ΣΧΟΛΗ ΧΡΗΜΑΤΟΟΙΚΟΝΟΜΙΚΗΣ & ΣΤΑΤΙΣΤΙΚΗΣ

ΤΜΗΜΑ ΣΤΑΤΙΣΤΙΚΗΣ & ΑΣΦΑΛΙΣΤΙΚΗΣ ΕΠΙΣΤΗΜΗΣ

Π.Μ.Σ. ΕΦΑΡΜΟΣΜΕΝΗΣ ΣΤΑΤΙΣΤΙΚΗΣ

ΣΤΑΤΙΣΤΙΚΗ ΜΗΧΑΝΙΚΗ ΜΑΘΗΣΗ

ΕΡΓΑΣΙΑ 2η

ΜΟΝΤΕΛΑ ΤΑΞΙΝΟΜΙΣΗΣ RF ΚΑΙ FNN (R & PYTHON)

Εμμανουήλ Φλωράκης florakis.emm@gmail.com ΜΕΣ22024

Αθήνα, 2024

Περιεχόμενα

1. Άσκηση 1 Random Forest 2

2. Υλοποίηση σε R 2

3. Υλοποίηση σε Python 13

4. Άσκηση 2 Feedforward Neural Network 18

5. Υλοποίηση σε R 18

6. Υλοποίηση σε Python 23

ΑΣΚΗΣΗ 1η

Βασισμένοι σε ένα δοθέν σύνολο δεδομένων όπου περιέχονται τιμές διαγνωστικών ιατρικών δεδομένων καθώς και αν η κάθε γυναίκα του δείγματος νοσεί από διαβήτη ή όχι, θα προχωρήσουμε σε κατασκευή και αξιολόγηση μοντέλων ταξινόμησης για την πρόβλεψη νοσηλείας. Οι μέθοδοι θα γίνουν επιμέρους μέσω του στατιστικού πακέτου R και της γλώσσας προγραμματισμού Python. Κατόπιν υλοποίησης εντολών μπορούν να εντοπισθούν και σχετικά σχόλια. Για την αξιολόγηση και βάσει της υπόθεσης θα κατασκευαστούν σχετικοί συγκεντρωτικοί πίνακες με την ακρίβεια των ταξινομητών στο σύνολο δεδομένων εκπαίδευσης και στο σύνολο δεδομένων ελέγχου. Θα παρατηρηθούν επίσης σχολιασμοί σε πινάκια και διαγράμματα για μία ευρύτερη αξιολόγηση.

- Random Forest

Στατιστικό πακέτο R

```
> ###STATISTICAL MACHINE LEARNING ASSIGNMENT II
>
> #Necessary Libraries
> library(caret)
> library(randomForest)
> library(keras)
> library(tensorflow)
> library(dplyr)
>
> #import data
> data<- read.csv(file = "diabetes.csv")
> str(data)
'data.frame':   768 obs. of  9 variables:
 $ Pregnancies             : int  6 1 8 1 0 5 3 10 2 8 ...
 $ Glucose                 : int  148 85 183 89 137 116 78 115 197 125 ...
 $ BloodPressure           : int  72 66 64 66 40 74 50 0 70 96 ...
 $ SkinThickness           : int  35 29 0 23 35 0 32 0 45 0 ...
 $ Insulin                 : 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 ...
 $ DiabetesPedigreeFunction: num  0.627 0.351 0.672 0.167 2.288 ...
 $ Age                     : int  50 31 32 21 33 30 26 29 53 54 ...
 $ Outcome                 : int  1 0 1 0 1 0 1 0 1 1 ...
>
> #change types of variables ton numeric and factor only
> data$Pregnancies <- as.numeric(data$Pregnancies)
> data$Glucose <- as.numeric(data$Glucose)
> data$BloodPressure <- as.numeric(data$BloodPressure)
> data$SkinThickness <- as.numeric(data$SkinThickness)
> data$Insulin <- as.numeric(data$Insulin)
> data$Age <- as.numeric(data$Age)
> data$Outcome = factor(data$Outcome, levels = c(0, 1))
>
> #Normalize data (variables only) and check the results
> data[-9]<-scale(data[-9])
> str(data)
'data.frame':   768 obs. of  9 variables:
 $ Pregnancies             : num  0.64 -0.844 1.233 -0.844 -1.141 ...
 $ Glucose                 : num  0.848 -1.123 1.942 -0.998 0.504 ...
 $ BloodPressure           : num  0.15 -0.16 -0.264 -0.16 -1.504 ...
 $ SkinThickness           : num  0.907 0.531 -1.287 0.154 0.907 ...
 $ Insulin                 : num  -0.692 -0.692 -0.692 0.123 0.765 ...
 $ BMI                     : num  0.204 -0.684 -1.103 -0.494 1.409 ...
 $ DiabetesPedigreeFunction: num  0.468 -0.365 0.604 -0.92 5.481 ...
 $ Age                     : num  1.4251 -0.1905 -0.1055 -1.0409 -0.0205 ...
 $ Outcome                 : Factor w/ 2 levels "0","1": 2 1 2 1 2 1 2 1 2 2 ...
>
```

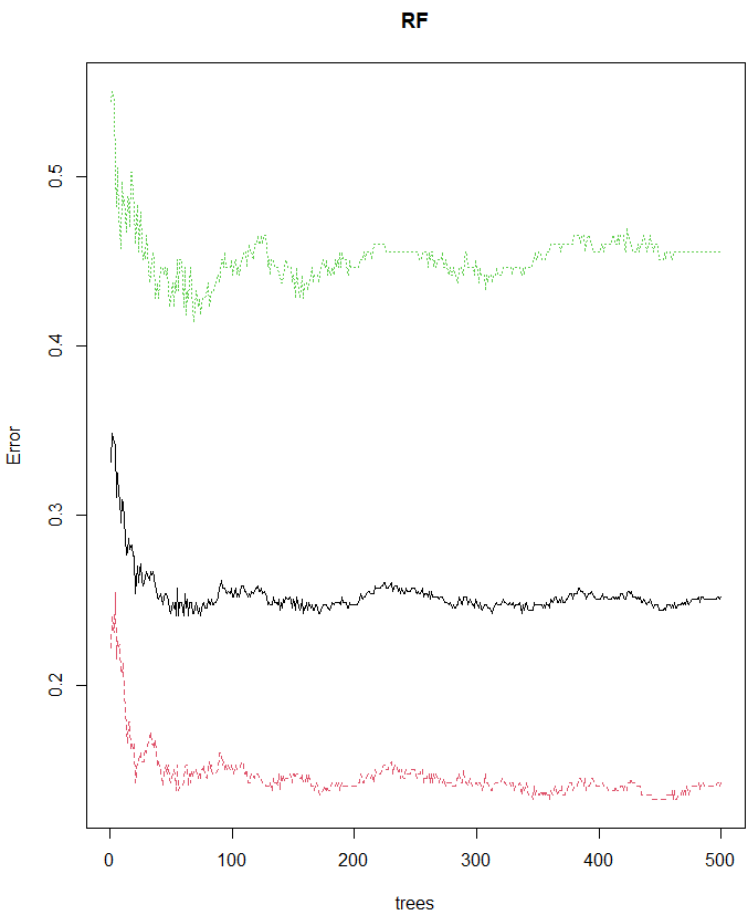
```
#i) Random Forest for all variables

# Reproducible random sampling
set.seed(123)

# Creating training data as 80% of the dataset
random_sample <- createDataPartition(data$Outcome, p = 0.80, list = F)

# Generating training dataset from the random_sample
training_dataset  <- data[random_sample, ]

# Generating testing dataset from rows which are not included in random_sample
testing_dataset <- data[-random_sample, ]
```

Διαμορφώνοντας ως άνω το dataset για όλες τις μεταβλητές προχωρούμε σε κανονικοποίηση και τυχαίο διαχωρισμό δεδομένων 80%-20% .

Κατόπιν προχωρούμε σε μοντελοποίηση με 500 ταξινομητές.

```
> # NTree --> 500
> RF <- randomForest(Outcome ~ ., data = training_dataset, ntree = 500, cv.fold = 5)
>
> # View the cross-validated forest results.
> print(RF)

Call:
 randomForest(formula = Outcome ~ ., data = training_dataset,      ntree = 500, cv.fold = 5)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 25.2%
Confusion matrix:
    0   1 class.error
0 343  57    0.142500
1  98 117    0.455814
>
> # Importance of each predictor.
> importance(RF, type = 2)
                         MeanDecreaseGini
Pregnancies                      22.88289
Glucose                          74.22349
BloodPressure                    25.67850
SkinThickness                    19.79991
Insulin                          18.93130
BMI                              44.20405
DiabetesPedigreeFunction         35.29578
Age                              35.92520
>
> # Plot the cross-validated random forest.
> plot(RF)
>
> # Variable importance plot for cross-validated random forest.
> varImpPlot(RF)
>
> # Make predictions on the training dataset.
> predictions_train <- predict(RF, data = training_dataset)
>
> # Calculate the accuracy.
> accuracy_train <- sum(predictions_train == training_dataset$Outcome) / length(training_dataset$Outcome)
>
> # Make predictions on the testing dataset.
> predictions_test <- predict(RF, testing_dataset)
>
> # Create a confusion matrix for the testing dataset.
> confusion_matrix_test <- confusionMatrix(data = predictions_test, reference = testing_dataset$Outcome)
>
> # Extract and print accuracy for the testing dataset.
> accuracy_test <- confusion_matrix_test$overall[1]
```

Όπου είναι εύκολο να παρατηρήσουμε στα πινάκια αλλά και στα παρακάτω διαγράμματα πως η εκτιμώμενη συχνότητα σφάλματος εκτός δείγματος (OOB) ανέρχεται σε 25,2% το οποίο παρουσιάζει ότι το μοντέλο έχει μια καλή γενίκευση και δεν υποφέρει από overfitting. Η ακρίβεια για την κατηγορία μη ανίχνευσης διαβήτη είναι καλή με σφάλμα ταξινόμησης 0.14 όμως για την κατηγορία ανίχνευσης διαβήτη είναι χαμηλή με σφάλμα 0.46. Οι σημαντικότερες μεταβλητές είναι η "Glucose", ο "BMI" και η "Age". Αναφορικά με το πλήθος ταξινομητών έχουμε μείωση του σφάλματος σύμφωνα με το παρακάτω διάγραμμα.

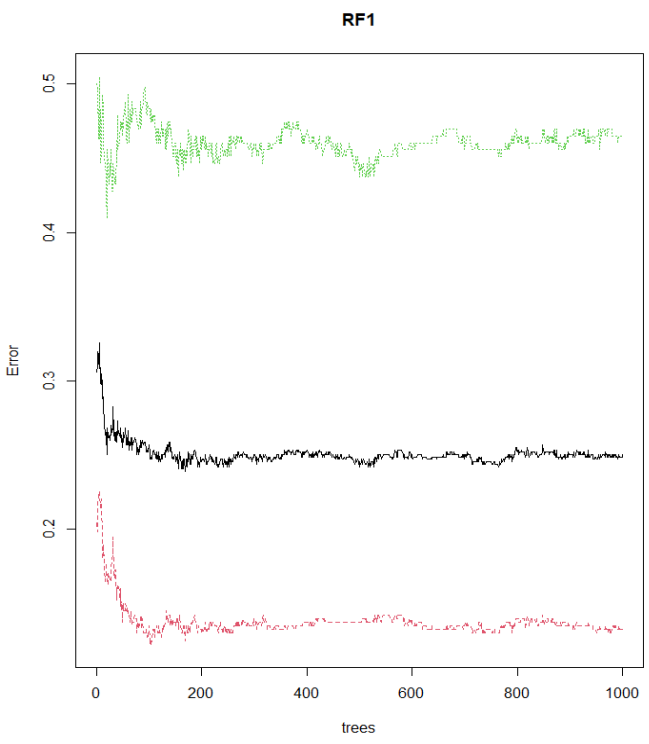


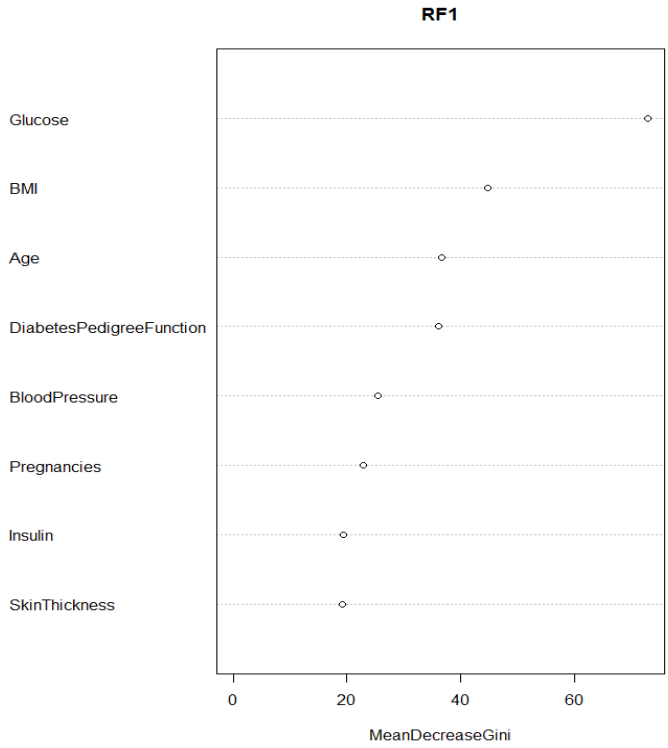
Συνεχίζοντας για 1000 ταξινομητές:

```
> # NTree --> 1000
> RF1 <- randomForest(Outcome ~ ., data = training_dataset, ntree = 1000, cv.fold = 5)
>
> # View the cross-validated forest results.
> print(RF1)
```

Είναι εξίσου εύκολο να παρατηρήσουμε στα επόμενα πινάκια αλλά και στα διαγράμματα που ακολουθούν πως η εκτιμώμενη συχνότητα σφάλματος εκτός δείγματος (OOB) ανέρχεται σε 25,04% όπου με μικρή διαφορά σε σχέση με το προηγούμενο μοντέλο έχει επίσης μια καλή γενίκευση και δεν υποφέρει από overfitting. Η ακρίβεια για την κατηγορία μη ανίχνευσης διαβήτη είναι επίσης καλή με σφάλμα ταξινόμησης 0.135 όμως για την κατηγορία ανίχνευσης διαβήτη παραμένει χαμηλή με σφάλμα 0.47. Οι σημαντικότερες μεταβλητές παραμένουν να είναι η "Glucose", ο "BMI" και η "Age" με ελάχιστες αυξομειώσεις. Εξίσου με το πλήθος ταξινομητών παρατηρούμε μείωση του σφάλματος βάσει του σχετικού διαγράμματος.

```
Call:
 randomForest(formula = Outcome ~ ., data = training_dataset,      ntree = 1000, cv.fold = 5)
               Type of random forest: classification
                     Number of trees: 1000
No. of variables tried at each split: 2

        OOB estimate of  error rate: 25.04%
Confusion matrix:
    0   1 class.error
0 346  54   0.1350000
1 100 115   0.4651163
>
> # Importance of each predictor.
> importance(RF1, type = 2)
                         MeanDecreaseGini
Pregnancies                      22.79489
Glucose                          72.93037
BloodPressure                    25.44401
SkinThickness                    19.25650
Insulin                          19.43423
BMI                              44.66567
DiabetesPedigreeFunction         36.07350
Age                              36.57135
>
> # Plot the cross-validated random forest.
> plot(RF1)
>
> # Variable importance plot for cross-validated random forest.
> varImpPlot(RF1)
>
> # Make predictions on the training dataset.
> predictions_train1 <- predict(RF1, data = training_dataset)
>
> # Calculate the accuracy.
> accuracy_train1 <- sum(predictions_train1 == training_dataset$Outcome) / length(training_dataset$Outcome)
>
> # Make predictions on the testing dataset.
> predictions_test1 <- predict(RF1, testing_dataset)
>
> # Create a confusion matrix for the testing dataset.
> confusion_matrix_test1 <- confusionMatrix(data = predictions_test1, reference = testing_dataset$Outcome)
>
> # Extract and print accuracy for the testing dataset.
> accuracy_test1 <- confusion_matrix_test1$overall[1]
```

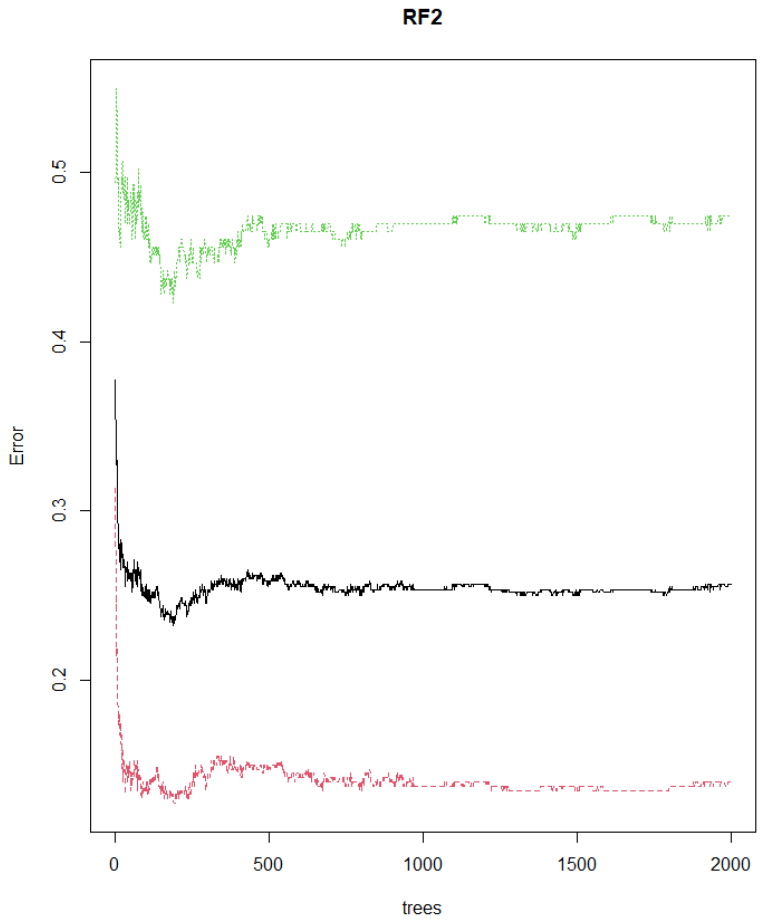


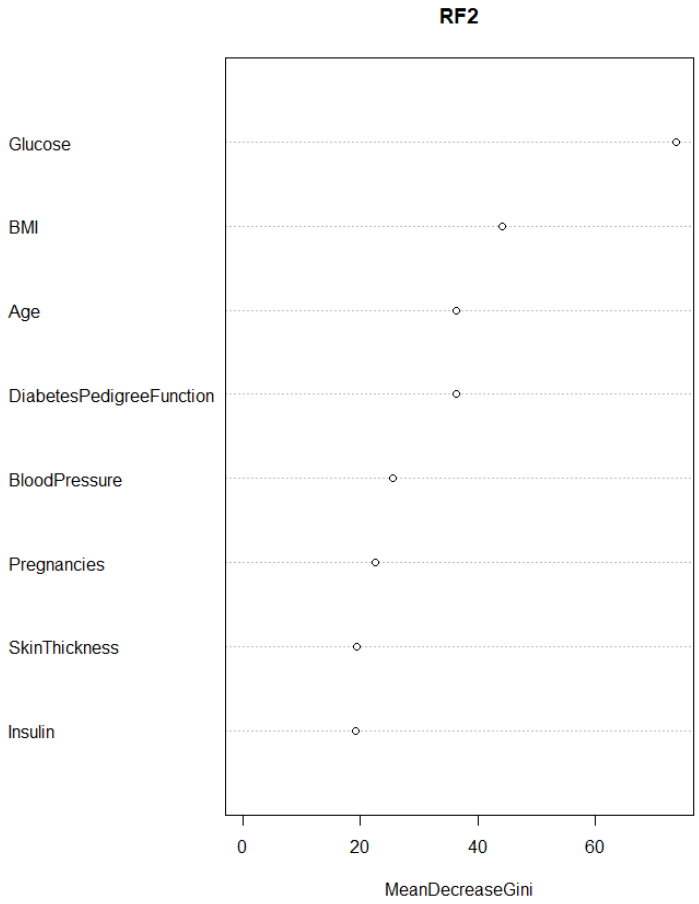


Τελικά θα προχωρήσουμε και σε μοντέλο με 2000 ταξινομητές ως κάτωθι:

```
> # NTree --> 2000
> RF2 <- randomForest(Outcome ~ ., data = training_dataset, ntree = 2000, cv.fold = 5)
>
> # View the cross-validated forest results.
> print(RF2)

Call:
 randomForest(formula = Outcome ~ ., data = training_dataset,      ntree = 2000, cv.fold = 5)
               Type of random forest: classification
                     Number of trees: 2000
No. of variables tried at each split: 2

        OOB estimate of  error rate: 25.69%
Confusion matrix:
    0   1 class.error
0 344  56   0.1400000
1 102 113   0.4744186
>
> # Importance of each predictor.
> importance(RF2, type = 2)
                         MeanDecreaseGini
Pregnancies                      22.48808
Glucose                          73.79604
BloodPressure                    25.56279
SkinThickness                    19.36002
Insulin                          19.28734
BMI                              44.23497
DiabetesPedigreeFunction         36.38425
Age                              36.41906
>
> # Plot the cross-validated random forest.
> plot(RF2)
>
> # Variable importance plot for cross-validated random forest.
> varImpPlot(RF2)
>
> # Make predictions on the training dataset.
> predictions_train2 <- predict(RF2, data = training_dataset)
>
> # Calculate the accuracy.
> accuracy_train2 <- sum(predictions_train2 == training_dataset$Outcome) / length(training_dataset$Outcome)
>
> # Make predictions on the testing dataset.
> predictions_test2 <- predict(RF2, testing_dataset)
>
> # Create a confusion matrix for the testing dataset.
> confusion_matrix_test2 <- confusionMatrix(data = predictions_test2, reference = testing_dataset$Outcome)
>
> # Extract and print accuracy for the testing dataset.
> accuracy_test2 <- confusion_matrix_test2$overall[1]
```

Στο οποίο διακρίνουμε στα σχετικά πινάκια αλλά και στα διαγράμματα που ακολουθούν πως η εκτιμώμενη συχνότητα σφάλματος εκτός δείγματος (OOB) ανέρχεται σε 25,69% όπου επίσης δεν διαφοροποιείτε με τα προηγούμενα μοντέλα, έχει μια εξίσου καλή γενίκευση και δεν υποφέρει από overfitting. Η ακρίβεια για την κατηγορία μη ανίχνευσης διαβήτη είναι επίσης καλή με σφάλμα ταξινόμησης 0.14 όμως για την κατηγορία ανίχνευσης διαβήτη έχουμε μία ακόμη σταδιακή αύξηση του σφάλματος στα 0,475. Οι σημαντικότερες μεταβλητές παραμένουν να είναι η "Glucose", ο "BMI" και η "Age" με ελάχιστες αυξομειώσεις. Σημαντική η είναι η πλέον οριακή ισοτιμία της μεταβλητής “Age” με την “DiabetesPedigreeFunction”. Το πλήθος των ταξινομητών βάσει του διαγράμματος εξίσου επιφέρει μείωση των σφαλμάτων.





Τελικά θα κατασκευάσουμε έναν συγκεντρωτικό πίνακα των αποτελεσμάτων της συνολικής ακρίβειας όλων των ταξινομητών ως εξής:

```
> #Creating a training and testing accuracy dataframe
> data.frame( modelName = c("RF   (NTree=500)","RF1 (NTree=1000)","RF2 (NTree=2000)"),
+             AccuracyTrain=c(accuracy_train,accuracy_train1,accuracy_train2),
+             AccuracyTest=c(accuracy_test,accuracy_test1,accuracy_test2))
         modelName AccuracyTrain AccuracyTest
1 RF   (NTree=500)     0.7479675    0.8104575
2 RF1 (NTree=1000)     0.7495935    0.8235294
3 RF2 (NTree=2000)     0.7430894    0.8169935
>
```

Όπου παρατηρούμε ότι η αύξηση του αριθμού των ταξινομητών δεν οδηγεί σε σημαντική αύξηση της ακρίβειας. Οι διαφορές δεν υποδεικνύουν overfitting και βάσει των παραπάνω αποτελεσμάτων μοντέλο με 1000 ταξινομητές επιτυγχάνει την καλύτερη ισορροπία μεταξύ γενίκευσης και ακρίβειας.

Συνεχίζοντας με τα μοντέλα που θα περιέχουν μόνο τις μεταβλητές “Glucose” & “BMI” βάσει της υπόθεσης:

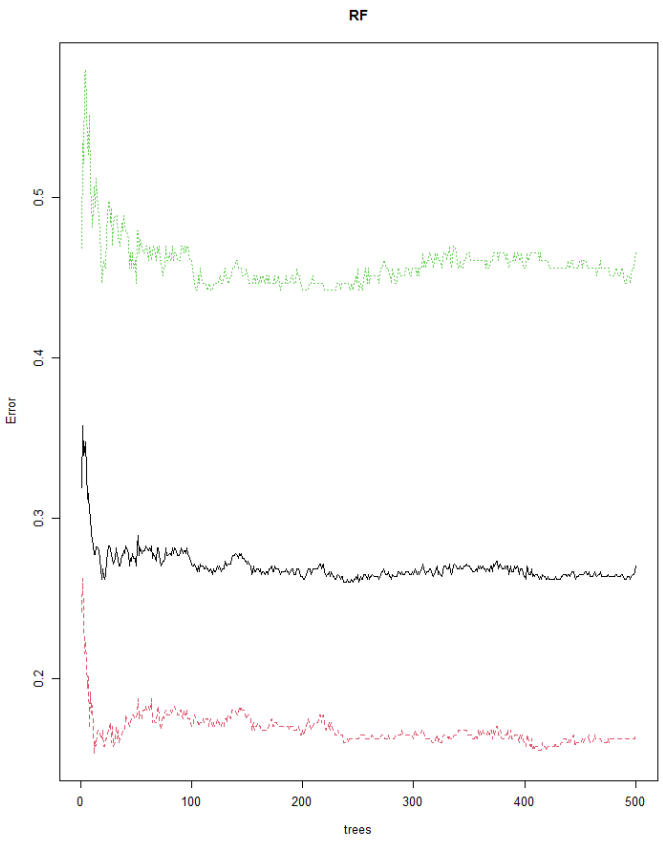
```
> #ii) Random Forest for “Glucose” & “BMI”
>
> #Downsizing the dataset as requested
> data<-subset(data,select = c(Glucose,BMI,Outcome))
> str(data)
'data.frame':   768 obs. of  3 variables:
 $ Glucose: num  0.848 -1.123 1.942 -0.998 0.504 ...
 $ BMI    : num  0.204 -0.684 -1.103 -0.494 1.409 ...
 $ Outcome: Factor w/ 2 levels "0","1": 2 1 2 1 2 1 2 1 2 2 ...
>
> # Reproducible random sampling
> set.seed(123)
>
> # Creating training data as 80% of the dataset
> random_sample <- createDataPartition(data$Outcome, p = 0.80, list = F)
>
> # Generating training dataset from the random_sample
> training_dataset  <- data[random_sample, ]
>
> # Generating testing dataset from rows which are not included in random_sample
> testing_dataset <- data[-random_sample, ]
```

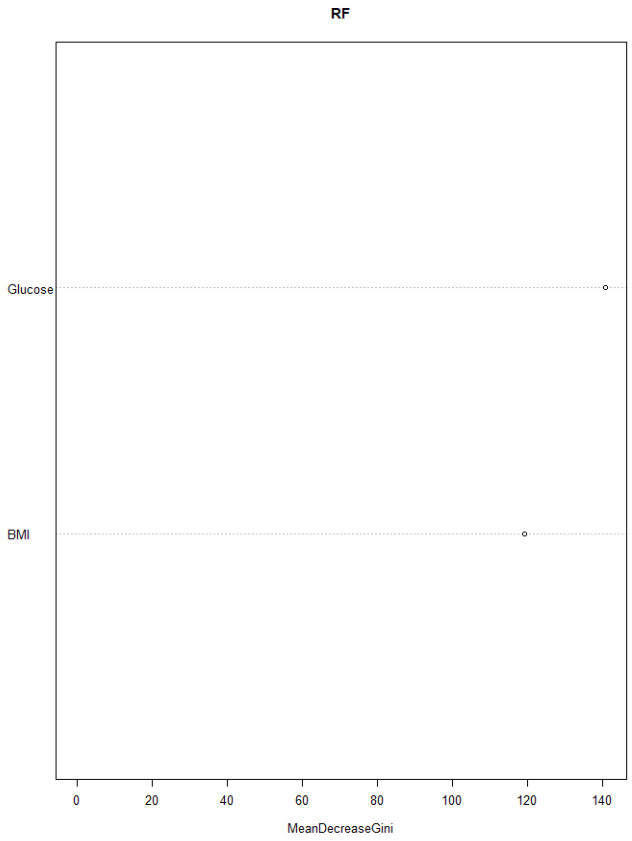
Διαμορφώνοντας το dataset για τις συγκεκριμένες μεταβλητές και προχωρούμε σε τυχαίο διαχωρισμό δεδομένων με 80%-20% εκπαίδευση και έλεγχο.

Κατόπιν προχωρούμε σε μοντελοποίηση με 500 ταξινομητές.

```
> # NTree --> 500
> RF <- randomForest(Outcome ~ ., data = training_dataset, ntree = 500, cv.fold = 5)
>
> # View the cross-validated forest results.
> print(RF)

Call:
 randomForest(formula = Outcome ~ ., data = training_dataset,      ntree = 500, cv.fold = 5)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 1

        OOB estimate of  error rate: 26.99%
Confusion matrix:
    0   1 class.error
0 334  66   0.1650000
1 100 115   0.4651163
>
> # Importance of each predictor.
> importance(RF, type = 2)
        MeanDecreaseGini
Glucose         140.6320
BMI             119.1707
>
> # Plot the cross-validated random forest.
> plot(RF)
>
> # Variable importance plot for cross-validated random forest.
> varImpPlot(RF)
>
> # Make predictions on the training dataset.
> predictions_train <- predict(RF, data = training_dataset)
>
> # Calculate the accuracy.
> accuracy_train <- sum(predictions_train == training_dataset$Outcome) / length(training_dataset$Outcome)
>
> # Make predictions on the testing dataset.
> predictions_test <- predict(RF, testing_dataset)
>
> # Create a confusion matrix for the testing dataset.
> confusion_matrix_test <- confusionMatrix(data = predictions_test, reference = testing_dataset$Outcome)
>
> # Extract and print accuracy for the testing dataset.
> accuracy_test <- confusion_matrix_test$overall[1]
```

Όπου είναι εύκολο να παρατηρήσουμε στα πινάκια αλλά και στα παρακάτω διαγράμματα πως η εκτιμώμενη συχνότητα σφάλματος εκτός δείγματος (OOB) ανέρχεται σε 26.99%. Αυτό υποδηλώνει ότι το μοντέλο πιθανότατα δυσκολεύεται λίγο παραπάνω να γενικεύσει σε νέα δεδομένα όμως δεν υποφέρει από overfitting. Η ακρίβεια για την κατηγορία μη ανίχνευσης διαβήτη είναι καλή με σφάλμα ταξινόμησης 0.165 όμως για την κατηγορία ανίχνευσης διαβήτη είναι χαμηλή με σφάλμα 0.465. Η μεταβλητή "Glucose" είναι σημαντικότερη από τον "BMI" κατά 20 μονάδες. Το πλήθος ταξινομητών φέρει μείωση του σφάλματος σύμφωνα με το παρακάτω διάγραμμα.



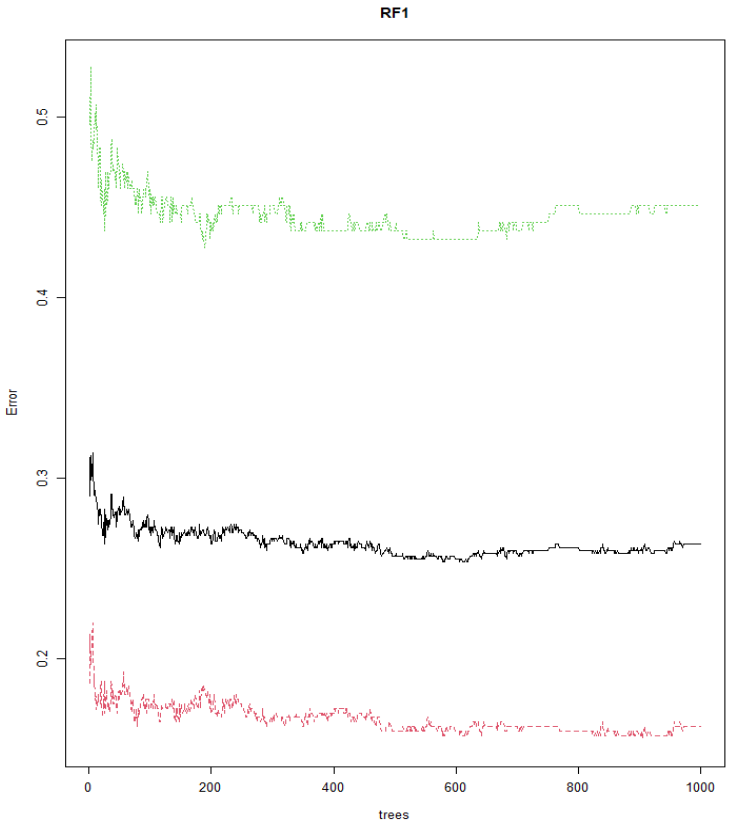


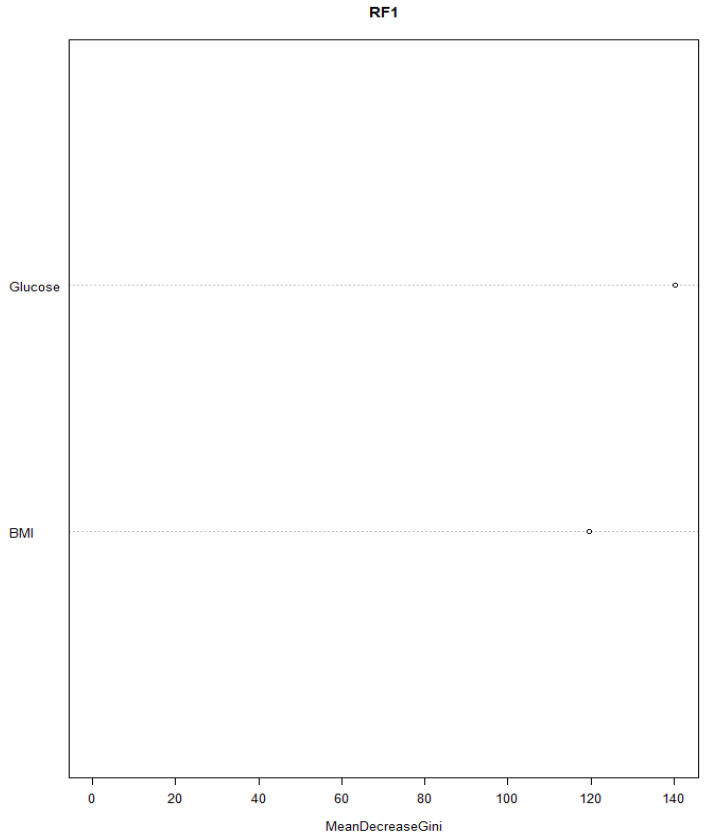
Συνεχίζοντας για 1000 ταξινομητές:

```
> # NTree --> 1000
> RF1 <- randomForest(Outcome ~ ., data = training_dataset, ntree = 1000, cv.fold = 5)
>
> # View the cross-validated forest results.
> print(RF1)

Call:
 randomForest(formula = Outcome ~ ., data = training_dataset,      ntree = 1000, cv.fold = 5)
               Type of random forest: classification
                     Number of trees: 1000
No. of variables tried at each split: 1

        OOB estimate of  error rate: 26.34%
Confusion matrix:
    0   1 class.error
0 335  65   0.1625000
1  97 118   0.4511628
>
> # Importance of each predictor.
> importance(RF1, type = 2)
        MeanDecreaseGini
Glucose         140.3491
BMI             119.5263
```

```
> # Plot the cross-validated random forest.
> plot(RF1)
>
> # Variable importance plot for cross-validated random forest.
> varImpPlot(RF1)
>
> # Make predictions on the training dataset.
> predictions_train1 <- predict(RF1, data = training_dataset)
>
> # Calculate the accuracy.
> accuracy_train1 <- sum(predictions_train1 == training_dataset$Outcome) / length(training_dataset$Outcome)
>
> # Make predictions on the testing dataset.
> predictions_test1 <- predict(RF1, testing_dataset)
>
> # Create a confusion matrix for the testing dataset.
> confusion_matrix_test1 <- confusionMatrix(data = predictions_test1, reference = testing_dataset$Outcome)
>
> # Extract and print accuracy for the testing dataset.
> accuracy_test1 <- confusion_matrix_test1$overall[1]
>
```

Είναι εξίσου εύκολο να παρατηρήσουμε στα παραπάνω πινάκια αλλά και στα διαγράμματα που ακολουθούν πως η εκτιμώμενη συχνότητα σφάλματος εκτός δείγματος (OOB) ανέρχεται σε 26,34% όπου με μικρή διαφορά σε σχέση με το προηγούμενο μοντέλο έχει μια καλή γενίκευση και δεν υποφέρει από overfitting. Η ακρίβεια για την κατηγορία μη ανίχνευσης διαβήτη είναι καλύτερη με σφάλμα ταξινόμησης 0.16 και το ίδιο ισχύει και για την κατηγορία ανίχνευσης διαβήτη με σφάλμα 0.45. Η μεταβλητή "Glucose" είναι σημαντικότερη από τον "BMI" με την διαφορά να μικραίνει. Εξίσου με το πλήθος ταξινομητών παρατηρούμε μείωση του σφάλματος βάσει του σχετικού διαγράμματος.

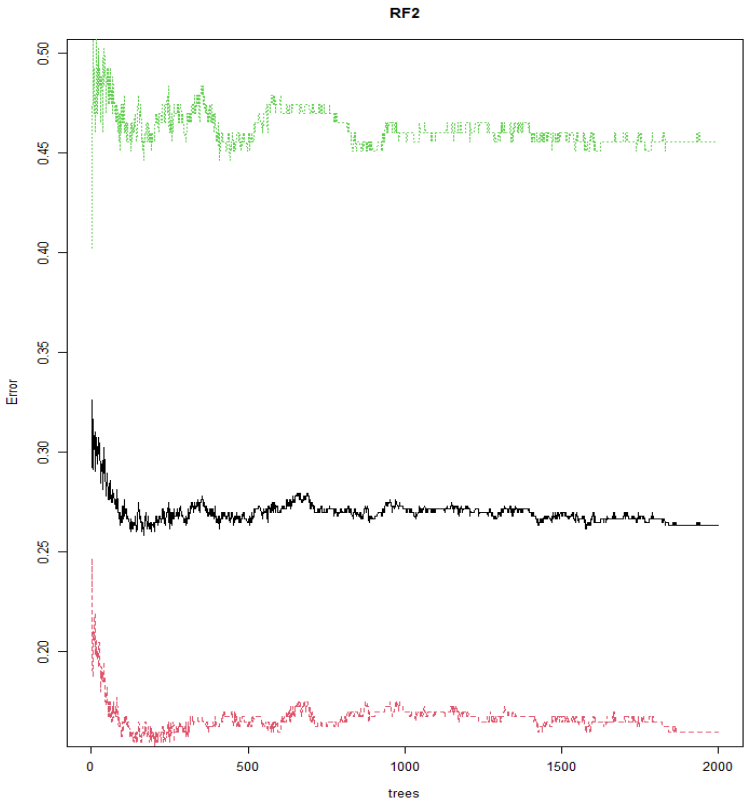


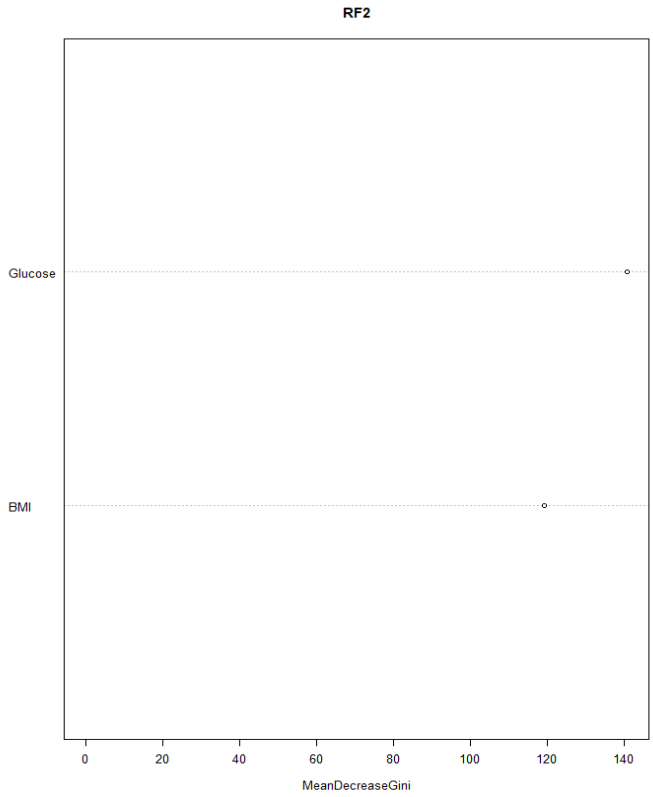


Τελικά θα προχωρήσουμε και σε μοντέλο με 2000 ταξινομητές ως κάτωθι:

```
> # NTree --> 2000
> RF2 <- randomForest(Outcome ~ ., data = training_dataset, ntree = 2000, cv.fold = 5)
>
> # View the cross-validated forest results.
> print(RF2)

Call:
 randomForest(formula = Outcome ~ ., data = training_dataset,      ntree = 2000, cv.fold = 5)
               Type of random forest: classification
                     Number of trees: 2000
No. of variables tried at each split: 1

        OOB estimate of  error rate: 26.34%
Confusion matrix:
    0   1 class.error
0 336  64    0.160000
1  98 117    0.455814
>
> # Importance of each predictor.
> importance(RF2, type = 2)
        MeanDecreaseGini
Glucose         140.7516
BMI             119.1828
>
> # Plot the cross-validated random forest.
> plot(RF2)
>
> # Variable importance plot for cross-validated random forest.
> varImpPlot(RF2)
>
> # Make predictions on the training dataset.
> predictions_train2 <- predict(RF2, data = training_dataset)
>
> # Calculate the accuracy.
> accuracy_train2 <- sum(predictions_train2 == training_dataset$Outcome) / length(training_dataset$Outcome)
>
> # Make predictions on the testing dataset.
> predictions_test2 <- predict(RF2, testing_dataset)
>
> # Create a confusion matrix for the testing dataset.
> confusion_matrix_test2 <- confusionMatrix(data = predictions_test2, reference = testing_dataset$Outcome)
>
> # Extract and print accuracy for the testing dataset.
> accuracy_test2 <- confusion_matrix_test2$overall[1]
```

Στο οποίο διακρίνουμε στα παραπάνω πινάκια αλλά και στα διαγράμματα που ακολουθούν πως η εκτιμώμενη συχνότητα σφάλματος εκτός δείγματος (OOB) ανέρχεται σε 26,34 όπου όμοια με προηγουμένως έχει μια καλή γενίκευση και δεν υποφέρει από overfitting. Η ακρίβεια για την κατηγορία μη ανίχνευσης διαβήτη είναι εξίσου ισότιμη με σφάλμα ταξινόμησης 0.16 όμως για την κατηγορία ανίχνευσης διαβήτη έχουμε μικρή αύξηση του σφάλματος στα 0,456. Οι σημαντικότερες μεταβλητές παραμένουν να είναι η "Glucose", ο "BMI" και η "Age" με ελάχιστες αυξομειώσεις. Η μεταβλητή "Glucose" είναι σημαντικότερη από τον "BMI" και το πλήθος των ταξινομητών βάσει του επόμενου διαγράμματος εξίσου επιφέρει μείωση των σφαλμάτων.





Τελικά θα κατασκευάσουμε έναν συγκεντρωτικό πίνακα των αποτελεσμάτων της συνολικής ακρίβειας όλων των ταξινομητών ως εξής:

```
> #Creating a training and testing accuracy dataframe
> data.frame( ModelName = c("RF   (NTree=500)","RF1 (NTree=1000)","RF2 (NTree=2000)"),
+             AccuracyTrain=c(accuracy_train,accuracy_train1,accuracy_train2),
+             AccuracyTest=c(accuracy_test,accuracy_test1,accuracy_test2))
         ModelName AccuracyTrain AccuracyTest
1 RF   (NTree=500)     0.7300813    0.7385621
2 RF1 (NTree=1000)     0.7365854    0.7647059
3 RF2 (NTree=2000)     0.7365854    0.7581699
```

Όπου παρατηρούμε ότι η αύξηση του αριθμού των ταξινομητών δεν οδηγεί σε σημαντική αύξηση της ακρίβειας. Οι διαφορές δεν υποδεικνύουν overfitting και βάσει των παραπάνω αποτελεσμάτων μοντέλο με 1000 ταξινομητές επιτυγχάνει την καλύτερη ισορροπία μεταξύ γενίκευσης και ακρίβειας. Σε σύγκριση με τα μοντέλα που χρησιμοποιήθηκε το σύνολο των μεταβλητών της υπόθεσης έχουμε μία καλύτερη γενίκευση δεδομένης της μικρής απόκλισης μεταξύ της συνολικής ακρίβειας στο σύνολο δεδομένων εκπαίδευσης και στο σύνολο δεδομένων ελέγχου όμως παρατηρείται κατά μέσο όρο μείωση της ακρίβειας. Μεταξύ των δύο βέλτιστων μοντέλων με 1000 ταξινομητές θα επιλέγαμε αυτό με τις μεταβλητές “Glucose” & “BMI” για λόγους οικονομίας και γενίκευσης.

Γλώσσα Προγραμματισμού Python

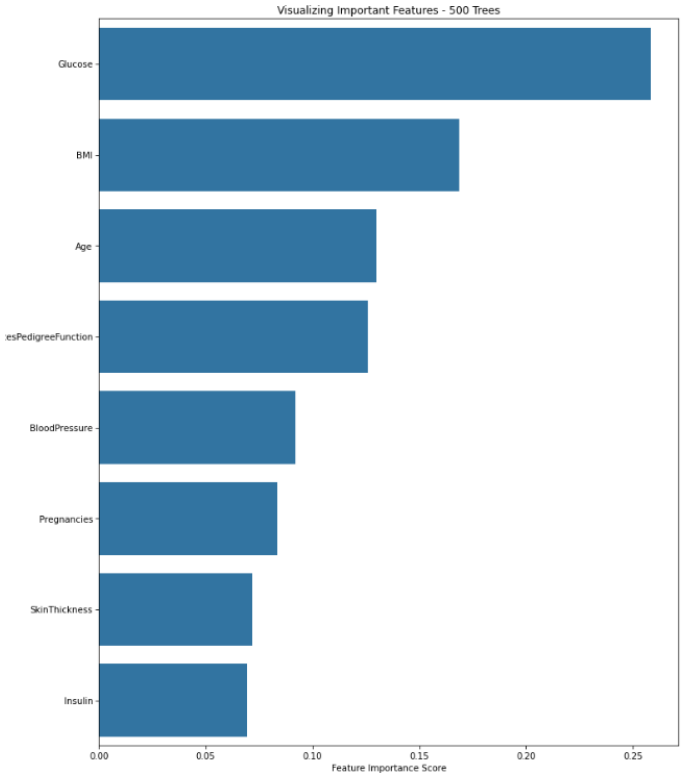
```
###STATISTICAL MACHINE LEARNING ASSIGNMENT II

#Necessary Libraries
#!pip install sklearn
#!pip install matplotlib
#!pip install pandas
#!pip install seaborn
#!pip install tensorflow
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predict
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

# Load Data
data = pd.read_csv('diabetes.csv')

##Random Forest

#i) Random Forest for all variables

# Standardize the data
scale = StandardScaler()
columns_to_scale = data.columns[:-1]  # Exclude the 'Outcome' column
data[columns_to_scale] = scale.fit_transform(data[columns_to_scale])

# Split data into features (x) and target variable (y)
x = data.iloc[:, :-1]
y = data['Outcome']

# Split the scaled data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.8, random_state=12345)
```

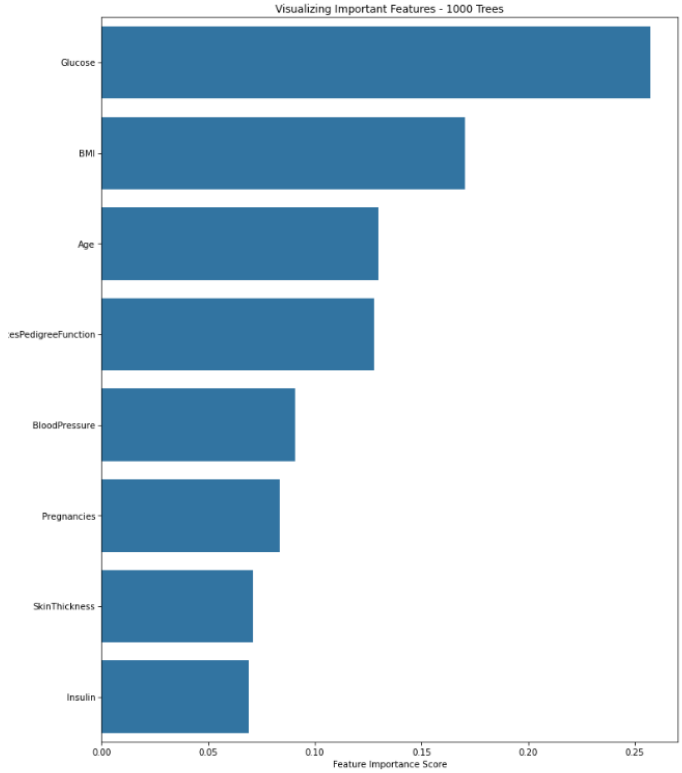
Ξεκινώντας εξίσου με τις απαραίτητες βιβλιοθήκες προχωρούμε σε κανονικοποίηση και κατόπιν προχωρούμε σε τυχαίο διαχωρισμό δεδομένων με 80%-20% για τα σύνολα εκπαίδευσης και ελέγχου για όλες τις μεταβλητές.

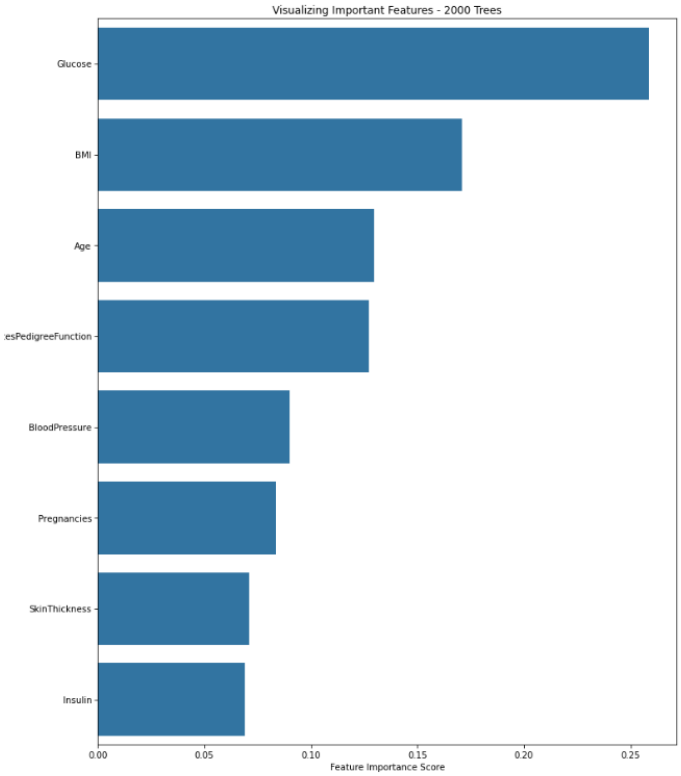
Στην συνέχεια και σε αντίθεση με την έως τώρα μορφολογία του κώδικα θα κατασκευάσουμε το παρακάτω for loop όπου θα κατασκευάσει μοντέλο για τις τρεις τιμές των ταξινομητών, θα επιστρέψει το ζητηθέν πίνακα ακρίβειας και θα παράξει σχετικά διαγράμματα σημαντικότητας των μεταβλητών.

```
>>> # Initialize results list
>>> results = []
>>>
>>> # Define the number of trees
>>> n_trees = [500, 1000, 2000]
>>>
>>> # Loop through different numbers of trees
>>> for n_tree in n_trees:
...     # Create RandomForestClassifier with the specified number of trees
...     rf_classifier = RandomForestClassifier(random_state=123, n_estimators=n_tree)
...
...     # Perform cross-validation
...     cv_scores = cross_val_score(rf_classifier, x_train, y_train, cv=5)
...
...     # Fit the model on the training set
...     rf_classifier.fit(x_train, y_train)
...
...     # Predictions on the testing dataset
...     predictions_train = rf_classifier.predict(x_train)
...
...     # Calculate testing accuracy
...     train_accuracy = accuracy_score(y_train, predictions_train)
...
...     # Predictions on the testing dataset
...     predictions_test = rf_classifier.predict(x_test)
...
...     # Calculate testing accuracy
...     test_accuracy = accuracy_score(y_test, predictions_test)
...
...     # Append results to the list
...     results.append([n_tree,cv_scores.mean(),test_accuracy])
...
...     # Visualize feature importance
...     feature_importances = pd.Series(rf_classifier.feature_importances_, index=x_train.columns).sort_values(ascending=False)
...     plt.figure(figsize=(10, 6))
...     sns.barplot(x=feature_importances, y=feature_importances.index)
...     plt.xlabel('Feature Importance Score')
...     plt.ylabel('Features')
...     plt.title(f"Visualizing Important Features - {n_tree} Trees")
...     plt.show()
...
... # Convert the list to a DataFrame
... results_df = pd.DataFrame(results, columns=['Number of Trees','Training Accuracy','Testing Accuracy'])
RandomForestClassifier(n_estimators=500, random_state=123)
<Figure size 1000x600 with 0 Axes>
<Axes: xlabel='None', ylabel='None'>
Text(0.5, 0, 'Feature Importance Score')
Text(0, 0.5, 'Features')
Text(0.5, 1.0, 'Visualizing Important Features - 500 Trees')
RandomForestClassifier(n_estimators=1000, random_state=123)
<Figure size 1000x600 with 0 Axes>
<Axes: xlabel='None', ylabel='None'>
Text(0.5, 0, 'Feature Importance Score')
Text(0, 0.5, 'Features')
Text(0.5, 1.0, 'Visualizing Important Features - 1000 Trees')
RandomForestClassifier(n_estimators=2000, random_state=123)
<Figure size 1000x600 with 0 Axes>
<Axes: xlabel='None', ylabel='None'>
Text(0.5, 0, 'Feature Importance Score')
Text(0, 0.5, 'Features')
Text(0.5, 1.0, 'Visualizing Important Features - 2000 Trees')
>>>
>>> # Print the results
>>> print(results_df)
   Number of Trees  Training Accuracy  Testing Accuracy
0              500           0.750806          0.818182
1             1000           0.749154          0.805195
2             2000           0.752432          0.805195
```

Στον παραπάνω κώδικα κατασκευάζουμε μοντέλο ταξινόμησης και κατόπιν υπολογίζουμε τις σχετικές ακρίβειες, όπου όπως φαίνονται στον παραπάνω πίνακα έχουμε πολύ ικανοποιητικά αποτελέσματα, δεν παρουσιάζουν σημαντικές διαφορές ανά πλήθος ταξινομητή και με την διαφορά στην ακρίβεια εκπαίδευσης (75%) και ελέγχου (81%) να κυμαίνεται στο 5% δεν υπάρχει μεγάλη πιθανότητα overfitting. Με την καλύτερη ισορροπία να εμφανίζεται για 500 ταξινομητές προχωρούμε στα επόμενα διαγράμματα.







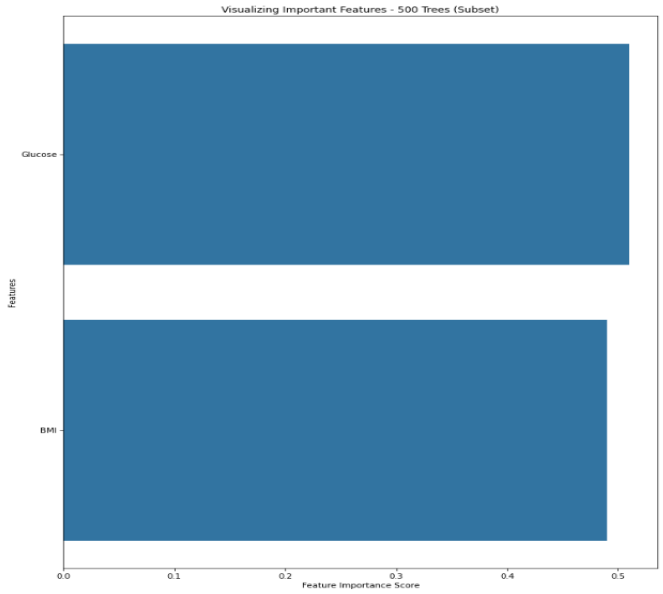
Ενώ στις ακρίβειες των συνόλων των δεδομένων ελέγχου και εκπαίδευσης εντοπίσαμε μικρές αυξομειώσεις σε σχέση με το στατιστικό πακέτο R (το οποίο μπορεί να οφείλετε και στα seed τυχαιότητας που ορίστηκαν) στα διαγράμματα εντοπίζονται πανομοιότυπα αποτελέσματα για τις σημαντικότερες μεταβλητές και τις επιμέρους αυξομειώσεις (Glucose, BMI και Age αντίστοιχα)

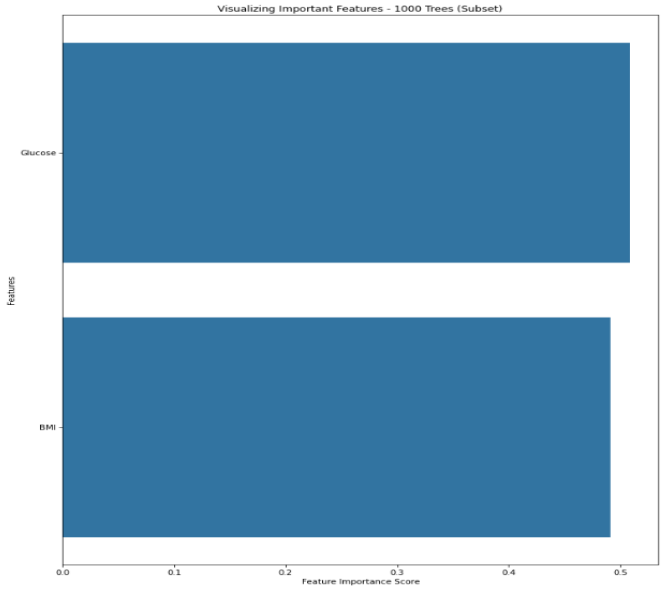
Συνεχίζοντας με τα μοντέλα που θα περιέχουν μόνο τις μεταβλητές “Glucose” & “BMI” βάσει της υπόθεσης:

```
>>> #ii) Random Forest for "Glucose" & "BMI"
>>>
>>> # Load Data
>>> data = pd.read_csv('diabetes.csv')
>>>
>>> # Select a subset of columns
>>> subset_columns = ["Glucose", "BMI"]
>>>
>>> # Standardize the subset data
>>> scale = StandardScaler()
>>> data[subset_columns] = scale.fit_transform(data[subset_columns])
>>>
>>> # Use only the subset columns
>>> x_subset = data[subset_columns]
>>> y = data['Outcome']
>>>
>>> # Split the scaled subset data into training and testing sets
>>> x_train_subset, x_test_subset, y_train, y_test = train_test_split(x_subset, y, train_size=0.8, random_state=12345)
>>>
>>> # Initialize results list
>>> results_subset = []
>>>
>>> # Define the number of trees
>>> n_trees = [500, 1000, 2000]
>>>
>>> # Loop through different numbers of trees
>>> for n_tree in n_trees:
...     # Create RandomForestClassifier with the specified number of trees
...     rf_classifier_subset = RandomForestClassifier(random_state=123, n_estimators=n_tree)
...
...     # Perform cross-validation on the subset
...     cv_scores_subset = cross_val_score(rf_classifier_subset, x_train_subset, y_train, cv=5)
...
...     # Fit the model on the training set (subset)
...     rf_classifier_subset.fit(x_train_subset, y_train)
...
...     # Predictions on the testing dataset (subset)
...     predictions_train_subset = rf_classifier_subset.predict(x_train_subset)
...
...     # Calculate testing accuracy (subset)
...     train_accuracy_subset = accuracy_score(y_train, predictions_train_subset)
...
...     # Predictions on the testing dataset (subset)
...     predictions_test_subset = rf_classifier_subset.predict(x_test_subset)
...
...     # Calculate testing accuracy (subset)
...     test_accuracy_subset = accuracy_score(y_test, predictions_test_subset)
...
...     # Append results to the list
...     results_subset.append([n_tree,cv_scores_subset.mean(),test_accuracy_subset])
...
...     # Visualize feature importance for the subset
...     feature_importances_subset = pd.Series(rf_classifier_subset.feature_importances_, index=x_train_subset.columns).sort_values(ascending=False)
...     plt.figure(figsize=(10, 6))
...     sns.barplot(x=feature_importances_subset, y=feature_importances_subset.index)
...     plt.xlabel('Feature Importance Score')
...     plt.ylabel('Features')
...     plt.title(f"Visualizing Important Features - {n_tree} Trees (Subset)")
...     plt.show()
...
... # Convert the list to a DataFrame for the subset
... results_df_subset = pd.DataFrame(results_subset, columns=['Number of Trees', 'Training Accuracy', 'Testing Accuracy'])
```

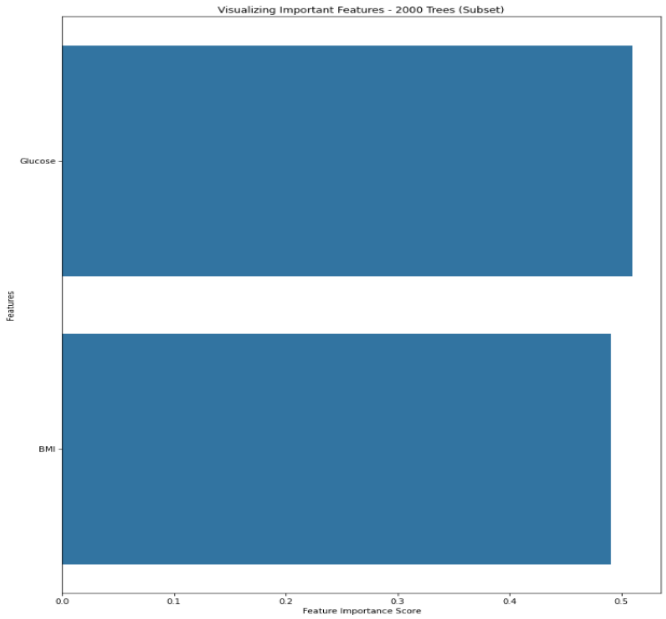
Όπου με την ίδια μορφολογία θα κατασκευάσουμε σχετικά μοντέλα ταξινόμισης για τα εκάστοτε πλήθη ταξινομητών. Στον παρακάτω πίνακα είναι εύκολο να παρατηρήσουμε πως βλέπουμε αρκετά ικανοποιητικές τιμές όμως μία γενικότερη μείωση της ακρίβειας στα δεδομένα εκπαίδευσης (72%) και στα δεδομένα ελέγχου (77%), χωρίς σημαντικές διαφορές ανά πλήθος ταξινομητή. Βέλτιστο εκ των τριών θα θεωρήσουμε το μοντέλο με τους 1000 ταξινομητές βάσει των αποτελεσμάτων του. Μεταξύ των δύο βέλτιστων μοντέλων των προσπαθειών για όλες τις μεταβλητές και για τις μεταβλητές "Glucose" & "BMI" θα έχουμε μία συμφωνία του στατιστικού πακέτου R με την γλώσσα προγραμματισμού python όπου θα επιλέξουμε τους 1000 ταξινομητές του μοντέλου με τις μεταβλητές "Glucose" & "BMI" για λόγους οικονομίας.

```
RandomForestClassifier(n_estimators=500, random_state=123)
<Figure size 1000x600 with 0 Axes>
<Axes: xlabel='None', ylabel='None'>
Text(0.5, 0, 'Feature Importance Score')
Text(0, 0.5, 'Features')
Text(0.5, 1.0, 'Visualizing Important Features - 500 Trees (Subset)')
RandomForestClassifier(n_estimators=1000, random_state=123)
<Figure size 1000x600 with 0 Axes>
<Axes: xlabel='None', ylabel='None'>
Text(0.5, 0, 'Feature Importance Score')
Text(0, 0.5, 'Features')
Text(0.5, 1.0, 'Visualizing Important Features - 1000 Trees (Subset)')
RandomForestClassifier(n_estimators=2000, random_state=123)
<Figure size 1000x600 with 0 Axes>
<Axes: xlabel='None', ylabel='None'>
Text(0.5, 0, 'Feature Importance Score')
Text(0, 0.5, 'Features')
Text(0.5, 1.0, 'Visualizing Important Features - 2000 Trees (Subset)')
>>>
>>> # Print the results for the subset
>>> print(results_df_subset)
   Number of Trees  Training Accuracy  Testing Accuracy
0              500           0.719872          0.766234
1             1000           0.718233          0.779221
2             2000           0.721485          0.766234
```





Επιπροσθέτως στα διαγράμματα σημαντικότητας των μεταβλητών παρατηρούμε κοντινές τιμές με ελάχιστες αυξομειώσεις, όπου η “Glucose” είναι σημαντικότερη του “BMI”.



ΑΣΚΗΣΗ 2η

- Feedforward Neural Network

Στατιστικό πακέτο R

```
> ##Feedforward Neural Network
>
> #import data
> data<- read.csv(file = "diabetes.csv")
>
> #change types of variables ton numeric and factor only
> data$Pregnancies <- as.numeric(data$Pregnancies)
> data$Glucose <- as.numeric(data$Glucose)
> data$BloodPressure <- as.numeric(data$BloodPressure)
> data$SkinThickness <- as.numeric(data$SkinThickness)
> data$Insulin <- as.numeric(data$Insulin)
> data$Age <- as.numeric(data$Age)
> data$Outcome <- as.factor(data$Outcome)
>
> #Normalize data (variables only) and check the results
> data[-9]<-scale(data[-9])
> str(data)
'data.frame':   768 obs. of  9 variables:
 $ Pregnancies             : num  0.64 -0.844 1.233 -0.844 -1.141 ...
 $ Glucose                 : num  0.848 -1.123 1.942 -0.998 0.504 ...
 $ BloodPressure           : num  0.15 -0.16 -0.264 -0.16 -1.504 ...
 $ SkinThickness           : num  0.907 0.531 -1.287 0.154 0.907 ...
 $ Insulin                 : num  -0.692 -0.692 -0.692 0.123 0.765 ...
 $ BMI                     : num  0.204 -0.684 -1.103 -0.494 1.409 ...
 $ DiabetesPedigreeFunction: num  0.468 -0.365 0.604 -0.92 5.481 ...
 $ Age                     : num  1.4251 -0.1905 -0.1055 -1.0409 -0.0205 ...
 $ Outcome                 : Factor w/ 2 levels "0","1": 2 1 2 1 2 1 2 1 2 2 ...
>
> #View(data) #if need to visually check the whole dataset
>
> #i)FNN for all variables
>
> # Create a reproducible random sampling
> set.seed(123)
>
> # Create training data as 80% of the dataset
> random_sample <- createDataPartition(data$Outcome, p = 0.8, list = FALSE)
> train_data <- data[random_sample, ]
> test_data <- data[-random_sample, ]
```

Ξεκινάμε διαμορφώνοντας κατάλληλα τα δεδομένα μας όσον αφορά τον τύπο των δεδομένων την κανονικοποίησή τους και τον διαχωρισμό τους σε 80%-20% σύνολα εκπαίδευσης και ελέγχου.

Στην συνέχεια θα προχωρήσουμε σε κατασκευή μοντέλων fnn μέσω for loop τα οποία βάσει της υπόθεσης θα έχουν σταθερά 2 κρυφά dense επίπεδα batch size 100, 300 epochs και συνάρτηση ενεργοποίησης RELU.

Μεταβλητέα θα δούμε τα πλήθη των νευρώνων με 4, 8 & 16 καθώς και το learning rate με 0.01 και 0.001 με adam βελτιωτή. Η συνάρτηση ενεργοποίησης της στοιβάδας εξόδου τέθηκε η sigmoid, η συνάρτηση loss function είναι η binary crossentropy.

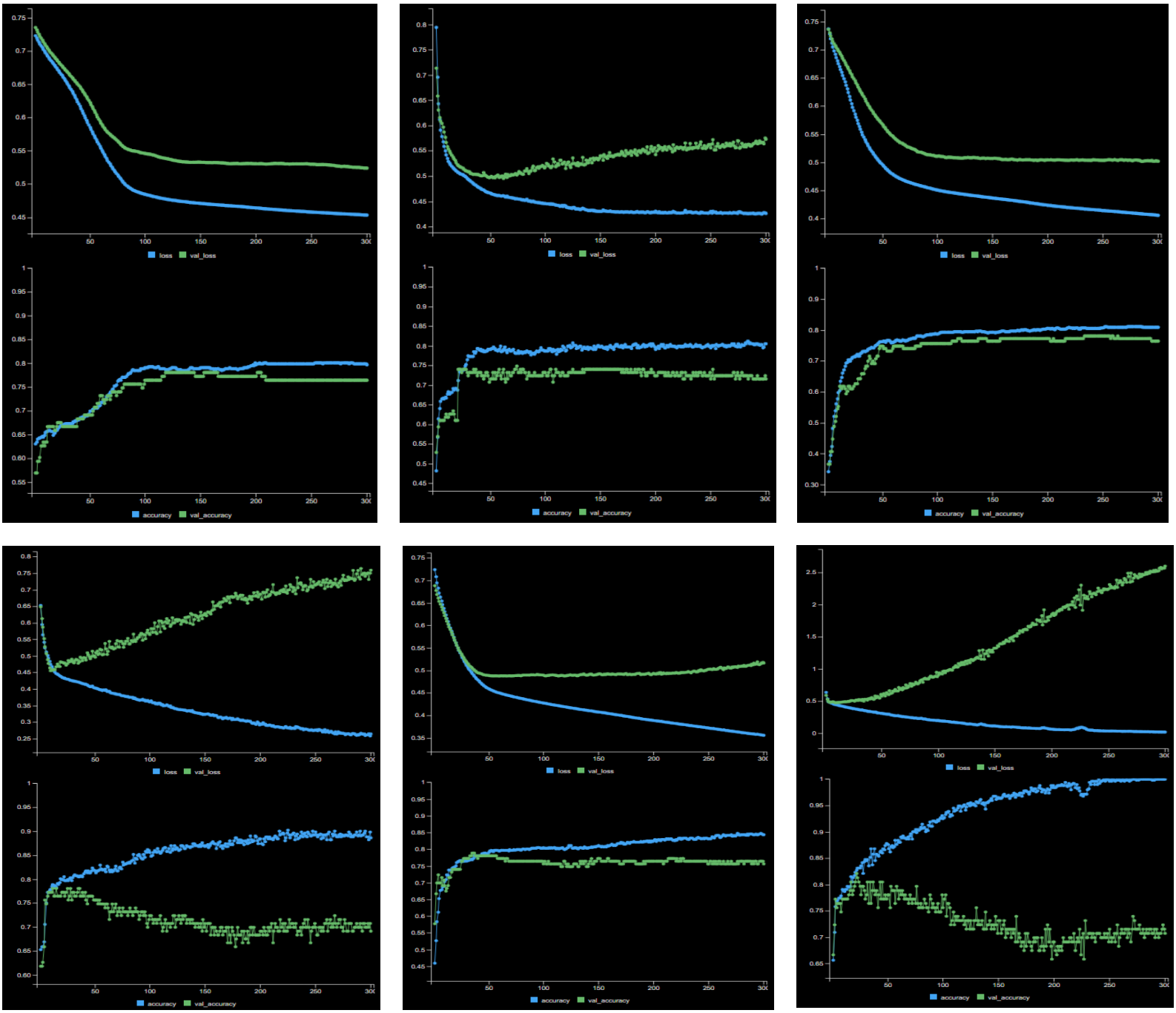
Ξεκινάμε με την λούπα κατασκευής, εκπαίδευσης και τραβήγματος των ακριβειών των συνόλων των δεδομένων εκπαίδευσης και ελέγχου. Για το μοντέλο χρησιμοποιήθηκαν οι βιβλιοθήκες Keras/Tensorflow.

```r
# Build and train the neural network model
train_and_evaluate_model <- function(neurons, learning_rate) {
  model <- keras_model_sequential() %>%
    layer_dense(units = neurons, activation = "relu", input_shape = c(8)) %>%
    layer_dense(units = neurons, activation = "relu") %>%
    layer_dense(units = 1, activation = "sigmoid")

  # Compile the model
  model %>% compile(
    loss = "binary_crossentropy",
    optimizer = optimizer_adam(learning_rate = learning_rate),
    metrics = "accuracy"
  )

  # Train the model
  history <- model %>% fit(
    x = as.matrix(train_data[, 1:8]),
    y = as.numeric(train_data$Outcome) - 1,
    epochs = 300,
    batch_size = 100,
    validation_split = 0.2
  )

  # Evaluate the model on the test set
  test_metrics <- model %>% evaluate(
    x = as.matrix(test_data[, 1:8]),
    y = as.numeric(test_data$Outcome) - 1
  )

  # Evaluate the model on the training set
  train_metrics <- model %>% evaluate(
    x = as.matrix(train_data[, 1:8]),
    y = as.numeric(train_data$Outcome) - 1
  )

  return(c(neurons, learning_rate, train_metrics["accuracy"], test_metrics["accuracy"]))
}
```

```r
# Matrix for results
results <- matrix(NA, nrow = 6, ncol = 4)
colnames(results) <- c("Neurons", "Learning Rate", "Train Accuracy", "Test Accuracy")

# Different parameters
neurons_list <- c(4, 8, 16)
learning_rates <- c(0.001, 0.01)

# Running the models and extracting results
index <- 1
for (neurons in neurons_list) {
  for (lr in learning_rates) {
    result <- train_and_evaluate_model(neurons, lr)
    results[index, ] <- result
    index <- index + 1
  }
}

# Display a summary table of results
summary_table <- as.data.frame(results)
print(summary_table)
```

Συνεχίζουμε με διαμόρφωση του πίνακα ακριβειών, καταχώρηση των μεταβλητών που προαναφέραμε βάσει υπόθεσης και τρέξιμο των 6 μοντέλων από τα οποία λαμβάνουμε τα επόμενα αποτελέσματα και διαγράμματα από το Viewer της R (για αυτό και δεν προχωρήσαμε σε περαιτέρω κατασκευή διαγραμμάτων).

```
> print(summary_table)
  Neurons Learning Rate Train Accuracy Test Accuracy
1       4          0.001      0.7902439     0.7973856
2       4          0.010      0.7886179     0.8169935
3       8          0.001      0.8000000     0.7908497
4       8          0.010      0.8552846     0.7973856
5      16          0.001      0.8325203     0.8039216
6      16          0.010      0.9414634     0.7581699
```



Έτσι μπορούμε να έχουμε μία συνολική εικόνα για τα παραπάνω μοντέλα. Διαδοχικά παρατηρούμε με τους περισσότερους νευρώνες και την μεγαλύτερη περίοδο μάθησης μεγαλύτερες ακρίβειες όμως με τις αποκλίσεις και το val loss των διαγραμμάτων υπάρχουν σε μερικές περιπτώσεις έντονες υποθέσεις για overfitting. Συγκεκριμένα οι προσπάθειες 8 και 16 νευρώνων με 0,01 learning rate έχουν εμφανά overfitting με πολύ αυξημένο validation loss και διαφορά στα accuracy και val accuracy στο δεύτερο μάλιστα υπάρχει και μεγάλη διαφορά στα accuracy ελέγχου και εκπαίδευσης. Τα υπόλοιπα μοντέλα είναι αποδεκτά ανάλογα με την περίσταση και το πιο ισορροπημένο μεταξύ γενίκευσης και ακρίβειας είναι το μοντέλο με 16 νευρώνες και learning rate 0.001 . Για μία ακόμα πιο ασφαλή επιλογή θα πηγαίναμε πάλι στο ίδιο learning rate με λιγότερους νευρώνες όπου θα χάναμε ακρίβεια άλλα θα κερδίζαμε στην γενίκευση.

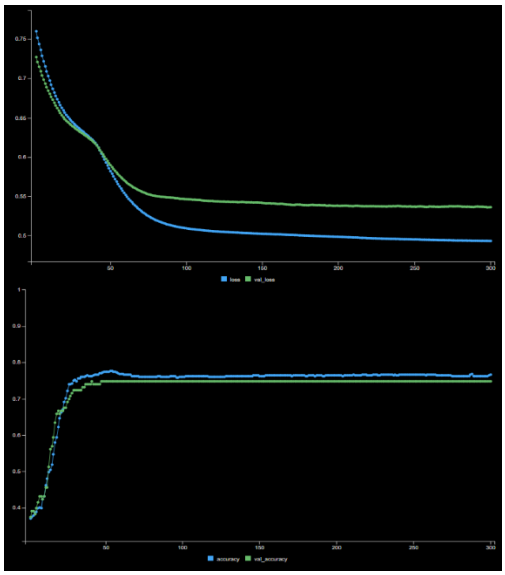
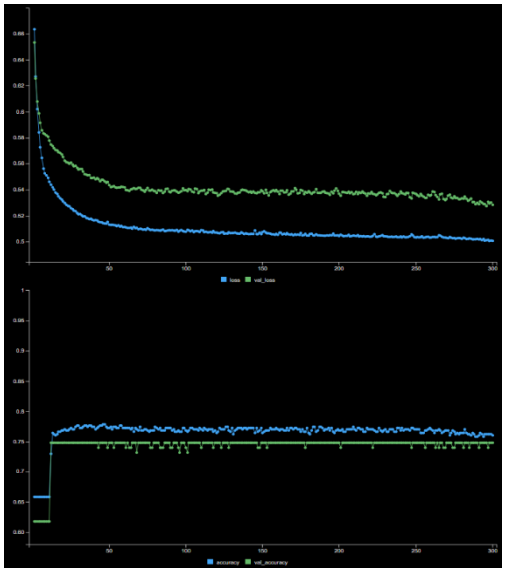
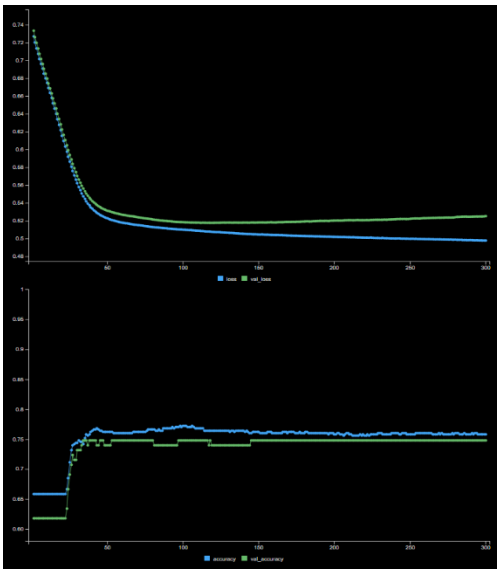
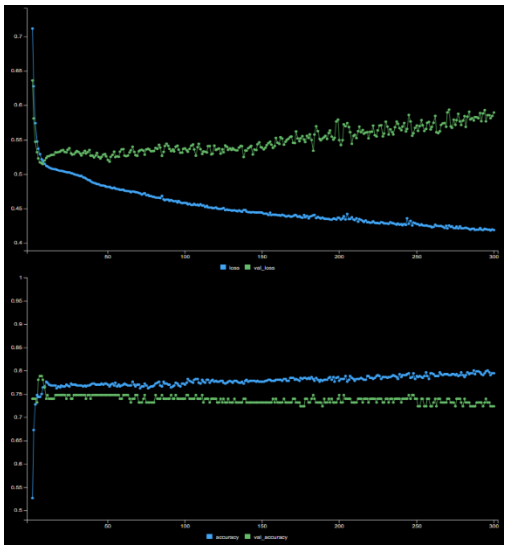
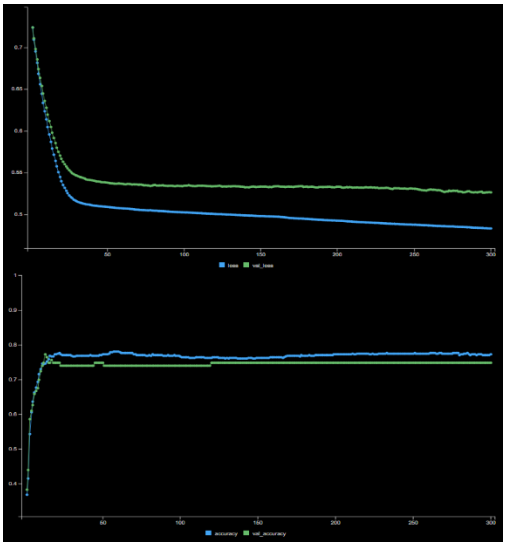
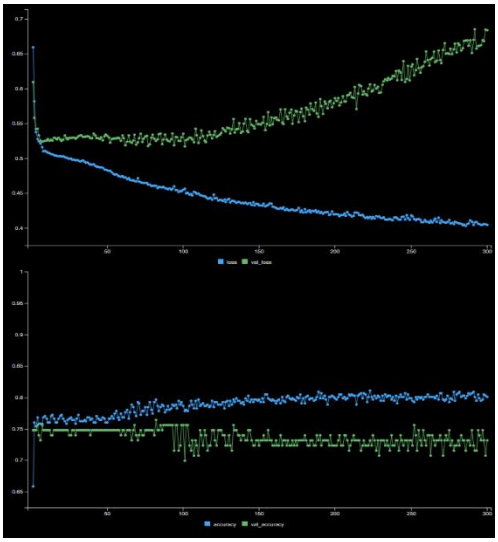
Συνεχίζοντας με τα μοντέλα που θα περιέχουν μόνο τις μεταβλητές “Glucose” & “BMI” βάσει της υπόθεσης:

```r
#ii) FNN for "Glucose" & "BMI"

# Build and train the neural network model
train_and_evaluate_model <- function(neurons, learning_rate) {
  model <- keras_model_sequential() %>%
    layer_dense(units = neurons, activation = "relu", input_shape = c(2)) %>%
    layer_dense(units = neurons, activation = "relu") %>%
    layer_dense(units = 1, activation = "sigmoid")

  model %>% compile(
    loss = "binary_crossentropy",
    optimizer = optimizer_adam(learning_rate = learning_rate),
    metrics = "accuracy"
  )

  history <- model %>% fit(
    x = as.matrix(train_data[, c("Glucose", "BMI")]),
    y = as.numeric(train_data$Outcome) - 1,
    epochs = 300,
    batch_size = 100,
    validation_split = 0.2
  )

  train_metrics <- model %>% evaluate(
    x = as.matrix(train_data[, c("Glucose", "BMI")]),
    y = as.numeric(train_data$Outcome) - 1
  )

  test_metrics <- model %>% evaluate(
    x = as.matrix(test_data[, c("Glucose", "BMI")]),
    y = as.numeric(test_data$Outcome) - 1
  )

  return(c(neurons, learning_rate, train_metrics["accuracy"], test_metrics["accuracy"]))
}

# Model parameters
neurons_list <- c(4, 8, 16)
learning_rates <- c(0.001, 0.01)

# Matrix for results
results <- matrix(NA, nrow = length(neurons_list) * length(learning_rates), ncol = 4)
colnames(results) <- c("Neurons", "Learning Rate", "Train Accuracy", "Test Accuracy")

# Running the models and extracting results
index <- 1
for (neurons in neurons_list) {
  for (lr in learning_rates) {
    result <- train_and_evaluate_model(neurons, lr)
    results[index, ] <- result
    index <- index + 1
  }
}

# Summary table of results
summary_table <- as.data.frame(results)
print(summary_table)
```

Αντίστοιχα προχωράμε σε κατασκευή for loop με τις ίδιες παραμέτρους όπου κατασκευάζει , εκπαιδεύει και υπολογίζει τις ακρίβειες στα δεδομένα μας.

Διαμορφώνουμε τον πίνακα ακριβειών, καταχωρούμε τις προς εξέταση μεταβλητές που προαναφέραμε βάσει υπόθεσης και τρέχουμε τα 6 μοντέλα από τα οποία λαμβάνουμε τα επόμενα αποτελέσματα και διαγράμματα από το Viewer της R.

```
> print(summary_table)
  Neurons Learning Rate Train Accuracy Test Accuracy
1       4         0.001      0.7609756     0.7777778
2       4         0.010      0.7593496     0.7777778
3       8         0.001      0.7560976     0.7712418
4       8         0.010      0.7788618     0.7908497
5      16         0.001      0.7674797     0.7843137
6      16         0.010      0.7853659     0.7908497
```

Σε αντίθεση με το προηγούμενο σετ δεδομένων παρατηρούμε λιγότερες διαφοροποιήσεις μεταξύ των ακριβειών με μία ελάχιστη αύξηση κατά το πλήθος των νευρώνων και μία πιο ομαλή γενίκευση ανά περίπτωση με τις εκάστοτε ακρίβειες να έχουν μειωθεί αλλά να αγγίζουν ακόμη πολύ καλά αποτελέσματα (μ.ο 0,77). Συγκεκριμένα αλλά σε πολύ μικρότερο βαθμό οι προσπάθειες 8 και 16 νευρώνων με 0,01 learning rate έχουν εμφανίζουν overfitting πράγμα που πλέον δεν φαίνεται στον πίνακα ακριβειών. Τα υπόλοιπα μοντέλα παραμένουν εξαιρετικά με βέλτιστο το μοντέλο με 16 νευρώνες και learning rate 0.001 το οποίο έχει τις υψηλότερες ακρίβειες των συνόλων εκπαίδευσης 0.77 και ελέγχου 0.78 και παράλληλα μία πολύ καλή γενίκευση.

Γλώσσα Προγραμματισμού Python

Με την ίδια αρχιτεκτονική ξεκινάμε διαμορφώνοντας κατάλληλα τα δεδομένα μας όσον αφορά τον τύπο των δεδομένων την κανονικοποίησή τους και τον διαχωρισμό τους σε 80%-20% σύνολα εκπαίδευσης και ελέγχου. Κατόπιν θα προχωρήσουμε σε κατασκευή μοντέλων fnn μέσω for loop τα οποία βάσει της υπόθεσης θα έχουν σταθερά 2 κρυφά dense επίπεδα batch size 100, 300 epochs και συνάρτηση ενεργοποίησης RELU.

Μεταβλητέα θα δούμε τα πλήθη των νευρώνων με 4, 8 & 16 καθώς και το learning rate με 0.01 και 0.001 με adam βελτιωτή. Η συνάρτηση ενεργοποίησης της στοιβάδας εξόδου τέθηκε η sigmoid, η συνάρτηση loss function είναι η binary crossentropy. Για το μοντέλο χρησιμοποιήθηκαν οι βιβλιοθήκες Keras/Tensorflow.

```
##Feedforward Neural Network

# Load Data
data = pd.read_csv('diabetes.csv')

# Convert outcome to a numeric array and explicitly cast to int64
y = np.array(data['Outcome'], dtype=np.int64)

# Normalize data (variables only)
scale = StandardScaler()
X = scale.fit_transform(data.iloc[:, :8])

# Create training data as 80% of the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)

# Build and train the neural network model
def build_and_train_model(neurons, learning_rate):
    model = Sequential([
        Dense(neurons, activation='relu', input_shape=(8,)),
        Dense(neurons, activation='relu'),
        Dense(1, activation='sigmoid')
    ])

    # Compile the model
    model.compile(
        loss='binary_crossentropy',
        optimizer=Adam(learning_rate=learning_rate),
        metrics=['accuracy']
    )

    # Train the model
    history = model.fit(
        x=X_train,
        y=y_train,
        epochs=300,
        batch_size=100,
        validation_split=0.2
    )

    # Evaluate the model on the test set
    test_metrics = model.evaluate(x=X_test, y=y_test)
    train_metrics = model.evaluate(x=X_train, y=y_train)

    # Plot training history
    plot_training_history(neurons, learning_rate, history)

    return neurons, learning_rate, train_metrics[1], test_metrics[1]
```

Ξεκινάμε με την λούπα κατασκευής εκπαίδευσης και τραβήγματος των ακριβειών των συνόλων των δεδομένων εκπαίδευσης και ελέγχου.

Συνεχίζουμε με κατασκευή διαγραμμάτων ακρίβειας και loss.

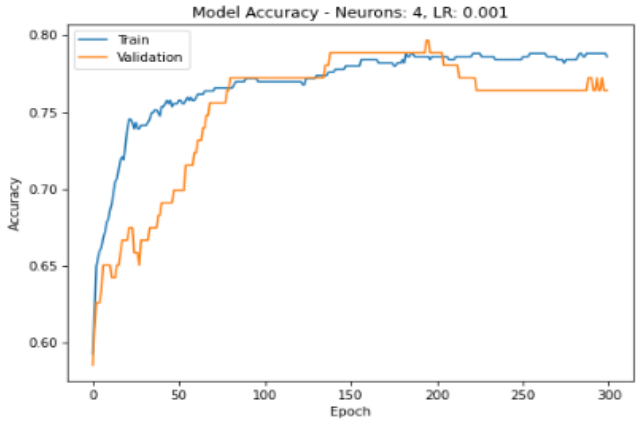
```python
# Function to plot training history
def plot_training_history(neurons, learning_rate, history):
    plt.figure(figsize=(12, 6))

    # Plot training & validation accuracy values
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title(f'Model Accuracy - Neurons: {neurons}, LR: {learning_rate}')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(['Train', 'Validation'], loc='upper left')

    # Plot training & validation loss values
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title(f'Model Loss - Neurons: {neurons}, LR: {learning_rate}')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(['Train', 'Validation'], loc='upper left')

    plt.tight_layout()
    plt.show()

# Matrix for results
results = []

# Different parameters
neurons_list = [4, 8, 16]
learning_rates = [0.001, 0.01]

# Running the models and extracting results
for neurons in neurons_list:
    for lr in learning_rates:
        result = build_and_train_model(neurons, lr)
        results.append(result)

# Display a summary table of results
summary_table = pd.DataFrame(results, columns=['Neurons', 'Learning Rate', 'Train Accuracy', 'Test Accuracy'])
print(summary_table)
```

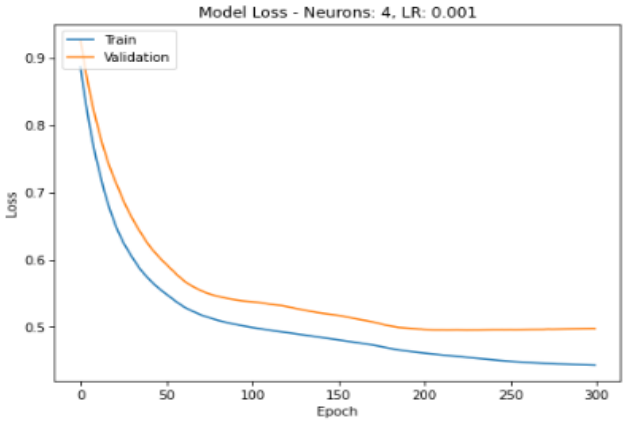
Και τελικά στην καταχώρηση των μεταβλητών που προαναφέραμε βάσει υπόθεσης όπου θα καταλήξουμε στο τρέξιμο των 6 μοντέλων από τα οποία λαμβάνουμε τα επόμενα αποτελέσματα και διαγράμματα.

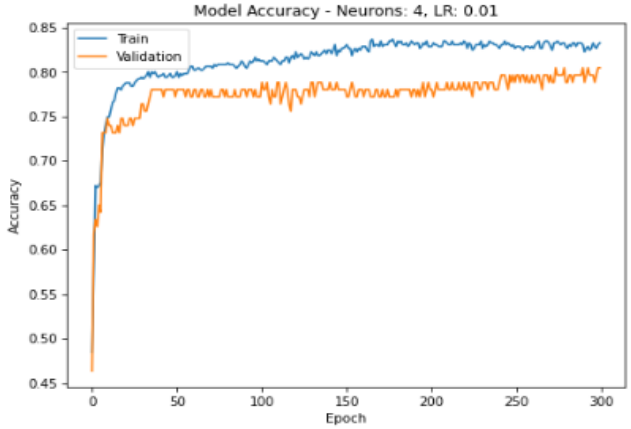
```
>>> print(summary_table)
   Neurons  Learning Rate  Train Accuracy  Test Accuracy
0        4          0.001        0.783388       0.785714
1        4          0.010        0.827362       0.772727
2        8          0.001        0.807818       0.753247
3        8          0.010        0.861564       0.733766
4       16          0.001        0.843648       0.772727
5       16          0.010        0.942997       0.733766
```

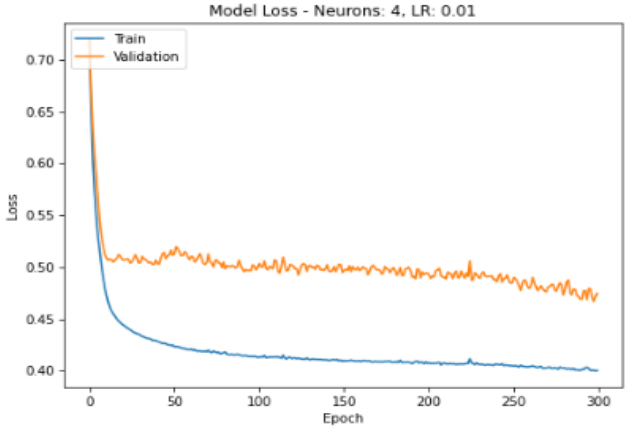
Υπάρχει μία εμφανής αστάθεια στις διακυμάνσεις των ακριβειών των συνόλων εκπαίδευσης και ελέγχου (αναμενόμενη από τα αποτελέσματα του πακέτου R) όπου σε συνδυασμό με τα επόμενα διαγράμματα μπορούμε να αποφανθούμε τα εξής:

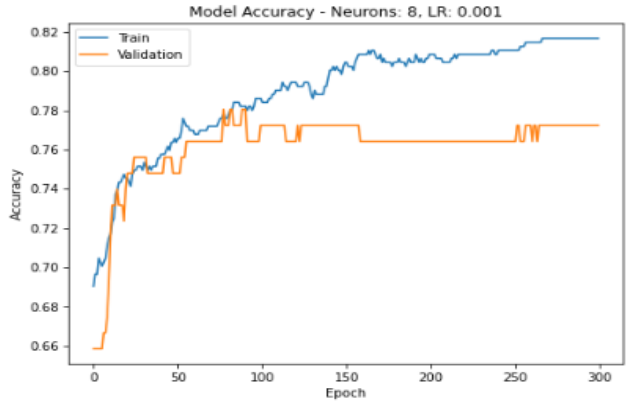
Όσο περισσότεροι νευρώνες τόσο μεγαλύτερες ακρίβειες. Τα σύνολα δεδομένων ελέγχουν φέρουν παρόμοιες ακρίβειες και είναι εμφανής οι περιπτώσεις που αφορούν overfitting. Συγκεκριμένα οι προσπάθειες 8 και 16 με μεγάλη διαφορά μεταξύ των ακριβειών εκπαίδευσης και ελέγχουν εμφανίζουν επίσης πολύ αυξημένο validation loss και διαφορά στα accuracy και val accuracy. Τα υπόλοιπα μοντέλα είναι αποδεκτά (με ειδικό χειρισμό για το 5^{ο} καθώς εμφανίζει δείγμα overfit) ανάλογα με την περίσταση. Ως βέλτιστο λόγω των μεγάλων αποκλίσεων και των διαγραμμάτων θα προχωρήσουμε με την ασφαλέστερη και πιο έτοιμη για γενίκευση προσπάθεια με το μοντέλο των 4 νευρώνων και 0,001 learning rate το οποίο εξίσου φέρει ένα εξαιρετικό accuracy 0.78 .

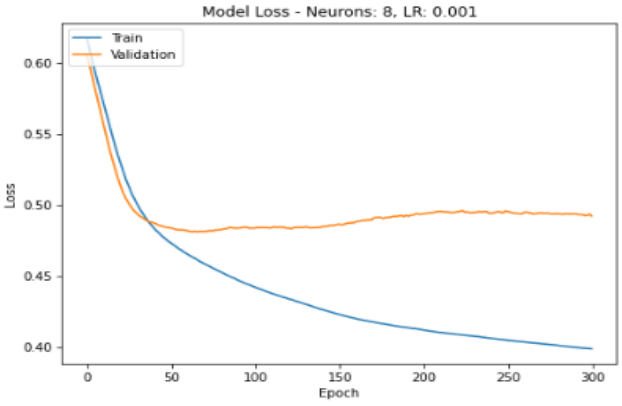

Model Accuracy - Neurons: 4, LR: 0.001
Train
Validation
Accuracy
Epoch

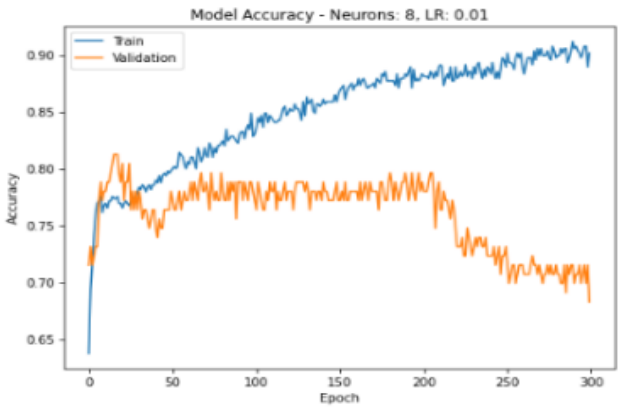

Model Loss - Neurons: 4, LR: 0.001
Train
Validation
Loss
Epoch

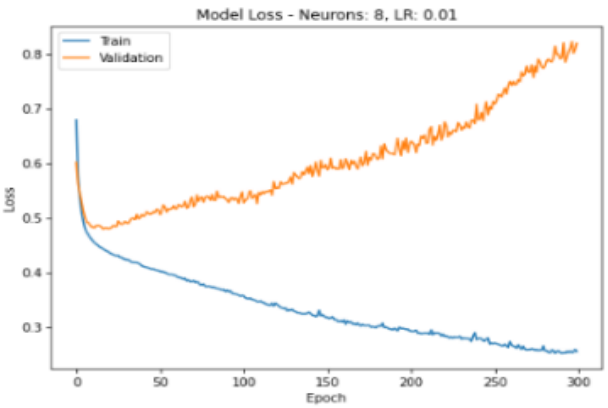

Model Accuracy - Neurons: 4, LR: 0.01
Train
Validation
Accuracy
Epoch

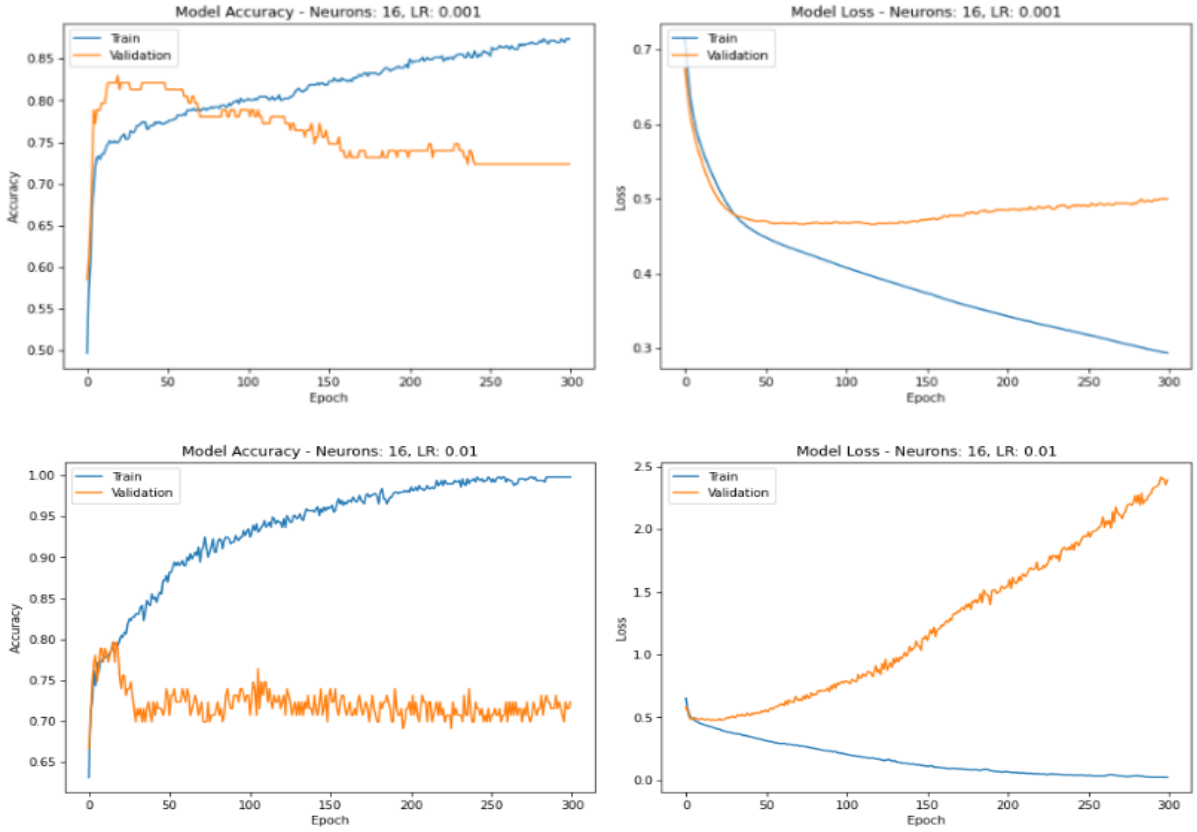

Model Loss - Neurons: 4, LR: 0.01
Train
Validation
Loss
Epoch


Model Accuracy - Neurons: 8, LR: 0.001
Train
Validation
Accuracy
Epoch


Model Loss - Neurons: 8, LR: 0.001
Train
Validation
Loss
Epoch


Model Accuracy - Neurons: 8, LR: 0.01
Train
Validation
Accuracy
Epoch


Model Loss - Neurons: 8, LR: 0.01
Train
Validation
Loss
Epoch



Τελικά θα προχωρήσουμε και στην υλοποίηση με τα μοντέλα που θα περιέχουν μόνο τις μεταβλητές “Glucose” & “BMI” βάσει της υπόθεσης:

```
#ii) FNN for “Glucose” & “BMI”

# Load Data
data = pd.read_csv('diabetes.csv')

# Change types of variables to numeric and factor only
data['Outcome'] = pd.Categorical(data['Outcome']).codes

# Normalize data (variables only) and check the results
scale = StandardScaler()
data[['Glucose', 'BMI']] = scale.fit_transform(data[['Glucose', 'BMI']])

# Convert outcome to a numeric array and explicitly cast to int64
y = np.array(data['Outcome'], dtype=np.int64)

# Create a reproducible random sampling
np.random.seed(123)
random_sample = np.random.rand(len(data)) < 0.8
train_data = data[random_sample]
test_data = data[~random_sample]
```

Αντίστοιχα προχωράμε διαμόρφωση των δεδομένων όσον αφορά τον τύπο των δεδομένων την κανονικοποίησή τους και τον διαχωρισμό τους σε 80%-20% σύνολα εκπαίδευσης και ελέγχου.

```python
# Build and train the neural network model
def build_and_train_model(neurons, learning_rate):
    model = Sequential()
    model.add(Dense(neurons, activation='relu', input_shape=(2,)))
    model.add(Dense(neurons, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    # Compile the model
    model.compile(
        loss='binary_crossentropy',
        optimizer='adam',
        metrics=['accuracy']
    )

    # Train the model
    history = model.fit(
        x=np.array(train_data[['Glucose', 'BMI']]),
        y=y[train_data.index],
        epochs=300,
        batch_size=100,
        validation_split=0.2
    )

    # Evaluate the model on the test set
    test_metrics = model.evaluate(
        x=np.array(test_data[['Glucose', 'BMI']]),
        y=y[test_data.index]
    )

    # Evaluate the model on the training set
    train_metrics = model.evaluate(
        x=np.array(train_data[['Glucose', 'BMI']]),
        y=y[train_data.index]
    )

    # Plot training history
    plot_training_history(neurons, learning_rate, history)

    return [neurons, learning_rate, train_metrics[1], test_metrics[1]]
```

Κατόπιν κατασκευάζουμε for loop με τις ίδιες παραμέτρους όπου κατασκευάζει , εκπαιδεύει και υπολογίζει τις ακρίβειες στα δεδομένα μας.

Συνεχίζουμε με κατασκευή διαγραμμάτων ακρίβειας και loss.

```python
# Function to plot training history
def plot_training_history(neurons, learning_rate, history):
    plt.figure(figsize=(12, 6))

    # Plot training & validation accuracy values
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title(f'Model Accuracy - Neurons: {neurons}, LR: {learning_rate}')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(['Train', 'Validation'], loc='upper left')

    # Plot training & validation loss values
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title(f'Model Loss - Neurons: {neurons}, LR: {learning_rate}')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(['Train', 'Validation'], loc='upper left')

    plt.tight_layout()
    plt.show()
```

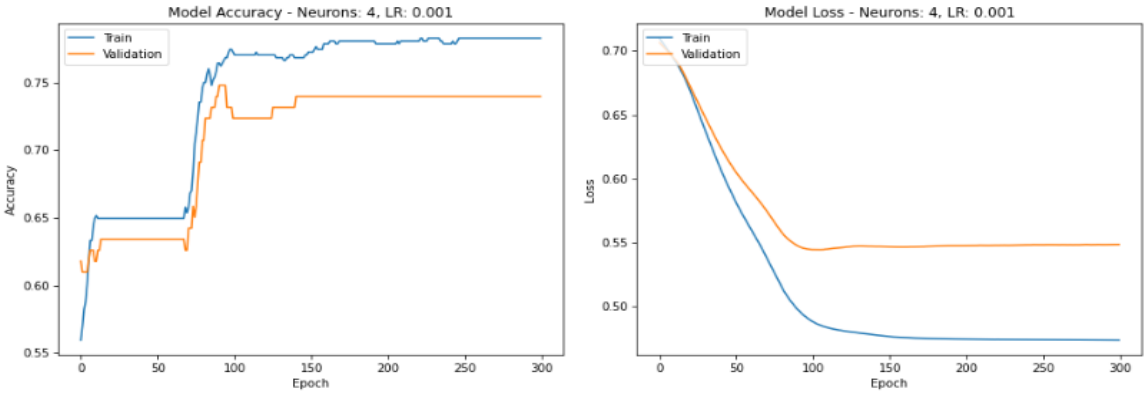
```
# Matrix for results
results = np.empty((6, 4))
results[:] = np.nan
colnames = ["Neurons", "Learning Rate", "Train Accuracy", "Test Accuracy"]

# Different parameters
neurons_list = [4, 8, 16]
learning_rates = [0.001, 0.01]

# Running the models and extracting results
index = 0
for neurons in neurons_list:
    for lr in learning_rates:
        result = build_and_train_model(neurons, lr)
        results[index, :] = result
        index += 1

# Display a summary table of results
summary_table = pd.DataFrame(results, columns=colnames)
print(summary_table)
```

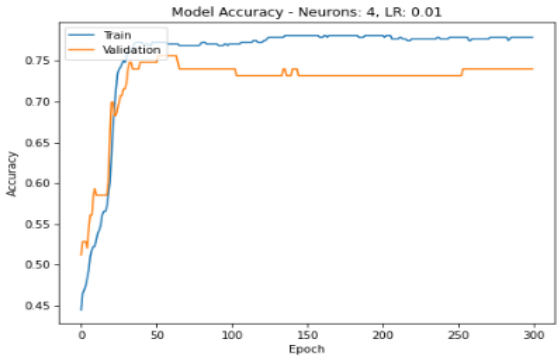
Και τελικά στην καταχώρηση των μεταβλητών που προαναφέραμε βάσει υπόθεσης όπου θα καταλήξουμε στο τρέξιμο των 6 μοντέλων από τα οποία λαμβάνουμε τα επόμενα αποτελέσματα και διαγράμματα.

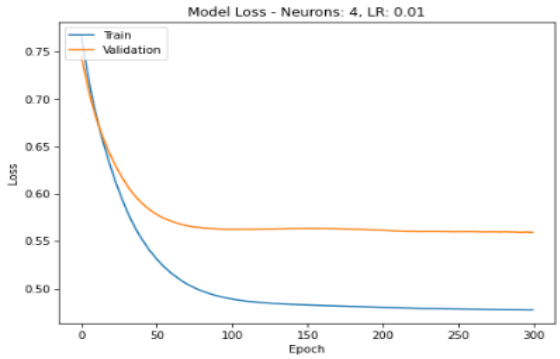
```
>>> print(summary_table)
   Neurons  Learning Rate  Train Accuracy  Test Accuracy
0      4.0          0.001        0.774141       0.777070
1      4.0          0.010        0.770867       0.777070
2      8.0          0.001        0.767594       0.751592
3      8.0          0.010        0.772504       0.745223
4     16.0          0.001        0.788871       0.726115
5     16.0          0.010        0.785597       0.738854
```

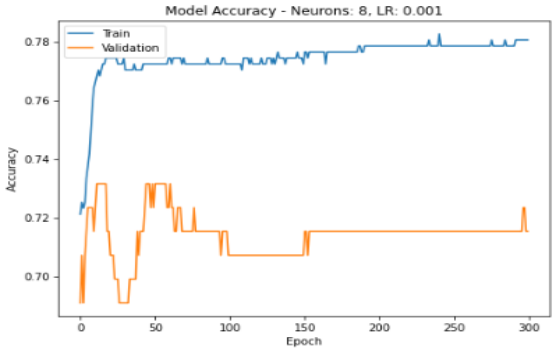
Υπάρχει μία εμφανής ισορροπία στις διακυμάνσεις των ακριβειών των συνόλων εκπαίδευσης και ελέγχου σε αντίθεση με τις προηγούμενες μεταβλητές όπου σε συνδυασμό με τα επόμενα διαγράμματα μπορούμε να αποφανθούμε τα εξής:

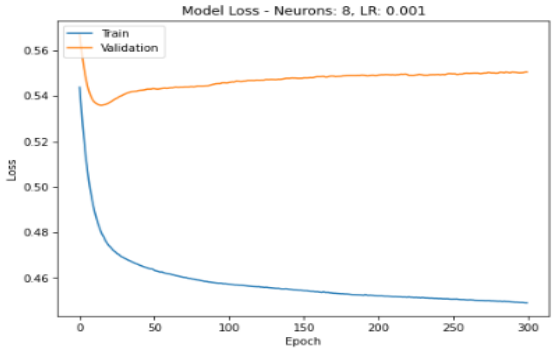
Σε αντίθεση με το προηγούμενο σετ δεδομένων παρατηρούμε λιγότερες διαφοροποιήσεις και διακυμάνσεις μεταξύ των ακριβειών με μία ελάχιστη αύξηση κατά το πλήθος των νευρώνων στα σύνολα εκπαίδευσης ενώ μείωση στα ελέγχου και μία πιο ομαλή γενίκευση ανά περίπτωση με τις εκάστοτε ακρίβειες να έχουν μειωθεί αλλά να αγγίζουν ακόμη πολύ καλά αποτελέσματα (μ.ο 0,77). Συγκεκριμένα αλλά σε πολύ μικρότερο βαθμό οι προσπάθειες 8 και 16 νευρώνων με 0,01 learning rate έχουν εμφανίζουν overfitting πράγμα που πλέον δεν φαίνεται στον πίνακα ακριβειών. Η περίπτωση 8 νευρώνων με 0.001 δεν αποκλείνει πολύ εάν παρατηρήσουμε τις κλίμακες ώστε να θεωρείται overfitting. Τα υπόλοιπα μοντέλα παραμένουν εξαιρετικά με βέλτιστο το μοντέλο με 4 νευρώνες και learning rate 0.001 το οποίο έχει τις υψηλότερες ακρίβειες των συνόλων εκπαίδευσης 0.77 και ελέγχου 0.77 και παράλληλα μία πολύ καλή γενίκευση. Επιμέρους, εμφανίστηκαν διαφορές μεταξύ R και Python, με τα καλύτερα μοντέλα me learning rate 0.001 στους 16 και 4 νευρώνες αντίστοιχα. Όπως προαναφέραμε το seed τυχαιποίησης σίγουρα συμβάλει σε αυτό. Γενικά παρατηρήσαμε ίδιες μορφολογίες και αναμενόμενες συμπεριφορές μεταξύ τους.

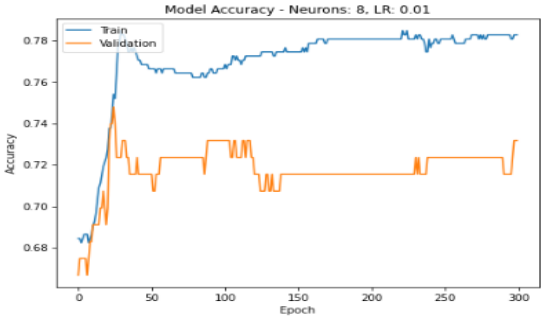


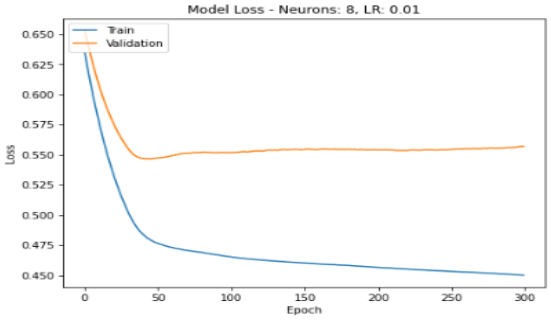

Model Accuracy - Neurons: 4, LR: 0.01
Train
Validation
Accuracy
Epoch

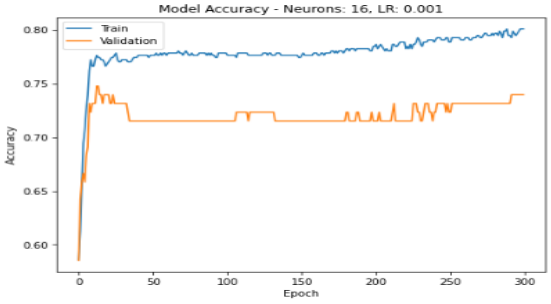

Model Loss - Neurons: 4, LR: 0.01
Train
Validation
Loss
Epoch

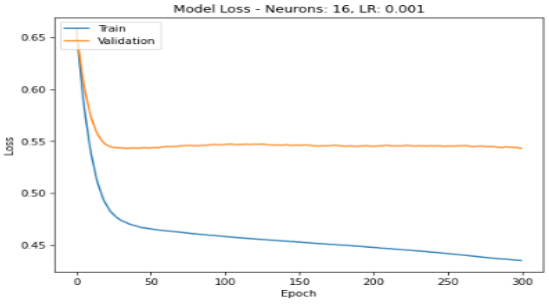

Model Accuracy - Neurons: 8, LR: 0.001
Train
Validation
Accuracy
Epoch

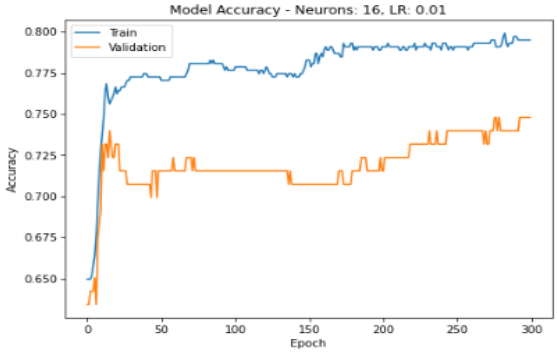

Model Loss - Neurons: 8, LR: 0.001
Train
Validation
Loss
Epoch

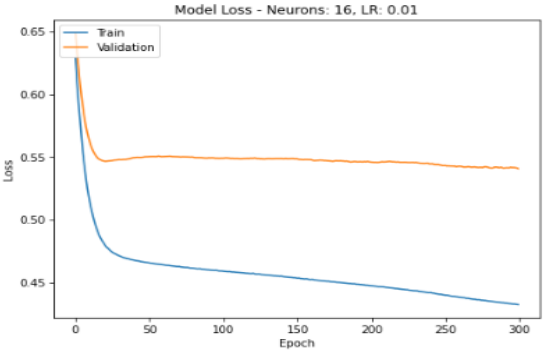

Model Accuracy - Neurons: 8, LR: 0.01
Train
Validation
Accuracy
Epoch


Model Loss - Neurons: 8, LR: 0.01
Train
Validation
Loss
Epoch


Model Accuracy - Neurons: 16, LR: 0.001
Train
Validation
Accuracy
Epoch


Model Loss - Neurons: 16, LR: 0.001
Train
Validation
Loss
Epoch


Model Accuracy - Neurons: 16, LR: 0.01
Train
Validation
Accuracy
Epoch


Model Loss - Neurons: 16, LR: 0.01
Train
Validation
Loss
Epoch