Computational Intelligence I Assignment

Gaelan Gu, Sunil Prakash

# Neural Network Ensemble for Diabetes Dataset

We will be using R to construct our Neural Network (NN) ensemble for the Diabetes dataset. In this dataset, we are attempting to predict the class variable, which we named as *positive.test*, to determine if a patient is positive for diabetes based on a given set of 8 continuous attributes.

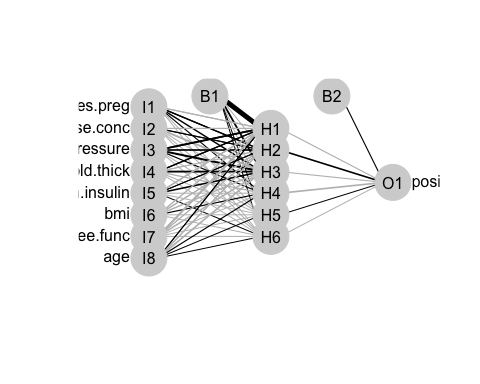
## no.of.times.preg plasma.glucose.conc diastolic.pressure  
## Min. : 0.000 Min. : 0.0 Min. : 0.00   
## 1st Qu.: 1.000 1st Qu.: 99.0 1st Qu.: 62.00   
## Median : 3.000 Median :117.0 Median : 72.00   
## Mean : 3.845 Mean :120.9 Mean : 69.11   
## 3rd Qu.: 6.000 3rd Qu.:140.2 3rd Qu.: 80.00   
## Max. :17.000 Max. :199.0 Max. :122.00   
## triceps.skin.fold.thick serum.insulin bmi   
## Min. : 0.00 Min. : 0.0 Min. : 0.00   
## 1st Qu.: 0.00 1st Qu.: 0.0 1st Qu.:27.30   
## Median :23.00 Median : 30.5 Median :32.00   
## Mean :20.54 Mean : 79.8 Mean :31.99   
## 3rd Qu.:32.00 3rd Qu.:127.2 3rd Qu.:36.60   
## Max. :99.00 Max. :846.0 Max. :67.10   
## diab.pedigree.func age positive.test  
## Min. :0.0780 Min. :21.00 0:500   
## 1st Qu.:0.2437 1st Qu.:24.00 1:268   
## Median :0.3725 Median :29.00   
## Mean :0.4719 Mean :33.24   
## 3rd Qu.:0.6262 3rd Qu.:41.00   
## Max. :2.4200 Max. :81.00

We have factorized our class variable and ascertained that there are no missing values in the dataset. We split our dataset into training and testing sets using the ratio 70:30. We will run 2 neural networks on the training set.

## Single-Layer Neural Network

Using a single-layer NN, we used 6 neurons for the hidden layer, set the maximum number of iterations at 10,000 and a learning rate of 0.0001. We keep the learning rate small in order not to exceed the local minimum.

## a 8-6-1 network with 61 weights  
## inputs: no.of.times.preg plasma.glucose.conc diastolic.pressure triceps.skin.fold.thick serum.insulin bmi diab.pedigree.func age   
## output(s): positive.test   
## options were - entropy fitting decay=1e-04



6 neurons are used in the hidden layer as this configuration produced the best accuracy, and also because it is optimally between the number of independent variables (8) and output node (1).

## Confusion Matrix and Statistics  
##   
## predicted  
## true 0 1  
## 0 141 9  
## 1 71 9  
##   
## Accuracy : 0.6522   
## 95% CI : (0.5868, 0.7136)  
## No Information Rate : 0.9217   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.0641   
## Mcnemar's Test P-Value : 9.104e-12   
##   
## Sensitivity : 0.6651   
## Specificity : 0.5000   
## Pos Pred Value : 0.9400   
## Neg Pred Value : 0.1125   
## Prevalence : 0.9217   
## Detection Rate : 0.6130   
## Detection Prevalence : 0.6522   
## Balanced Accuracy : 0.5825   
##   
## 'Positive' Class : 0   
##

Accuracy of **65.2%** achieved on test set with single-layer neural network.

## Neural Networks with Feature Extraction

We will apply principal component analysis (PCA) to the training set before applying a single-layer NN to it. We will also keep the number of neurons, maxit and decay values the same as before.

## Neural Network Model with PCA Pre-Processing  
##   
## Created from 538 samples and 8 variables  
## PCA needed 8 components to capture 99 percent of the variance  
##   
## a 8-6-2 network with 68 weights  
## options were - decay=1e-04

## Confusion Matrix and Statistics  
##   
## predicted  
## true 0 1  
## 0 130 20  
## 1 39 41  
##   
## Accuracy : 0.7435   
## 95% CI : (0.6819, 0.7986)  
## No Information Rate : 0.7348   
## P-Value [Acc > NIR] : 0.41574   
##   
## Kappa : 0.4014   
## Mcnemar's Test P-Value : 0.01911   
##   
## Sensitivity : 0.7692   
## Specificity : 0.6721   
## Pos Pred Value : 0.8667   
## Neg Pred Value : 0.5125   
## Prevalence : 0.7348   
## Detection Rate : 0.5652   
## Detection Prevalence : 0.6522   
## Balanced Accuracy : 0.7207   
##   
## 'Positive' Class : 0   
##

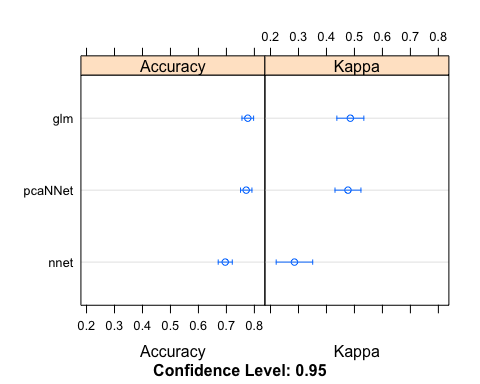
A higher accuracy rate of **74.4%** was achieved on the test set using a single-layer NN with a PCA step (PCANN).

## Ensemble Learning

Let's see if the accuracy rates can be improved with a NN ensemble with the models. The type of ensemble which we are using is of the stacking type. We train the ensemble for combining the predictions of the individual learning algorithms to achieve a higher accuracy.

The train control which we will use in training is a repeated 10-fold cross-validation iterated 3 times. We train the individual models first.

## Call:  
## summary.resamples(object = result1)  
##   
## Models: glm, nnet, pcaNNet   
## Number of resamples: 30   
##   
## Accuracy   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## glm 0.6481 0.7593 0.7778 0.7764 0.8148 0.8704 0  
## nnet 0.5741 0.6478 0.6852 0.6958 0.7170 0.8519 0  
## pcaNNet 0.6481 0.7407 0.7778 0.7708 0.8113 0.8704 0  
##   
## Kappa   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## glm 0.19970 0.4352 0.5068 0.4853 0.5735 0.7123 0  
## nnet -0.04722 0.1693 0.2838 0.2855 0.3316 0.6752 0  
## pcaNNet 0.21920 0.4191 0.5008 0.4768 0.5554 0.6897 0



## A glm ensemble of 2 base models: glm, nnet, pcaNNet  
##   
## Ensemble results:  
## Generalized Linear Model   
##   
## 1614 samples  
## 3 predictor  
## 2 classes: 'X0', 'X1'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 3 times)   
## Summary of sample sizes: 1453, 1452, 1453, 1452, 1453, 1453, ...   
## Resampling results:  
##   
## Accuracy Kappa   
## 0.7722184 0.4732185

The glm-stacked ensemble mode performs at an accuracy of **77.2%**, which is higher than the individual models. As stacking works well by combining multiple models, it will be simpler and easy to implement by using a single stacking script for more models.

# Neural Network Ensemble for Wine Quality Dataset

For the Wine dataset, we are attempting to predict the *quality* variable, which ranges from 0 to 10. This will be a regression exercise using 11 continuous predictor variables, which describe several properties of the wine content.

## fixed.acidity volatile.acidity citric.acid residual.sugar   
## Min. : 3.800 Min. :0.0800 Min. :0.0000 Min. : 0.600   
## 1st Qu.: 6.300 1st Qu.:0.2100 1st Qu.:0.2700 1st Qu.: 1.700   
## Median : 6.800 Median :0.2600 Median :0.3200 Median : 5.200   
## Mean : 6.855 Mean :0.2782 Mean :0.3342 Mean : 6.391   
## 3rd Qu.: 7.300 3rd Qu.:0.3200 3rd Qu.:0.3900 3rd Qu.: 9.900   
## Max. :14.200 Max. :1.1000 Max. :1.6600 Max. :65.800   
## chlorides free.sulfur.dioxide total.sulfur.dioxide  
## Min. :0.00900 Min. : 2.00 Min. : 9.0   
## 1st Qu.:0.03600 1st Qu.: 23.00 1st Qu.:108.0   
## Median :0.04300 Median : 34.00 Median :134.0   
## Mean :0.04577 Mean : 35.31 Mean :138.4   
## 3rd Qu.:0.05000 3rd Qu.: 46.00 3rd Qu.:167.0   
## Max. :0.34600 Max. :289.00 Max. :440.0   
## density pH sulphates alcohol   
## Min. :0.9871 Min. :2.720 Min. :0.2200 Min. : 8.00   
## 1st Qu.:0.9917 1st Qu.:3.090 1st Qu.:0.4100 1st Qu.: 9.50   
## Median :0.9937 Median :3.180 Median :0.4700 Median :10.40   
## Mean :0.9940 Mean :3.188 Mean :0.4898 Mean :10.51   
## 3rd Qu.:0.9961 3rd Qu.:3.280 3rd Qu.:0.5500 3rd Qu.:11.40   
## Max. :1.0390 Max. :3.820 Max. :1.0800 Max. :14.20   
## quality   
## Min. :3.000   
## 1st Qu.:5.000   
## Median :6.000   
## Mean :5.878   
## 3rd Qu.:6.000   
## Max. :9.000

Similarly, the dataset will be split into training and testing sets using the same ratio as before. We conduct an additional step of scaling the predictor variables using the minimum and maximum values of each column.

# Support Vector Machine (SVM) with Radial Basis Function (RBF) Kernel

We will first build a SVM with a RBF kernel, but we leave the determination of the hyperparameters up to the kernel. This is done by setting *kpar* parameter to "automatic".

## Support Vector Machine object of class "ksvm"   
##   
## SV type: eps-svr (regression)   
## parameter : epsilon = 0.1 cost C = 1   
##   
## Gaussian Radial Basis kernel function.   
## Hyperparameter : sigma = 0.0774829127912879   
##   
## Number of Support Vectors : 2986   
##   
## Objective Function Value : -1589.486   
## Training error : 0.5218

The kernel recommends a sigma hyperparameter of 0.077. In the case of an epsilon regression, the parameters recommended are epsilon = 0.1 and cost = 1.

## RMSE Rsquared   
## 0.16278898 0.06573323

We obtain a root mean squared error (RMSE) of **0.16** from the SVM with RBF kernel.

## Stacked AutoEncoder Deep Neural Network

Next we will use an autoencoder deep neural network for training, however we will only be using one hidden layer with 8 neurons. The purpose of the autoencoder in this case is to perform feature extraction on the training set. It tries to learn aspects of the input in the hidden layer so that it is able to reconstruct the output based on those representations of the input.

Learning rate will be set to a small figure to avoid missing the local minima.

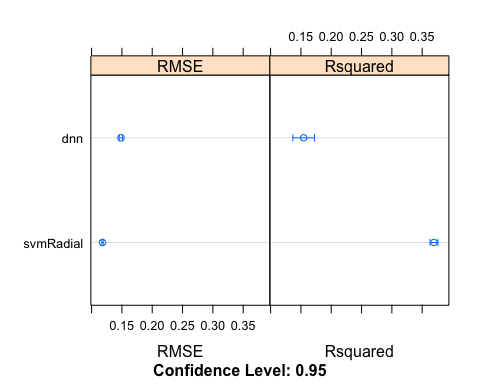
## Length Class Mode   
## input\_dim 1 -none- numeric   
## output\_dim 1 -none- numeric   
## hidden 1 -none- numeric   
## size 3 -none- numeric   
## activationfun 1 -none- character  
## learningrate 1 -none- numeric   
## momentum 1 -none- numeric   
## learningrate\_scale 1 -none- numeric   
## hidden\_dropout 1 -none- numeric   
## visible\_dropout 1 -none- numeric   
## output 1 -none- character  
## W 2 -none- list   
## vW 2 -none- list   
## B 2 -none- list   
## vB 2 -none- list   
## post 3 -none- list   
## pre 3 -none- list   
## e 30 -none- numeric   
## L 350 -none- numeric

## RMSE Rsquared   
## 0.1481129 0.0352256

We obtain a slightly improved RMSE of **0.14** in this case.

## Ensemble Learning

We will stack the models in this ensemble to determine if we can obtain an even lower RMSE. 25 bootstrapped samples will be obtained from the training set to train the models and this is reflected in the train control parameter.

## Call:  
## summary.resamples(object = result2)  
##   
## Models: svmRadial, dnn   
## Number of resamples: 25   
##   
## RMSE   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## svmRadial 0.1131 0.1160 0.1190 0.1180 0.1203 0.1218 0  
## dnn 0.1398 0.1446 0.1464 0.1484 0.1504 0.1683 0

## A glm ensemble of 2 base models: svmRadial, dnn  
##   
## Ensemble results:  
## Generalized Linear Model   
##   
## 31641 samples  
## 2 predictor  
##   
## No pre-processing  
## Resampling: Bootstrapped (25 reps)   
## Summary of sample sizes: 31641, 31641, 31641, 31641, 31641, 31641, ...   
## Resampling results:  
##   
## RMSE Rsquared   
## 0.1178194 0.3674986

The glm-stacked ensemble model has performed slightly better at a RMSE of **0.12**, as compared to the individual models.

The full R code may be found in Appendix A.

# Appendix A

## Neural Network Ensemble for the Diabetes Dataset

# Data import

diabetes = read.csv('./datasets/Diabetes.csv', header = F)

colnames(diabetes) = c('no.of.times.preg', 'plasma.glucose.conc', 'diastolic.pressure',

'triceps.skin.fold.thick', 'serum.insulin', 'bmi', 'diab.pedigree.func',

'age', 'positive.test')

# Factorizing class variable

diabetes$positive.test = as.factor(as.character(diabetes$positive.test))

summary(diabetes)

# ------------------------------------------------------------------------

# Train-test split

library(caTools)

set.seed(111)

diabetes$split = sample.split(diabetes$positive.test, SplitRatio = 0.7)

diabetes\_train = subset(diabetes, split == T)

diabetes\_test = subset(diabetes, split == F)

# Excluding split columns

diabetes\_train = diabetes\_train[-10]

diabetes\_test = diabetes\_test[-10]

# Writing CSV for train and test

write.table(diabetes\_train, sep = ',', file = './datasets/Diabetes\_train.csv', row.names = F)

write.table(diabetes\_test, sep = ',', file = './datasets/Diabetes\_test.csv', row.names = F)

# ------------------------------------------------------------------------

# Single-Layer Neural Network

# Building the neural net model

library(nnet)

diabetes\_nnet = nnet(positive.test ~ ., data = diabetes\_train,

size = 8, maxit = 10000, decay = 0.0001)

summary(diabetes\_nnet)

# ------------------------------------------------------------------------

# Neural network plot

require(RCurl)

root.url<-'https://gist.githubusercontent.com/fawda123'

raw.fun<-paste(

root.url,

'5086859/raw/cc1544804d5027d82b70e74b83b3941cd2184354/nnet\_plot\_fun.r',

sep='/'

)

script<-getURL(raw.fun, ssl.verifypeer = FALSE)

eval(parse(text = script))

rm('script','raw.fun')

plot(diabetes\_nnet)

# ------------------------------------------------------------------------

# Prediction and confusion matrix

test\_pred = predict(diabetes\_nnet, newdata = diabetes\_test, type = 'class')

library(caret)

cm\_nn = confusionMatrix(table(true = diabetes\_test$positive.test, predicted = test\_pred))

cm\_nn

# ------------------------------------------------------------------------

# Neural Networks with Feature Extraction

# Building the neural net model

diabetes\_pcann = pcaNNet(positive.test ~ ., data = diabetes\_train,

size = 8, maxit = 10000, decay = 0.0001)

summary(diabetes\_pcann)

# ------------------------------------------------------------------------

# Prediction and confusion matrix

test\_pred\_pcann = predict(diabetes\_pcann, newdata = diabetes\_test, type = 'class')

cm\_pcann = confusionMatrix(table(true = diabetes\_test$positive.test, predicted = test\_pred\_pcann))

cm\_pcann

# ------------------------------------------------------------------------

# Ensemble Learning

# Coercing class variable into syntactically valid names

diabetes\_train$positive.test = make.names(diabetes\_train$positive.test)

library(caretEnsemble)

set.seed(111)

control1 = trainControl(method = 'repeatedcv', number = 10, repeats = 3,

savePredictions = 'all', classProbs = TRUE)

algorithmList1 = c('glm', 'nnet', 'pcaNNet')

# Model Fit

models1 = caretList(positive.test ~ ., data = diabetes\_train,

trControl = control1, methodList = algorithmList1)

# Result summary

result1 = resamples(models1)

summary(result1)

dotplot(result1)

stack\_control1 = trainControl(method = 'repeatedcv', number = 10, repeats = 3)

stack\_glm1 = caretStack(models1, method = 'glm', metric = 'Accuracy', trControl = stack\_control1)

print(stack\_glm1)

# ------------------------------------------------------------------------

## Neural Network Emsemble for Wine Quality Dataset

# Data import

wine = read.csv('./datasets/winequality-white.csv')

summary(wine)

# ------------------------------------------------------------------------

# Train-test split

set.seed(111)

wine$split = sample.split(wine$quality, SplitRatio = 0.7)

wine\_train = subset(wine, split == T)

wine\_test = subset(wine, split == F)

normalize <- function(x) {

return ((x - min(x)) / (max(x) - min(x)))

}

wine\_train <- as.data.frame(lapply(wine\_train, normalize))

wine\_test <- as.data.frame(lapply(wine\_test, normalize))

# Excluding split columns

wine\_train = wine\_train[-13]

wine\_test = wine\_test[-13]

# Writing CSV for train and test

write.table(wine\_train, sep = ',', file = './datasets/winequality\_train.csv', row.names = F)

write.table(wine\_test, sep = ',', file = './datasets/winequality\_test.csv', row.names = F)

# ------------------------------------------------------------------------

# SVM with Radial Basis Function Kernel

# Building the neural network model

library(kernlab)

wine\_svmrbf = ksvm(quality ~ .,

data = wine\_train,

kernel = 'rbfdot',

kpar = 'automatic')

summary(wine\_svmrbf)

# ------------------------------------------------------------------------

# Prediction

rbf\_pred = predict(wine\_svmrbf, newdata = wine\_test[-12])

model\_values1 = data.frame(obs = as.numeric(wine\_test$quality), pred = rbf\_pred)

defaultSummary(model\_values1)

# ------------------------------------------------------------------------

# Stacked AutoEncoder Deep Neural Network

# Building the neural network model

library(deepnet)

wine\_dnn = dbn.dnn.train(x = as.matrix(wine\_train[1:11]),

y = as.matrix(wine\_train[12]),

learningrate = 0.001,

hidden = 8,

numepochs = 10)

summary(wine\_dnn)

# ------------------------------------------------------------------------

# Prediction

wine\_pred\_dnn = nn.predict(wine\_dnn, x = as.matrix(wine\_test[1:11]))

model\_values2 = data.frame(obs = as.numeric(wine\_test$quality), pred = wine\_pred\_dnn)

defaultSummary(model\_values2)

## ------------------------------------------------------------------------

# Ensemble Learning

library(caret)

library(caretEnsemble)

set.seed(111)

control2 <- trainControl(

method="boot",

number=25,

savePredictions="final",

classProbs=TRUE,

index=createResample(wine\_train$quality, 25))

models2 <- caretList(

quality~., data=wine\_train,

trControl=control2,

methodList=c("svmRadial", "dnn"))

greedy\_ensemble <- caretEnsemble(

models2,

trControl=control2)

summary(greedy\_ensemble)

result2 <- resamples(models2)

# Result summary

summary(result2)

dotplot(result2)

# ------------------------------------------------------------------------

set.seed(111)

stack\_control2 = trainControl(method = 'boot', number = 25)

stack\_glm2 = caretStack(models2, method = 'glm', metric = 'RMSE', trControl = stack\_control2)

print(stack\_glm2)

# ------------------------------------------------------------------------