Diabetes Classification

David

2024-11-15

Introduction

Diabetes is a metabolic condition causing excessive blood sugar levels (MSD Manual). Diabetes occurs when the pancreas either doesn't produce enough insulin or can't use the insulin it produces effectively. Insulin is a hormone that regulates blood glucose. The hormone insulin transfers sugar from the blood into the cells for storage or energy use. Patients with the potential of diabetes have to go through a series of tests and examinations to diagnose the disease properly, which are expensive (Tasin, Nabil, Islam, & Khan, 2022). A predictive model that can accurately detect diabetes is therefore needed, as it will help in early detection and treatment/management of the disease.

Study Objective

The main objective of this study was to develop a predictive model that can accurately predict (with high precision) the likelihood of developing diabetes, and identify the most important predictors of diabetes.

```
# Load packages
suppressMessages(
  {
    library(tidyverse)
    library(janitor)
    library(caret)
    library(mlr)
    library(tidymodels)
    library(pROC)
    library(vip)
    library(parallel)
    library(parallelMap)
  }
)

# Import data
diabetes <- read_csv("diabetes.txt")

## Rows: 15000 Columns: 10
## ── Column specification
────────────────────────────────────────────────────────
## Delimiter: ","
## dbl (10): PatientID, Pregnancies, PlasmaGlucose, DiastolicBloodPressure,
```

```
Tri...
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this
message.

# View the structure of the dataset
diabetes |> glimpse()

## Rows: 15,000
## Columns: 10
## $ PatientID              <dbl> 1354778, 1147438, 1640031, 1883350,
1424119, 16…
## $ Pregnancies            <dbl> 0, 8, 7, 9, 1, 0, 0, 0, 8, 1, 1, 3, 5, 7,
0, 3,…
## $ PlasmaGlucose          <dbl> 171, 92, 115, 103, 85, 82, 133, 67, 80, 72,
88,…
## $ DiastolicBloodPressure <dbl> 80, 93, 47, 78, 59, 92, 47, 87, 95, 31, 86,
96,…
## $ TricepsThickness       <dbl> 34, 47, 52, 25, 27, 9, 19, 43, 33, 40, 11,
31, …
## $ SerumInsulin           <dbl> 23, 36, 35, 304, 35, 253, 227, 36, 24, 42,
58, …
## $ BMI                    <dbl> 43.50973, 21.24058, 41.51152, 29.58219,
42.6045…
## $ DiabetesPedigree       <dbl> 1.21319135, 0.15836498, 0.07901857,
1.28286985,…
## $ Age                    <dbl> 21, 23, 23, 43, 22, 26, 21, 26, 53, 26, 22,
23,…
## $ Diabetic               <dbl> 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
0, 1,…
```

The data contains 15000 observations of 10 variables. All the variables are numeric (double).

Clean the data

```
# Clean variable names
diabetes <- diabetes |> clean_names()

# Check for missing values
diabetes |> map_dbl(~ sum(is.na(.)))

##               patient_id              pregnancies          plasma_glucose
##                        0                        0                       0
## diastolic_blood_pressure        triceps_thickness            serum_insulin
##                        0                        0                       0
##                      bmi        diabetes_pedigree                      age
##                        0                        0                       0
##                 diabetic
##                        0
```

There are no missing values in the data.

```
# Check for duplicated observations
which(duplicated(diabetes))

## integer(0)
```

There are no duplicated observations.

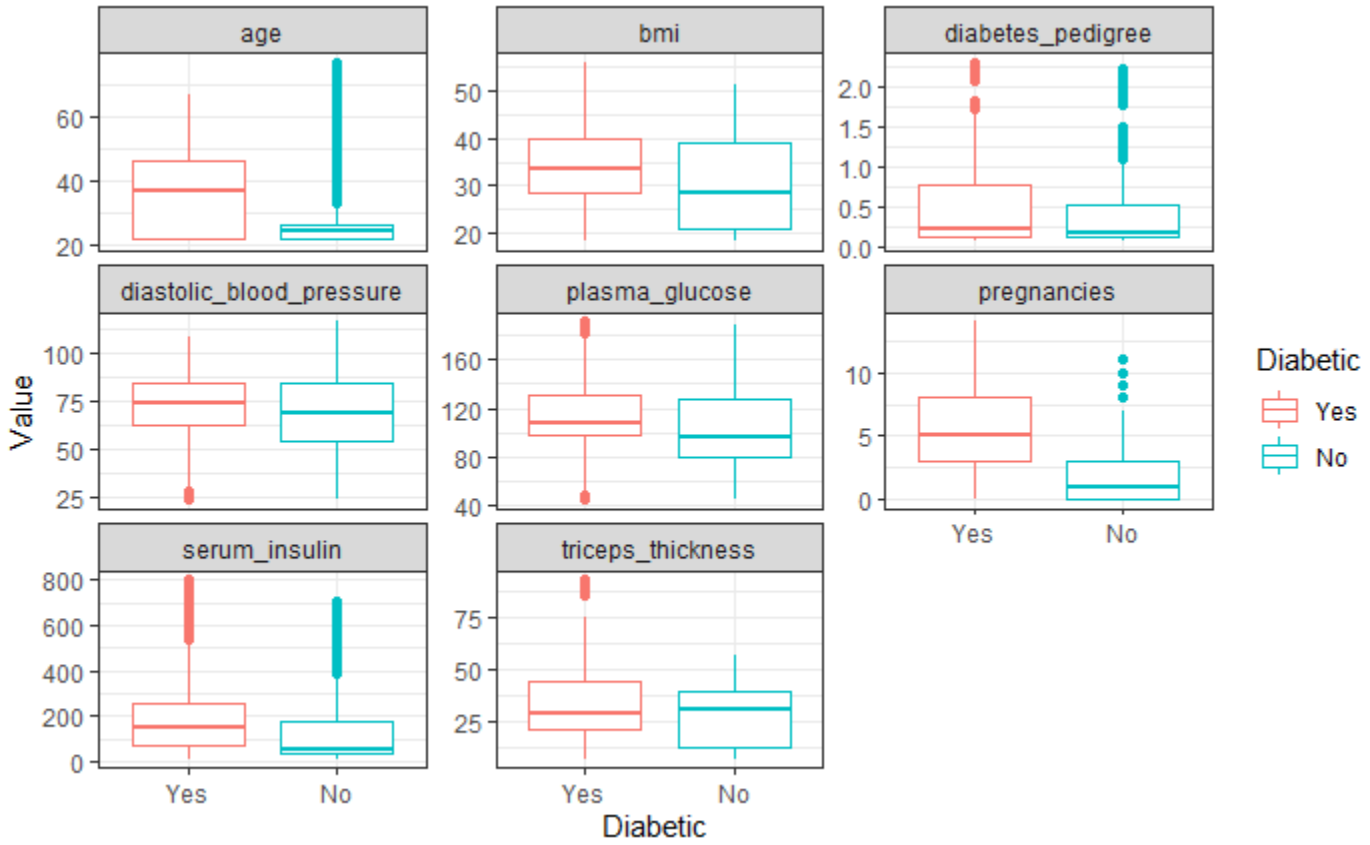
```
# Deselect patient_id variable
dbTib <- diabetes |> select(-patient_id)
# Factor the response variable and reverse levels to begin with the positive
class
dbTib[["diabetic"]] <- factor(dbTib[["diabetic"]],
                              levels = rev(c(0,1)),
                              labels = rev(c("No", "Yes")))
```

EDA

```
# Generate summary statistics for each and every variable.
dbTib |> summary()

##   pregnancies     plasma_glucose  diastolic_blood_pressure
triceps_thickness
##  Min.   : 0.000   Min.   : 44.0   Min.   : 24.00           Min.   : 7.00
##  1st Qu.: 0.000   1st Qu.: 84.0   1st Qu.: 58.00           1st Qu.:15.00
##  Median : 2.000   Median :104.0   Median : 72.00           Median :31.00
##  Mean   : 3.225   Mean   :107.9   Mean   : 71.22           Mean   :28.81
##  3rd Qu.: 6.000   3rd Qu.:129.0   3rd Qu.: 85.00           3rd Qu.:41.00
##  Max.   :14.000   Max.   :192.0   Max.   :117.00           Max.   :93.00
##  serum_insulin        bmi        diabetes_pedigree      age
diabetic
##  Min.   : 14.0   Min.   :18.20   Min.   :0.07804   Min.   :21.00   Yes:
5000
##  1st Qu.: 39.0   1st Qu.:21.26   1st Qu.:0.13774   1st Qu.:22.00   No
:10000
##  Median : 83.0   Median :31.77   Median :0.20030   Median :24.00
##  Mean   :137.9   Mean   :31.51   Mean   :0.39897   Mean   :30.14
##  3rd Qu.:195.0   3rd Qu.:39.26   3rd Qu.:0.61629   3rd Qu.:35.00
##  Max.   :799.0   Max.   :56.03   Max.   :2.30159   Max.   :77.00
```

- The minimum and first quartile for the number of pregnancies is zero. Male patients definitely have zero number of pregnancies. The median number of pregnancies is 2 and the maximum number of pregnancies is 14.
- The median values for plasma glucose, diastolic blood pressure, triceps thickness, serum insulin, BMI and diabetes pedigree are 104, 72, 31, 83, 31.77 and 0.2 units respectively. The median age is 24 years.
- Diabetic patients are 5000 while non-diabetic patients are 10,000 in number. There's class imbalance in the data.

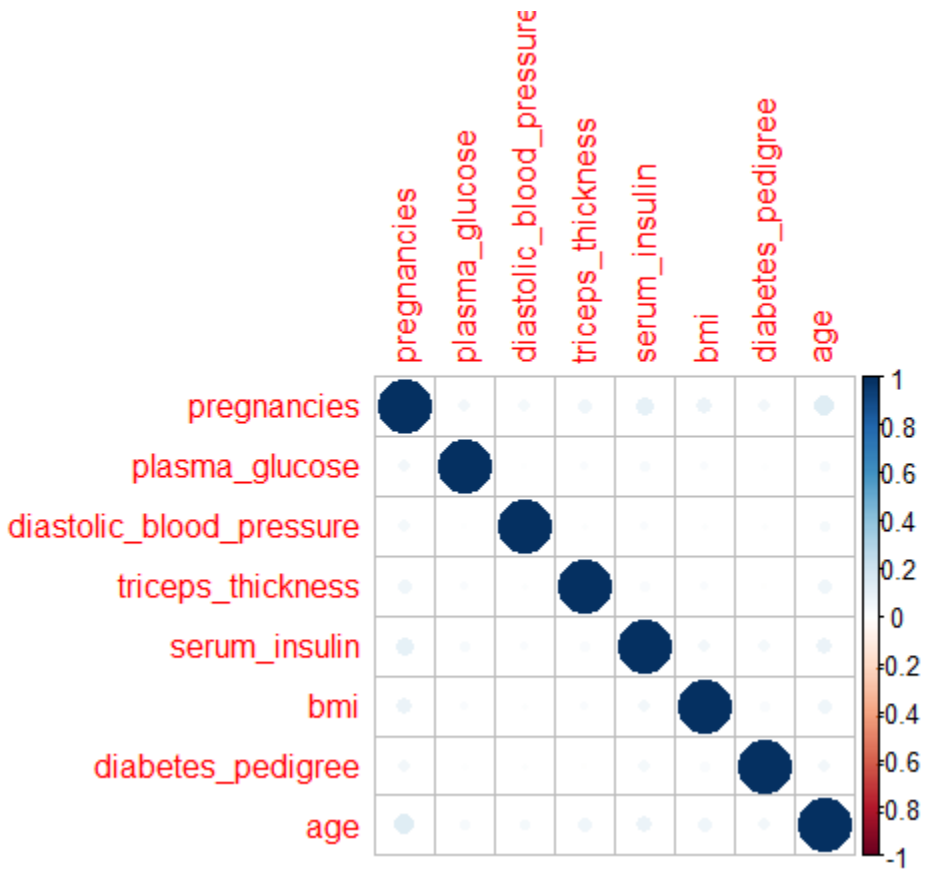
```r
# Convert the data into longer format for visualization
diabetesUntidy <- gather(dbTib, key = "Variable", value = "Value", -
diabetic)

# Visualize the data
diabetesUntidy |> ggplot(aes(diabetic, as.numeric(Value), colour = diabetic))
+
  geom_boxplot() +
  facet_wrap(~ Variable, scales = "free_y") +
  theme_bw() +
  labs(x = "Diabetic", y = "Value",
       colour = "Diabetic")
```



- Most of non-diabetic patients are younger (less than 30 years). There are also numerous elder patients who are non-diabetic.
- Diabetic patients have higher BMI, high serum insulin, high plasma glucose and thicker triceps.
- Female diabetic patients are more likely to have had high number of pregnancies, or given birth to many children.

Based on the distributions of age, BMI, diabetes pedigree, plasma glucose, number of pregnancies and serum insulin, the two classes seem to be separable.

```
# Check for highly correlated features
corrplot::corrplot(cor(dbTib[-9]))
```



All the features are lowly correlated. There's no problem of multicollinearity.

Model Training

I'll try 6 different algorithms i.e Logistic Regression, Naive Bayes classifier, KNN, Random Forest, XGBoost and an Artificial Neural Network. Before training the models, I'll first split the data into training and test sets. The training set will be used to train and fine-tune the models with cross-validation, and the test sets will be used for model validation.

```
# Partition the data into training and test sets (use 75/25 split)
# Set random seed for reproducibility
set.seed(1234)

# Data partitioning
train_index <- createDataPartition(dbTib$diabetic, p = 0.75, list = FALSE)
```

```r
# Assign 75% to training set
training_data <- dbTib[train_index, ]
# Assign the remaining 25% to test set
test_data <- dbTib[-train_index, ]
```

Logistic Regression model

```r
## Define classification task
diabetesTask <- makeClassifTask(data = training_data, target = "diabetic")

# Define learner
logReg <- makeLearner("classif.logreg", predict.type = "prob")

# Train the model
logRegModel <- train(logReg, diabetesTask)

# Cross-validate model training process

# Specify resampling strategy
kFold <- makeResampleDesc(method = "RepCV", folds = 6,
                          reps = 50, stratify = TRUE)
# Cross-validate
logRegCV <- resample(learner = logReg, task = diabetesTask,
                             resampling = kFold,
                             measures = list(acc, fpr, fnr),
                     show.info = FALSE)
# View model results
logRegCV$aggr

## acc.test.mean fpr.test.mean fnr.test.mean
##      0.788272      0.118088      0.399008
```

The Logistic Regression model has a training accuracy of 78.83%, which is good based on the simple nature of the model. The model generalizes well. The model however, has a high false negative rate.

Naive Bayes model

```r
# I already defined a classification task, I'll just define a learner for the
model
naiveLearner <- makeLearner("classif.naiveBayes", predict.type = "prob")
# Train the model
bayesModel <- train(naiveLearner, diabetesTask)

# Cross-validate the model building procedure

# Define resampling description
kFold <- makeResampleDesc(method = "RepCV", folds = 6,
                          reps = 50, stratify = TRUE)
# Cross-validate
bayesCV <- resample(learner = naiveLearner,
```

```
                    task = diabetesTask,
                    resampling = kFold,
                    measures = list(mmce, acc, fpr, fnr),
                    show.info = FALSE)
# Check performance
bayesCV$aggr

## mmce.test.mean  acc.test.mean  fpr.test.mean  fnr.test.mean
##      0.2131182      0.7868818      0.1226373      0.3940800
```

The Naive Bayes model has a training accuracy of 78.68%. This model also has a high False Negative rate. The model performs slightly lower than the Logistic Regression model.

KNN model

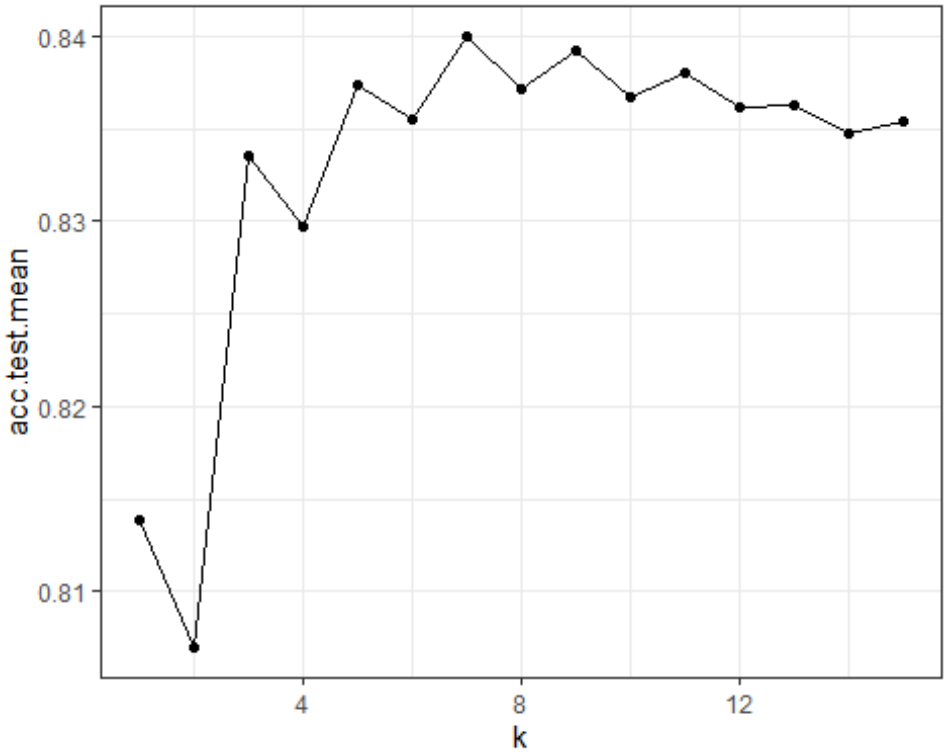
```
# Make learner
knnLearner <- makeLearner("classif.knn")

# Define hyperparameter space for tuning k
knnParamSpace <- makeParamSet(makeDiscreteParam("k", values = 1:15))
# Define tuning grid
gridSearch <- makeTuneControlGrid()
# Define CV for tuning
cvForTuning <- makeResampleDesc("RepCV", folds = 6, reps = 50)

# Tune the model with cross-validation
tunedK <- tuneParams(knnLearner, task = diabetesTask,
                     resampling = cvForTuning,
                     par.set = knnParamSpace,
                     control = gridSearch,
                     measures = list(acc, fpr, fnr),
                     show.info = FALSE)
# Obtain the optimal hyperparameter
tunedK$x

## $k
## [1] 7
```

Optimal value of k = 7.

```
# Check mmce value
tunedK$y

## acc.test.mean fpr.test.mean fnr.test.mean
##    0.83999289    0.09799023    0.28394713
```

The model has a lower mmce value (0.16), implying a good performance (has a training accuracy of 84%). The KNN model performs better than the Logistic Regression and Naive Bayes models.

```r
# Visualize the tuning process
# Obtained model data
knnTuningData <- generateHyperParsEffectData(tunedK)
# Plot
plotHyperParsEffect(knnTuningData, x = "k", y = "acc.test.mean",
plot.type = "line") +
theme_bw()
```



Accuracy is highest at k = 7.

```r
# Set hyperparameters for the final model
tunedKnn <- setHyperPars(makeLearner("classif.knn"),
                         par.vals = tunedK$x)
# Train the final model
tunedKnnModel <- train(tunedKnn, diabetesTask)
```

Random Forest model

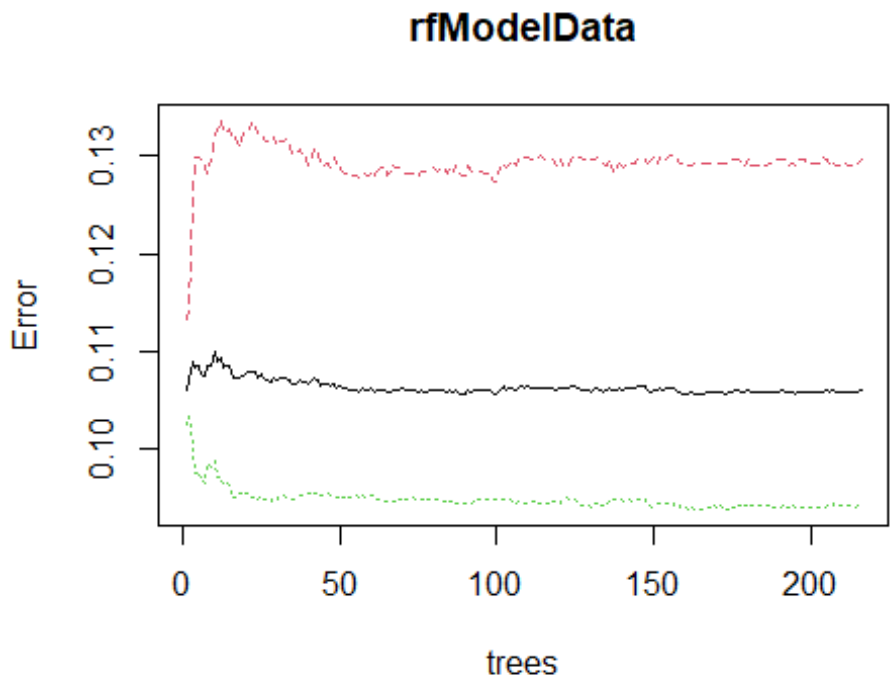
```r
# Define learner
rf <- makeLearner("classif.randomForest", predict.type = "prob")
# I'll continue using the task I defined earlier

# Define hyperparameter space for tuning
rf_ParamSpace <- makeParamSet(makeIntegerParam("ntree", lower = 200, upper =
300),
                              makeIntegerParam("mtry", lower = 4, upper =
10),
```

```
                                  makeIntegerParam("nodesize", lower = 3,
upper = 25),
                                  makeIntegerParam("maxnodes", lower = 5,
upper = 20))

# Define random search method with 200 iterations
randSearch <- makeTuneControlRandom(maxit = 200)
# Define a 6-fold CV strategy
cvForTuning <- makeResampleDesc("CV", iters = 6)

# Start parallelization
parallelStartSocket(cpus = detectCores())

## Starting parallelization in mode=socket with cpus=4.

# Tune the hyperparameters using cross-validation
tuned_rf_Pars <- tuneParams(rf, task = diabetesTask,
                            resampling = cvForTuning,
                            par.set = rf_ParamSpace,
                            control = randSearch,
                            measures = list(acc, fpr, fnr),
                            show.info = FALSE)

## Exporting objects to slaves for mode socket: .mlr.slave.options

## Mapping in parallel: mode = socket; level = mlr.tuneParams; cpus = 4;
elements = 200.

# Stop parallelization
parallelStop()

## Stopped parallelization. All cleaned up.

# View CV results
tuned_rf_Pars

## Tune result:
## Op. pars: ntree=216; mtry=10; nodesize=21; maxnodes=20
## acc.test.mean=0.8934222,fpr.test.mean=0.0944601,fnr.test.mean=0.1309094
```

The RF model has a training accuracy of 89.44%, which is good. The model performs better than the Logistic Regression, Naïve Bayes and KNN models.

```
# Set the optimal hyperparameters for the final model
tuned_rf <- setHyperPars(rf, par.vals = tuned_rf_Pars$x)
# Train the final model with the optimal hyperparameters
tuned_rf_Model <- train(tuned_rf, diabetesTask)

## Warning in randomForest.default(m, y, ...): invalid mtry: reset to within
valid
## range
```

```r
# Check if there are enough trees in the Random Forest model

# First extract model data
rfModelData <- getLearnerModel(tuned_rf_Model)
# Plot
plot(rfModelData)
```



The mean out-of-bag error begins to stabilize early, at about 50 trees. This implies that I have enough number of trees in the forest. The positive class has high mean out-of-bag error rate.

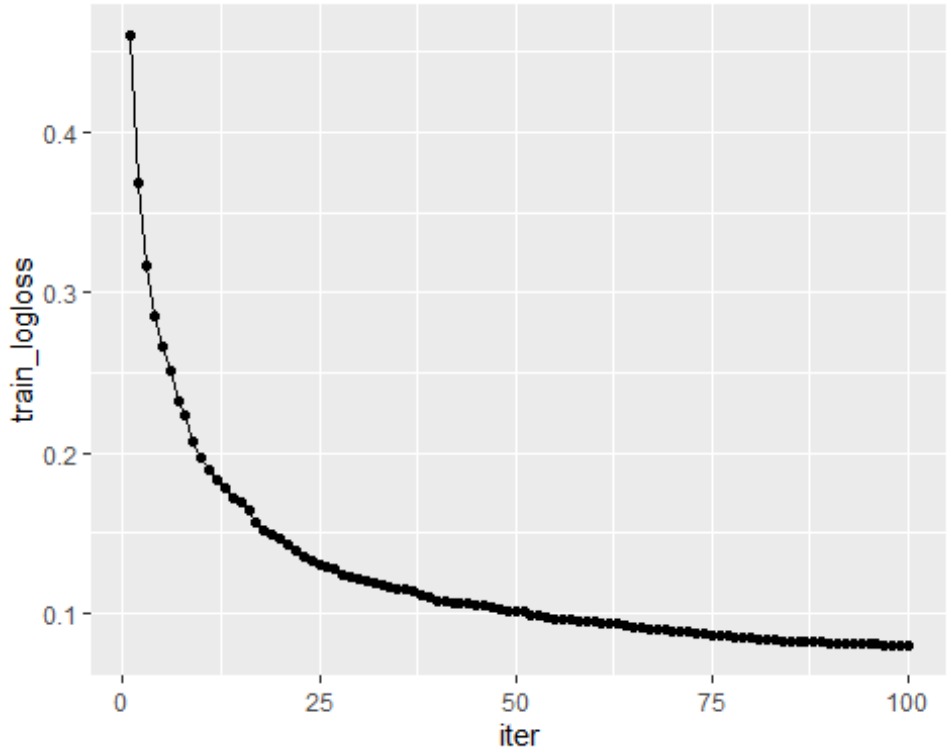
XGBoost

```r
# Define learner
XGB <- makeLearner("classif.xgboost", predict.type = "prob")

# Define hyperparameter space for tuning
xgbParamSpace <- makeParamSet(
makeNumericParam("eta", lower = 0, upper = 1),
makeNumericParam("gamma", lower = 0, upper = 5),
makeIntegerParam("max_depth", lower = 1, upper = 10),
makeNumericParam("min_child_weight", lower = 1, upper = 10),
makeNumericParam("subsample", lower = 0.5, upper = 1),
makeNumericParam("colsample_bytree", lower = 0.5, upper = 1),
makeIntegerParam("nrounds", lower = 100, upper = 100))
```

```
# Define hyperparameter search strategy
randSearch <- makeTuneControlRandom(maxit = 700)
# Make resampling description for CV
cvForTuning <- makeResampleDesc("CV", iters = 6)

# Tune the model
tunedXgbPars <- tuneParams(learner = XGB, task = diabetesTask,
                           resampling = cvForTuning,
                           par.set = xgbParamSpace,
                           control = randSearch,
                           measures = list(acc, fpr, fnr),
                           show.info = FALSE)
# Check performance
tunedXgbPars$y

## acc.test.mean fpr.test.mean fnr.test.mean
##    0.96071111    0.02916851    0.05957756
```

A training accuracy 96.07% is very good, though the model might be overfitting the training data. XGBoost outperforms all the previous models.

```
# Train the final model using optimal hyperparameters

# Set the optimal hyperparameters for the final model
tunedXgb <- setHyperPars(XGB, par.vals = tunedXgbPars$x)
# Train the final model
tunedXgbModel <- train(tunedXgb, diabetesTask)

# Check if there are enough trees in the model

# Extract model data
xgbModelData <- getLearnerModel(tunedXgbModel)
# Plot
ggplot(xgbModelData$evaluation_log, aes(iter, train_logloss)) +
  geom_line() + geom_point()
```



Training log loss stabilizes after 75 iterations and decreases steadily. I used enough trees.

Neural Network

```
# Define learner for the neural network
nnet <- makeLearner("classif.nnet", predict.type = "prob")

# Define hyperparameter space for tuning
nnetParam_set <- makeParamSet(
  makeIntegerParam("size", lower = 3, upper = 20), # Number of units in
hidden layer
  makeNumericParam("decay", lower = 0.001, upper = 0.8)) # Weight decay

# Define search strategy to use random search
randSearch <- makeTuneControlRandom(maxit = 150)

# Make resampling description for CV
cvForTuning <- makeResampleDesc("CV", iters = 6, stratify = TRUE)

# Set random seed for reproducibility
set.seed(1234)

# Start parallelization
parallelStartSocket(cpus = detectCores())
```

```
## Starting parallelization in mode=socket with cpus=4.

# Tune the model with cross-validation
tunedNnetPars <- tuneParams(learner = nnet, task = diabetesTask,
                            resampling = cvForTuning,
                            par.set = nnetParam_set,
                            control = randSearch,
                            measures = list(mmce, acc, fpr, fnr),
                            show.info = FALSE)

## Exporting objects to slaves for mode socket: .mlr.slave.options

## Mapping in parallel: mode = socket; level = mlr.tuneParams; cpus = 4;
elements = 150.

# Stop parallelization
parallelStop()

## Stopped parallelization. All cleaned up.

# View CV results
print(tunedNnetPars$x)

## $size
## [1] 6
##
## $decay
## [1] 0.3855826

tunedNnetPars$y

## mmce.test.mean  acc.test.mean  fpr.test.mean  fnr.test.mean
##      0.1432889      0.8567111      0.1114667      0.2069333
```

The Neural Net performs better than Logistic Regression, Naive Bayes and KNN classifiers, but is outperformed by Random Forest and XGBoost. The Neural Net has a training accuracy of 85.67%. The optimal hyperparameters are size of 6 neurons and decay rate of 0.385.

```
# Train the final model using optimal hyperparameters

# Set the best parameters for the final model
tunedNnet <- setHyperPars(nnet, par.vals = tunedNnetPars$x)

# Train the final model
tunedNnetModel <- train(tunedNnet, diabetesTask)

## # weights:  61
## initial  value 16193.872951
## iter  10 value 7173.257497
## iter  20 value 6958.835329
## iter  30 value 6667.220271
```

```
## iter  40 value 6130.795307
## iter  50 value 5872.353183
## iter  60 value 5131.380692
## iter  70 value 4675.695522
## iter  80 value 4441.830627
## iter  90 value 4352.662468
## iter 100 value 4257.294982
## final  value 4257.294982
## stopped after 100 iterations
```

Benchmark the model-building processes

```
# First create a list of learners
learners = list(logReg, naiveLearner, tunedKnn, tuned_rf, tunedXgb,
tunedNnet)

# Make a resampling description for benchmarking
benchCV <- makeResampleDesc("RepCV", folds = 5, reps = 20)

# Benchmark
bench <- benchmark(learners, diabetesTask, benchCV,
                   show.info = FALSE, measures = list(acc, kappa))

# View the benchmarking results
bench
```

```
##         task.id           learner.id acc.test.mean kappa.test.mean
## 1 training_data       classif.logreg     0.7883600       0.5036092
## 2 training_data   classif.naiveBayes     0.7865556       0.5011454
## 3 training_data          classif.knn     0.8393778       0.6303253
## 4 training_data classif.randomForest     0.8928000       0.7622245
## 5 training_data      classif.xgboost     0.9602089       0.9103687
## 6 training_data         classif.nnet     0.8268356       0.6141528
```

According to this benchmarking results, XGBoost is likely to give me the best-performing model, with an accuracy of more than 90%. Random Forest also performs well.

Model Evaluation

I'll use the two performing models to make predictions on test data (RF & XGB) and evaluate how my models would perform on unseen data.

```
# Use the RF model to make predictions on test data
rf_Preds <- predict(tuned_rf_Model, newdata = test_data)

# Collect prediction
rf_Preds_data <- rf_Preds$data
```

```r
# Calculate confusion matrix
confusionMatrix(table(rf_Preds_data$truth, rf_Preds_data$response))
```

```
## Confusion Matrix and Statistics
##
##
##        Yes   No
##   Yes 1079  171
##   No   244 2256
##
##                Accuracy : 0.8893
##                  95% CI : (0.8789, 0.8992)
##     No Information Rate : 0.6472
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7546
##
##  Mcnemar's Test P-Value : 0.0004088
##
##             Sensitivity : 0.8156
##             Specificity : 0.9295
##          Pos Pred Value : 0.8632
##          Neg Pred Value : 0.9024
##              Prevalence : 0.3528
##          Detection Rate : 0.2877
##    Detection Prevalence : 0.3333
##       Balanced Accuracy : 0.8726
##
##        'Positive' Class : Yes
##
```

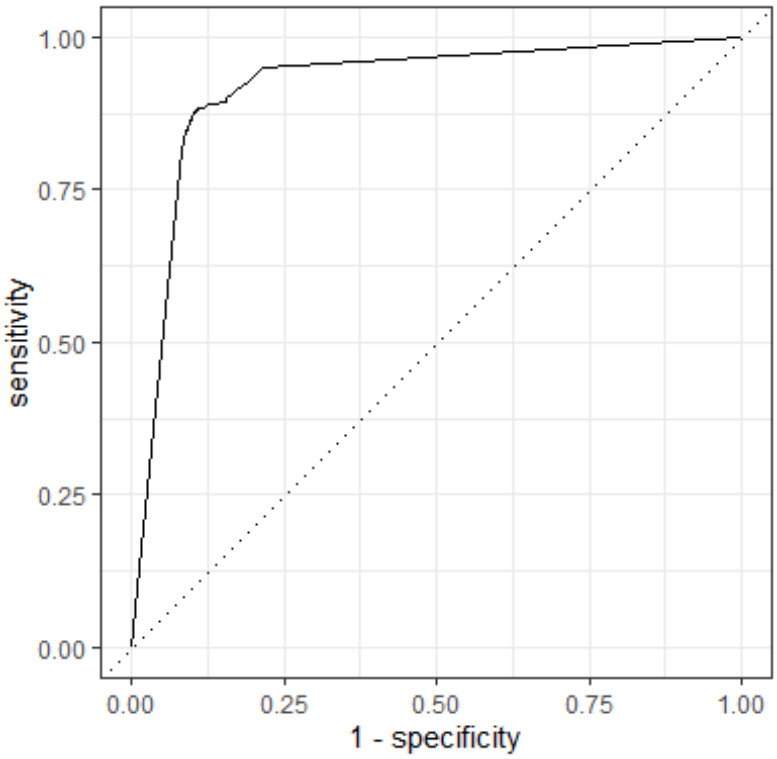
The Random Forest model has a validation accuracy of 88.93%, with a precision of 86.32% which are all good. The model has a higher Specificity (0.9295) than Sensitivity (0.8156). The training accuracy is slightly higher than the validation accuracy, implying that the model didn't overfit the training data.

```r
# Calculate ROC AUC value
rf_Preds_data |> roc_auc(truth = truth, prob.Yes)
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.919
```

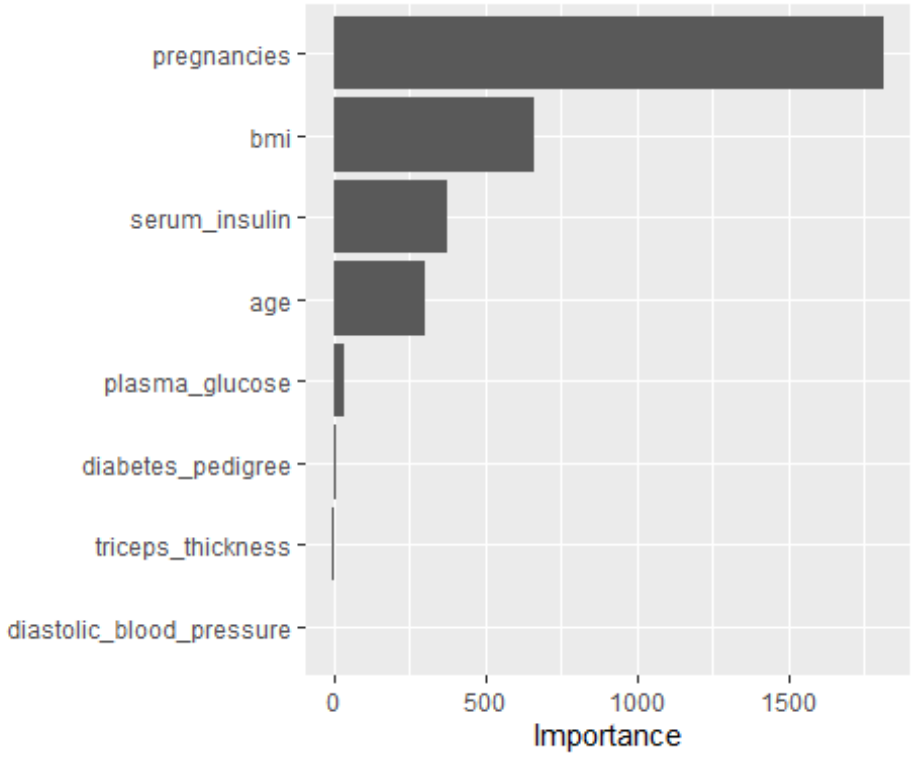
A ROC AUC value of 0.919 for the Random Forest model is very good, implying that the model fits the data very well.

```r
# Plot ROC curve
rf_Preds_data |> roc_curve(truth = truth, prob.Yes) |> autoplot()
```



ROC curve looks good, very steady and is closely approaching the top left corner where AUC value is 1.

```
# Plot variable importance for the Random Forest model
vip(tuned_rf_Model)
```



The Random Forest model finds number of pregnancies as the most important predictor of diabetes, followed by BMI, serum insulin, age, plasma glucose, diabetes pedigree and triceps thickness respectively.

```
# Use the XGBoost model to make predictions on test data
xgbPreds <- predict(tunedXgbModel, newdata = test_data)

# Collect prediction
xgbPreds_data <- xgbPreds$data

# Calculate confusion matrix
confusionMatrix(table(xgbPreds_data$truth, xgbPreds_data$response))
```

```
## Confusion Matrix and Statistics
##
##
##        Yes   No
##   Yes 1165   85
##   No    76 2424
##
##                Accuracy : 0.9571
##                  95% CI : (0.9501, 0.9633)
##     No Information Rate : 0.6691
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9032
##
```

```
##  Mcnemar's Test P-Value : 0.5284
##
##             Sensitivity : 0.9388
##             Specificity : 0.9661
##          Pos Pred Value : 0.9320
##          Neg Pred Value : 0.9696
##              Prevalence : 0.3309
##          Detection Rate : 0.3107
##    Detection Prevalence : 0.3333
##       Balanced Accuracy : 0.9524
##
##        'Positive' Class : Yes
##
```

Wow! 95.71% validation accuracy. The XGBoost model has an excellent performance.

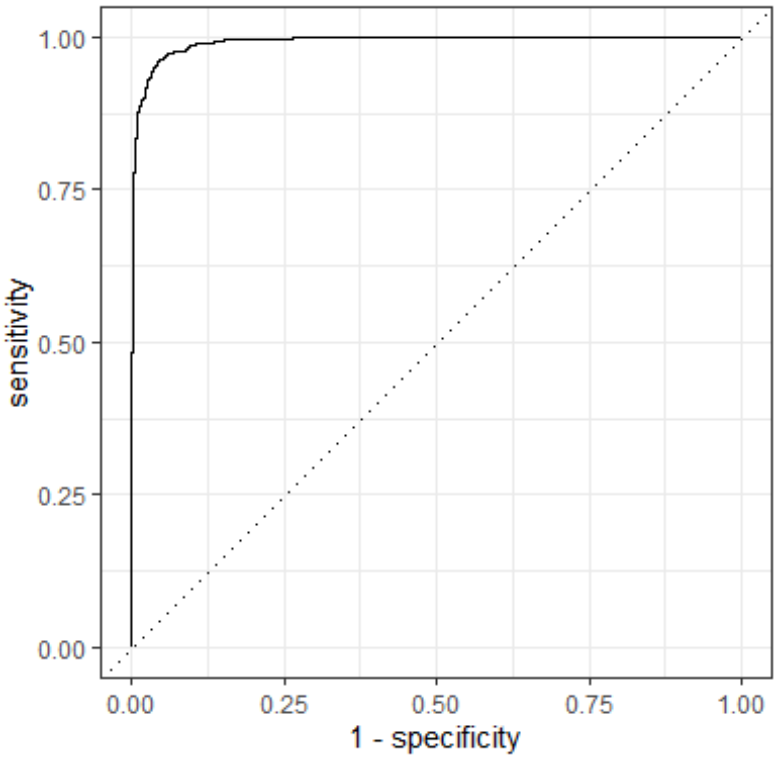
- Sensitivity and Specificity for the XGBoost model are all good, with Specificity being high. The trade-off between Sensitivity and Specificity is small. The Sensitivity and Precision for this model are also good (93.88% and 93.20% respectively).

```
# Calculate ROC AUC value
xgbPreds_data |> roc_auc(truth = truth, prob.Yes)
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.992
```

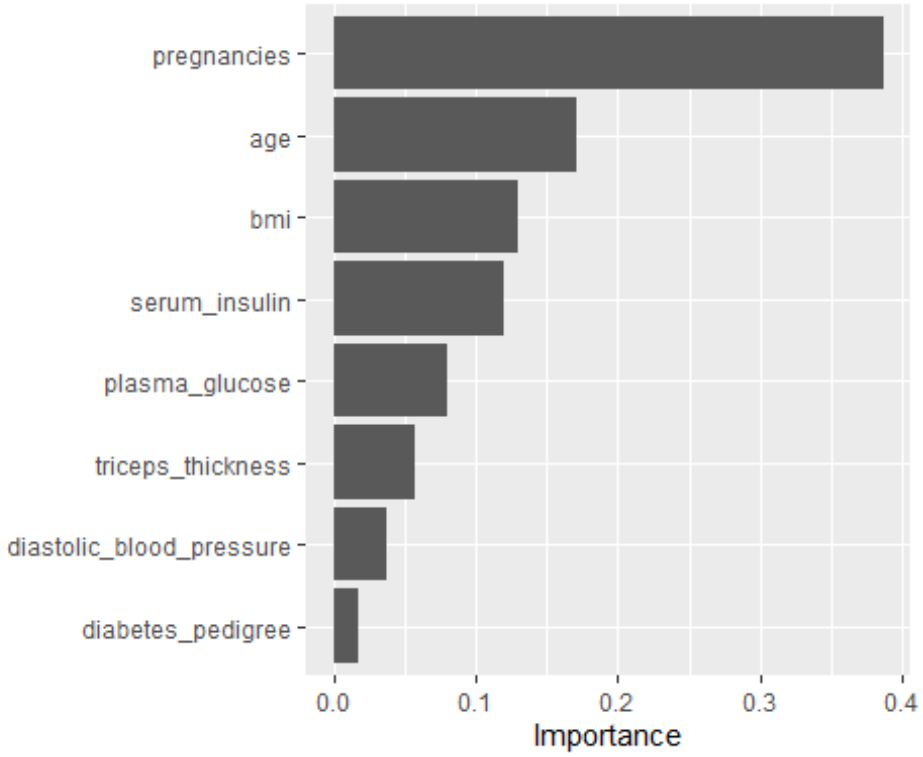
The XGBoost model has a ROC AUC value of 0.99, which is excellent. The model fits the data very well.

```
# Plot ROC curve
xgbPreds_data |> roc_curve(truth = truth, prob.Yes) |> autoplot()
```



The curve is almost touching the top left corner, near 1.

```
# Plot variable importance for the XGBoost model
vip(tunedXgbModel, type = "gain")
```



Based on information gain ratio score, number of pregnancies is the most important predictor of diabetes, followed by age, BMI, serum insulin, plasma glucose, triceps thickness, diastolic blood pressure and diabetes pedigree respectively.

- However, the main limitation of this analysis is that I did not handle class imbalance in the data.

References

Rhys, H. I. (2020). Machine learning with R, the tidyverse, and mlr. Manning Publications. https://livebook.manning.com/book/machine-learning-with-r-the-tidyverse-and-mlr/about-this-book

Tasin, I., Nabil, T. U., Islam, S., & Khan, R. (2022). Diabetes prediction using machine learning and explainable AI techniques. Healthcare Technology Letters, 10(1-2), 1-10. https://pmc.ncbi.nlm.nih.gov/articles/PMC10107388/#htl212039-sec-0010