adjust them in proportion to the negative of the change in the sum of squared errors with respect to the change in weights because of the behavior of the slope of the SSE function.

That is,

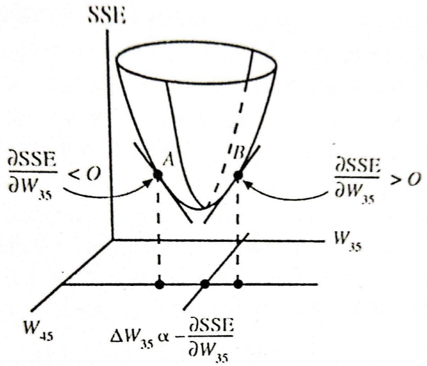
$$W_{ij}(new) = W_{ij}(old) + \Delta W_{ij}(new)$$

$$\Delta W_{ij} \propto -\frac{\delta SSE}{\delta W_{ij}}$$

ANN

Backpropagation

A hypothetical SSE versus the value of two network weights, W_{35} and W_{45}.


SSE
$\frac{\partial SSE}{\partial W_{35}} < O$
A
B
$\frac{\partial SSE}{\partial W_{35}} > O$
W_{35}
W_{45}
$\Delta W_{35} \propto -\frac{\partial SSE}{\partial W_{35}}$

At point A, increase ΔW_{35}, $-\frac{\partial SSE}{\partial W_{35}} > O$

At point B, decrease ΔW_{35}, $-\frac{\partial SSE}{\partial W_{35}} < O$

ANN

Backpropagation

In actuality, partial differentiation is used to change the weights in proportion to the derivative of the SSE with respect to weights. The training formula used in most software and the spreadsheets enclosed with this text use what is referred to as the **Generalized Delta Rule**.

$$\Delta Wij(new) = \eta\delta_{ij}O_j + \alpha\Delta W_{ij}(old)$$

where,

$\eta =$ Learning Coefficient, which users set between 0 to 1.

$\delta_{ij} =$ Derivatives of the SSEs with respect to weight at a node, sometimes referred to as the node share of the error.

$O_j =$ Output value at node j.

$\alpha =$ Momentum coefficient varying between 0 to 1.

$\Delta W_{ij}(old) =$ Previous changes in W_{ij}.

ANN

Mathematics of a Backpropagation NN

Each noninput node has an output level O_j where:

$$O_j = \frac{1}{1 + e^{-I_j}}$$

$$I_j = \sum_i W_{ij} O_i$$

where, O_i is each of the signals to node j.

The derivation of the backpropagation formula involves the use of the chain rule of partial derivatives and equals:

$$\delta_{ij} = \frac{\delta SSE}{\delta W_{ij}} = \left(\frac{\delta SSE}{\delta O_j}\right)\left(\frac{\delta O_j}{\delta I_j}\right)\left(\frac{\delta I_j}{\delta W_{ij}}\right)$$

ANN

Mathematics of a Backpropagation NN

Now, The error is defined as

$$e_i = (D_j - O_j)$$

$$SSE = \sum_j (D_j - O_j)^2$$

Therefore,

$$\left(\frac{\delta SSE}{\delta O_j}\right) = -2\sum_j (D_j - O_j)$$

The output of an output node is

$$O_j = \frac{1}{1 + e^{-I_j}}$$

$$\left(\frac{\delta O_j}{\delta I_j}\right) = O_j(1 - O_j)$$

ANN

Mathematics of a Backpropagation NN

The input to an output nodes is:

$$I_j = \sum W_{ij} O_i$$

$$\left(\frac{\delta I_j}{\delta W_{ij}} \right) = O_i$$

Therefore, the δ_{ij} becomes

$$\delta_{ij} = -2 e_j O_j (1 - O_j) O_i$$

Now, the old weight is updated by the following equation:

$$\Delta Wij(new) = \eta \delta_{ij} O_j + \alpha \Delta W_{ij}(old)$$

ANN

Mathematics of a Backpropagation NN

For hidden layers, the calculations are very similar. The only change is how the ANN output error is backpropagated to the hidden layer nodes. The output error at the ith hidden node depends on the output errors of all nodes in the output layer. The relationship is,

$$e_i = \sum W_{ij} e_j$$

After calculating the output error for the hidden layer, the update rules for the weights in that layer are the same as the previous update,

$$\delta_{ij} = -2 e_j O_j (1 - O_j) O_i$$

R Code to Fit Neural Network

Problem

Generate X_1 form $U(a = 2, b = 9)$, X_2 from $\chi^2(df = 5)$, X_3 from $\chi^2(df = 2)$, and E from $N(\mu = 3, \sigma = 3)$ of size $N = 100$, then generate $Y = 0.35X_1^3 + 2X_2 + 5X_3 + E$. Fit a neural network model 3x3x2x1 where X_1, X_2, X_3 be the input variables and Y be the target variable. Use your Examination Roll (1111) as a seed point to generate the data, training data, and test data. Find the fitted model, MSE, and other results.

R Code to Fit Neural Network

R Code for ANN

```
rm(list=ls())
library(MASS)
#install.packages('neuralnet')
library(neuralnet)

##==Set a seed====#
set.seed(1111)
N=100
X1=runif(N,2,9)
X2=rchisq(N,5)
X3=rchisq(N,2)
E=rnorm(N,3,3)
Y=0.35*X1^3+2*X2+5*X3+E
```

R Code to Fit Neural Network

R Code for ANN

```
DT=data.frame(cbind(X1,X2,X3,Y))
head(DT)

#==Check that no data is missing ===#
##==HERE 1 for row, and 2 for column ===#
apply(DT,2,function(x) sum(is.na(x)))
apply(DT,1,function(x) sum(is.na(x)))

#==Train and Test Data set random splitting==#

Index=sample(1:nrow(DT), round(0.75*nrow(DT)),
replace=FALSE)
Train=DT[Index,]
Test=DT[-Index,]
```

R Code to Fit Neural Network

R Code for ANN

```
#-------------------------------------#
#==Neural net fitting==#

#===Scaling data for the NN ===#
Maxs = apply(DT, 2, max)
Mins = apply(DT, 2, min)

Scaled = as.data.frame(scale(DT, center = Mins,
                         scale = Maxs - Mins))

#==Scaled Train-Test split ===##
Train_S = Scaled[Index,]
Test_S = Scaled[-Index,]
```

R Code to Fit Neural Network

R Code for ANN

```
#====NN training=====#

n = names(Train_S)
f = as.formula(paste("Y ~", paste(n[!n %in% "Y"],
             collapse = " + ")))

nn = neuralnet(f,data=Train_S,hidden=c(3,3),
                 linear.output=T)

#===Visual plot of the model===#

plot(nn)
```

R Code to Fit Neural Network

R Code for ANN

```
#==Results from NN are Normalized (scaled)===#
#=Predict value for Training and Test and ALL Data =#

Pr.nn_TR=compute(nn,Train_S[,1: (ncol(DT)-1)])
Pr.nn_TS=compute(nn,Test_S[,1: (ncol(DT)-1)])
Pr.nn_AL=compute(nn,Scaled[,1: (ncol(DT)-1)])

#===Descaling for comparison ===#

D_Pr.nn_TR = Pr.nn_TR$net.result*(max(DT$Y)-min(DT$Y))
              +min(DT$Y)
D_Pr.nn_TS = Pr.nn_TS$net.result*(max(DT$Y)-min(DT$Y))
              +min(DT$Y)
D_Pr.nn_AL = Pr.nn_AL$net.result*(max(DT$Y)-min(DT$Y))
              +min(DT$Y)
```

R Code to Fit Neural Network

R Code for ANN

```
#===Calculating MSE and RMS ====##

MSE.nn_TR = sum((D_Pr.nn_TR - Train$Y)^2)/nrow(Train)
MSE.nn_TS = sum((D_Pr.nn_TS - Test$Y)^2)/nrow(Test)
MSE.nn_AL = sum((D_Pr.nn_AL - DT$Y)^2)/nrow(DT)

RMS_TR=sqrt(MSE.nn_TR)
RMS_TS=sqrt(MSE.nn_TS)
RMS_AL=sqrt(MSE.nn_AL)

#==PREDICTION ERROR===#
PE_TR =mean(abs(D_Pr.nn_TR - Train$Y))/mean(Train$Y)
PE_TS =mean(abs(D_Pr.nn_TS - Test$Y))/mean(Test$Y)
PE_AL =mean(abs(D_Pr.nn_AL - DT$Y))/mean(DT$Y)
```

R Code to Fit Neural Network

R Code for ANN

```
jpeg(filename = "PE.jpg",width = 5, height = 5,
             units = "in",res=500)
#dev.new(width=5,height=5)
par(cex=1,mai=c(1.0,1,0.7,0.4))
barplot(c(PE_TR,PE_TS,PE_AL),
names.arg=c("Training", "Test", "Over All"),
main="Plot for Prediction Error",
col="indianred2",
ylab="Prediction Error \n",ylim=c(0,0.035),
col.main="black",col.axis="black",col.lab="black",
cex.main="1",cex.axis="1",cex.lab="1",
font.main="2",font.axis="1",font.lab="1",las=1)
dev.off()
```

R Code to Fit Neural Network

R Code for ANN

```
#==Plot predictions==#  #dev.new(width=10,height=5)
jpeg(filename = "AP.jpg",width = 10, height = 5,
       units = "in",res=500)
par(mfrow=c(1,2))
plot(Train$Y,D_Pr.nn_TR,col='red',
main='Actual vs Predicted',pch=20,cex=1,
xlab="Actual",ylab="Predicted")
abline(0,1,lwd=2)
points(Test$Y,D_Pr.nn_TS,col='blue',
main='Actual vs Predicted',pch=20, cex=1,)
legend('bottomright',legend=c('Train','Test'),
pch=c(20,20), col=c('red','blue'), bty='n', cex=.95)
```

R Code to Fit Neural Network

R Code for ANN

```
plot(DT$Y,D_Pr.nn_AL,col='indianred2',pch=20, cex=1,
main='Actual vs Predicted',xlab="Actual",
ylab="Predicted")
abline(0,1,lwd=2)
legend('bottomright',legend='ALL',pch=18,
col='indianred2', bty='n', cex=.95)
dev.off()
```

R Code to Fit Neural Network

R Code for ANN

```
#-----------------------------------#
#===Cross validating ===#

library(boot)
#===Neural net cross validation===#
set.seed(11111)
cv.error <- NULL
k <- 10

#==Initialize progress bar
#install.packages('plyr')
library(plyr)
pbar <- create_progress_bar('text')
pbar$init(k)
```

R Code to Fit Neural Network

R Code for ANN

```
for(i in 1:k){
index <- sample(1:nrow(DT),round(0.9*nrow(DT)))
train.cv <- Scaled[Index,]
test.cv <- Scaled[-Index,]
nn <- neuralnet(f,data=train.cv,hidden=c(3,3),
      linear.output=T)
pr.nn <- compute(nn,test.cv[,1:(ncol(DT)-1)])
pr.nn <- pr.nn$net.result*(max(DT$Y)-min(DT$Y))
     +min(DT$Y)
test.cv.r <- (test.cv$Y)*(max(DT$Y)-min(DT$Y))
     +min(DT$Y)
cv.error[i]=sum((test.cv.r - pr.nn)^2)/nrow(test.cv)
pbar$step()
}
```

R Code to Fit Neural Network

R Code for ANN

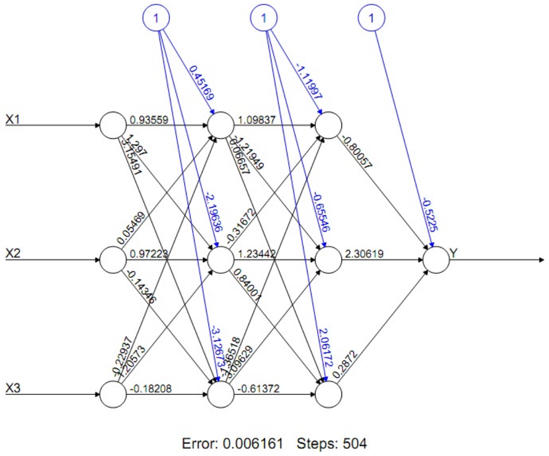
```
# Average MSE
mean(cv.error)

# MSE vector from CV
cv.error

jpeg(filename = "BOXP.jpg",width = 5, height = 5,
    units = "in",res=500)
# Visual plot of CV results
#dev.new(width=5,height=5)
boxplot(cv.error,xlab='MSE',col='cyan',
border='blue',names='MSE',
main='Box-and-Whisker Plot \n for MSE(Test Data)',
      horizontal=TRUE)
dev.off()
```

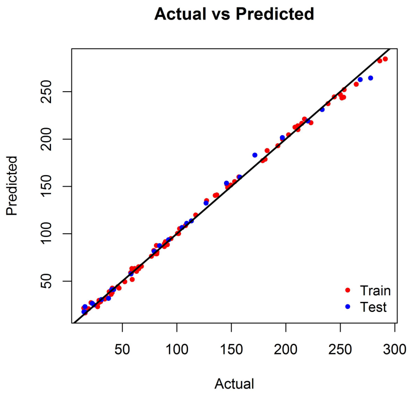
Neural Network

Results of Fitted ANN



Neural Network

Results of Fitted ANN



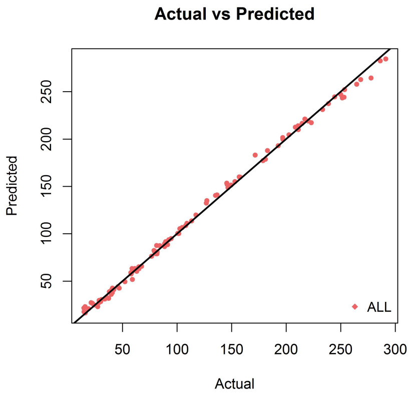


Figure: Actual vs Predicted value for Training, Test, and Over All data

Neural Network

Results of Fitted ANN

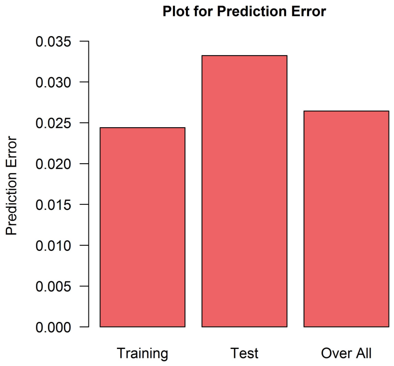


Figure: Prediction Error

Neural Network

Results of Fitted ANN

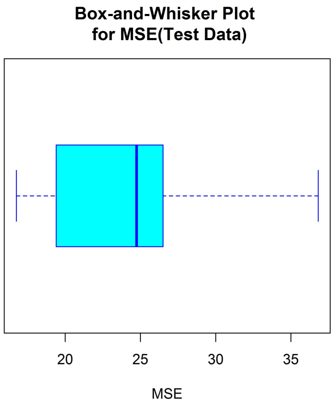
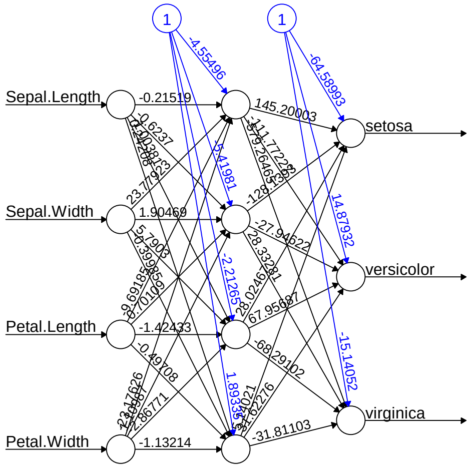


Figure: Box and Whisker Plot for MSE

ANN Model to classify the IRIS Data

```
library(neuralnet)##==ANN Classification for IRIS Data=#
data(iris)
I=iris
set.seed(12357)
ID=sample(1:nrow(I),round(nrow(I)*0.75,0),replace=FALSE)
TR=I[ID,]   # Training Data
TS=I[-ID,]  # Test Data
TRD=TR
TRD=cbind(TRD,TR$Species=="setosa")
TRD=cbind(TRD,TR$Species=="versicolor")
TRD=cbind(TRD,TR$Species=="virginica")
names(TRD)[6]="setosa"; names(TRD)[7]="versicolor"
names(TRD)[8]="virginica"; head(TRD[,5:8])
nn1=neuralnet(setosa+versicolor+virginica~
    Sepal.Length+Sepal.Width+Petal.Length+Petal.Width,
    data=TRD, hidden=c(4),stepmax=1e+50, algorithm="rprop+",
    err.fct="sse",act.fct = "logistic", linear.output =FALSE)
print(nn1); plot(nn1)
PR=compute(nn1, TS[-5]); PR1=PR$net.result
PR_ID=function(x){return(which(x==max(x)))}
IDX=apply(PR1,c(1),PR_ID)
PR_nn=c('setosa','versicolor','virginica')[IDX]
T=table(TS$Species,PR_nn); T
ATS=sum(diag(T))/sum(T);ATS;1-ATS
```

Machine Learning for Data Science



Error: 0.979225   Steps: 5791

Confusion Matrix for ANN Model

		Predicted		
	Species	setosa	versicolor	virginica
	setosa	15	0	0
Actual	versicolor	0	10	1
	virginica	0	1	11

Accuracy rate $= 0.9473684$ (94.7%) and Error rate $= 0.05263158$ (5.3%).

Python Code to Fit Neural Network

Problem

Generate X_1 form $U(a = 2, b = 9)$, X_2 from $\chi^2(df = 5)$, X_3 from $\chi^2(df = 2)$, and E from $N(\mu = 3, \sigma = 3)$ of size $N = 300$, then generate $Y = 0.35X_1^3 + 2X_2 + 5X_3 + E$. Fit a neural network model 3x3x2x1 where X_1, X_2, X_3 are the input variables and Y is the target variable. Use your Examination Roll as a seed point to generate the data, training data, and test data. Find the fitted model, MSE, and other results.

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor

```
###==Multi-layer Perceptron Regressor.##
import scipy
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_absolute_percentage_error
```

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor

```
np.random.seed(104729)
N=300
X1=pd.DataFrame((9-2)*np.random.random(N)+2)
X2=pd.DataFrame(np.random.chisquare(5, N))
X3=pd.DataFrame(np.random.chisquare(2, N))
E=pd.DataFrame(np.random.normal(3,3,N))
Y1=pd.DataFrame(0.35*X1**3+2*X2+5*X3+E)
ID=np.arange(0,N)
D=pd.DataFrame(pd.concat([X1,X2,X3,Y1],axis=1,
               ignore_index=True))
# adding column name to the respective columns
D.columns=['X1', 'X2', 'X3','Y1']
# Check for missing data
#print(D.isna().sum(axis=1))  # for rows
print(D.isna().sum(axis=0))  # for columns
```

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor

```
X=D.drop(['Y1'], axis=1)
Y=D[['Y1']]
np.random.seed(104729)
# Train and Test Data set random splitting
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,
                        test_size=0.25, random_state=1111)

# Scaling data for the NN
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
Scaled = pd.DataFrame(scaler.fit_transform(D), columns=D.colu
print(Scaled.head())
X_S=Scaled.drop(['Y1'], axis=1)
Y_S=Scaled[['Y1']]
```

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor

```
#=====Scaled Train-Test Data===#
X_train_S=X_S.iloc[X_train.index,]
X_test_S=X_S.iloc[X_test.index,]
Y_train_S=Y_S.iloc[Y_train.index,]
Y_test_S=Y_S.iloc[Y_test.index,]

Y_test, Y_test_S
```

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor

```
'''#===Using MLP======#'''
'''###==Multi-layer Perceptron Regressor.##'''

ANN = MLPRegressor(hidden_layer_sizes=(3,2),
    activation='logistic', solver='lbfgs', max_iter=99999999)
ANN.fit(X_train_S, Y_train_S)

ANN.score(X_test_S, Y_test_S)
ANN.intercepts_
ANN.coefs_
```

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor

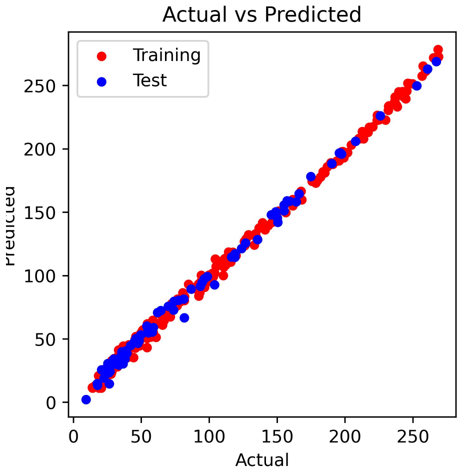


Figure: Actuasl vs Predicted

Python Code to Fit Neural Network

Python Code for Tensorflow (if you want to use)

```
'''#====Using Tensorflow Package====#'''
#!pip install tensorflow
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
# Define and train the Sequential model
ANN = Sequential()
ANN.add(Dense(3, input_dim=3))
ANN.add(Dense(3,activation="sigmoid"))
ANN.add(Dense(2,activation="sigmoid"))
ANN.add(Dense(1,activation="sigmoid"))
ANN.compile(optimizer='adam', loss='mean_squared_error')
ANN.fit(X_train_S, Y_train_S, epochs=1000,
        batch_size=10, verbose=1)
```

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor

The following outputs are produced using "MLPRegressor"

```
#=====Predict===#
PR=ANN.predict(X_train_S)
PS=ANN.predict(X_test_S)
```

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor

```
Y_train_pred=pd.DataFrame(PR*(D['Y1'].max() -
                  D['Y1'].min())+D['Y1'].min())
Y_test_pred=pd.DataFrame(PS*(D['Y1'].max() -
                  D['Y1'].min())+D['Y1'].min())

# Evaluate the model
from sklearn.metrics import mean_squared_error,
mean_absolute_error, mean_absolute_percentage_error, r2_score
# Training metrics
mse_train = mean_squared_error(Y_train, Y_train_pred)
mae_train = mean_absolute_error(Y_train, Y_train_pred)
mape_train = mean_absolute_percentage_error(Y_train,
                             Y_train_pred)
r2_train = r2_score(Y_train, Y_train_pred)
```

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor

```
mse_train, mae_train, mape_train, r2_train
(16.921732751135604,  3.382070014742195,
 0.05599049697242429,  0.9967653416107658)

# Test metrics
mse_test = mean_squared_error(Y_test, Y_test_pred)
mae_test = mean_absolute_error(Y_test, Y_test_pred)
mape_test = mean_absolute_percentage_error(Y_test,
                               Y_test_pred)
r2_test = r2_score(Y_test, Y_test_pred)
mse_test, mae_test, mape_test, r2_test

(18.274555168187195, 3.1791940152955607,
 0.07038717043882335, 0.9958722697338295)
```

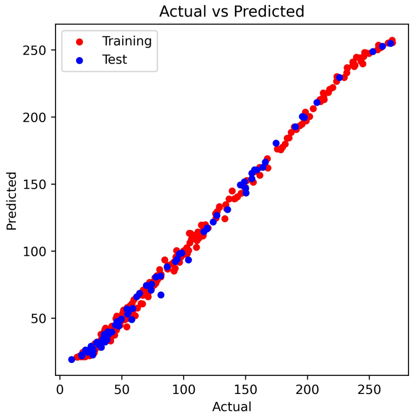
Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor

```
plt.figure(figsize=(4,4))
plt.scatter(Y_train,Y_train_pred,color='red',
                label='Training',s=20)
plt.scatter(Y_test,Y_test_pred,color='blue',
                label='Test',s=20)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs Predicted')
plt.legend()
```

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor



Python Code to Fit Neural Network

Multi-layer Perceptron Regressor for Climate Data

Problem Fit MLP to predict the Rainfall based on Temperature, Dew Point Temperature, Wind Speed, Humidity, and Sea Level Pressure.

ID	TEM	DPT	WIS	HUM	SLP	RAN
1	16.9	11.3	2	73.39	1016	0.48
2	21.4	12.6	1.7	66.34	1013	0
3	24.1	14.9	2.3	64.13	1011.4	2.23
.	.	.	.	.	.	.
.	.	.	.	.	.	.
653	23.1	18.7	1.7	81.73	1013.7	0.43
654	18.3	14.9	1.8	82.68	1015.9	0.16

Table: Mymensingh Data

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor for Climate Data

```
#====MLP to predict the Rainfall Level===#
import scipy
import numpy as np
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics mean_absolute_percentage_error
```

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor for Climate Data

```
path='E:/Village of Study/PMASDS/PM-ASDS 22 MACHINELEARNING FO

DD=pd.read_csv(path)
DD.dropna(how='any', inplace=True, axis=0)
DD.head()
#DD.count()

D=pd.DataFrame(DD.drop(['ID'], axis=1))
D.head()

scaler_x = MinMaxScaler()
SC=pd.DataFrame(scaler_x.fit_transform(D),columns=D.columns)
SC.head()
#SC.count()
```

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor for Climate Data

```
XX=pd.DataFrame(D.drop(['RAN'], axis=1))
YY=pd.DataFrame(D['RAN'])
X=pd.DataFrame(SC.drop(['RAN'],axis=1))
Y=pd.DataFrame(SC['RAN'])
XX.head(), X.head()

np.random.seed(104729)
Xo_train, Xo_test, Yo_train, Yo_test = train_test_split(XX,
YY, test_size=0.25, random_state=11)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.25, random_state=11)

np.sum(Xo_train.index-X_train.index)
np.sum(Yo_test.index-Y_test.index)
X_train.head(), X_train.index
```

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor for Climate Data

```
MLP = MLPRegressor(hidden_layer_sizes=(3,2),
        activation='logistic',learning_rate_init=0.001,
        momentum=0.8, solver='lbfgs', max_iter=999929)
MLP.fit(X_train, Y_train)
MLP.score(X_test, Y_test)

PR=MLP.predict(X_train)
PS=MLP.predict(X_test)

n=len(X.index)
n1=len(X_train.index)
n2=len(X_test.index)
n, n1, n2
```

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor for Climate Data

```
YH_tr=PR*(D['RAN'].max() -D['RAN'].min())+D['RAN'].min()

#===R-Square===#
r2_score(YH_tr,Yo_train)

#===Root Mean Square Error===#
np.sqrt(mean_squared_error(YH_tr,Yo_train))
mean_absolute_error(YH_tr,Yo_train)
mean_absolute_percentage_error(YH_tr,Yo_train)

#===Root Mean Square Error===#
E_tr=np.array(Yo_train['RAN'])-np.array(YH_tr)
RMSE_tr=np.sqrt(np.sum((E_tr)**2)/len(X_train))
RMSE_tr
```

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor for Climate Data

```
YH_ts=PS*(D['RAN'].max() -D['RAN'].min())+D['RAN'].min()
pd.DataFrame(YH_ts)

#===R-Square===#
r2_score(YH_ts,Yo_test)

#===Root Mean Square Error===#
np.sqrt(mean_squared_error(YH_ts,Yo_test))
mean_absolute_error(YH_ts,Yo_test)
mean_absolute_percentage_error(YH_ts,Yo_test)

#===Root Mean Square Error===#
E_ts=np.array(Yo_test['RAN'])-np.array(YH_ts)
RMSE_ts=np.sqrt(np.sum((E_ts)**2)/len(Yo_test.index))
RMSE_ts
```

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor for Climate Data

```
#=Prediction Error=======#
PE_tr=np.mean(np.abs(E_tr))/np.mean(Yo_train)
PE_ts=np.mean(np.abs(E_ts))/np.mean(Yo_test)
PE_tr, PE_ts
```

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor for Climate Data

```
MLP.coefs_
[array([[-3.15467852,  3.34291696,  1.02824667],
        [-4.85116818,  4.37172069, -2.71909758],
        [-1.11587446,  1.44012872, -3.84914094],
        [-2.77185936,  1.96120387, -5.61147083],
        [ 3.60630568, -2.38385056,  3.16575063]]),
 array([[-2.38795483, -0.04964771],
        [ 0.35062063, -0.35589035],
        [-6.53784306,  2.02430811]]),
 array([[3.08571396],
        [0.19289553]])]
MLP.intercepts_
[array([ 0.49261518, -0.25068592,  2.52949839]),
 array([-1.88656851,  0.13393256]),
 array([-0.13724781])]
```

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor for Climate Data

```
plt.figure(figsize=(5,5))
plt.scatter(Yo_train,YH_tr,color='red',label='Training')
plt.scatter(Yo_test,YH_ts,color='blue',label='Test')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs Predicted')
plt.legend()
plt.savefig("D:/PE_Actual_vs_Predicted for Mymensingh Data.jp
```

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor for Climate Data

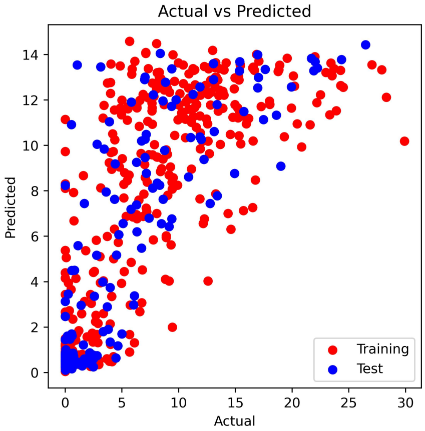


Figure: Schematic plot for Actual Rainfall vs Predicted Rainfall.

Python Code to Fit Neural Network

Python Code for Multi-layer Perceptron Regressor for Climate Data

```
data = {'Training': PE_tr[0], 'Test': PE_ts[0]}
CT = list(data.keys())
CV = list(data.values())
fig = plt.figure(figsize = (3,3))
# creating the bar plot
plt.bar(CT, CV, color ='blue', width = 0.5)
plt.xlabel("Category")
plt.ylabel("Prediction Error")
plt.title("Prediction Errors")
plt.savefig("D:/PE_Prediction Error for Mymensingh Data.jpg",
dpi=500)
```

Python Code to Fit Neural Network

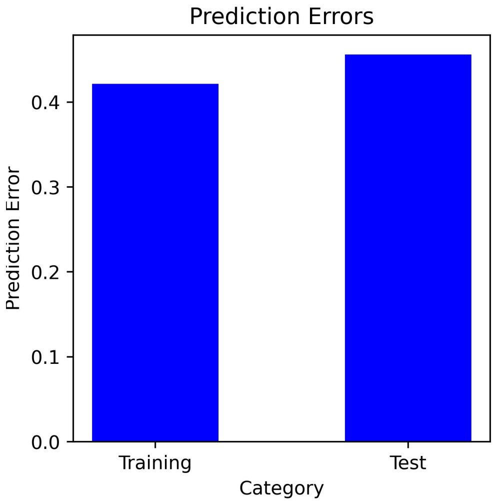


Figure: Plot for Prediction Error.

Python Code to Fit MLP Classification for IRIS Data

Problem
Fit an MLP to classify the Species category of IRIS Flower from IRIS data.

Python Code to Fit MLP Classification for IRIS Data

Python Code

```
#===MLP or ANN Classifier for IRIS Data.ipynb#

import numpy as np
import pandas as pd
import warnings
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.exceptions import ConvergenceWarning
```

Python Code to Fit MLP Classification for IRIS Data

Python Code

```
# Stat Models
import statsmodels.tools.tools as stattools
from sklearn.metrics import accuracy_score
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import classification_report
from sklearn.metrics importconfusion_matrix
#from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
from sklearn.metrics import recall_score
```

Python Code to Fit MLP Classification for IRIS Data

Python Code

```
D=datasets.load_iris()
x=D.data
y=D.target

# Data Normalization
Sm= MinMaxScaler()
Sn=Sm.fit_transform(x)
np.random.seed(104729)
X_train, X_test, Y_train, Y_test = train_test_split(Sn,
y,test_size=0.2,random_state=1)

MLP = MLPClassifier(solver='lbfgs', alpha=1e-5,
max_iter=1000,activation='logistic', verbose=True,
hidden_layer_sizes=(4,3), random_state=1)
MLP.fit(X_train, Y_train)
```

Python Code to Fit MLP Classification for IRIS Data

Python Code

```
# Return the mean accuracy on the given test data and labels.
MLP.score(X_train, Y_train)
# Return the mean accuracy on the given test data and labels.
MLP.score(X_test, Y_test)

# Predict using the multi-layer perceptron classifier.
PR=MLP.predict(X_train[:,:])

print(confusion_matrix(Y_train, PR))
accuracy_score(Y_train, PR)
print(classification_report(Y_train, PR))
# Probability estimates.
np.round(MLP.predict_proba(X_train[:,:]),2)
```

Python Code to Fit MLP Classification for IRIS Data

Python Code

```
# Predict using the multi-layer perceptron classifier.
PS=MLP.predict(X_test[:,:])
PS

# Probability estimates.
np.round(MLP.predict_proba(X_test[:,:]),2)

print(confusion_matrix(Y_test, PS))

accuracy_score(Y_test, PS)

#MLP.get_params

print(classification_report(Y_test, PS))
```

Python Code to Fit MLP Classification for IRIS Data

Results for Training Data

```
 print(classification_report(Y_train, PR))
              precision    recall  f1-score   support
           0       1.00      1.00      1.00        39
           1       0.97      0.97      0.97        37
           2       0.98      0.98      0.98        44

    accuracy                           0.98       120
   macro avg       0.98      0.98      0.98       120
weighted avg       0.98      0.98      0.98       120

print(confusion_matrix(Y_train, PR))
[[39  0  0]
 [ 0 36  1]
 [ 0  1 43]]
```

Python Code to Fit MLP Classification for IRIS Data

Results for Test Data

```
print(classification_report(Y_test, PS))
             precision    recall  f1-score   support
           0       1.00      1.00      1.00        11
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00         6

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

print(confusion_matrix(Y_test, PS))
[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
```

Look for Details

Han et al. (2022), Hastie et al. (2009), DeLurgio (1998)

References I

DeLurgio, S. A. (1998). *Forecasting principles and applications*, Vol. 49, Irwin/McGraw-Hill New York.

Han, J., Pei, J. and Tong, H. (2022). *Data Mining: Concepts and Techniques*, 4th edn, Morgan Kaufmann, Burlington, MA.

Hastie, T., Tibshirani, R. and Friedman, J. (2009). The elements of statistical learning: data mining, inference, and prediction.

Machine Learning for Data Science

***** - /// - There are no End .. - /// - *****

****

****