COMPUTATIONAL INTELLIGENCE ASSIGNMENT REPORT

(MTECH-KEFT-27)

HYBRID NEURAL NET USING R, RAPIDMINER AND SPSS

TEAM MEMBERS
ASHUTOSH GAUR(A0134613N)
AKSHAY SETH(A0134589R)

TABLE OF CONTENT

1.1 INTRODUCTION:

An Artificial Neural Network (ANN) is an information-processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well[1].

Architecture of neural networks

Feed-forward networks

Feed-forward ANNs (figure 1) allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer. Feed-forward ANNs tend to be straightforward networks that associate inputs with outputs. They are extensively used in pattern recognition. This type of organization is also referred to as bottom-up or top-down.

Feedback networks

Feedback networks (figure 1.1) can have signals travelling in both directions by introducing loops in the network. Feedback networks are very powerful and can get extremely complicated. Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found. Feedback architectures are also referred to as interactive or recurrent, although the latter term is often used to denote feedback connections in single-layer organizations[2].

---

[1]http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#What%20is%20a%20Neural%20Network

[2]http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#What%20is%20a%20Neural%20Network

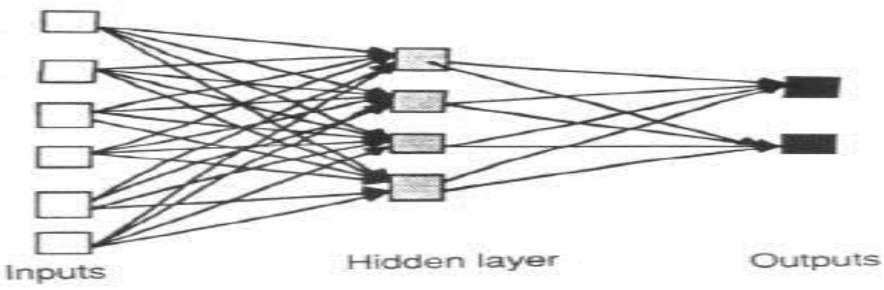


Figure 1.1 An example of a simple feed forward network

1.2PROBLEM STATEMENT

Two dataset have been given to perform Classification/ Regression task using soft computing techniques. The outcome of this assignment is to get good understanding of the different neural network architecture as well as get hand on experience of development of the hybrid neural network to solve the real world problems. We are required to train group of different type of Neural Network using different tools to solve the problem of classification of regression. **We have chosen R, Rapid Miner and SPSS as tool to work on Multi Layer Perceptron (MLP), Support Vector Machine (SVM) and General Regression Neural Network (GRNN) architecture.** All the coding and findings are enclosed herewith in this file in the experiments section.

Lets have a look on the given dataset. The set s is consistent so there is no need to process it further. The details of the dataset are given below:

1.1 Diabetes Data Pima Indians Diabetes Database is the consolidated data of female are at least of 21 years old and is provided by National Institute of Diabetes and Digestive and Kidney Diseases on 9 May 1990. It contains 768 numbers of instances with 8 inputs and one target ‘class’.

Input Attributes: (all numeric-valued)

- Number of times pregnant
- Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- Diastolic blood pressure (mm Hg)
- Triceps skin fold thickness (mm)
- 2-Hour serum insulin (mu U/ml)
- Body mass index (weight in kg/(height in m)^2)

- Diabetes pedigree function
- Age (years)

Output variable:

- Class variable (0 or 1)

Class Distribution: (class value 1 is interpreted as "tested positive for diabetes")

Class Value	Number of instances
0	500
1	268

1.2 Wine Data

Wine Quality is consolidated data of white wine samples and is created by Paulo Cortez (Univ. Minho), Antonio Cerdeira, Fernando Almeida, Telmo Matos and Jose Reis (CVRVV) in 2009. The inputs include objective tests (e.g. PH values) and the output is based on sensory data (median of at least 3 evaluations made by wine experts). Each expert graded the wine quality between 0 (very bad) and 10 (very excellent It contain 4898 number of instances with 11 input and one target 'quality'.

Input Attribute (based on physicochemical tests):

- fixed acidity
- volatile acidity
- citric acid
- residual sugar
- chlorides
- free sulfur dioxide
- total sulfur dioxide
- density

- pH
- sulphates
- alcohol

Output variable (based on sensory data): quality (score between 0 and 10)

2. DIABETES DATASET

2.1 Network1

Tool used: R Programming Language using nnet package

Architecture used: Feed Forward Neural Network (Multi Layer Perceptron)

Data Set: Diabetes Dataset

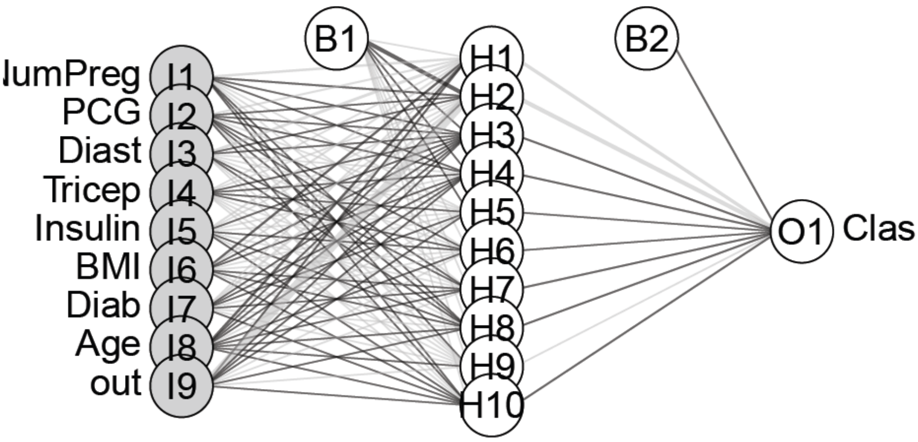
2.1 BACK PROPAGATION CODE

```
#######################################################################
Back Propagation method for Diabetes dataset
#######################################################################

#Diabetes.csv
#Installing Packages
#install.packages('nnet')      # Neural Network package
#install.packages('caTools')   # Utility package
#install.packages('caret')     # package for computing Confusion matrix
library(nnet)                  # installing Library nnet
require(caTools)                 # installing Library caTools
library(caret)                 # installing Library caret

setwd("C:/Users/Akshay.Akshay-PC.000/Downloads/CI SVM")
#reading dataset from the local drive
Diab <- read.csv("Diabetes.csv",header=TRUE)
#reading dataset from the local drive
Diab=read.csv("~/Documents/Diab.csv", head=TRUE)
#Defining target
Diab$Clas<-as.factor(Diab$out)
#dispalying the details of the data
str(Diab)

#output

#'data.frame':  768 obs. of  10 variables:
#  $ NoPr  : int  6 1 8 1 0 5 3 10 2 8 ...
#$ Pl.Glu: int  148 85 183 89 137 116 78 115 197 125 ...
#$ BP    : int  72 66 64 66 40 74 50 0 70 96 ...
#$ TSFT  : int  35 29 0 23 35 0 32 0 45 0 ...
#$ INSU  : int  0 0 0 94 168 0 88 0 543 0 ...
#$ BMI   : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
```

```
#$ DPF   : num  0.627 0.351 0.672 0.167 2.288 ...
#$ Age   : int  50 31 32 21 33 30 26 29 53 54 ...
#$ Clas  : Factor w/ 2 levels "0","1": 2 1 2 1 2 1 2 1 2 2 ...
#$ Result: Factor w/ 2 levels "negetive","positive": 2 1 2 1 2 1 2 1 2 2 ...

#setting seed value
#set.seed(1000)

#spliting the data in to training and testing set 75:25
check<-sample.split(Diab$Clas,SplitRatio=0.75)

# allocation of training data
train<-subset(Diab, check==TRUE)

#Allocation of Testing data
test<-subset(Diab,check==FALSE)

#Creat Neural Network Model
model_nnet <- nnet(Clas ~ ., data=train, size=10, maxit=1000)

#output
# weights:  111
#initial  value 490.826832
#iter  10 value 356.843702
#iter  20 value 346.001007
#iter  30 value 333.321303
#iter  40 value 320.004147
#iter  50 value 306.344904
#iter  60 value 281.575144
#iter  70 value 215.347420
#iter  80 value 167.680977
#iter  90 value 148.423714
#iter 100 value 134.809676
#iter 110 value 107.502788
#iter 120 value 76.753335
#iter 130 value 29.778354
#iter 140 value 19.113442
#iter 150 value 7.957232
#iter 160 value 0.327102
#iter 170 value 0.038201
#iter 180 value 0.010042
#iter 190 value 0.007195
#iter 200 value 0.005167
#iter 210 value 0.000618
#iter 220 value 0.000395
#iter 230 value 0.000114
#iter 240 value 0.000112
#final  value 0.000088
#converged
#Setting Prediction variable
pred<- predict(model_nnet, test, type="class")

#table(true=test$Clas, predicted=pred)
xtab <- table(true=test$Clas, predicted=pred)

#Creating confusion Matrix
confusionMatrix(xtab)
```

```
#output
#predicted
#true      0    1
#       0  124    1
#       1   0    67

#Accuracy : 0.7448
#95% CI : (0.677, 0.8048)
#No Information Rate : 0.625
#P-Value [Acc > NIR] : 0.0002918

#Kappa : 0.4479
#Mcnemar's Test P-Value : 0.5677092

#          Sensitivity : 0.8167
#          Specificity : 0.6250
#       Pos Pred Value : 0.7840
#       Neg Pred Value : 0.6716
#           Prevalence : 0.6250
#  Detection Prevalence : 0.6510
#     Balanced Accuracy : 0.7208

#     'Positive' Class : 0

#Result
result<- cbind(test, data.frame(pred))
#Writing Output
write.csv(result, file="D:\\resultBP.csv")

### TO PLOT NEURAL NET
library(devtools)

#import the function from Github
source_url('https://gist.githubusercontent.com/Peque/41a9e20d6687f2f3108d/raw/85e14f3a292e126f1454864427e3a189c2fe33f3/nnet_plot_update.r')

#plot each model
pdf('./nn-example.pdf', width = 7, height = 7)
plot.nnet(model_nnet, alpha.val = 0.5, circle.col = list('lightgray', 'white'), bord.col = 'black')
dev.off()
```

Output: Accuracy=



2.2 Network2

Tool used: R Programming Language using grnn package

Architecture used: General Regression Neural Network

Data Set: Diabetes Dataset

2.2 GRNN CODE

```
###########################################################################
#GRNN for Diabetes dataset
###########################################################################
#install.packages("grnn")
library("grnn")
Diab=read.csv("~/Documents/Diab.csv")
#view(Diab)
#wait
size=nrow(Diab)
length=ncol(Diab)
idex=1:size
positions<-sample(idex,trunc(size*0.75))
training<-Diab[positions,]
testing<-Diab[-positions,1:length-1]
result=Diab[-positions, ]
result$actual=result[,length]
result$predict=-5
nn<-learn(training,variable.column=length)
nn<-smooth(nn,sigma=0.95)
for(i in 1:nrow(testing))
{
  vec<-as.matrix(testing[i,])
  res<-guess(nn,vec)
    #res1<-res
  if(is.nan(res))
  {
    cat("Entry",i,"generated non results")
  }
  else
  {
```

```
    result$predict[i]<-round(res)
  }
}

#  Entry 30 generated non resultsEntry 91 generated non resultsEntry 123 generated non results
result.size=nrow(result)
a1=result.correct=nrow(result[round(result$predict)==result$actual,])
cat('\nNo. of test cases=',result.size,"\n")
#No. of test cases= 154
cat("correct preditions=",result.correct,"\n")
#correct preditions= 108
cat("Accuracy=",result.correct/result.size*100,"\n")
#Accuracy= 70.12987
result2<-cbind(testing,data.frame(result))
plotRegressionError(result.correct, result$atual)
```

Output:

```
> result.size = nrow(result)
> result.correct = nrow(result[round(result$predict) == result$actual,])
> cat("No of test cases = ",result.size,"\n")
No of test cases =  154
> cat("Correct predictions = ", result.correct ,"\n")
Correct predictions =  108
> cat("Accuracy = ", result.correct / result.size * 100 ,"\n")
Accuracy =  70.12987
>
>
> result2<- cbind(testing_grnn, data.frame(result))
>
> write.csv(result, file="C:/Users/Akshay.Akshay-PC.000/Documents/resultGRNN_d.csv")
> |
```

2.3 Network3

Tool used: R Programming Language using RSNNS package (mlp network)

Architecture used: Feed Forward Neural Network (Multi Layer Perceptron)

Data Set: Diabetes Dataset

2.3 MLP CODE in RSNNS package

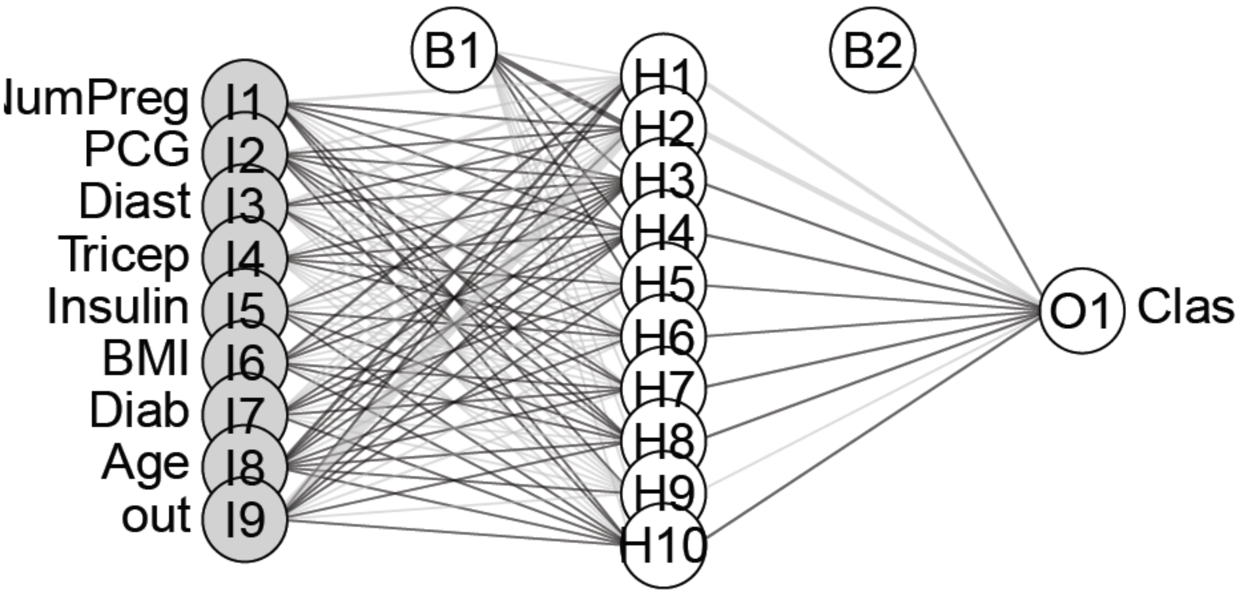
```
#########################################################################################
#Back Propagation method for Diabities dataset using Multy Layer Perceptron (MLP)
#
# MLPs are fully connected feed- forward networks, and probably the most common
#network architecture in use. Training is usually performed by error backpropagation
#or a related procedure.
########################################################################################

install.packages('RSNNS') #   Neural Networks in R using the Stuttgart Neural Network
                        #  Simulator (SNNS)

library('RSNNS')         # installing Library RSNNS

Diab=read.csv("~/Documents/Diab.csv", head=TRUE)  # Reading File from local storage

#shuffle the vector
Diab <- Diab[sample(1:nrow(Diab),length(1:nrow(Diab))),1:ncol(Diab)]

DiabValues <- Diab[,1:8]                      # selecting Input data
DiabTargets <- decodeClassLabels(Diab[,9])      # selecting target data

# Spliting of training and test data in 75:25 ratio
Diab <- splitForTrainingAndTest(DiabValues, DiabTargets, ratio=0.33)

Diab <- normTrainingAndTestSet(Diab)          # Normalization

# Building MLP model with 8 neurons in Input layer, 8 neurons in Hidden layer

model <- mlp(Diab$inputsTrain, Diab$targetsTrain, size=8, learnFuncParams=c(0.1),
            maxit=100, inputsTest=Diab$inputsTest, targetsTest=Diab$targetsTest)

summary(model)       # summary of model
model                #  model description

#Class: mlp->rsnns
#Number of inputs: 8
#Number of outputs: 2
#Maximal iterations: 100
#Initialization function: Randomize_Weights
#Initialization function parameters: -0.3 0.3
#Learning function: Std_Backpropagation
#Learning function parameters: 0.1
#Update function:Topological_Order
#Update function parameters: 0
#Patterns are shuffled internally: TRUE
#Compute error in every iteration: TRUE
```

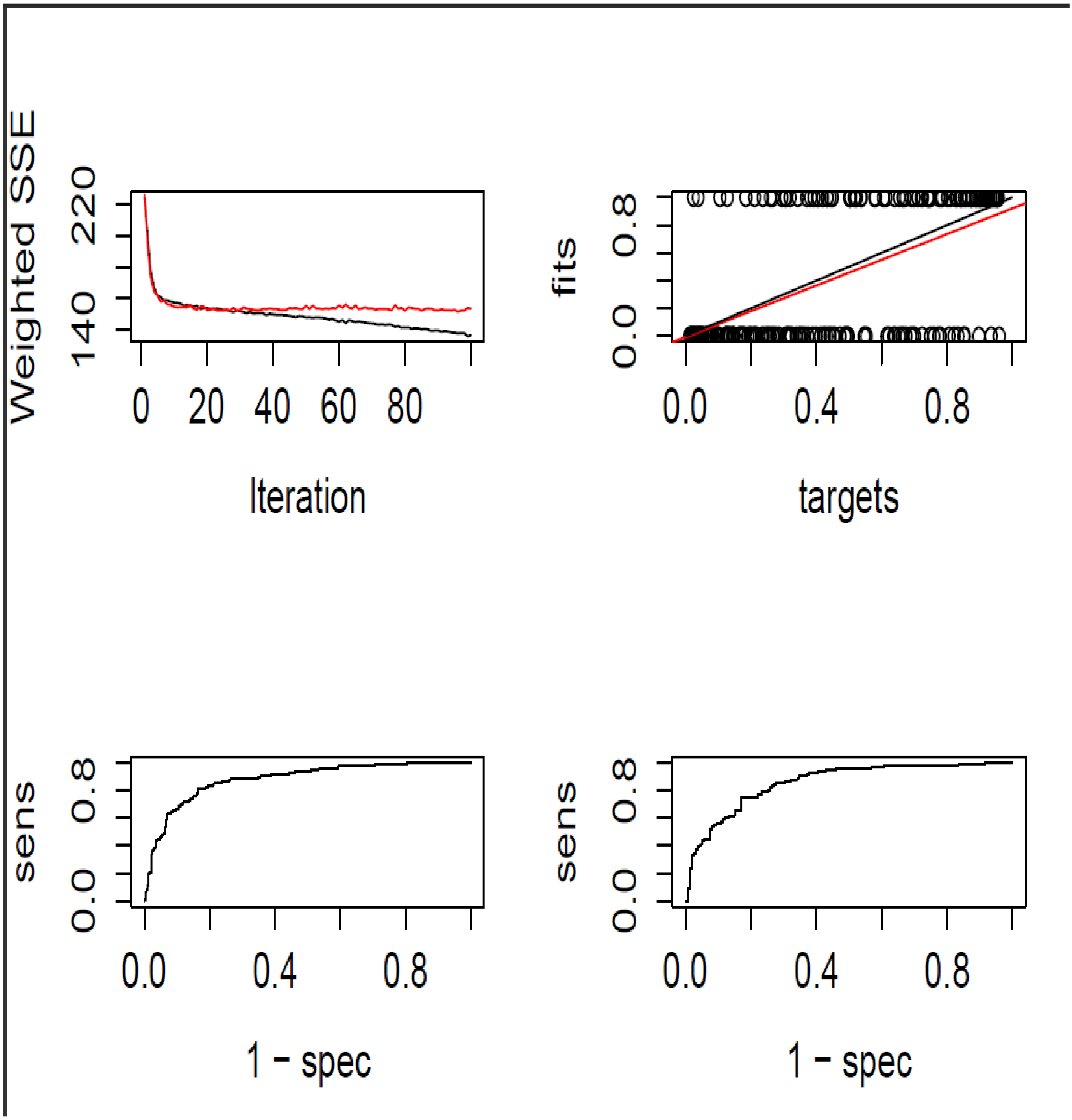
```
#Architecture Parameters:
#  $size
#[1] 8

#All members of model:
#  [1] "nInputs"               "maxit"               "initFunc"
#[4] "initFuncParams"        "learnFunc"            "learnFuncParams"
#[7] "updateFunc"            "updateFuncParams"      "shufflePatterns"
#[10] "computeIterativeError" "snnsObject"           "archParams"
#[13] "IterativeFitError"     "IterativeTestError"    "fitted.values"
#[16] "fittedTestValues"      "nOutputs"

weightMatrix(model)        # Displaying Weight Matrix

extractNetInfo(model)        # Displaying Information

par(mfrow=c(2,2))               #parameter adjustments

plotIterativeError(model)         # Plot Iterative error

predictions <- predict(model,Diab$inputsTest)         # executing prediction

plotRegressionError(predictions[,2], Diab$targetsTest[,2])      # plot regression line

confusionMatrix(Diab$targetsTrain,fitted.values(model))      # confusion matrix of training data


# predictions
#   targets   1     2
#       1    299    43
#       2    49     123
confusionMatrix(Diab$targetsTest,predictions)        # confusion matrix of test data


#  predictions
#      targets   1   2
#          1    131  27
#          2     24  72
plotROC(fitted.values(model)[,2], Diab$targetsTrain[,2])      # Ploting ROC curve of training data

plotROC(predictions[,2], Diab$targetsTest[,2])              # Ploting ROC curve for test data

#confusion matrix with 402040-method

confusionMatrix(Diab$targetsTrain,   encodeClassLabels(fitted.values(model),method="402040",   l=0.4,
h=0.6))

#predictions
#      targets   0   1   2
#          1     43 275  24
#          2     34  30 108

# TO PLOT NEURAL NET

library(devtools)

#import the function from Github
```

source_url('https://gist.githubusercontent.com/Peque/41a9e20d6687f2f3108d/raw/85e14f3a292e126f1454864427e3a189c2fe33f3/nnet_plot_update.r')

#plot each model
pdf('./nn-example.pdf', width = 27, height = 107)
plot.nnet(model_nnet, alpha.val = 0.5, circle.col = list('lightgray', 'white'), bord.col = 'black')
dev.off()
plot.nnet(model_nnet, alpha.val = 0.5, circle.col = list('lightgray', 'white'), bord.col = 'black')

```
predictions <- predict(model,Diab$inputsTest)  # executing prediction

plotRegressionError(predictions[,2], Diab$targetsTest[,2])   # plot regression line

confusionMatrix(Diab$targetsTrain,fitted.values(model))   # confusion matrix of training data


# predictions
#   targets   1     2
#       1    299    43
#       2    49     123
confusionMatrix(Diab$targetsTest,predictions)  # confusion matrix of test data


#  predictions
#      targets   1   2
#          1    131  27
#          2     24  72
plotROC(fitted.values(model)[,2], Diab$targetsTrain[,2])  # Ploting ROC curve of training data

plotROC(predictions[,2], Diab$targetsTest[,2])  # Ploting ROC curve for test data
#confusion matrix with 402040-method
confusionMatrix(Diab$targetsTrain, encodeClassLabels(fitted.values(model),method="402040", l=0.4, h=0.6))

#predictions
#      targets   0   1   2
#          1     43 275  24
#          2     34  30 108
```




Weighted SSE
140
220
0
20
40
60
80
Iteration
fits
0.0
0.8
0.0
0.4
0.8
targets
sens
0.0
0.8
0.0
0.4
0.8
1 − spec
sens
0.0
0.8
0.0
0.4
0.8
1 − spec

2.4 Network4

Tool used: R Programming Language using e1071 package (SVM)

Architecture used: Support Vector Machine

Data Set: Diabetes Dataset

2.4 SVM CODE with RBF, Polynomial, Linear and radial Kernel

```
############################################################################
#SVM for Diabetes dataset with different kernel
############################################################################
#Diabetes.csv
install.packages(‘caTools)
install.packages(‘e1071’)
require(caTools)
require(e1071)

setwd("C:/Users/Akshay.Akshay-PC.000/Downloads/CI SVM")

# Input data
data <- read.csv("Diabetes.csv",header=TRUE)

# convert output class(target) into a factor
data$out <- as.factor(data$out)

# spliting the data into Train & Test
set.seed(2000)
check <- sample.split(data$out, SplitRatio=0.75)
train <- subset(data,check==TRUE)
test <- subset(data,check==FALSE)

# DEFAULT SVM Model
set.seed(2000)
default <- svm(out~.,train)
pr <- predict(default,test)
table(pr,test$out)
#pr    0   1
#0 108  38
#1  17  29
# Polynomial SVM
set.seed(2000)
poly <- svm(out~., train, kernel="polynomial")
pr <- predict(poly,test)
table(pr,test$out)
#pr    0   1
#0 115  50
#1  10  17
# Radial SVM
set.seed(2000)
poly <- svm(out~., train, kernel="radial")
pr <- predict(poly,test)
table(pr,test$out)
#pr    0   1
```

```
#0 108  38
#1  17  29
set.seed(2001)
tune.out <- tune(svm,out~.,data=train,ranges=list(cost=c(0.00001,0.001,0.034,0.6,0.98,1.9,3,99)))
# we find that best cost paramters = 0.6
radial <- svm(out~., train, kernel="radial", cost=0.6)
pr <- predict(radial, test)
table(pr,test$out)
#pr    0   1
#0 110  39
#1  15  28
```

```
> # DEFAULT SVM Model
> set.seed(2000)
> default <- svm(out~.,train)
> pr <- predict(default,test)
> table(pr,test$out)

pr    0   1
  0 108  38
  1  17  29
> # Polynomial SVM
> set.seed(2000)
> poly <- svm(out~., train, kernel="polynomial")
> pr <- predict(poly,test)
> table(pr,test$out)

pr    0   1
  0 115  50
  1  10  17
> # Radial SVM
> set.seed(2000)
> poly <- svm(out~., train, kernel="radial")
> pr <- predict(poly,test)
> table(pr,test$out)

pr    0   1
  0 108  38
  1  17  29
> set.seed(2001)
> tune.out <- tune(svm,out~.,data=train,ranges=list(cost=c(0.00001,0.001,0.034,0.6,0.98,1.9,3,99)))
> # we find that best cost paramters = 0.6
> radial <- svm(out~., train, kernel="radial", cost=0.6)
> pr <- predict(radial, test)
> table(pr,test$out)

pr    0   1
  0 110  39
  1  15  28
```

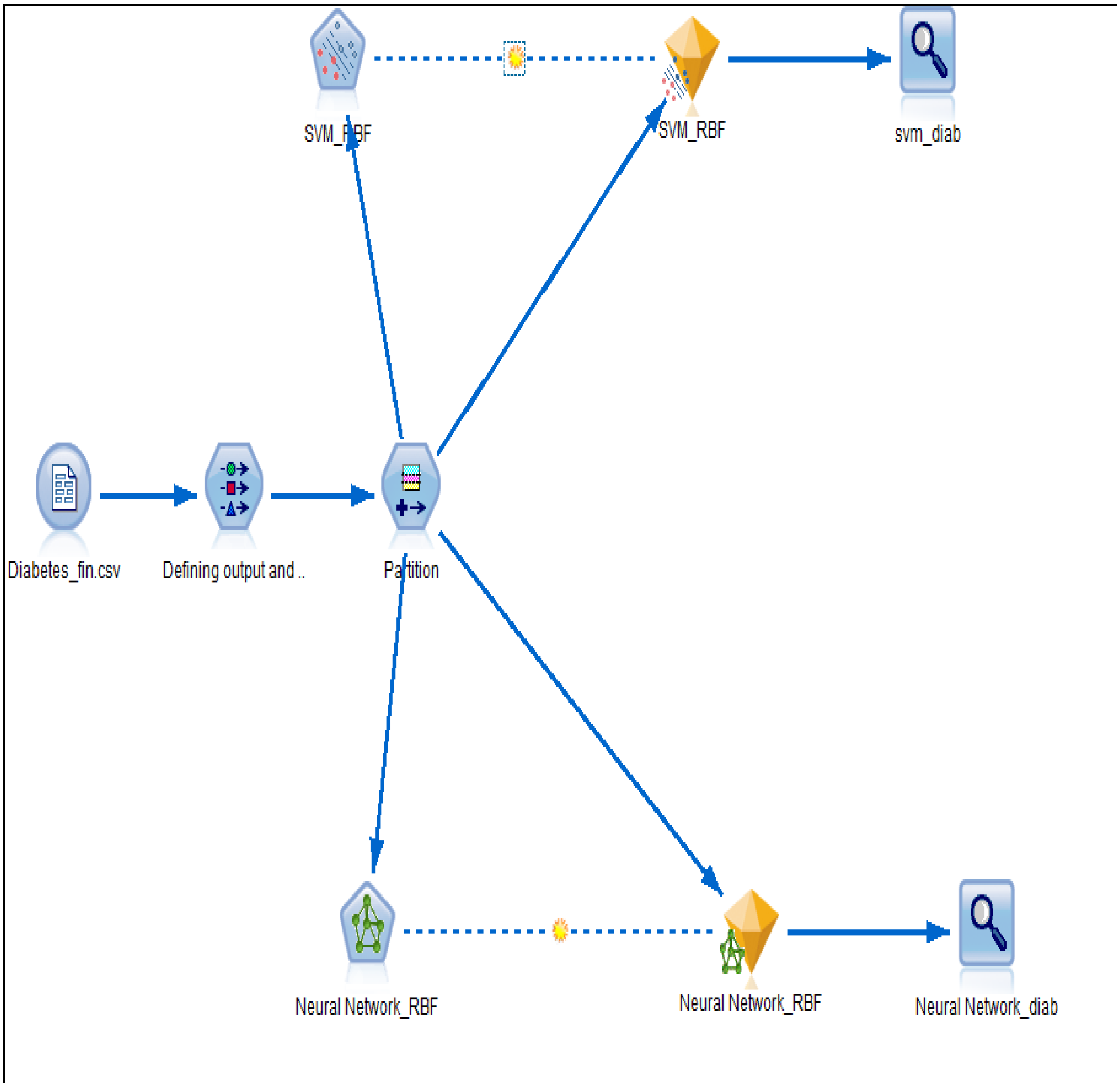
2.5 Network5

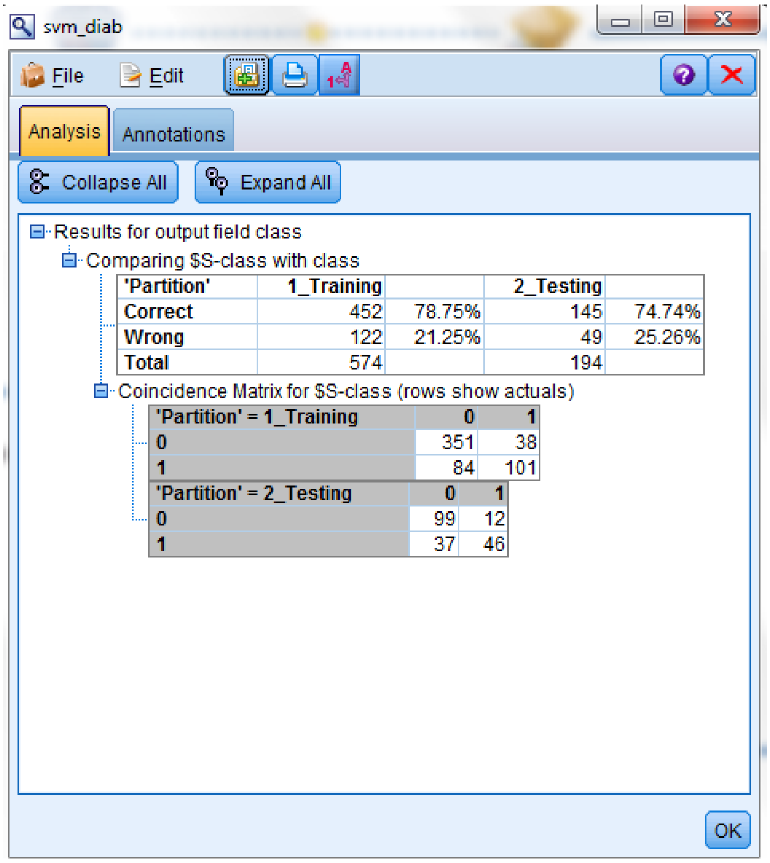
Tool used: IBM SPSS Modular

Architecture used: Support Vector Machine & Neural Network separately

Data Set: Diabetes Dataset

2.5 SVM and Neural Network

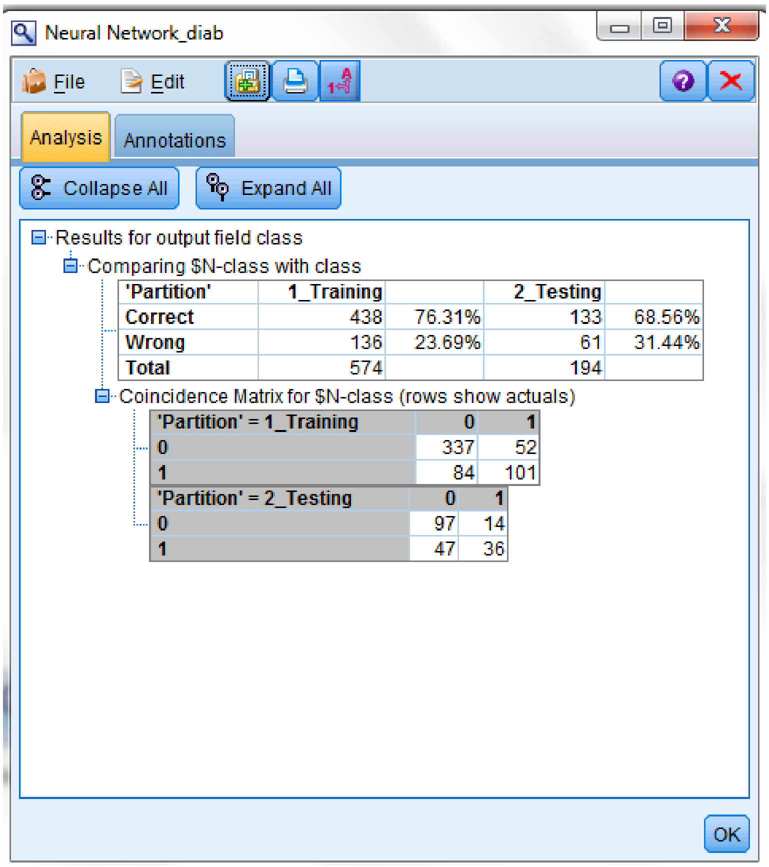




'Partition'	1_Training		2_Testing	
Correct	452	78.75%	145	74.74%
Wrong	122	21.25%	49	25.26%
Total	574		194	

'Partition' = 1_Training	0	1
0	351	38
1	84	101

'Partition' = 2_Testing	0	1
0	99	12
1	37	46



'Partition'	1_Training		2_Testing	
Correct	438	76.31%	133	68.56%
Wrong	136	23.69%	61	31.44%
Total	574		194	

'Partition' = 1_Training	0	1
0	337	52
1	84	101

'Partition' = 2_Testing	0	1
0	97	14
1	47	36

2.6 Network6 (Hybrid 1 SVM Polynomial with Neural Network MLP)

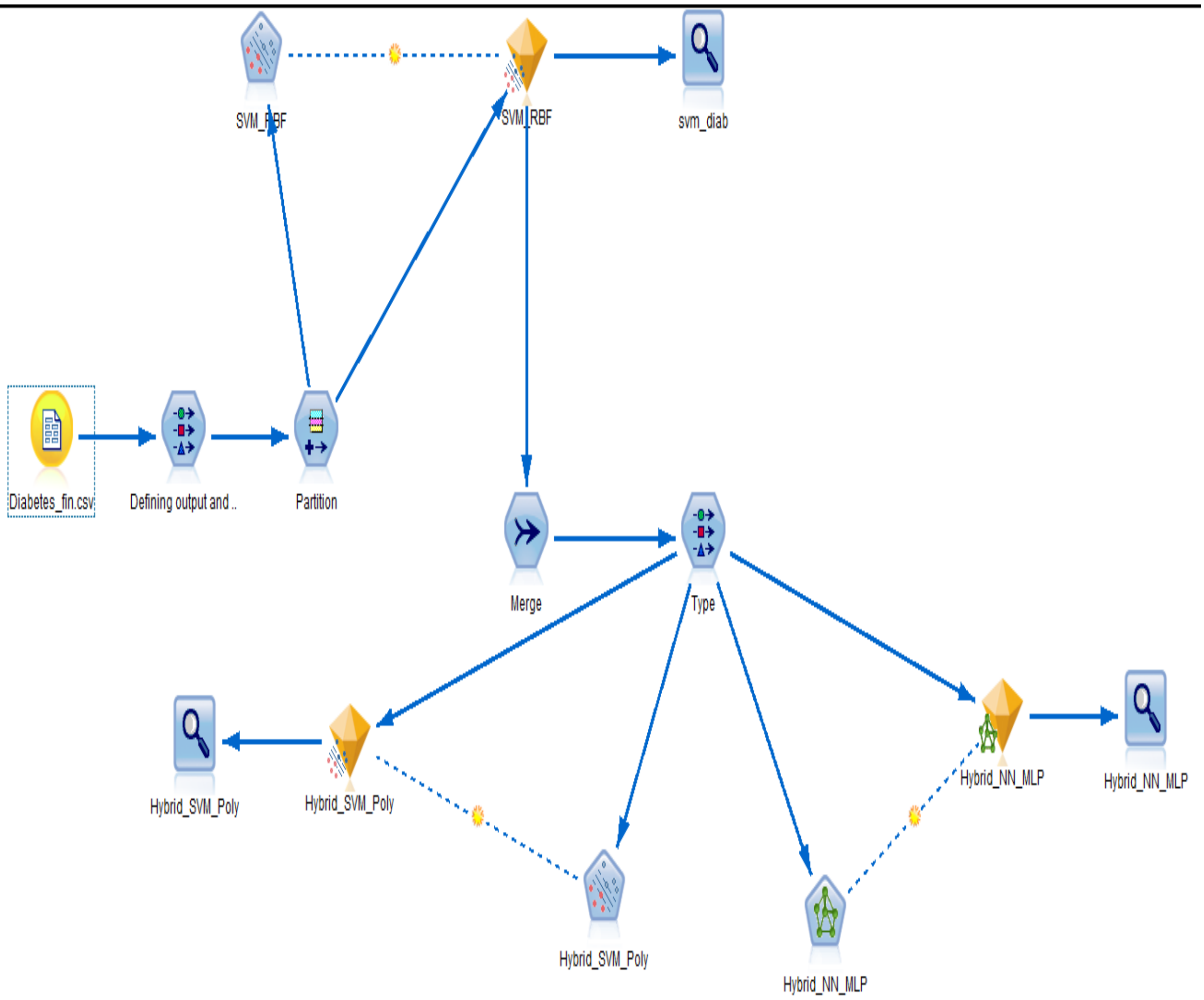
Tool used: IBM SPSS Modular

Architecture used: Support Vector Machine & Neural Network (Hybrid Network)

Data Set: Diabetes Dataset

2.6 SVM and Neural Network

HYBRID- 1

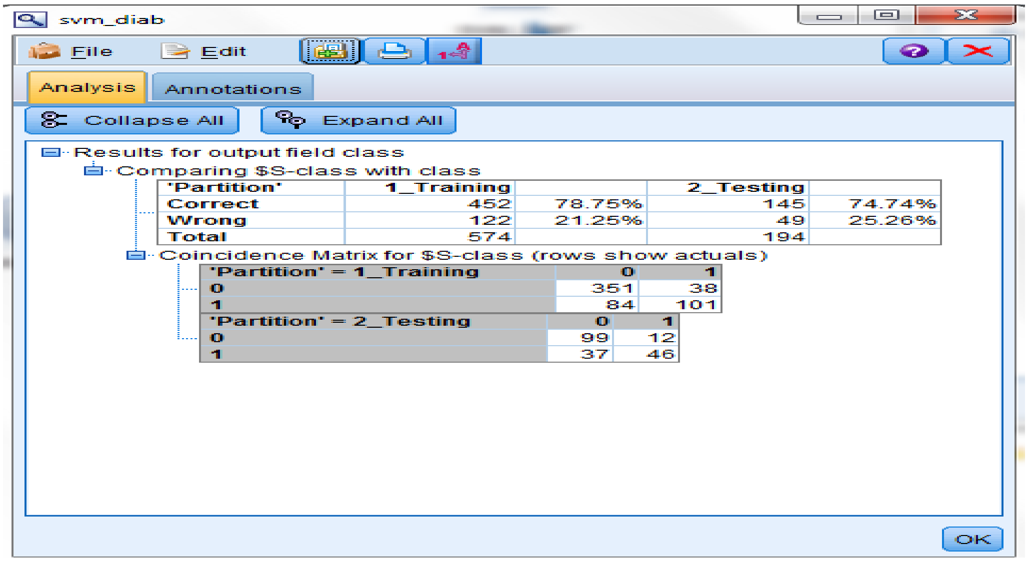


Output: SVM RBF= 74.74%

Hybrid NN=73.71%

Hybrid SVM=74.74%

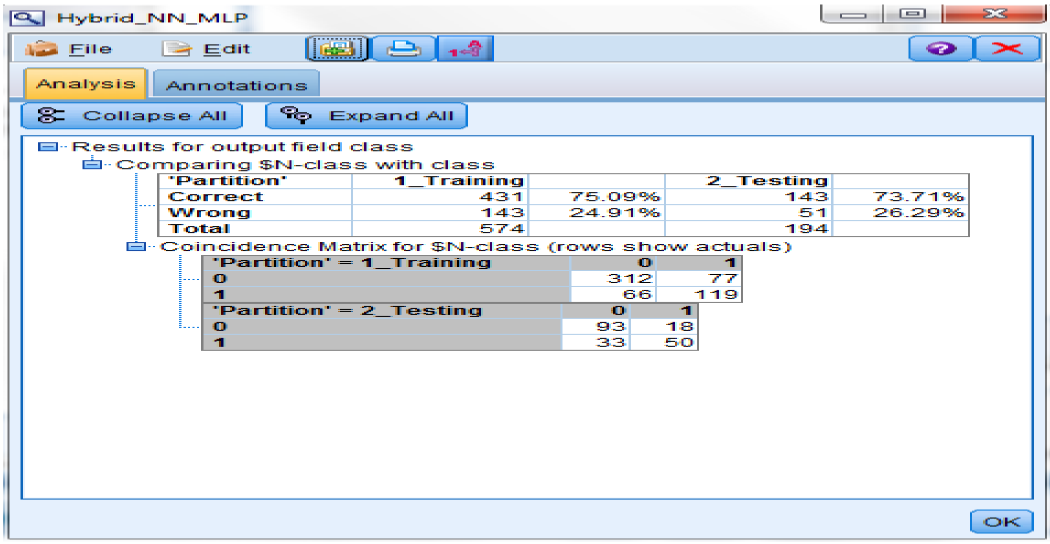
Finding: SVM is performing much better than neural network. However Polynomial and RBF kernel have the same result.



'Partition'	1_Training		2_Testing	
Correct	452	78.75%	145	74.74%
Wrong	122	21.25%	49	25.26%
Total	574		194	

'Partition' = 1_Training	0	1
0	351	38
1	84	101

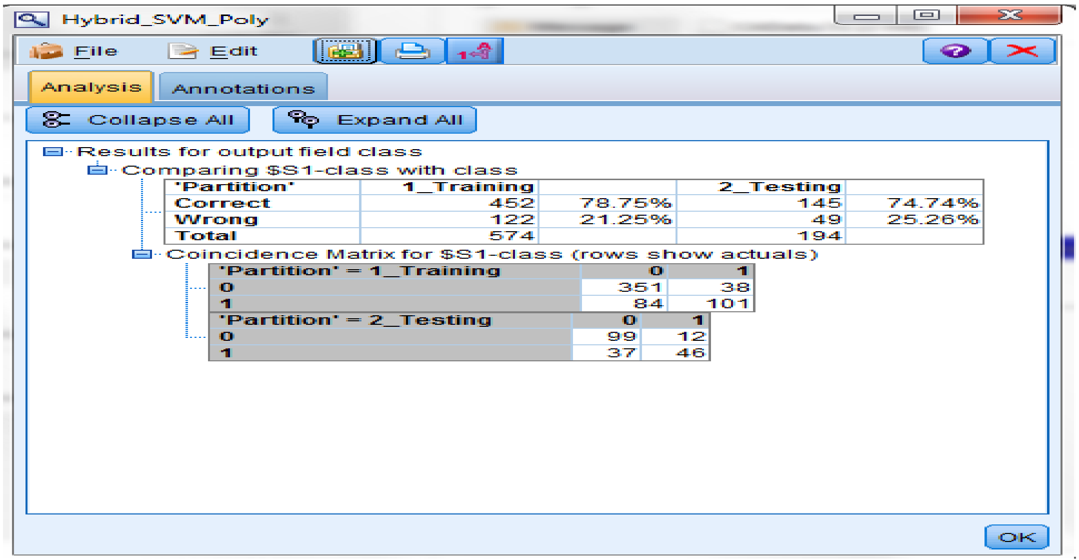
'Partition' = 2_Testing	0	1
0	99	12
1	37	46



'Partition'	1_Training		2_Testing	
Correct	431	75.09%	143	73.71%
Wrong	143	24.91%	51	26.29%
Total	574		194	

'Partition' = 1_Training	0	1
0	312	77
1	66	119

'Partition' = 2_Testing	0	1
0	93	18
1	33	50



'Partition'	1_Training		2_Testing	
Correct	452	78.75%	145	74.74%
Wrong	122	21.25%	49	25.26%
Total	574		194	

'Partition' = 1_Training	0	1
0	351	38
1	84	101

'Partition' = 2_Testing	0	1
0	99	12
1	37	46

2.7 Network7 (Hybrid 2 with SVM RBF and Neural Network RBF)

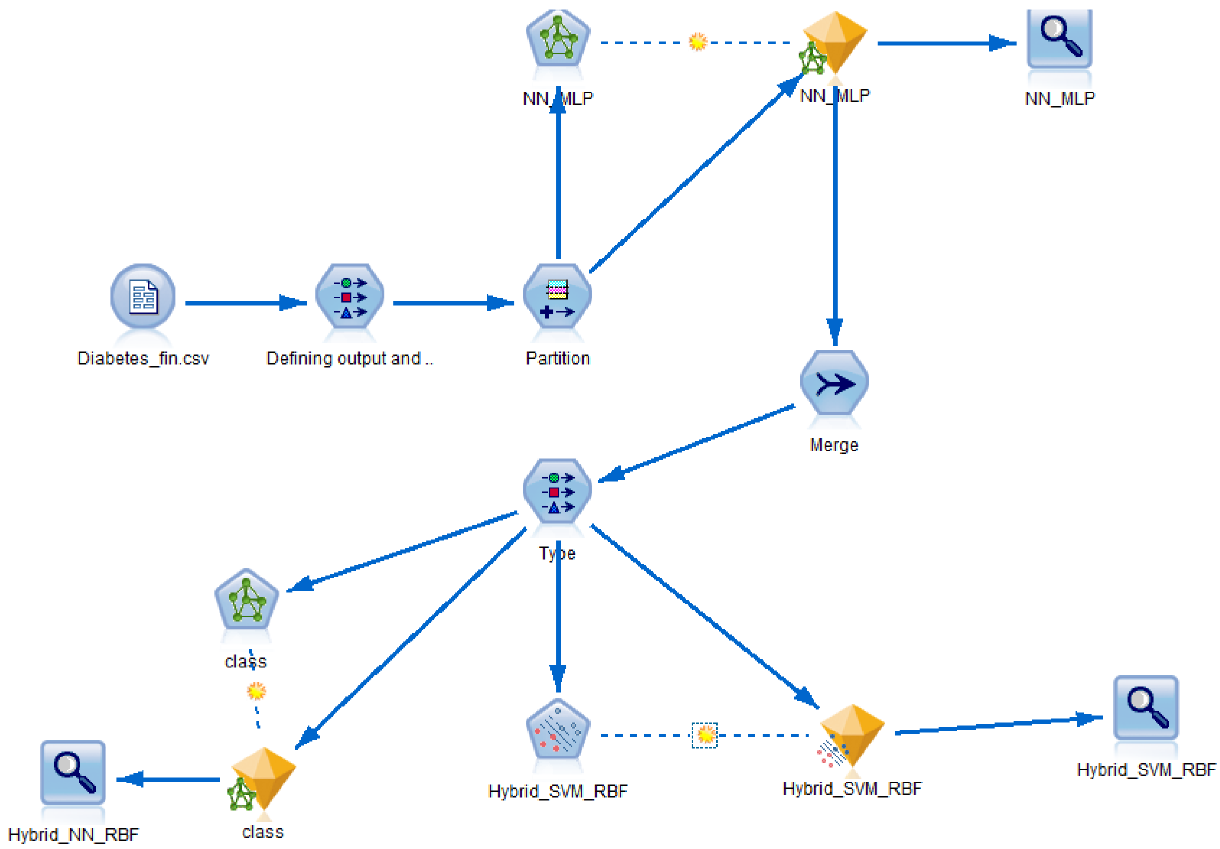
Tool used: IBM SPSS Modular

Architecture used: Support Vector Machine & Neural Network (Hybrid Network)

Data Set: Diabetes Dataset

2.7 SVM and Neural Network

HYBRID- 2

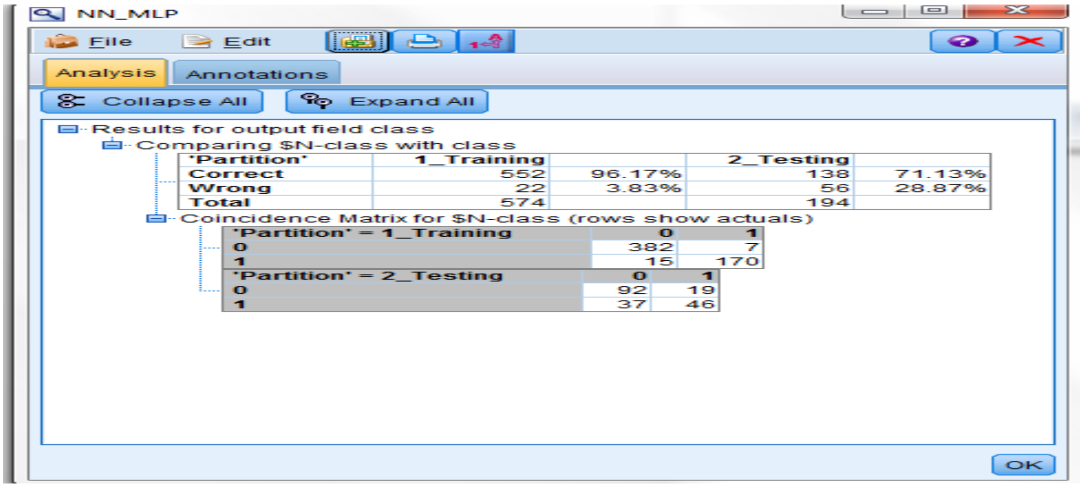


Output: NN = 71.13%

Hybrid SVM=75.26%

Hybrid NN=71.13%

Finding: SVM is performing much better than neural network, even in hybrid network, SVM is outperforms the Neural Network. However in neural netwok using MLP gives better results then neural network with RBF.



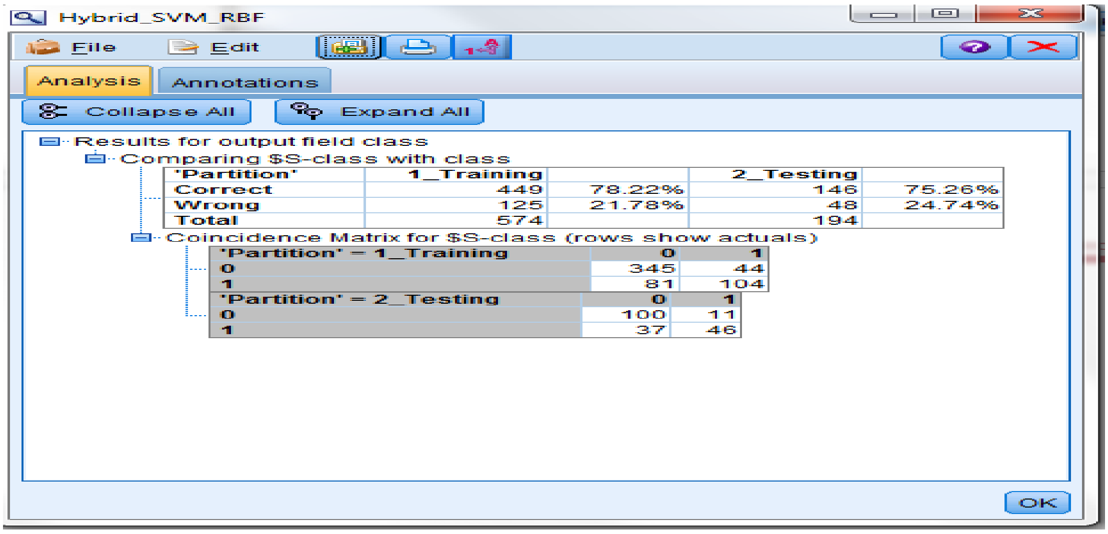
Results for output field class
Comparing $N-class with class

'Partition'	1_Training		2_Testing	
Correct	552	96.17%	138	71.13%
Wrong	22	3.83%	56	28.87%
Total	574		194	

Coincidence Matrix for $N-class (rows show actuals)

'Partition' = 1_Training	0	1
0	382	7
1	15	170

'Partition' = 2_Testing	0	1
0	92	19
1	37	46



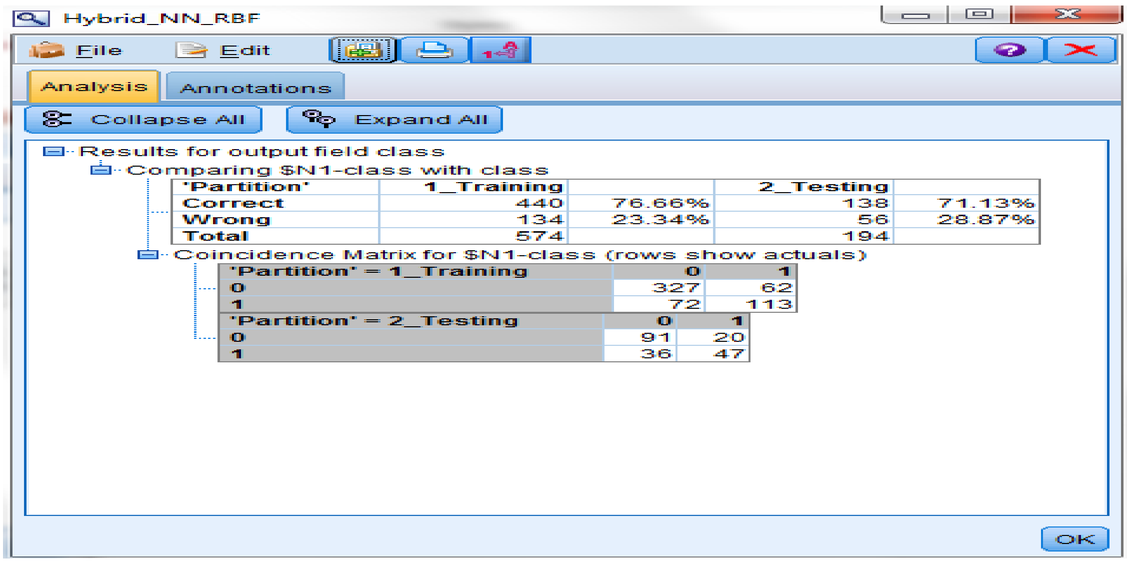
Results for output field class
Comparing $S-class with class

'Partition'	1_Training		2_Testing	
Correct	449	78.22%	146	75.26%
Wrong	125	21.78%	48	24.74%
Total	574		194	

Coincidence Matrix for $S-class (rows show actuals)

'Partition' = 1_Training	0	1
0	345	44
1	81	104

'Partition' = 2_Testing	0	1
0	100	11
1	37	46



Results for output field class
Comparing $N1-class with class

'Partition'	1_Training		2_Testing	
Correct	440	76.66%	138	71.13%
Wrong	134	23.34%	56	28.87%
Total	574		194	

Coincidence Matrix for $N1-class (rows show actuals)

'Partition' = 1_Training	0	1
0	327	62
1	72	113

'Partition' = 2_Testing	0	1
0	91	20
1	36	47

2.8 Network8

Tool used: Rapid Minor

Architecture used: Feed Forward Neural Network (MLP)

Data Set: Diabetes Dataset

back propagation algorithm (multi-layer perceptron)

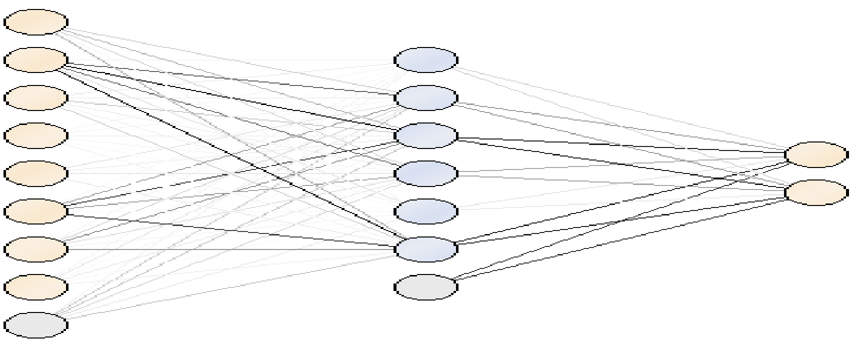
(training cycle: 500, learing rate: 0.3, momentum= 0.2)

accuracy: 76.43% +/- 5.01% (mikro: 76.43%)

	true 1	true 0	class precision
pred. 1	156	69	69.33%
pred. 0	112	431	79.37%
class recall	58.21%	86.20%	

PerformanceVector

```
PerformanceVector:
accuracy: 76.43% +/- 5.01% (mikro: 76.43%)
ConfusionMatrix:
True:   1       0
1:      156     69
0:      112     431
precision: 79.53% +/- 4.13% (mikro: 79.37%) (positive class: 0)
ConfusionMatrix:
True:   1       0
1:      156     69
0:      112     431
recall: 86.20% +/- 5.47% (mikro: 86.20%) (positive class: 0)
ConfusionMatrix:
True:   1       0
1:      156     69
0:      112     431
AUC (optimistic): 0.834 +/- 0.048 (mikro: 0.834) (positive class: 0)
AUC: 0.834 +/- 0.048 (mikro: 0.834) (positive class: 0)
AUC (pessimistic): 0.834 +/- 0.048 (mikro: 0.834) (positive class: 0)
```



2.9 Network9

Tool used: Rapid Minor

Architecture used: Feed Forward Neural Network (Auto MLP)

Data Set: Diabetes Dataset

AUTOMLP

(AutoMLP is a simple algorithm for both learning rate and size adjustment of neural networks during training. The algorithm combines ideas from genetic algorithms and stochastic optimization. It maintains a small ensemble of networks that are trained in parallel with different rates and different numbers of hidden units.)

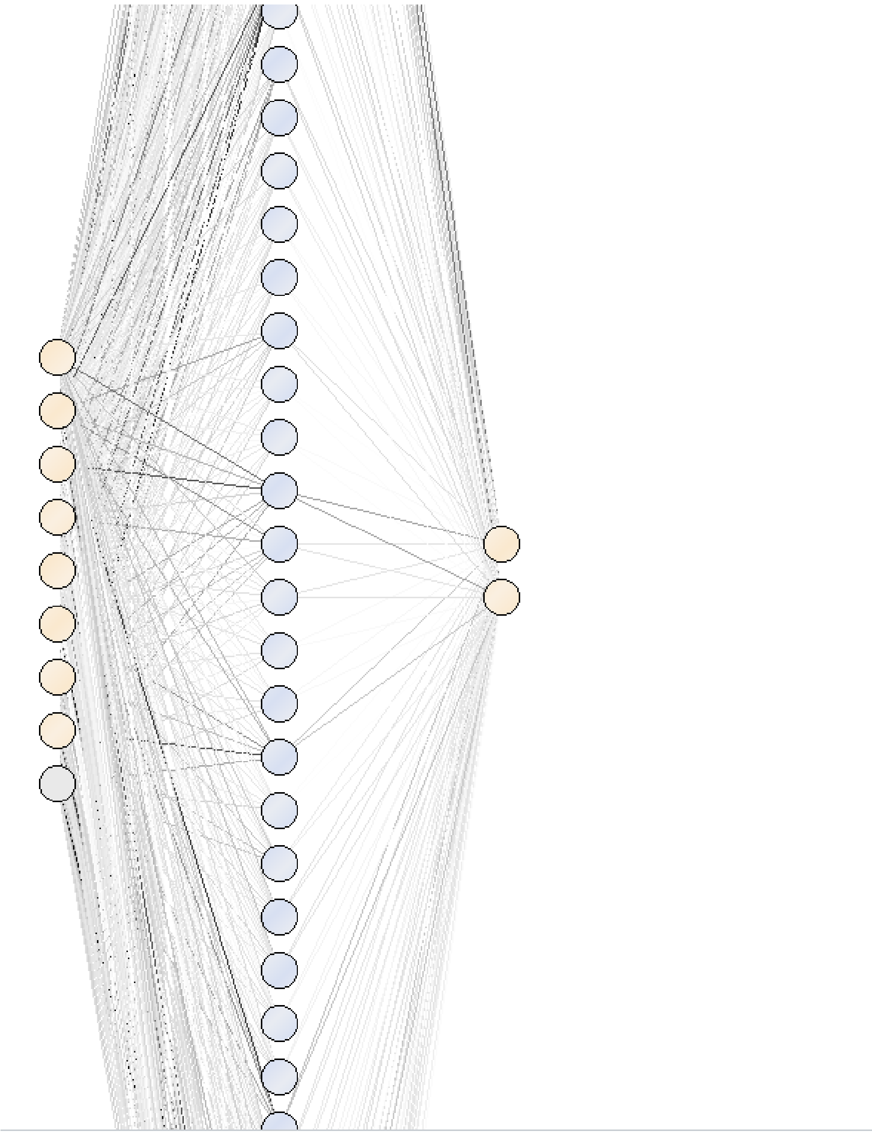
Training cycle: 10, number of generations: 10, number of esemble mlps: 4

accuracy: 76.18% +/- 4.15% (mikro: 76.17%)

	true 1	true 0	class precision
pred. 1	161	76	67.93%
pred. 0	107	424	79.85%
class recall	60.07%	84.80%	

PerformanceVector

```
PerformanceVector:
accuracy: 76.18% +/- 4.15% (mikro: 76.17%)
ConfusionMatrix:
True:   1       0
1:      161     76
0:      107     424
precision: 79.98% +/- 3.45% (mikro: 79.85%) (positive class: 0)
ConfusionMatrix:
True:   1       0
1:      161     76
0:      107     424
recall: 84.80% +/- 5.31% (mikro: 84.80%) (positive class: 0)
ConfusionMatrix:
True:   1       0
1:      161     76
0:      107     424
AUC (optimistic): 0.813 +/- 0.067 (mikro: 0.813) (positive class: 0)
AUC: 0.813 +/- 0.067 (mikro: 0.813) (positive class: 0)
AUC (pessimistic): 0.813 +/- 0.067 (mikro: 0.813) (positive class: 0)
```



3 WINE QUALITY DATA SET

3.1 Network1

Tool used: R Programming Language using nnet package

Architecture used: Feed Forward Neural Network (Multi Layer Perceptron)

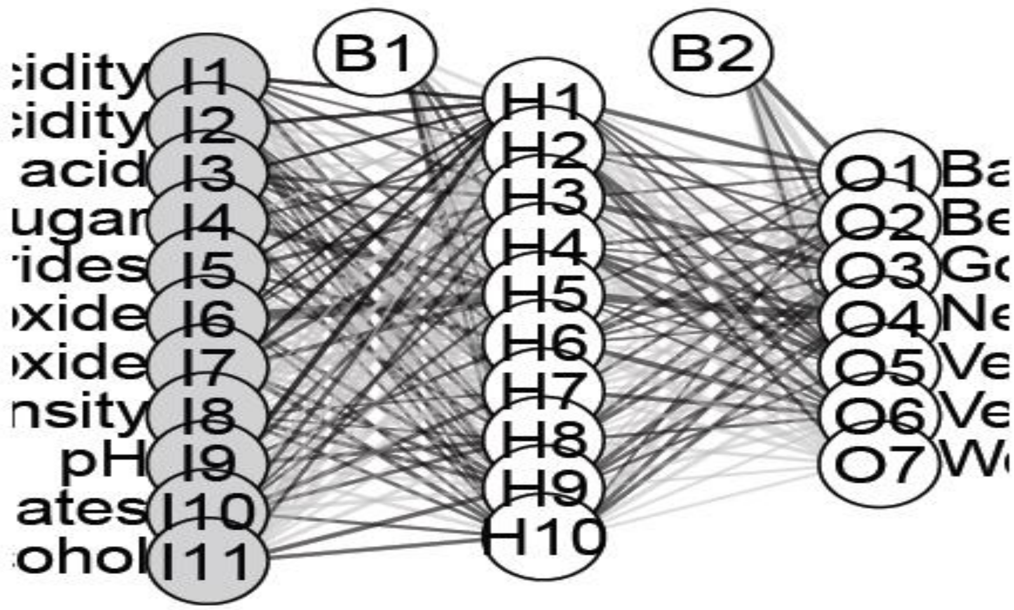
Data Set: Wine Quality

3.1 BP

```
install.packages('nnet')
library(nnet)
wine <- read.csv("winequality-BP.csv")
# shuffle
x <- wine[sample(1:nrow(wine)),]

# Create training and testing data
train <- x[1:3000,]
test <- x[3001:4898,]

model_nnet <- nnet(quality ~ ., data=train, size=10, maxit=1000)

pred<- predict(model_nnet, test, type="class")

table(true=test$quality, predicted=pred)

result<- cbind(test, data.frame(pred))


write.csv(result, file="C:/Users/Akshay.Akshay-PC.000/Documents/resultBP_w.csv")
```

```
converged
>
>
> pred<- predict(model_nnet, test, type="class")
>
>
> table(true=test$quality, predicted=pred)
           predicted
true         Bad Good Neutral Very Bad Very Good Worst
  Bad        287    6     237        3         0     1
  Best         0    1       2        0         0     0
  Good         9   95     196        0         1     0
  Neutral    145   66     522        1         0     1
  Very Bad    29    0      22        5         0     0
  Very Good    2   20      41        0         1     0
  Worst        2    0       2        1         0     0
>
> result<- cbind(test, data.frame(pred))
> |
```

Accuracy= (287+95+522+5+1+0)/ 1698= 53.6%



3.2 Network2

Tool used: R Programming Language using grnn package

Architecture used: General Regression Neural Network

Data Set: Wine Quality

3.2 GRNN

```
#install.packages("grnn")
library("grnn")
Diab=read.csv("~/Documents/wine.csv")
#view(Diab)
#wait
size=nrow(Diab)
length=ncol(Diab)
idex=1:size
positions<-sample(idex,trunc(size*0.75))
training<-Diab[positions,]
testing<-Diab[-positions,1:length-1]
result=Diab[-positions, ]
result$actual=result[,length]
result$predict=-5
nn<-learn(training,variable.column=length)
nn<-smooth(nn,sigma=0.95)
for(i in 1:nrow(testing))
{
  vec<-as.matrix(testing[i,])
  res<-guess(nn,vec)

  #res1<-res
  if(is.nan(res))
  {
    cat("Entry",i,"generated non results")
  }
  else
  {
    result$predict[i]<-round(res)
  }
}
result.size=nrow(result)
a1=result.correct=nrow(result[round(result$predict)==result$actual,])

cat('\nNo. of test cases=',result.size,"\n")
cat("correct preditions=",result.correct,"\n")
cat("Accuracy=",result.correct/result.size*100,"\n")
result2<-cbind(testing,data.frame(result))


#output
#Entry 697 generated non results
# result.size=nrow(result)
# a1=result.correct=nrow(result[round(result$predict)==result$actual,])
```

#cat('\nNo. of test cases=',result.size,"\n")

#No. of test cases= 1225
cat("correct preditions=",result.correct,"\n")
#correct preditions= 728
cat("Accuracy=",result.correct/result.size*100,"\n")
#Accuracy= 59.26531

```
+   else
+   {
+     result$predict[i] <- res
+   }
+ }
Entry  696  Generated NaN result!
> result.size = nrow(result)
> result.correct = nrow(result[round(result$predict) == result$actual,])
> cat("No of test cases = ",result.size,"\n")
No of test cases =  1225
> cat("Correct predictions = ", result.correct ,"\n")
Correct predictions =  726
> cat("Accuracy = ", result.correct / result.size * 100 ,"\n")
Accuracy =  59.26531
> result2<- cbind(testing, data.frame(result))
>
> write.csv(result, file="C:/Users/Akshay.Akshay-PC.000/Documents/resultGRNN_w.csv")
> 
```

Output: #Accuracy= 59.26531
Findings: GRNN performing much better than Back propagation in FF neural network.

3.3 Network3

Tool used: R Programming Language using RSNSS (mlp) package

Architecture used: Feed Forward Neural Network (Multi Layer Perceptron)

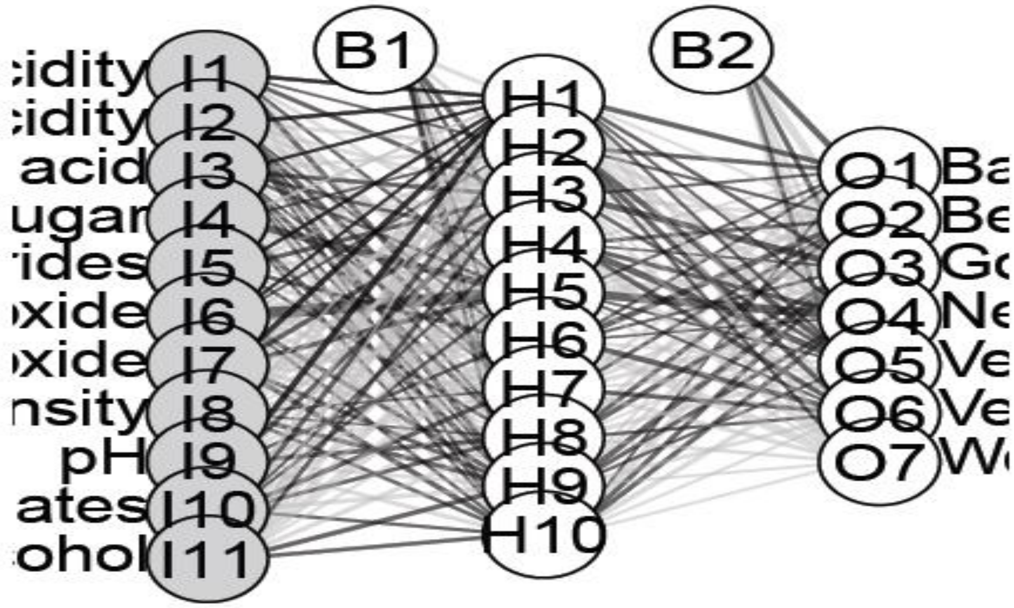
Data Set: Wine Quality

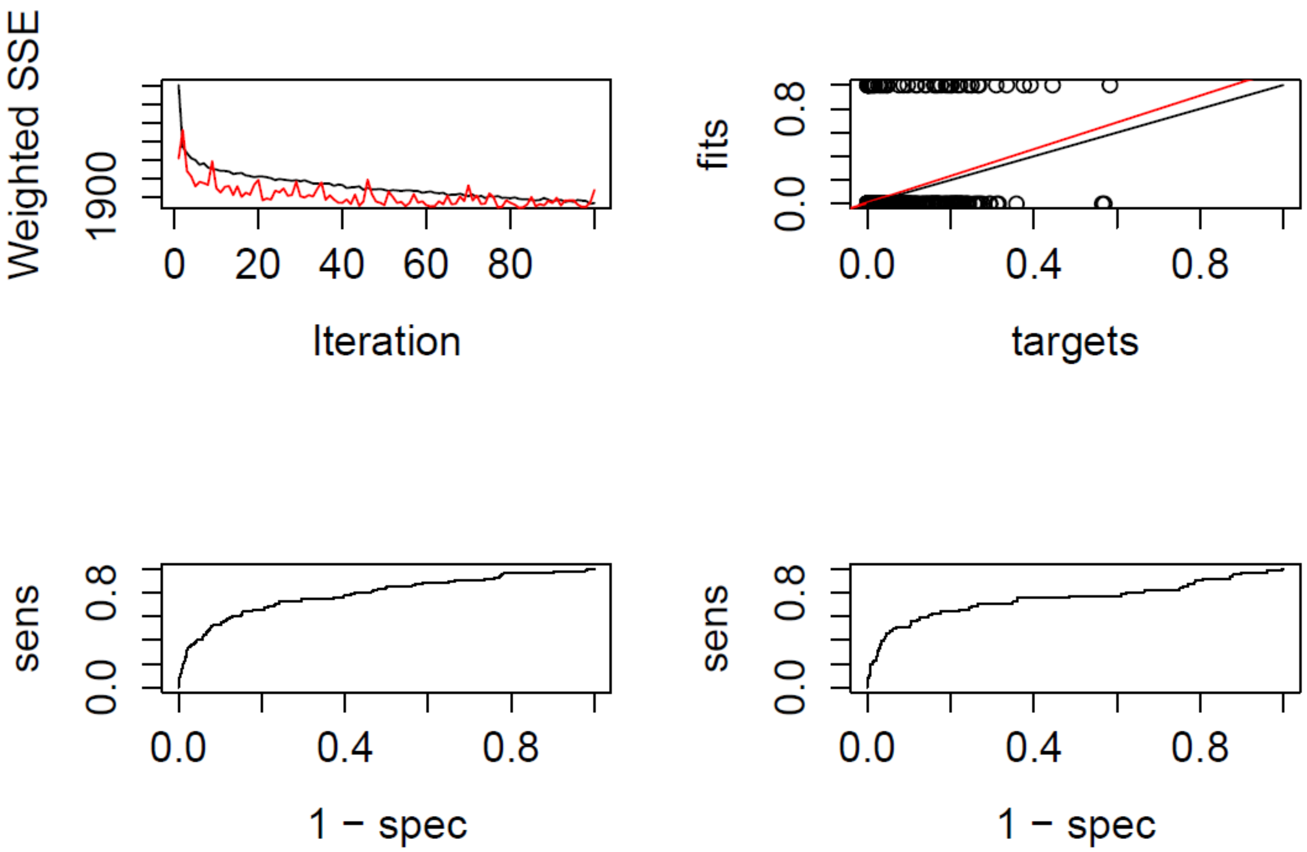
3.3 MLP

```
#########################################################################################
# Wine.csv
#Back Propagation method for Diabities dataset using Multy Layer Perceptron (MLP)
#
# MLPs are fully connected feed- forward networks, and probably the most common
#network architecture in use. Training is usually performed by error backpropagation
#or a related procedure.
#
########################################################################################

#install.packages('RSNNS')
#install.packages('caret')
install.packages('devtools')
install.packages('mlp')
library('RSNNS')
#library('caret')

win=read.csv("~/Documents/wine.csv", head = TRUE)

#shuffle the vector
win <- win[sample(1:nrow(win),length(1:nrow(win))),1:ncol(win)]
#normalizeData 31
winValues <- win[,1:11]
winTargets <- decodeClassLabels(win[,12])
win <- splitForTrainingAndTest(winValues, winTargets, ratio=0.33)
win <- normTrainingAndTestSet(win)
model <- mlp(win$inputsTrain, win$targetsTrain, size=8, learnFuncParams=c(0.1),
        maxit=100, inputsTest=win$inputsTest, targetsTest=win$targetsTest)
summary(model)
model
weightMatrix(model)
extractNetInfo(model)
par(mfrow=c(2,2))
plotIterativeError(model)
predictions <- predict(model,win$inputsTest)
plotRegressionError(predictions[,2], win$targetsTest[,2])
confusionMatrix(win$targetsTrain,fitted.values(model))

#   predictions
#     targets    2    3    4    5
#       1      1    3   10    0
#       2     10   48   43    1
#       3      4  443  506    8
#       4      3  171 1254   49
#       5      0   11  445  137
#       6      0    0   94   35
```

```
#       7     0    0    2    3
confusionMatrix(win$targetsTest,predictions)

#predictions
#   targets   2   3   4   5
#       1     1   2   2   1
#       2     2  35  23   1
#       3     2 237 255   2
#       4     0  86 598  37
#       5     0   5 221  61
#       6     0   0  37   9
plotROC(fitted.values(model)[,2], win$targetsTrain[,2])
plotROC(predictions[,2], win$targetsTest[,2])

#predictions
#   targets   0   3   4
#         1   8   2   4
#         2  73  17  12
#         3 587 144 230
#         4 698  34 745
#         5 337   1 255
#         6  81   0  48
#       7   4   0   1
```

```
weightMatrix(model)
extractNetInfo(model)
par(mfrow=c(2,2))
plotIterativeError(model)
predictions <- predict(model,win$inputsTest)
plotRegressionError(predictions[,2], win$targetsTest[,2])
confusionMatrix(win$targetsTrain,fitted.values(model))

#   predictions
#     targets     2     3     4     5
#         1       1     3    10     0
#         2      10    48    43     1
#         3       4   443   506     8
#         4       3   171  1254    49
#         5       0    11   445   137
#         6       0     0    94    35
#         7       0     0     2     3

confusionMatrix(win$targetsTest,predictions)

#predictions
#   targets    2    3    4    5
#        1     1    2    2    1
#        2     2   35   23    1
#        3     2  237  255    2
#        4     0   86  598   37
#        5     0    5  221   61
#        6     0    0   37    9

plotROC(fitted.values(model)[,2], win$targetsTrain[,2])
plotROC(predictions[,2], win$targetsTest[,2])

#predictions
#    targets    0    3    4
#          1    8    2    4
#          2   73   17   12
#          3  587  144  230
#          4  698   34  745
#          5  337    1  255
#          6   81    0   48
#          7    4    0    1
```


B1
B2
I1
I2
I3
I4
I5
I6
I7
I8
I9
I10
I11
H1
H2
H3
H4
H5
H6
H7
H8
H9
H10
O1
O2
O3
O4
O5
O6
O7
pH


Weighted SSE
1900
0
20
40
60
80
Iteration
fits
0.0
0.8
0.0
0.4
0.8
targets
sens
0.0
0.8
0.0
0.4
0.8
1 − spec
sens
0.0
0.8
0.0
0.4
0.8
1 − spec

3.4 Network4

Tool used: R Programming Language using e1071 package (SVM)

Architecture used: Support Vector Machine

Data Set: Wine Quality Dataset

3.4 SVM CODE with RBF, Polynomial, Linear and radial Kernel

```
##########################################################################
###

#SVMfor Wine dataset

##########################################################################
#

#wine.csv

require(caTools)
require(e1071)

setwd("C:/Users/Akshay.Akshay-PC.000/Downloads/CI SVM")

# Input Data
data <- read.csv("winequality-white.csv",header=TRUE)

# convert output class(target) into a factor
data$quality <- as.factor(data$quality)

# spliting data set into Train & Test
set.seed(2000)
check <- sample.split(data$quality, SplitRatio=0.75)
train <- subset(data,check==TRUE)
test <- subset(data,check==FALSE)

# DEFAULT SVM Model
set.seed(2000)
default <- svm(quality~.,train)
pr <- predict(default,test)
table(pr,test$quality)

#pr   3   4   5   6   7   8   9
#3    0   0   0   0   0   0   0
#4    0   2   1   0   0   0   0
#5    2  27 220 113   9   0   0
#6    3  12 143 410 165  36   1
#7    0   0   0  27  46   8   0
#8    0   0   0   0   0   0   0
#9    0   0   0   0   0   0   0
```

```
# Polynomial SVM
set.seed(2000)
poly <- svm(quality~., train, kernel="polynomial")
pr <- predict(poly,test)
table(pr,test$quality)

#pr   3   4   5   6   7   8   9
#3   1   0   0   0   0   0   0
#4   0   5   2   1   0   0   0
#5   2  19 138  70   1   0   0
#6   2  17 223 468 199  37   1
#7   0   0   1  11  19   7   0
#8   0   0   0   0   1   0   0
#9   0   0   0   0   0   0   0

# radial SVM
set.seed(2000)
poly <- svm(quality~., train, kernel="radial")
pr <- predict(poly,test)
table(pr,test$quality)

#pr   3   4   5   6   7   8   9
#3   0   0   0   0   0   0   0
#4   0   2   1   0   0   0   0
#5   2  27 220 113   9   0   0
#6   3  12 143 410 165  36   1
#7   0   0   0  27  46   8   0
#8   0   0   0   0   0   0   0
#9   0   0   0   0   0   0   0


set.seed(2001)
tune.out <- tune(svm,quality~.,data=train, kernel="radial",ranges=list(cost=c(0.00001,0.001,0.034,0.6,0.98,1.9,3,99)))
# we find that best cost paramters = 99
radial <- svm(quality~., train, kernel="radial", cost=99)
pr <- predict(radial, test)
table(pr,test$quality)
#pr   3   4   5   6   7   8   9
#3   0   0   1   0   0   1   0
#4   1   9   5   7   0   0   0
#5   4  19 220  99   3   2   0
#6   0  11 130 383 106  10   1
#7   0   0   5  58 103  16   0
#8   0   2   3   3   8  15   0
#9   0   0   0   0   0   0   0
```

```
> # DEFAULT SVM Model
> set.seed(2000)
> default <- svm(quality~.,train)
> pr <- predict(default,test)
> table(pr,test$quality)

pr    3   4   5   6   7   8   9
  3   0   0   0   0   0   0   0
  4   0   2   1   0   0   0   0
  5   2  27 220 113   9   0   0
  6   3  12 143 410 165  36   1
  7   0   0   0  27  46   8   0
  8   0   0   0   0   0   0   0
  9   0   0   0   0   0   0   0
> # Polynomial SVM
> set.seed(2000)
> poly <- svm(quality~., train, kernel="polynomial")
> pr <- predict(poly,test)
> table(pr,test$quality)

pr    3   4   5   6   7   8   9
  3   1   0   0   0   0   0   0
  4   0   5   2   1   0   0   0
  5   2  19 138  70   1   0   0
  6   2  17 223 468 199  37   1
  7   0   0   1  11  19   7   0
  8   0   0   0   0   1   0   0
  9   0   0   0   0   0   0   0
> # radial SVM
> set.seed(2000)
> poly <- svm(quality~., train, kernel="radial")
> pr <- predict(poly,test)
> table(pr,test$quality)

pr    3   4   5   6   7   8   9
  3   0   0   0   0   0   0   0
  4   0   2   1   0   0   0   0
  5   2  27 220 113   9   0   0
  6   3  12 143 410 165  36   1
  7   0   0   0  27  46   8   0
  8   0   0   0   0   0   0   0
  9   0   0   0   0   0   0   0
```

```
> set.seed(2001)
> tune.out <- tune(svm,quality~.,data=train, kernel="radial",ranges=list(cost=c(0.00001,0.001,0.034,0.6,0.98,1.9,
3,99)))
> # we find that best cost paramters = 99
> radial <- svm(quality~., train, kernel="radial", cost=99)
> pr <- predict(radial, test)
> table(pr,test$quality)

pr    3   4   5   6   7   8   9
  3   0   0   1   0   0   1   0
  4   1   9   5   7   0   0   0
  5   4  19 220  99   3   2   0
  6   0  11 130 383 106  10   1
  7   0   0   5  58 103  16   0
  8   0   2   3   3   8  15   0
  9   0   0   0   0   0   0   0
>
```

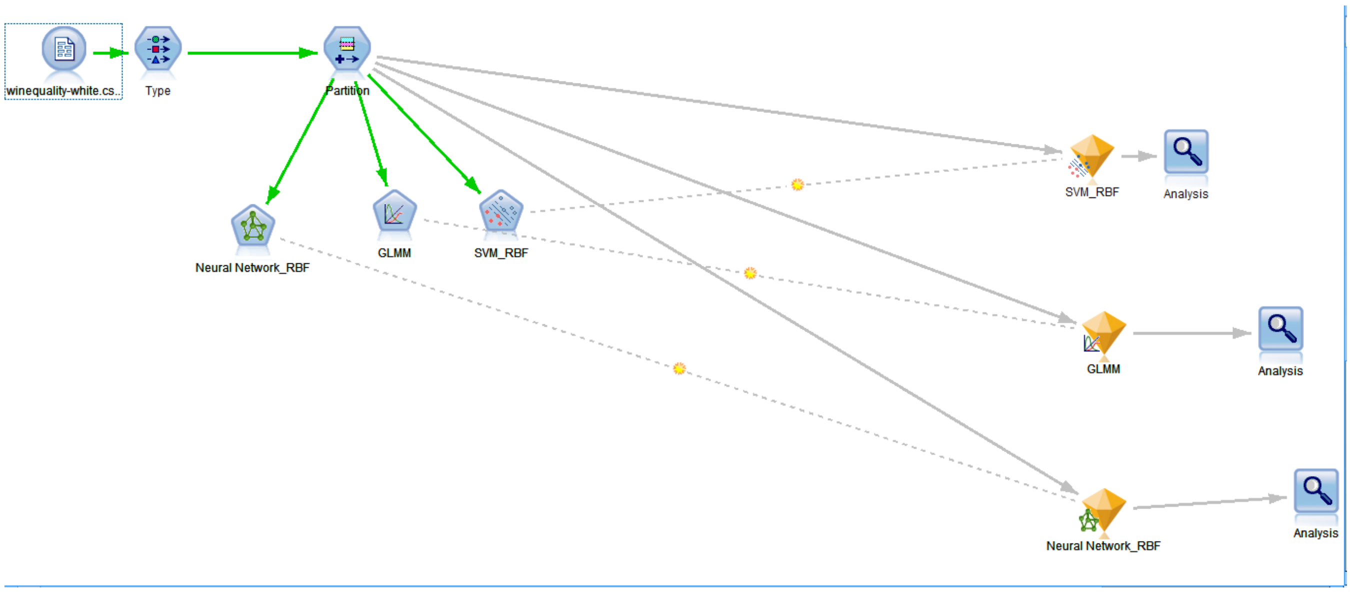
3.5 Network5

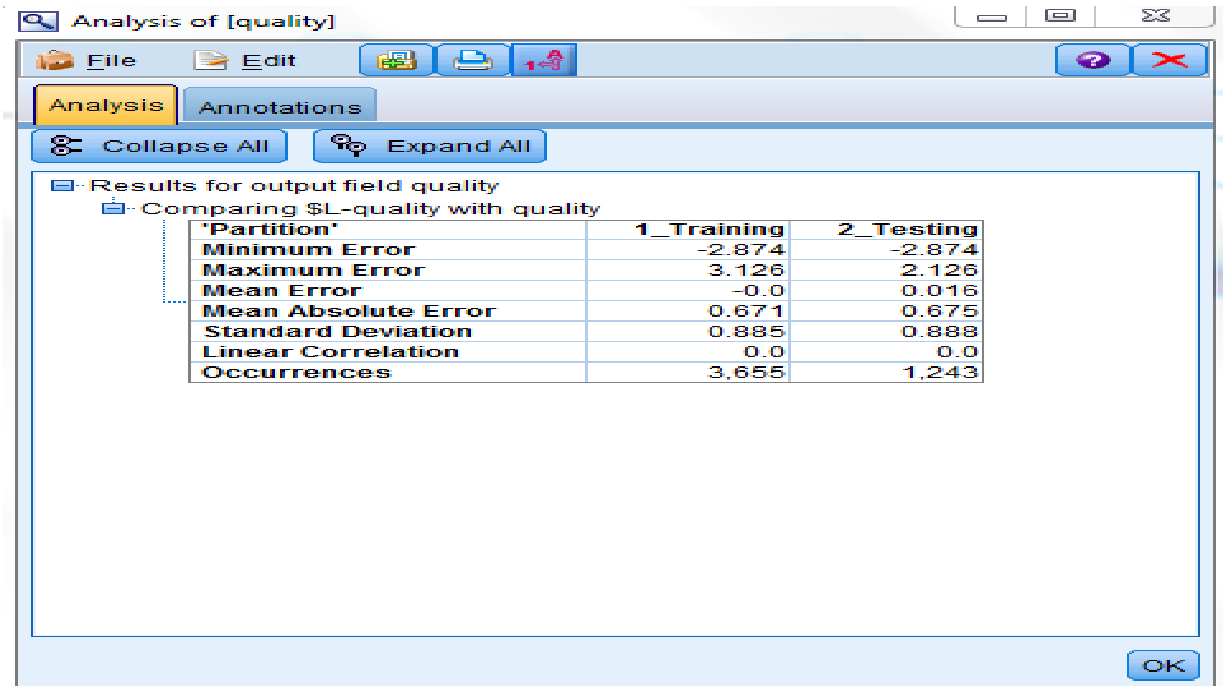
Tool used: IBM SPSS Modular

Architecture used: Support Vector Machine & Neural Network separately

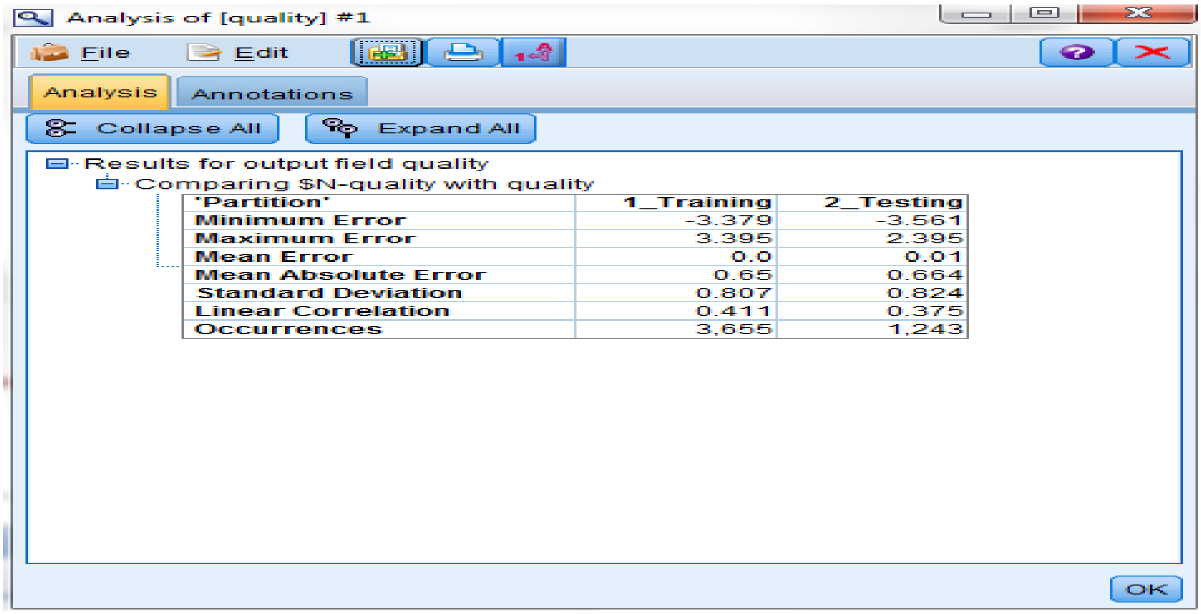
Data Set: Diabetes Dataset

3.5 SVM and Neural Network

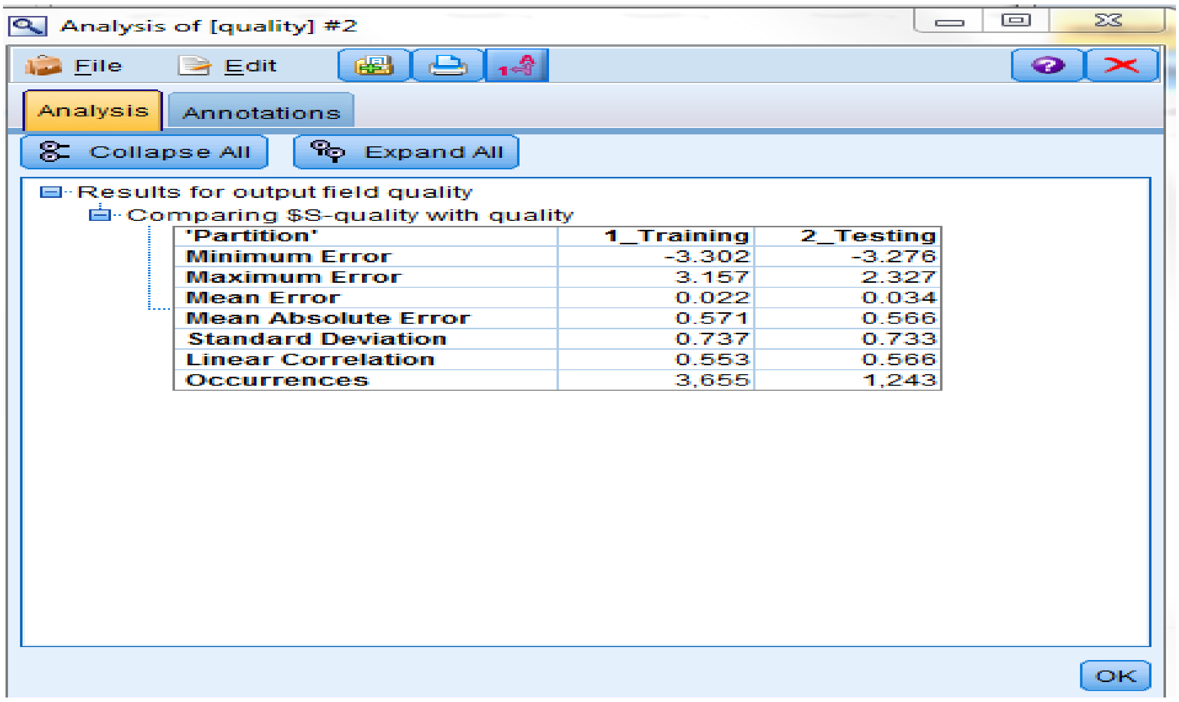




'Partition'	1_Training	2_Testing
Minimum Error	-2.874	-2.874
Maximum Error	3.126	2.126
Mean Error	-0.0	0.016
Mean Absolute Error	0.671	0.675
Standard Deviation	0.885	0.888
Linear Correlation	0.0	0.0
Occurrences	3,655	1,243



'Partition'	1_Training	2_Testing
Minimum Error	-3.379	-3.561
Maximum Error	3.395	2.395
Mean Error	0.0	0.01
Mean Absolute Error	0.65	0.664
Standard Deviation	0.807	0.824
Linear Correlation	0.411	0.375
Occurrences	3,655	1,243



'Partition'	1_Training	2_Testing
Minimum Error	-3.302	-3.276
Maximum Error	3.157	2.327
Mean Error	0.022	0.034
Mean Absolute Error	0.571	0.566
Standard Deviation	0.737	0.733
Linear Correlation	0.553	0.566
Occurrences	3,655	1,243

3.6 Network6 (Hybrid 1 SVM Polynomial with Neural Network MLP)

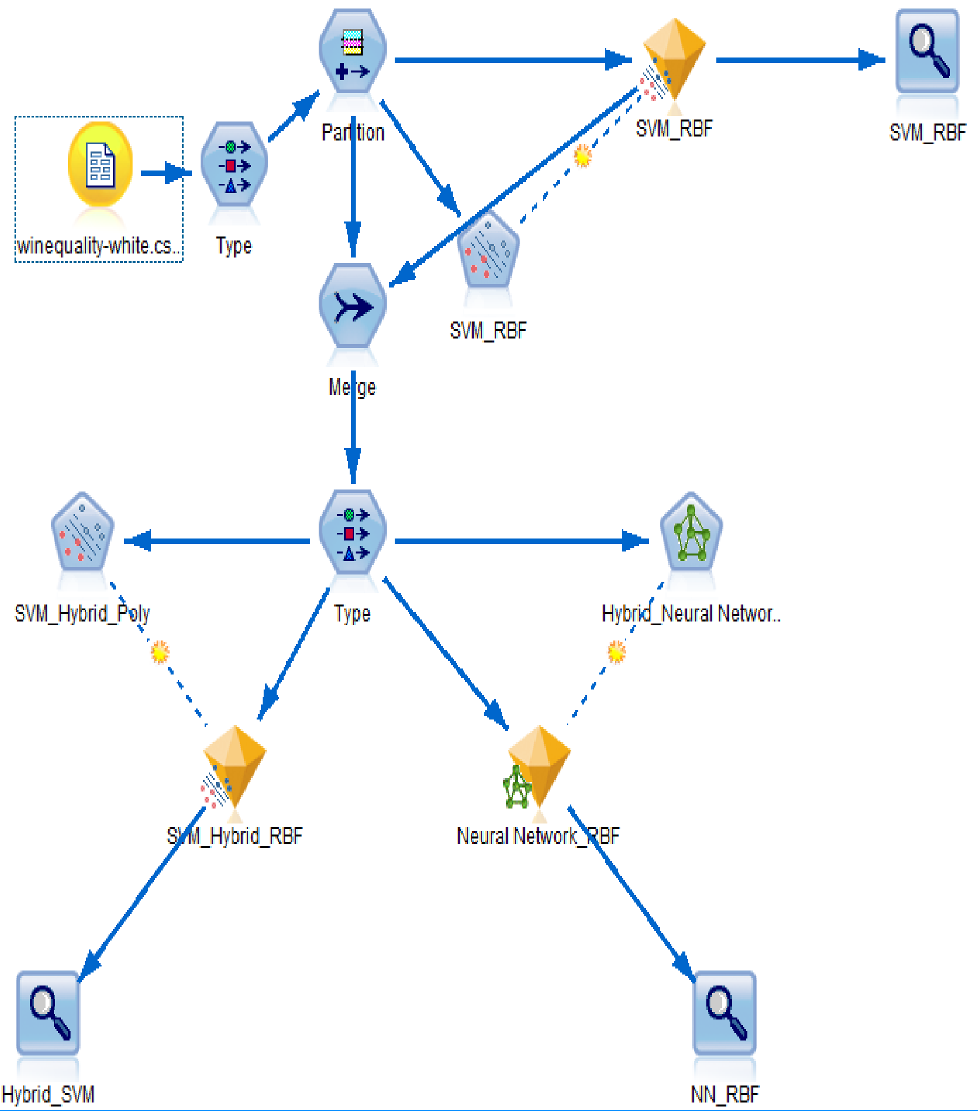
Tool used: IBM SPSS Modular

Architecture used: Support Vector Machine & Neural Network (Hybrid Network)

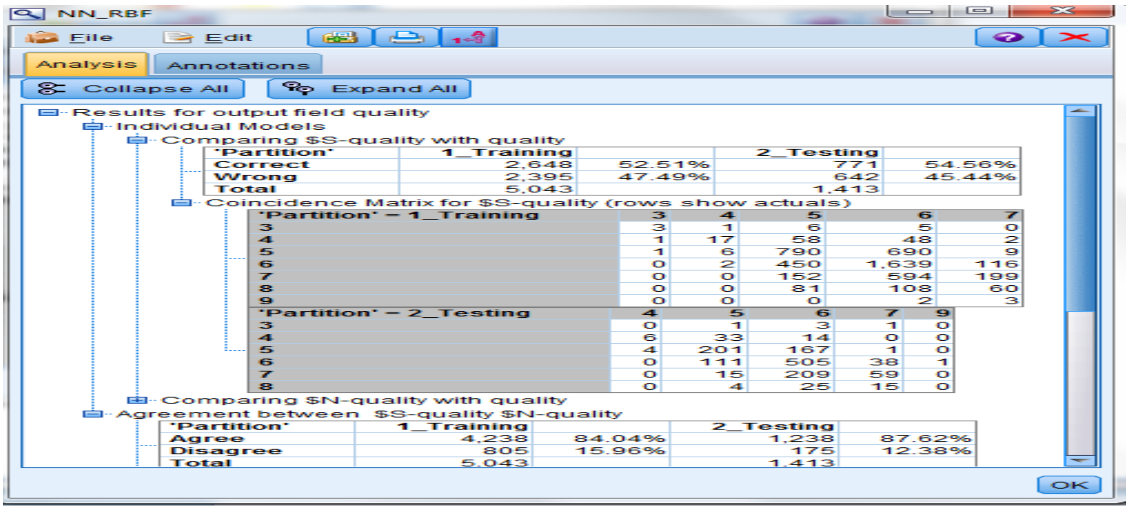
Data Set: Diabetes Dataset

3.6 SVM and Neural Network

HYBRID- 1



Finding: SVM is performing much better than neural network. However Polynomial and RBF kernel have the same result.



NN_RBF

File Edit

Analysis Annotations

Collapse All Expand All

Results for output field quality

Individual Models

Comparing $S-quality with quality

'Partition'	1_Training		2_Testing	
Correct	2,648	52.51%	771	54.56%
Wrong	2,395	47.49%	642	45.44%
Total	5,043		1,413	

Coincidence Matrix for $S-quality (rows show actuals)

'Partition' = 1_Training	3	4	5	6	7
3	3	1	6	5	0
4	1	17	58	48	2
5	1	6	790	690	9
6	0	2	450	1,639	116
7	0	0	152	594	199
8	0	0	81	108	60
9	0	0	0	2	3

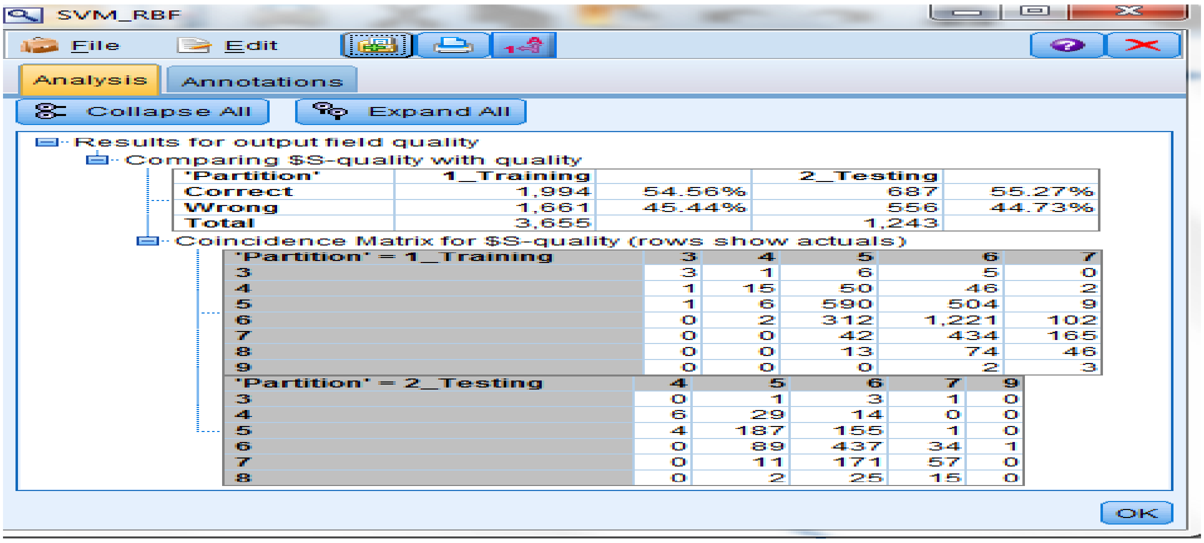
'Partition' = 2_Testing	4	5	6	7	9
3	0	1	3	1	0
4	6	33	14	0	0
5	4	201	167	1	0
6	0	111	505	38	1
7	0	15	209	59	0
8	0	4	25	15	0

Comparing $N-quality with quality

Agreement between $S-quality $N-quality

'Partition'	1_Training		2_Testing	
Agree	4,238	84.04%	1,238	87.62%
Disagree	805	15.96%	175	12.38%
Total	5,043		1,413	

OK



SVM_RBF

File Edit

Analysis Annotations

Collapse All Expand All

Results for output field quality

Comparing $S-quality with quality

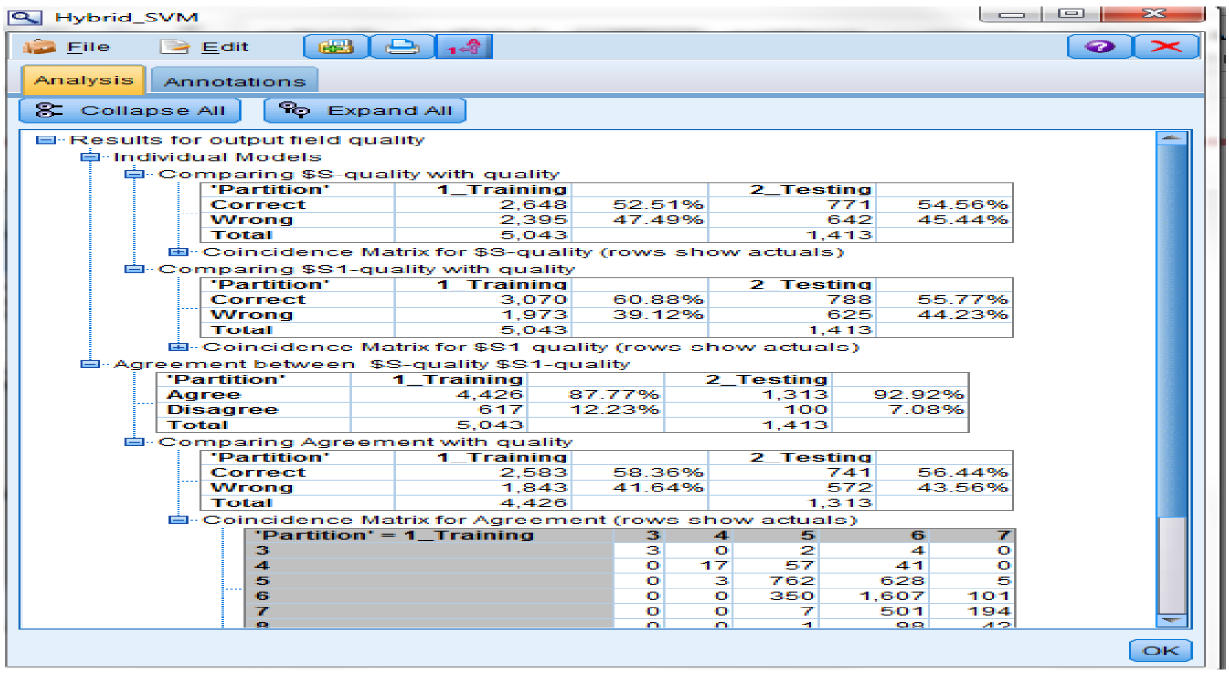
'Partition'	1_Training		2_Testing	
Correct	1,994	54.56%	687	55.27%
Wrong	1,661	45.44%	556	44.73%
Total	3,655		1,243	

Coincidence Matrix for $S-quality (rows show actuals)

'Partition' = 1_Training	3	4	5	6	7
3	3	1	6	5	0
4	1	15	50	46	2
5	1	6	590	504	9
6	0	2	312	1,221	102
7	0	0	42	434	165
8	0	0	13	74	46
9	0	0	0	2	3

'Partition' = 2_Testing	4	5	6	7	9
3	0	1	3	1	0
4	6	29	14	0	0
5	4	187	155	1	0
6	0	89	437	34	1
7	0	11	171	57	0
8	0	2	25	15	0

OK



Hybrid_SVM

File Edit

Analysis Annotations

Collapse All Expand All

Results for output field quality

Individual Models

Comparing $S-quality with quality

'Partition'	1_Training		2_Testing	
Correct	2,648	52.51%	771	54.56%
Wrong	2,395	47.49%	642	45.44%
Total	5,043		1,413	

Coincidence Matrix for $S-quality (rows show actuals)

Comparing $S1-quality with quality

'Partition'	1_Training		2_Testing	
Correct	3,070	60.88%	788	55.77%
Wrong	1,973	39.12%	625	44.23%
Total	5,043		1,413	

Coincidence Matrix for $S1-quality (rows show actuals)

Agreement between $S-quality $S1-quality

'Partition'	1_Training		2_Testing	
Agree	4,426	87.77%	1,313	92.92%
Disagree	617	12.23%	100	7.08%
Total	5,043		1,413	

Comparing Agreement with quality

'Partition'	1_Training		2_Testing	
Correct	2,583	58.36%	741	56.44%
Wrong	1,843	41.64%	572	43.56%
Total	4,426		1,313	

Coincidence Matrix for Agreement (rows show actuals)

'Partition' = 1_Training	3	4	5	6	7
3	3	0	2	4	0
4	0	17	57	41	0
5	0	3	762	628	5
6	0	0	350	1,607	101
7	0	0	7	501	194
8	0	0	1	[illegible]	[illegible]

OK

3.7Network7

Tool used: Rapid Minor

Architecture used: Feed Forward Neural Network (MLP)

Data Set: Wine Quality Dataset

back propagation algorithm (multi-layer perceptron)

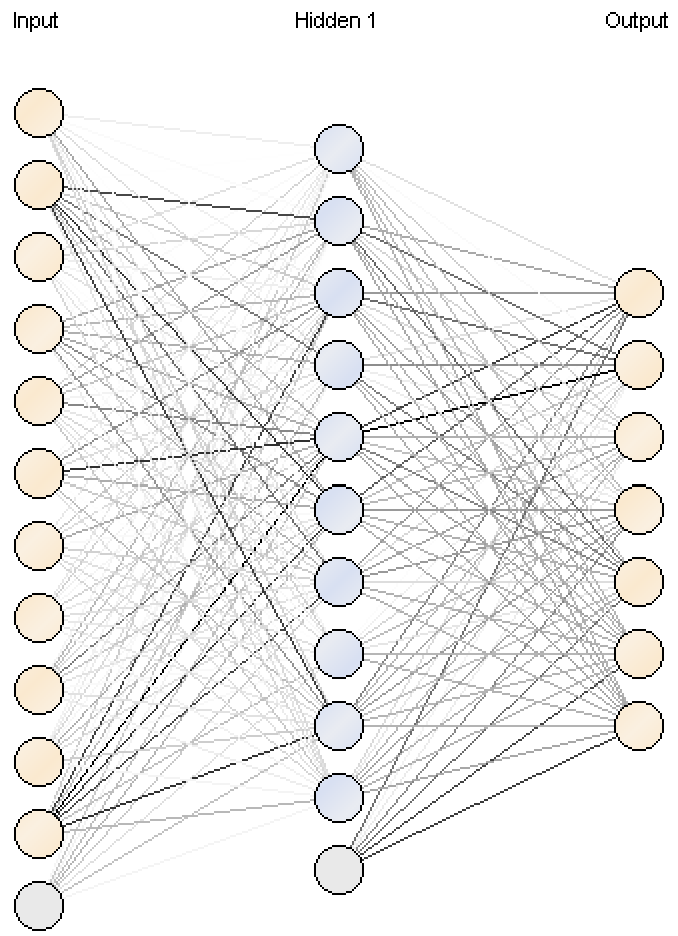
(training cycle: 600, learning rate: 0.3, momentum: 0.2)

accuracy: 54.43% +/- 2.04% (mikro: 54.43%)

	true 6	true 5	true 7	true 8	true 4	true 3	true 9	class precision
pred. 6	1632	634	611	103	58	9	2	53.53%
pred. 5	428	809	44	11	102	9	0	57.66%
pred. 7	138	14	225	61	3	2	3	50.45%
pred. 8	0	0	0	0	0	0	0	0.00%
pred. 4	0	0	0	0	0	0	0	0.00%
pred. 3	0	0	0	0	0	0	0	0.00%
pred. 9	0	0	0	0	0	0	0	0.00%
class recall	74.25%	55.53%	25.57%	0.00%	0.00%	0.00%	0.00%	

PerformanceVector

```
PerformanceVector:
accuracy: 54.43% +/- 2.04% (mikro: 54.43%)
ConfusionMatrix:
True:   6       5       7       8       4       3       9
6:      1632    634     611     103     58      9       2
5:      428     809     44      11      102     9       0
7:      138     14      225     61      3       2       3
8:      0       0       0       0       0       0       0
4:      0       0       0       0       0       0       0
3:      0       0       0       0       0       0       0
9:      0       0       0       0       0       0       0
kappa: 0.264 +/- 0.030 (mikro: 0.264)
ConfusionMatrix:
True:   6       5       7       8       4       3       9
6:      1632    634     611     103     58      9       2
5:      428     809     44      11      102     9       0
7:      138     14      225     61      3       2       3
8:      0       0       0       0       0       0       0
4:      0       0       0       0       0       0       0
3:      0       0       0       0       0       0       0
9:      0       0       0       0       0       0       0
```


Input
Hidden 1
Output

3.8 Network8

Tool used: Rapid Minor

Architecture used: Feed Forward Neural Network (Auto MLP)

Data Set: Wine Quality Dataset

7.2 AUTOMLP

(AutoMLP is a simple algorithm for both learning rate and size adjustment of neural networks during training. The algorithm combines ideas from genetic algorithms and stochastic optimization. It maintains a small ensemble of networks that are trained in parallel with different rates and different numbers of hidden units.)

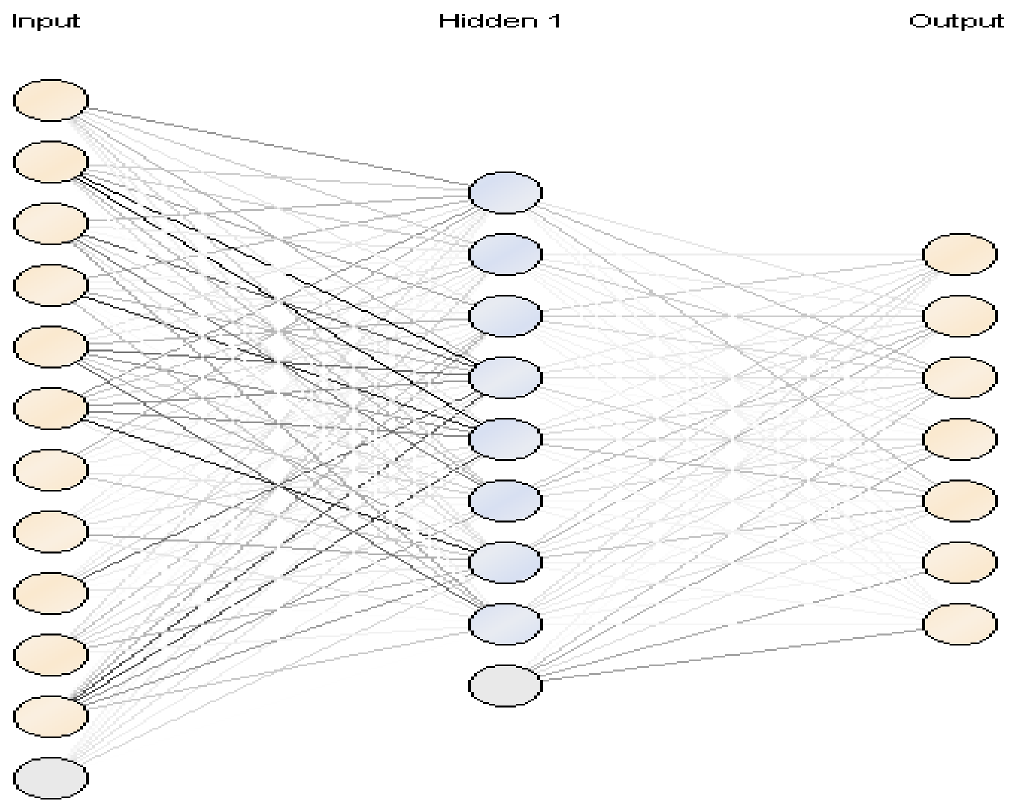
Training cycle: 10, number of generations: 10, number of esemble mlps: 4

accuracy: 54.72% +/- 2.43% (mikro: 54.72%)

	true 6	true 5	true 7	true 8	true 4	true 3	true 9	class precision
pred. 6	1553	576	572	100	54	10	1	54.19%
pred. 5	469	847	38	7	90	8	1	58.01%
pred. 7	168	16	265	67	3	0	3	50.77%
pred. 8	1	1	4	0	1	1	0	0.00%
pred. 4	7	17	1	1	15	1	0	35.71%
pred. 3	0	0	0	0	0	0	0	0.00%
pred. 9	0	0	0	0	0	0	0	0.00%
class recall	70.66%	58.13%	30.11%	0.00%	9.20%	0.00%	0.00%	

PerformanceVector

```
PerformanceVector:
accuracy: 54.72% +/- 2.43% (mikro: 54.72%)
ConfusionMatrix:
True:   6       5       7       8       4       3       9
6:      1553    576     572     100     54      10      1
5:      469     847     38      7       90      8       1
7:      168     16      265     67      3       0       3
8:      1       1       4       0       1       1       0
4:      7       17      1       1       15      1       0
3:      0       0       0       0       0       0       0
9:      0       0       0       0       0       0       0
kappa: 0.279 +/- 0.050 (mikro: 0.280)
ConfusionMatrix:
True:   6       5       7       8       4       3       9
6:      1553    576     572     100     54      10      1
5:      469     847     38      7       90      8       1
7:      168     16      265     67      3       0       3
8:      1       1       4       0       1       1       0
4:      7       17      1       1       15      1       0
3:      0       0       0       0       0       0       0
9:      0       0       0       0       0       0       0
```



3.10 Hybrid Using EXCEL

Data Set	Wine
Corrected Predicted	468
Total Number	1225
Accurancy	38.20408

Data Set	Diabetes	
Actual/Hybrid	0	1
0	79	51
1	17	45
Correct Prediction	124	
Acurany	64.58333	

Findings: It was a nice experince of developing different neural network architectureon different tools. We learned a lot about the learning and tarining methods of neural networks. Considering the datasets, Diabetes data set have much more accurate result in all of the networks. The reasone may be the posibility of the classification in to two classes i.e. either positive or negetive. However SVM gives better result as compare to our balck box neural network. SVM using its kernel to solve non linear classification problems which are quite difficult in neural networks. GRNN was also better than MLP.

In case of wine dataset, multiple class clasiification was the criteria of the classification. Total 6 types of qualities of wine is available in which the given data is to be classified. This is little bit complex problem as compare to diabetes data set. Here in this dataset, SVM perform much better among neural network and GRNN. However GRNN performed well as compared to neural network.