Capstone project - Diabetes

Eli F Mayost

November 16, 2019

Introduction

More and more people worlwide are affected by Diabetes.

The number of affected people has been rising almost four folds from 1980 to 2014 [1].

Diabetes is a major cause of blindness, kidney failure, strokes, heart attacks, and lower limb amputation.

46% of people with diabetes are undiagnosed [2].

The goal of this project is to develop a machine learning model to try and predict diabetes in women.

[1] World Health Organization.
[2] diabetes.co.uk.

The data

A population of women who were at least 21 years old, of Pima Indian heritage and living near Phoenix, Arizona, were tested for diabetes according to World Health Organization criteria.

The data is in diabetes.csv:

```
data <- readr::read_csv("diabetes.csv")
```

Checking that we do not have missing values:

```
data %>% is.na() %>% any()
```

```
## [1] FALSE
```

The dataset consists of 768 observations, with 9 features [3] for each observation:

```
data %>% dim()
```

```
## [1] 768   9
```

The first 10 observations:

```
data %>% head(10) %>% knitr::kable()
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31.0	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0.0	0.232	54	1

We can already see some illogical values (BMI, Glucose, SkinThickness, and BloodPressure at 0...).

As we do not have any NA, it appears that a zero was filled in for missing values (the 8th observation has a Blood Pressure of 0).

[3] The 9 features are: Number of pregnancies, plasma glucose concentration, diastolic blood pressure (mm Hg), triceps skin fold thickness (in mm), 2 hour serum insulin measure, body mass index, diabetes pedigree function, age, and outcome (0 for negative, and 1 for positive diabetes).

Let's find out how many in each of the above columns:

```
columns <- data %>%
  dplyr::select(-Outcome, -Pregnancies, -Insulin, -DiabetesPedigreeFunction) %>%
  colnames()

for(c in columns){
  glue::glue("{as.character(c)} has {sum(data[as.character(c)] == 0)} zeroes") %>%
  print()
}
```

```
## Glucose has 5 zeroes
## BloodPressure has 35 zeroes
## SkinThickness has 227 zeroes
## BMI has 11 zeroes
## Age has 0 zeroes
```

236 rows are affected:

```
data %>% dplyr::filter_at(vars(BMI, SkinThickness, BloodPressure, Glucose), any_vars(. == 0)) %>%
  nrow()
```

```
## [1] 236
```

So the zero values are in the 4 columns previously mentioned. We could eliminate all those 236 observations, but we would be loosing an important portion of observations for training our model later, as well as important information contained in those rows.

I will replace the zero values with the mean of the other observations having the same outcome.

```
bmi_mean <- data %>%
  dplyr::filter(! BMI == 0) %>%
  dplyr::group_by(Outcome) %>%
  dplyr::summarize(avg = mean(BMI)) %>%
  .$avg

bp_mean <- data %>%
  dplyr::filter(! BloodPressure == 0) %>%
  dplyr::group_by(Outcome) %>%
  dplyr::summarize(avg = mean(BloodPressure)) %>%
  .$avg

st_mean <- data %>%
  dplyr::filter(! SkinThickness == 0) %>%
  dplyr::group_by(Outcome) %>%
  dplyr::summarize(avg = mean(SkinThickness)) %>%
  .$avg

gl_mean <- data %>%
  dplyr::filter(! Glucose == 0) %>%
  dplyr::group_by(Outcome) %>%
  dplyr::summarize(avg = mean(Glucose)) %>%
  .$avg

data <- data %>%
```

```
  dplyr::mutate(BMI = ifelse(BMI == 0 & Outcome == 0, bmi_mean[1],
    ifelse(BMI == 0 & Outcome == 1, bmi_mean[2], BMI))) %>%

  dplyr::mutate(BloodPressure = ifelse(BloodPressure == 0 & Outcome == 0, bp_mean[1],
    ifelse(BloodPressure == 0 & Outcome == 1, bp_mean[2], BloodPressure))) %>%

  dplyr::mutate(SkinThickness = ifelse(SkinThickness == 0 & Outcome == 0, st_mean[1],
    ifelse(SkinThickness == 0 & Outcome == 1, st_mean[2], SkinThickness))) %>%

  dplyr::mutate(Glucose = ifelse(Glucose == 0 & Outcome == 0, gl_mean[1],
    ifelse(Glucose == 0 & Outcome == 1, gl_mean[2], Glucose)))

data %>% head(10) %>% round(digits = 2) %>% knitr::kable()
```

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72.00	35.00	0	33.60	0.63	50	1
1	85	66.00	29.00	0	26.60	0.35	31	0
8	183	64.00	33.00	0	23.30	0.67	32	1
1	89	66.00	23.00	94	28.10	0.17	21	0
0	137	40.00	35.00	168	43.10	2.29	33	1
5	116	74.00	27.24	0	25.60	0.20	30	0
3	78	50.00	32.00	88	31.00	0.25	26	1
10	115	70.88	27.24	0	35.30	0.13	29	0
2	197	70.00	45.00	543	30.50	0.16	53	1
8	125	96.00	33.00	0	35.41	0.23	54	1

Now, we should not have any zero value, and we keep the same observations' total:

```
data %>%
  dplyr::filter_at(vars(BMI, SkinThickness, Age, BloodPressure), any_vars(. == 0)) %>%
  nrow()
```

```
## [1] 0
```

Exploring the data

A quick summary of the features:

```
data %>%
  dplyr::select(Pregnancies, Glucose, BloodPressure, SkinThickness) %>%
  summary() %>%
  knitr::kable()
```

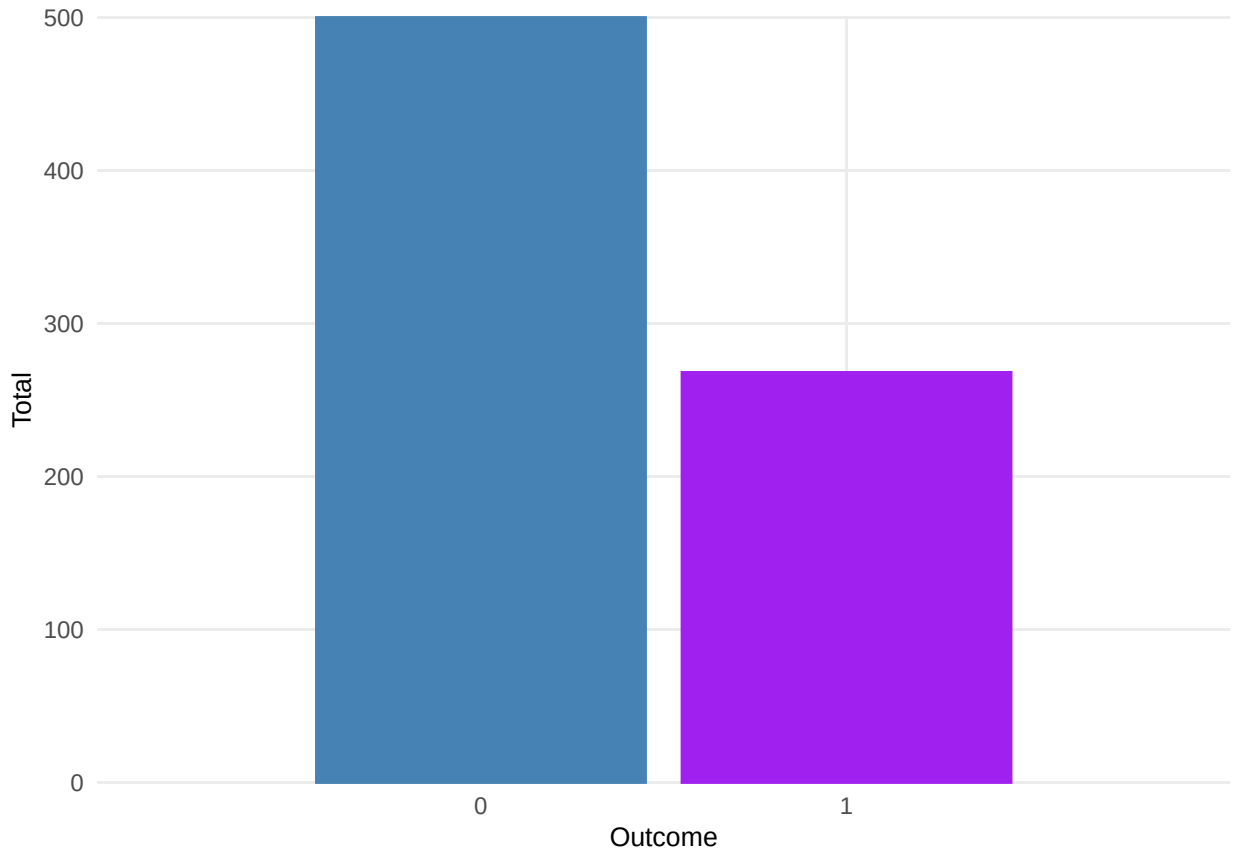
Pregnancies	Glucose	BloodPressure	SkinThickness
Min. : 0.000	Min. : 44.00	Min. : 24.00	Min. : 7.00
1st Qu.: 1.000	1st Qu.: 99.75	1st Qu.: 64.00	1st Qu.:25.00
Median : 3.000	Median :117.00	Median : 72.00	Median :28.00
Mean : 3.845	Mean :121.70	Mean : 72.43	Mean :29.25
3rd Qu.: 6.000	3rd Qu.:141.00	3rd Qu.: 80.00	3rd Qu.:33.00
Max. :17.000	Max. :199.00	Max. :122.00	Max. :99.00

```
data %>%
  dplyr::select(Insulin, BMI, Age, DiabetesPedigreeFunction) %>%
  summary() %>%
  knitr::kable()
```

Insulin	BMI	Age	DiabetesPedigreeFunction
Min. : 0.0	Min. :18.20	Min. :21.00	Min. :0.0780
1st Qu.: 0.0	1st Qu.:27.50	1st Qu.:24.00	1st Qu.:0.2437
Median : 30.5	Median :32.05	Median :29.00	Median :0.3725
Mean : 79.8	Mean :32.45	Mean :33.24	Mean :0.4719
3rd Qu.:127.2	3rd Qu.:36.60	3rd Qu.:41.00	3rd Qu.:0.6262
Max. :846.0	Max. :67.10	Max. :81.00	Max. :2.4200

For the outcome, total of negative and positive observations:

```
data %>%
  ggplot(aes(Outcome)) +
  geom_bar(aes(color = factor(Outcome), fill = factor(Outcome)), show.legend = F) +
  scale_x_discrete(limits = c(0, 1)) +
  scale_y_discrete(name = "Total", limits = seq(0, sum(data$Outcome == 0), 100)) +
  scale_color_manual(values = c("steelblue", "purple")) +
  scale_fill_manual(values = c("steelblue", "purple")) +
  theme_minimal() +
  theme(axis.title.x = element_text(size = 10),
        axis.title.y = element_text(size = 10))
```



Exploring the correlation between features and outcome:

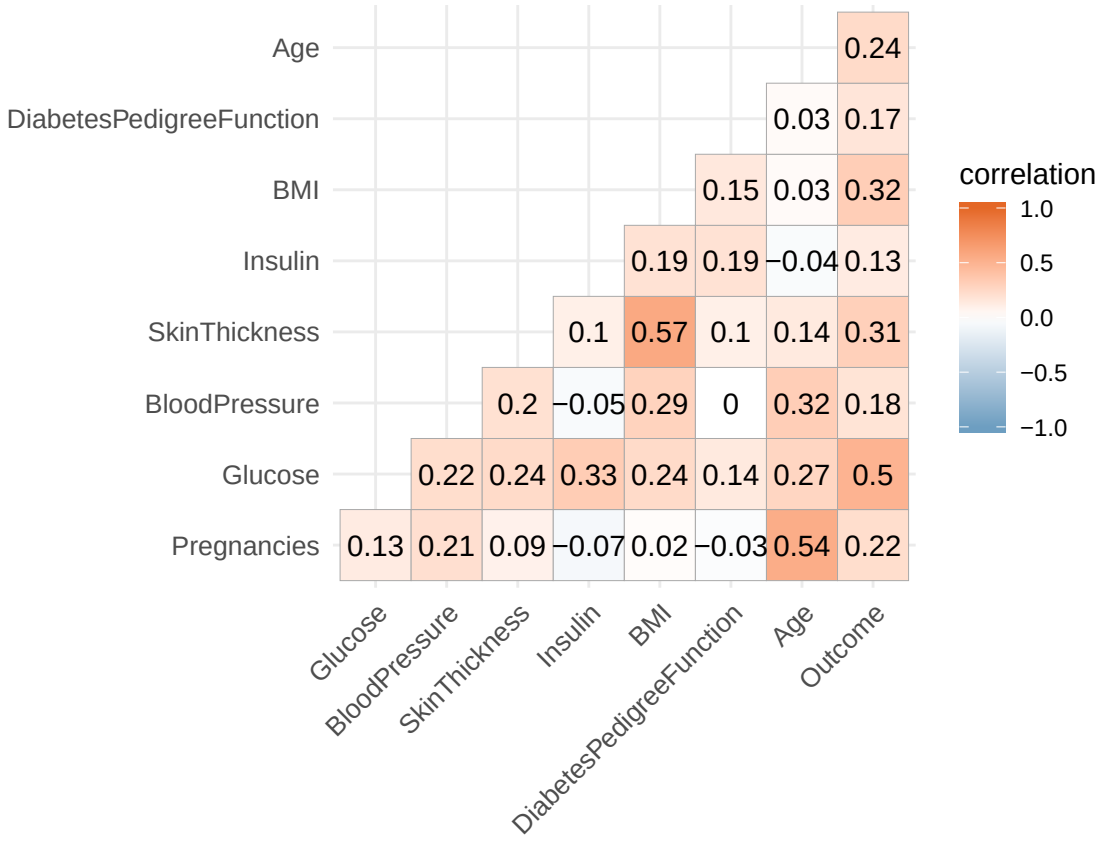
```
data %>%
  dplyr::mutate(Outcome = as.integer(Outcome)) %>%
  cor %>%
  round(digits = 2) %>%
  knitr::kable()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
Pregnancies	1.00	0.13	0.21	0.09	-0.07	0.02	
Glucose	0.13	1.00	0.22	0.24	0.33	0.24	
BloodPressure	0.21	0.22	1.00	0.20	-0.05	0.29	
SkinThickness	0.09	0.24	0.20	1.00	0.10	0.57	
Insulin	-0.07	0.33	-0.05	0.10	1.00	0.19	
BMI	0.02	0.24	0.29	0.57	0.19	1.00	
DiabetesPedigreeFunction	-0.03	0.14	0.00	0.10	0.19	0.15	
Age	0.54	0.27	0.32	0.14	-0.04	0.03	
Outcome	0.22	0.50	0.18	0.31	0.13	0.32	

There is a positive correlation between all the features and the Outcome (Glucose being the strongest one).

We can visualize the correlations like so:

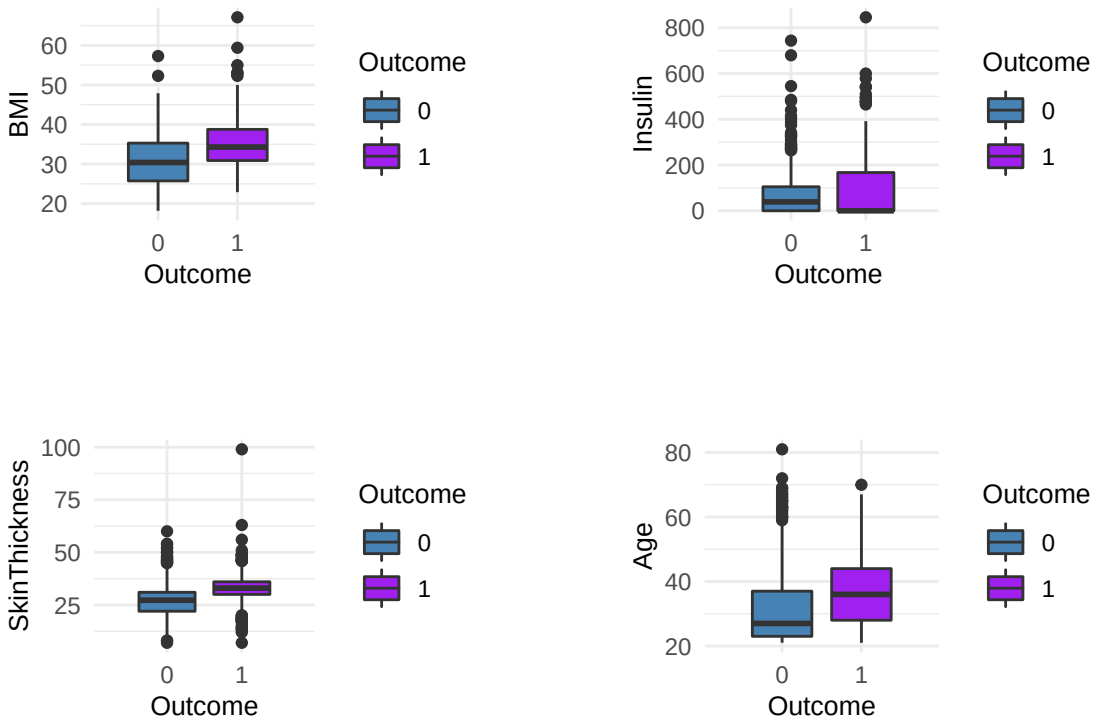
```
data %>%
  cor %>%
  ggcorrplot(type = "lower",
             lab = T,
             colors = c("#6D9EC1", "white", "#E46726"),
             outline.color = "darkgrey",
             tl.cex = 10,
             legend.title = "correlation")
```



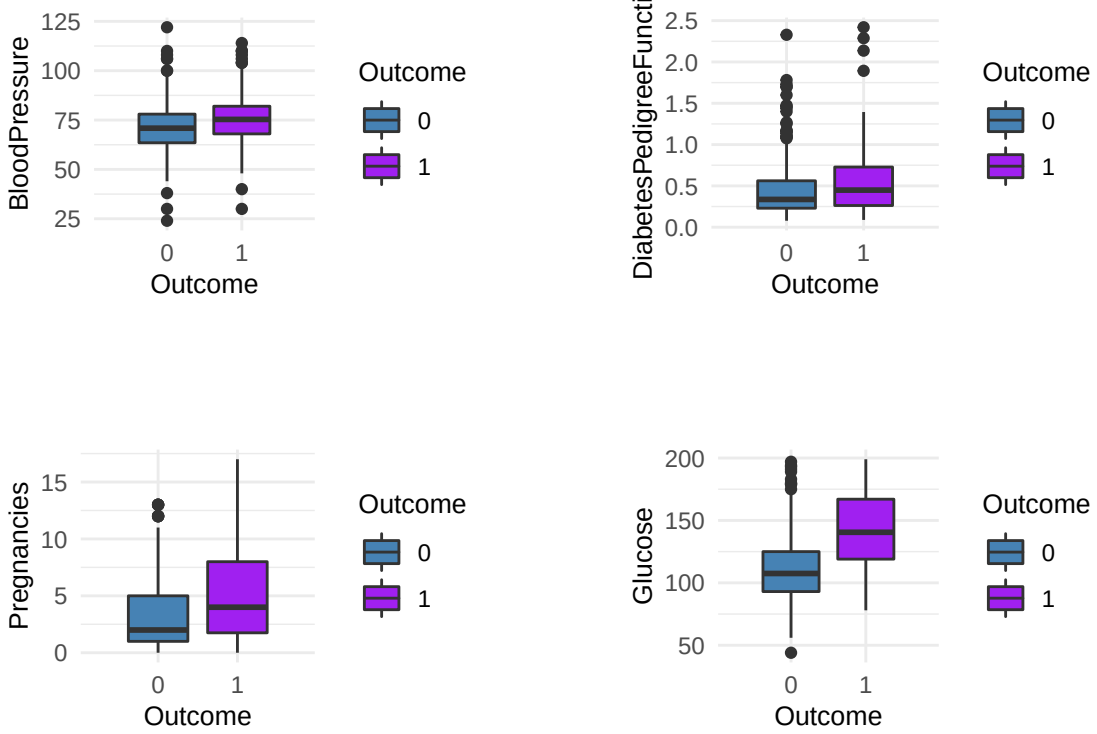
We can see if the data is grouped, the outliers, and the reliationship betwen a categorical feature (outcome), and a continuous one (Glucose, BMI etc...) with boxplots:

```
make_plot <- function(feature){
  data %>%
  ggplot(aes(x=Outcome, y=!!rlang::sym(feature))) +

    geom_boxplot(aes(fill=factor(Outcome))) +
    scale_x_discrete(limits = c(0, 1)) +
    scale_color_manual(values = c("steelblue", "purple")) +
    scale_fill_manual("Outcome", values = c("steelblue", "purple")) +
    theme_minimal() +
    theme(axis.title.x = element_text(size = 10),
          axis.title.y = element_text(size = 10),
          legend.title = element_text(size = 10),
          plot.margin = unit(c(1,1,1,1), "cm"))
```

```
}

p1 <- make_plot("BMI")
p2 <- make_plot("Insulin")
p3 <- make_plot("SkinThickness")
p4 <- make_plot("Age")
p5 <- make_plot("BloodPressure")
p6 <- make_plot("DiabetesPedigreeFunction")
p7 <- make_plot("Pregnancies")
p8 <- make_plot("Glucose")

grid.arrange(
  grobs = list(p1, p2, p3, p4),
  layout_matrix = rbind(c(1, 2),
                        c(3, 4))
)
```




```
grid.arrange(
  grobs = list(p5, p6, p7, p8),
  layout_matrix = rbind(c(1, 2),
                        c(3, 4))
)
```



For the models I will use all the features, but a possible future improvement could be to keep only BMI, SkinThickness, and Glucose. The features where there is a clear demarcation between the two outcomes, and that contain relatively few outliers.

Preparing train and test sets

We will partition the data in two; 70% for the train set, and 30% for the test set:

```
ind <- caret::createDataPartition(data$Outcome, times = 1, p = 0.3, list = F)

train_set <- data %>% dplyr::slice(-ind)
test_set <- data %>% dplyr::slice(ind)
```

Checking how many observations in each set:

```
glue::glue("The train set has {nrow(train_set)} observations.")
```

```
## The train set has 537 observations.
```

```
glue::glue("The test set has {nrow(test_set)} observations.")
```

```
## The test set has 231 observations.
```

Modelling

From the caret package we will make sure that we are choosing the methods that are fit for classification.

We will see the performance of a few models, and an ensemble of three, in order to choose the best one at the end.

I will try and tune them as we go.

Guessing

The simplest to try, is guessing:

```
set.seed(1976)

y_hat_guess <- sample(c(0, 1), size = nrow(test_set), replace = T)
conf_matrix_guess <- confusionMatrix(factor(y_hat_guess), factor(test_set$Outcome))
conf_matrix_guess
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 66 53
##          1 78 34
##
##                Accuracy : 0.4329
##                  95% CI : (0.3681, 0.4995)
##     No Information Rate : 0.6234
##     P-Value [Acc > NIR] : 1.000
##
##                   Kappa : -0.1427
##
##  Mcnemar's Test P-Value : 0.036
##
##             Sensitivity : 0.4583
##             Specificity : 0.3908
##          Pos Pred Value : 0.5546
##          Neg Pred Value : 0.3036
##              Prevalence : 0.6234
##          Detection Rate : 0.2857
##    Detection Prevalence : 0.5152
##       Balanced Accuracy : 0.4246
##
##        'Positive' Class : 0
##
```

Not great, as we are unable to predict the outcome in roughly one case out of two (43.29%). That is to be expected, and similar to a coin toss.

Logistic regression

Next we will try logistic regression:

```
set.seed(1976)
fit_glm <- train(factor(Outcome) ~ ., data = train_set, method = "glm")
y_hat_glm <- predict(fit_glm, newdata = test_set)
conf_matrix_glm <- confusionMatrix(factor(y_hat_glm), factor(test_set$Outcome))
conf_matrix_glm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 120  42
##          1  24  45
##
##                Accuracy : 0.7143
##                  95% CI : (0.6514, 0.7716)
##     No Information Rate : 0.6234
##     P-Value [Acc > NIR] : 0.002322
##
##                   Kappa : 0.3655
##
##  Mcnemar's Test P-Value : 0.036389
##
##             Sensitivity : 0.8333
##             Specificity : 0.5172
##          Pos Pred Value : 0.7407
##          Neg Pred Value : 0.6522
##              Prevalence : 0.6234
##          Detection Rate : 0.5195
##    Detection Prevalence : 0.7013
##       Balanced Accuracy : 0.6753
##
##        'Positive' Class : 0
##
```

Already a sizeable improvement over somply guessing.

Recursive Partitioning

Here, the available parameter for tuning is cp:

```
getModelInfo()$rpart$parameters
```

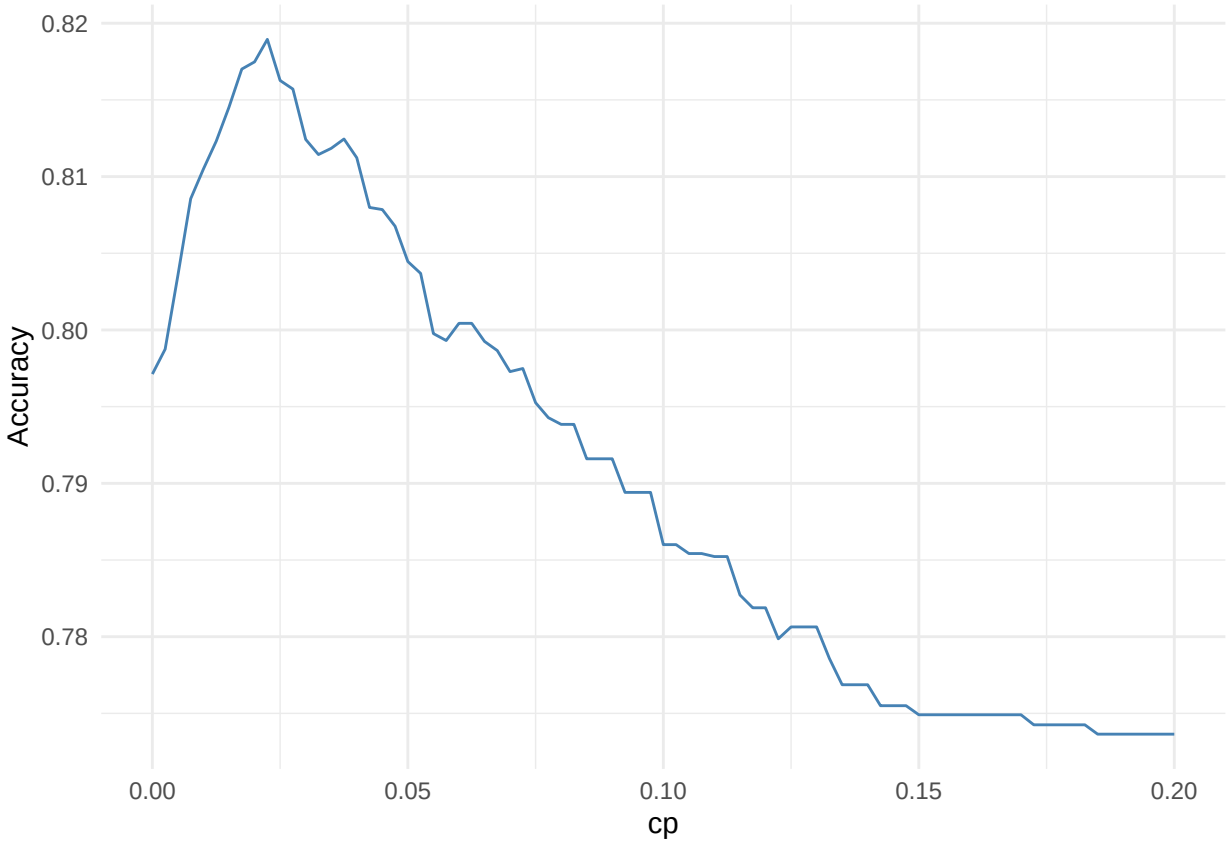
```
##   parameter   class                label
## 1        cp numeric Complexity Parameter
```

```
set.seed(1976)

tune_grid_rpart <- expand.grid(cp = seq(0, 0.2, 0.0025))

fit_rpart <- train(factor(Outcome) ~ ., data = train_set, method = "rpart", tuneGrid = tune_grid_rpart)
y_hat_rpart <- predict(fit_rpart, type = "raw", newdata = test_set)
conf_matrix_rpart <- confusionMatrix(factor(y_hat_rpart), factor(test_set$Outcome))
conf_matrix_rpart
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 120  25
##          1  24  62
##
##                Accuracy : 0.7879
##                  95% CI : (0.7295, 0.8388)
##     No Information Rate : 0.6234
##     P-Value [Acc > NIR] : 5.759e-08
##
##                   Kappa : 0.5472
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8333
##             Specificity : 0.7126
##          Pos Pred Value : 0.8276
##          Neg Pred Value : 0.7209
##              Prevalence : 0.6234
##          Detection Rate : 0.5195
##    Detection Prevalence : 0.6277
##       Balanced Accuracy : 0.7730
##
##        'Positive' Class : 0
##
```

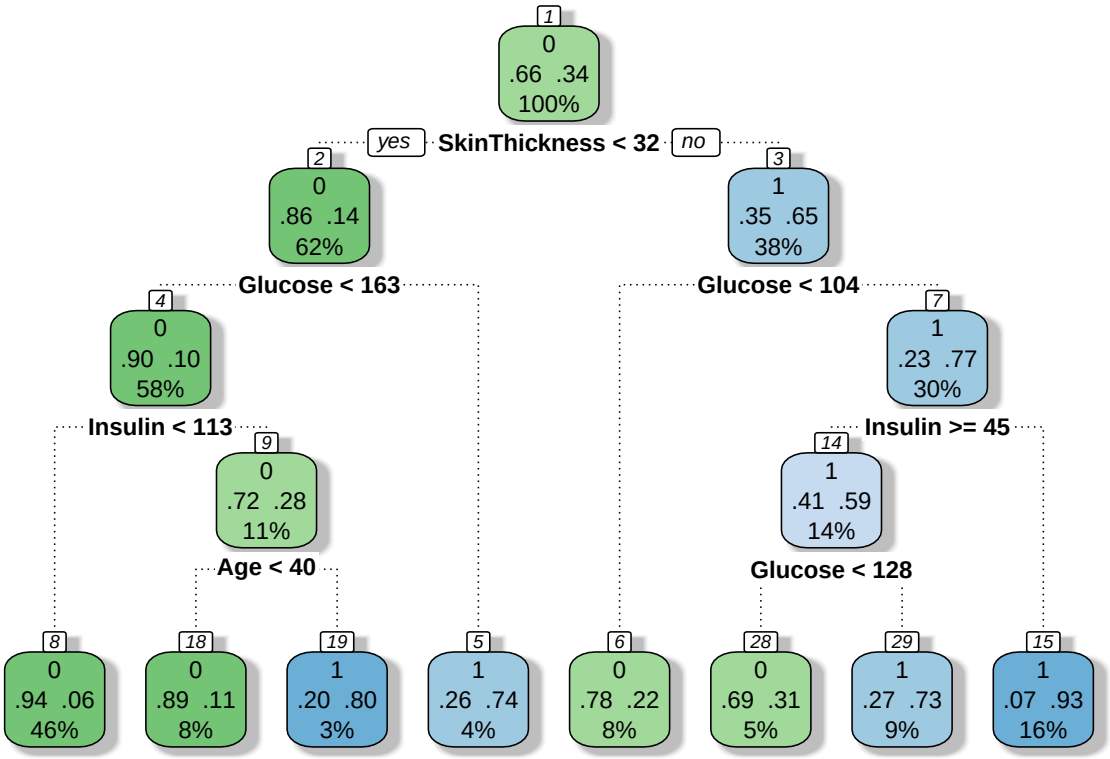
The best cp value is 0.0225:

```
fit_rpart$results %>%
  as_tibble() %>%
  ggplot(aes(x = cp, y = Accuracy)) +
    geom_line(color = "steelblue") +
    theme_minimal()
```



We can visualize the final model's decision tree like so:

```
rattle::fancyRpartPlot(fit_rpart$finalModel, caption = "")
```


1
0
.66 .34
100%
yes
SkinThickness < 32
no
2
0
.86 .14
62%
3
1
.35 .65
38%
Glucose < 163
Glucose < 104
4
0
.90 .10
58%
7
1
.23 .77
30%
Insulin < 113
Insulin >= 45
9
0
.72 .28
11%
14
1
.41 .59
14%
Age < 40
Glucose < 128
8
0
.94 .06
46%
18
0
.89 .11
8%
19
1
.20 .80
3%
5
1
.26 .74
4%
6
0
.78 .22
8%
28
0
.69 .31
5%
29
1
.27 .73
9%
15
1
.07 .93
16%

Random Forest

Here, also only one parameter, mtry, to tune.

Maximum value for mtry is the number of predictors, so 8 in our case. We tune using tuneGrid, and see which one gives the best accuracy.

```
set.seed(1976)
fit_rf <- train(factor(Outcome) ~ .,
             data = train_set,
             method = "rf",
             tuneGrid = data.frame(mtry = seq(train_set %>% colnames() %>% length() - 1)))
y_hat_rf <- predict(fit_rf, newdata = test_set)
conf_matrix_rf <- confusionMatrix(factor(y_hat_rf), factor(test_set$Outcome))
conf_matrix_rf
```

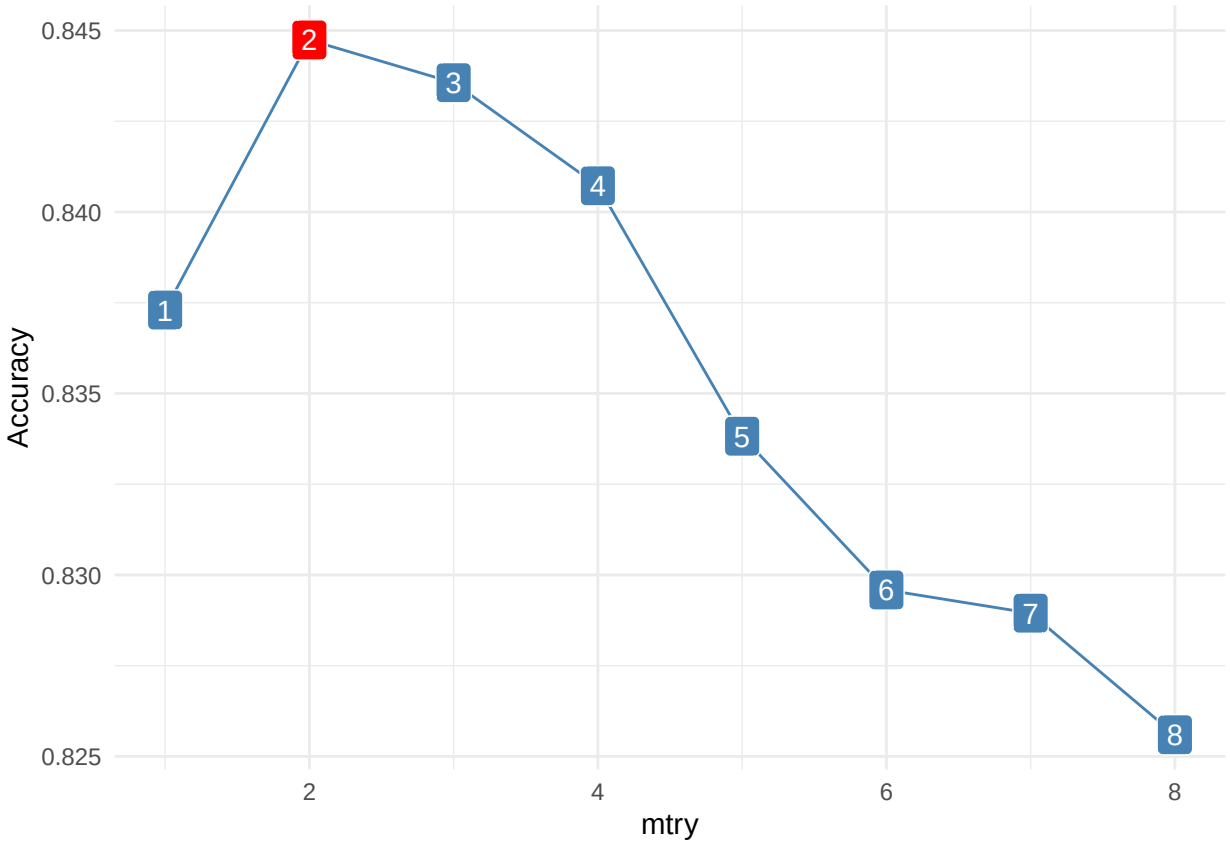
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 117  26
##          1  27  61
##
##                Accuracy : 0.7706
##                  95% CI : (0.7109, 0.8232)
##     No Information Rate : 0.6234
##     P-Value [Acc > NIR] : 1.259e-06
##
##                   Kappa : 0.5125
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8125
##             Specificity : 0.7011
##          Pos Pred Value : 0.8182
##          Neg Pred Value : 0.6932
##              Prevalence : 0.6234
##          Detection Rate : 0.5065
##    Detection Prevalence : 0.6190
##       Balanced Accuracy : 0.7568
##
##        'Positive' Class : 0
##
```

We can see that we have a high percentage when coming to detect people without diabetes (81.25%). The accuracy is not that high when we try and detect positive outcome with only (70.11%).

The best mtry value is 2. This can be seen in the following graph:

```
cols <- replicate(fit_rf$results %>% nrow, "steelblue")
cols[which.max(fit_rf$results$Accuracy)] = "red"

fit_rf$results %>%
  as_tibble() %>%
  ggplot(aes(x = mtry, y = Accuracy)) +
```

```
    geom_line(color = "steelblue") +
    geom_label(aes(label = mtry), fill = cols, color = "white") +
    theme_minimal()
```



K Nearest Neighbours

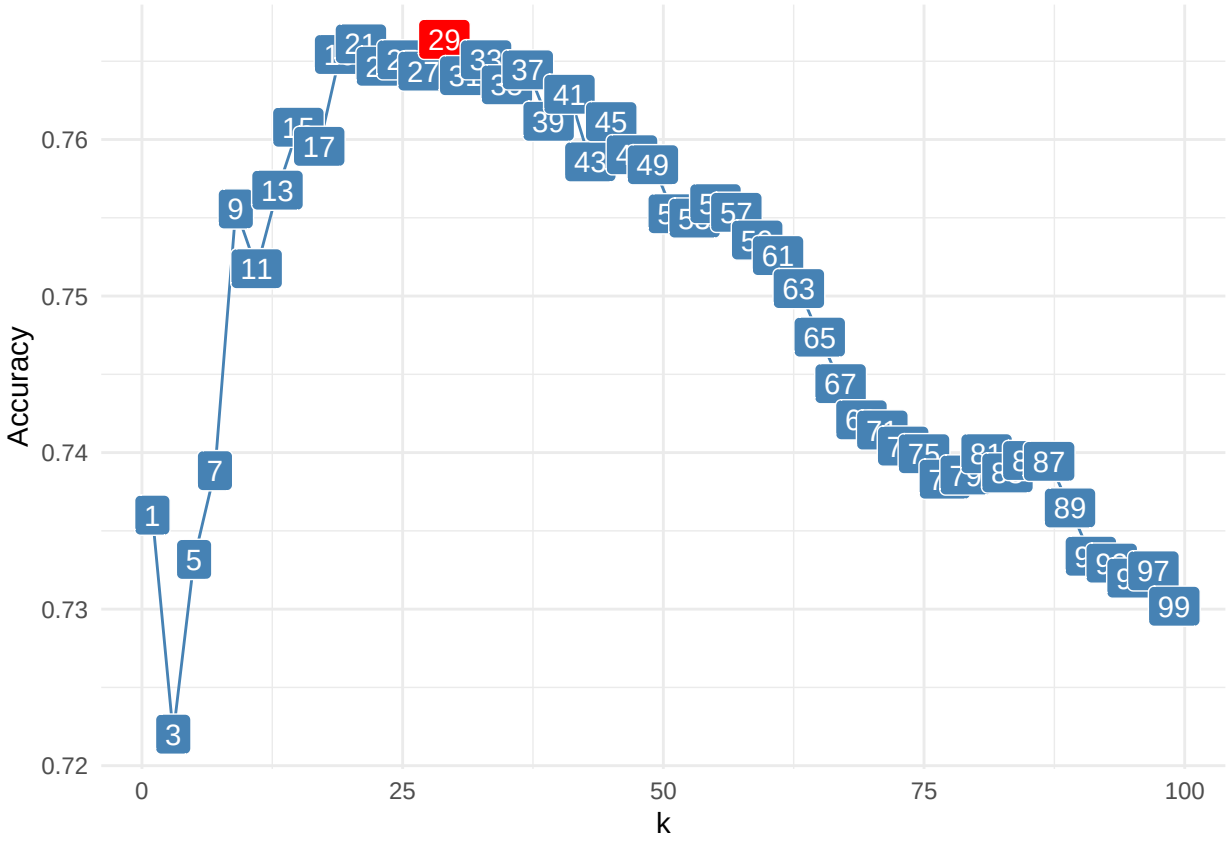
k is the tuning parameter. We will try every second k values from 1 to 100.

```
set.seed(1976)
fit_knn <- train(factor(Outcome) ~ ., data = train_set, method = "knn", tuneGrid = data.frame(k = seq(1
y_hat_knn <- predict(fit_knn, newdata = test_set)
conf_matrix_knn <- confusionMatrix(factor(y_hat_knn), factor(test_set$Outcome))
conf_matrix_knn
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 113  41
##          1  31  46
##
##                Accuracy : 0.6883
##                  95% CI : (0.6243, 0.7474)
##     No Information Rate : 0.6234
##     P-Value [Acc > NIR] : 0.02342
##
##                   Kappa : 0.3208
##
##  Mcnemar's Test P-Value : 0.28884
##
##             Sensitivity : 0.7847
##             Specificity : 0.5287
##          Pos Pred Value : 0.7338
##          Neg Pred Value : 0.5974
##              Prevalence : 0.6234
##          Detection Rate : 0.4892
##    Detection Prevalence : 0.6667
##       Balanced Accuracy : 0.6567
##
##        'Positive' Class : 0
##
```

We can graph the accuracy vs. the k value like so:

```
cols <- replicate(fit_knn$results %>% nrow, "steelblue")
cols[which.max(fit_knn$results$Accuracy)] = "red"

fit_knn$results %>%
  as_tibble() %>%
  ggplot(aes(x = k, y = Accuracy)) +
    geom_line(color = "steelblue") +
    geom_label(aes(label = k), fill = cols, color = "white") +
    theme_minimal()
```



The best k value here is 29.

Gradient Boosting Machine

To tune gbm, the following parameters are required:

```
getModelInfo()$gbm$parameters
```

```
##           parameter   class                     label
## 1           n.trees numeric   # Boosting Iterations
## 2 interaction.depth numeric          Max Tree Depth
## 3         shrinkage numeric               Shrinkage
## 4    n.minobsinnode numeric Min. Terminal Node Size
```

For interaction.depth, the max value is the number of predictors, so 8 in our case. For shrinkage, generally, the smaller the number the better the predicting value keeping in mind the modest computational resources of a laptop. n.minobsinnode's default is 10, but for classification smaller number work well. To minimize the time it runs, we will leave it at 10, but we could supply it a sequence like seq(1, 11, 2). For small shrinkage, we need more trees. We will go up to 5000 trees.

```
set.seed(1976)

tune_grid_gbm <- expand.grid(n.trees = seq(50, 5000, 50),
                             interaction.depth = seq(1, 9, 2),
                             shrinkage = 0.01,
                             n.minobsinnode = 10)
fit_gbm <- train(factor(Outcome) ~ ., data = train_set, method = "gbm", tuneGrid = tune_grid_gbm, verbos
y_hat_gbm <- predict(fit_gbm, newdata = test_set)
conf_matrix_gbm <- confusionMatrix(factor(y_hat_gbm), factor(test_set$Outcome))
conf_matrix_gbm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 121  29
##          1  23  58
##
##                Accuracy : 0.7749
##                  95% CI : (0.7155, 0.8271)
##     No Information Rate : 0.6234
##     P-Value [Acc > NIR] : 6.037e-07
##
##                   Kappa : 0.514
##
##  Mcnemar's Test P-Value : 0.4881
##
##             Sensitivity : 0.8403
##             Specificity : 0.6667
##          Pos Pred Value : 0.8067
##          Neg Pred Value : 0.7160
##              Prevalence : 0.6234
##          Detection Rate : 0.5238
##    Detection Prevalence : 0.6494
##       Balanced Accuracy : 0.7535
##
```

```
##        'Positive' Class : 0
##
```

ensemble rpart, rf, and gbm

We take an average of the different models' predictions, and declare an outcome of 1 when the average is equal or above 2/3 (either 2 or 3 of the methods precict diabetes).

```
y_hat_ensemble <- (y_hat_rf %>% as.character() %>% as.numeric() +
                   y_hat_gbm %>% as.character() %>% as.numeric() +
                   y_hat_rpart %>% as.character() %>% as.numeric()) / 3

y_hat_ensemble <-  replace(y_hat_ensemble, y_hat_ensemble >= 2/3, 1)
y_hat_ensemble <-  replace(y_hat_ensemble, y_hat_ensemble < 2/3, 0)

conf_matrix_ensemble <- confusionMatrix(factor(y_hat_ensemble), factor(test_set$Outcome))
conf_matrix_ensemble
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 118  28
##          1  26  59
##
##                Accuracy : 0.7662
##                  95% CI : (0.7063, 0.8192)
##     No Information Rate : 0.6234
##     P-Value [Acc > NIR] : 2.563e-06
##
##                   Kappa : 0.4999
##
##  Mcnemar's Test P-Value : 0.8918
##
##             Sensitivity : 0.8194
##             Specificity : 0.6782
##          Pos Pred Value : 0.8082
##          Neg Pred Value : 0.6941
##              Prevalence : 0.6234
##          Detection Rate : 0.5108
##    Detection Prevalence : 0.6320
##       Balanced Accuracy : 0.7488
##
##        'Positive' Class : 0
##
```

Let's summarize the accuracies of the different methods:

```
  accuracies <- c(
    conf_matrix_guess$overall["Accuracy"],
    conf_matrix_glm$overall["Accuracy"],
    conf_matrix_rpart$overall["Accuracy"],
    conf_matrix_rf$overall["Accuracy"],
    conf_matrix_knn$overall["Accuracy"],
    conf_matrix_gbm$overall["Accuracy"],
    conf_matrix_ensemble$overall["Accuracy"]
  )
```

```r
accuracies_table <- accuracies %>%
  as.table() %>%
  setNames(c("guessing", "glm", "rpart", "rf", "knn", "gbm", "ensemble"))

accuracies_table
```

```
##  guessing       glm     rpart        rf       knn       gbm  ensemble
## 0.4329004 0.7142857 0.7878788 0.7705628 0.6883117 0.7748918 0.7662338
```

Conclusion

When approaching a choice of a method, in this particular case, we will choose the method that produces the highest overall accuracy as the model will, undoubtedly, be doubled by proper medical (repeated?) detection.

That method is rpart with an accuracy of 0.7879%.

Further improvement in accuracy could probably be gained by removing some features, and by tuning further, althouthg that would come at a price of computational time and resources.