An R Markdown document converted from "breast-cancer-predictive-model.ipynb"

Predicting Breast Cancer

This looks into using various R machine learning and data-science libraries in an attempt to build a machine learning model capable to predict whether or not a person has breast cancer based on their medical attributes.

Project approach:

1. Problem definition
2. Data
3. Evaluation
4. Features
5. Modelling
6. Experimentation

1. Problem definition

In a statement:

> Based on a model trained on synthetic data, how well can said model predict whether or not a clinical patient have breast cancer?

2. Data

The original data came from the Coimbra data of the UCI Machine Learning Repository. https://archive.ics.uci.edu/dataset/451/breast+cancer+coimbra

It will be used alongside a synthetic data found in Kaggle. https://www.kaggle.com/datasets/atom1991/breast-cancer-coimbra

3. Evaluation

> The model will be evaluated on various metrics: accuracy, precision, recall, f1-score, confusion matrix.

4. Features

A list of important information regarding the features of the data.

Create data dictionary Quantitative Attributes:

1. Age (years): Represents the age of individuals in the dataset.
2. BMI (kg/m^2): Body Mass Index, a measure of body fat based on weight and height.
3. Glucose (mg/dL): Reflects blood glucose levels, a vital metabolic indicator.
4. Insulin (µU/mL): Indicates insulin levels, a hormone associated with glucose regulation.
5. HOMA: Homeostatic Model Assessment, a method assessing insulin resistance and beta-cell function.
6. Leptin (ng/mL): Represents leptin levels, a hormone involved in appetite and energy balance regulation.
7. Adiponectin (µg/mL): Reflects adiponectin levels, a protein associated with metabolic regulation.
8. Resistin (ng/mL): Indicates resistin levels, a protein implicated in insulin resistance.
9. MCP-1 (pg/dL): Reflects Monocyte Chemoattractant Protein-1 levels, a cytokine involved in inflammation.
10. Labels:
    - 1: Healthy controls
    - 2: Patients with breast cancer

Import libraries

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(dplyr)
```

Import dataset:

```r
data <- read_csv("Coimbra_breast_cancer_dataset.csv")
```

```
## Rows: 4000 Columns: 10
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (10): Age, BMI, Glucose, Insulin, HOMA, Leptin, Adiponectin, Resistin, M...
##
## 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.
```

```r
head(data)
```

```
## # A tibble: 6 x 10
##     Age   BMI Glucose Insulin  HOMA Leptin Adiponectin Resistin MCP.1
##   <dbl> <dbl>   <dbl>   <dbl> <dbl>  <dbl>       <dbl>    <dbl> <dbl>
```

```
## 1    54  35.2     103    5.64  1.38   65.7        9.74    31.2   198.
## 2    52  23.0     132    6.05  1.15   47.5        3.63    23.0   423.
## 3    32  21.1      87    5.67  1.01   50.5        5.07     9.51  897.
## 4    42  26.8     132    2.88  1.00   16.9       10.1      9.76  268.
## 5    55  34.2      76    3.12  1.60   17.7       11.8     17.2   789.
## 6    47  26.6     136    3.38  4.52   29.5       10.4      5.59  354.
## # i 1 more variable: Classification <dbl>
```

```
colnames(data)
```

```
##  [1] "Age"            "BMI"            "Glucose"        "Insulin"
##  [5] "HOMA"           "Leptin"         "Adiponectin"    "Resistin"
##  [9] "MCP.1"          "Classification"
```

```
# org_data <- read_csv("/kaggle/input/breastcancercoimbra-csv/BreastCancerCoimbra.csv")
org_data <- read_csv("dataR2.csv")
```

```
## Rows: 116 Columns: 10
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (10): Age, BMI, Glucose, Insulin, HOMA, Leptin, Adiponectin, Resistin, M...
##
## 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.
```

```
head(org_data)
```

```
## # A tibble: 6 x 10
##     Age   BMI Glucose Insulin  HOMA Leptin Adiponectin Resistin MCP.1
##   <dbl> <dbl>   <dbl>   <dbl> <dbl>  <dbl>       <dbl>    <dbl> <dbl>
## 1    48  23.5      70    2.71 0.467   8.81        9.70     8.00  417.
## 2    83  20.7      92    3.12 0.707   8.84        5.43     4.06  469.
## 3    82  23.1      91    4.50 1.01   17.9        22.4      9.28  555.
## 4    68  21.4      77    3.23 0.613   9.88        7.17    12.8   928.
## 5    86  21.1      92    3.55 0.805   6.70        4.82    10.6   774.
## 6    49  22.9      92    3.23 0.732   6.83       13.7     10.3   530.
## # i 1 more variable: Classification <dbl>
```

```
colnames(org_data)
```

```
##  [1] "Age"            "BMI"            "Glucose"        "Insulin"
##  [5] "HOMA"           "Leptin"         "Adiponectin"    "Resistin"
##  [9] "MCP.1"          "Classification"
```

```
# Check the similarity of feature names of both datasets
all.equal(colnames(data), colnames(org_data))
```

```
## [1] TRUE
```

Exploratory Data Analysis

EDA is applied to the two datasets, the synthetic and the original datasets.

```r
# View the dimension
dim(data)
```

Synthetic dataset

```
## [1] 4000   10
```

What is the class of the columns in the dataframe?

```r
str(data)
```

```
## spc_tbl_ [4,000 x 10] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Age           : num [1:4000] 54 52 32 42 55 47 32 53 57 83 ...
##  $ BMI           : num [1:4000] 35.2 23 21.1 26.8 34.2 ...
##  $ Glucose       : num [1:4000] 103 132 87 132 76 136 134 131 76 76 ...
##  $ Insulin       : num [1:4000] 5.64 6.05 5.67 2.88 3.12 ...
##  $ HOMA          : num [1:4000] 1.38 1.15 1.01 1 1.6 ...
##  $ Leptin        : num [1:4000] 65.7 47.5 50.5 16.9 17.7 ...
##  $ Adiponectin   : num [1:4000] 9.74 3.63 5.07 10.1 11.85 ...
##  $ Resistin      : num [1:4000] 31.17 23.03 9.51 9.76 17.22 ...
##  $ MCP.1         : num [1:4000] 198 423 897 268 789 ...
##  $ Classification: num [1:4000] 1 2 2 1 2 1 2 1 2 1 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Age = col_double(),
##   ..   BMI = col_double(),
##   ..   Glucose = col_double(),
##   ..   Insulin = col_double(),
##   ..   HOMA = col_double(),
##   ..   Leptin = col_double(),
##   ..   Adiponectin = col_double(),
##   ..   Resistin = col_double(),
##   ..   MCP.1 = col_double(),
##   ..   Classification = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

Convert features into their appropriate datatypes. Features with values without decimal points are converted into integers and features with classes are converted into factors. Features with decimal points will remain as numeric. - `Age`: `integer` - `BMI`: `numeric` - `Glucose`: `integer` - `Insulin`: `numeric` - `HOMA`: `numeric` - `Leptin`: `numeric` - `Adiponectin`: `numeric` - `Resistin`: `numeric` - `MCP.1`: `integer` - `Classification`: `factor`

```r
# Convert features datatype from numeric to integer
data$Age <- as.integer(data$Age)
data$Glucose <- as.integer(data$Glucose)
data$MCP.1 <- as.integer(data$MCP.1)

# Convert target feature datatype from numeric to factor
data$Classification <- as.factor(data$Classification)
```

```r
str(data)
```

```
## spc_tbl_ [4,000 x 10] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Age           : int [1:4000] 54 52 32 42 55 47 32 53 57 83 ...
##  $ BMI           : num [1:4000] 35.2 23 21.1 26.8 34.2 ...
##  $ Glucose       : int [1:4000] 103 132 87 132 76 136 134 131 76 76 ...
##  $ Insulin       : num [1:4000] 5.64 6.05 5.67 2.88 3.12 ...
##  $ HOMA          : num [1:4000] 1.38 1.15 1.01 1 1.6 ...
##  $ Leptin        : num [1:4000] 65.7 47.5 50.5 16.9 17.7 ...
##  $ Adiponectin   : num [1:4000] 9.74 3.63 5.07 10.1 11.85 ...
##  $ Resistin      : num [1:4000] 31.17 23.03 9.51 9.76 17.22 ...
##  $ MCP.1         : int [1:4000] 197 423 897 267 788 354 331 415 585 209 ...
##  $ Classification: Factor w/ 2 levels "1","2": 1 2 2 1 2 1 2 1 2 1 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Age = col_double(),
##   ..   BMI = col_double(),
##   ..   Glucose = col_double(),
##   ..   Insulin = col_double(),
##   ..   HOMA = col_double(),
##   ..   Leptin = col_double(),
##   ..   Adiponectin = col_double(),
##   ..   Resistin = col_double(),
##   ..   MCP.1 = col_double(),
##   ..   Classification = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```r
# View count of null values
sum(is.na(data))
```

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

This data does not contain any missing values so we do not need to remove or replace any rows.

Display the summary statistics for the numeric variables (which includes both integer and floats):

```r
# Subset of data containing only numeric data
numeric_data <- data[, -which(colnames(data) == "Classification")]
```

```r
summary(numeric_data)
```

```
##       Age             BMI           Glucose         Insulin
##  Min.   :32.00   Min.   :20.69   Min.   : 76.0   Min.   : 2.821
##  1st Qu.:39.00   1st Qu.:23.08   1st Qu.: 76.0   1st Qu.: 4.422
##  Median :56.00   Median :27.56   Median :131.0   Median : 5.818
##  Mean   :56.21   Mean   :27.42   Mean   :113.9   Mean   : 8.654
##  3rd Qu.:72.00   3rd Qu.:30.81   3rd Qu.:134.0   3rd Qu.:10.466
##  Max.   :85.00   Max.   :36.21   Max.   :138.0   Max.   :30.211
##       HOMA            Leptin        Adiponectin        Resistin
##  Min.   :0.5900   Min.   : 6.832   Min.   : 3.192   Min.   : 4.190
##  1st Qu.:0.9701   1st Qu.:12.713   1st Qu.: 5.580   1st Qu.: 7.022
```

```
##  Median :1.3738   Median :19.805   Median : 8.287   Median :10.693
##  Mean   :2.0243   Mean   :25.138   Mean   : 9.365   Mean   :13.054
##  3rd Qu.:2.5028   3rd Qu.:36.670   3rd Qu.:10.918   3rd Qu.:17.295
##  Max.   :8.2185   Max.   :68.507   Max.   :22.537   Max.   :42.618
##      MCP.1
##  Min.   : 137.0
##  1st Qu.: 299.0
##  Median : 482.0
##  Mean   : 511.7
##  3rd Qu.: 690.5
##  Max.   :1227.0
```

```r
head(data)
```

```
## # A tibble: 6 x 10
##     Age   BMI Glucose Insulin  HOMA Leptin Adiponectin Resistin MCP.1
##   <int> <dbl>   <int>   <dbl> <dbl>  <dbl>       <dbl>    <dbl> <int>
## 1    54  35.2     103    5.64  1.38   65.7        9.74    31.2    197
## 2    52  23.0     132    6.05  1.15   47.5        3.63    23.0    423
## 3    32  21.1      87    5.67  1.01   50.5        5.07     9.51   897
## 4    42  26.8     132    2.88  1.00   16.9       10.1      9.76   267
## 5    55  34.2      76    3.12  1.60   17.7       11.8     17.2    788
## 6    47  26.6     136    3.38  4.52   29.5       10.4      5.59   354
## # i 1 more variable: Classification <fct>
```

```r
head(numeric_data)
```

```
## # A tibble: 6 x 9
##     Age   BMI Glucose Insulin  HOMA Leptin Adiponectin Resistin MCP.1
##   <int> <dbl>   <int>   <dbl> <dbl>  <dbl>       <dbl>    <dbl> <int>
## 1    54  35.2     103    5.64  1.38   65.7        9.74    31.2    197
## 2    52  23.0     132    6.05  1.15   47.5        3.63    23.0    423
## 3    32  21.1      87    5.67  1.01   50.5        5.07     9.51   897
## 4    42  26.8     132    2.88  1.00   16.9       10.1      9.76   267
## 5    55  34.2      76    3.12  1.60   17.7       11.8     17.2    788
## 6    47  26.6     136    3.38  4.52   29.5       10.4      5.59   354
```

Histograms of numeric variables:

```r
# Setting up the layout for multiple histograms
par(mfrow = c(3, 3))

# Looping through each numeric variable to create histograms
for (i in 1:ncol(numeric_data)) {
    hist(numeric_data[[i]], main = colnames(numeric_data)[i], xlab = colnames(numeric_data)[i])
}
```

Find the count of each class in target feature.

```
counts <- table(data$Classification)

counts
```

```
##
##    1    2
## 1784 2216
```

```
barplot(counts, main="Target Variable Value Counts", xlab="target")
```

Calculate the correlation matrix between the different variables:

```
correlation_matrix <- cor(numeric_data)
correlation_matrix
```

```
##                      Age          BMI       Glucose       Insulin          HOMA
## Age          1.000000000  0.019139084  0.0026106626  0.0007423400  0.0246307970
## BMI          0.019139084  1.000000000 -0.0144685660  0.0115743909  0.0127951212
## Glucose      0.002610663 -0.014468566  1.0000000000  0.0005577622 -0.0182322119
## Insulin      0.000742340  0.011574391  0.0005577622  1.0000000000 -0.0131416590
## HOMA         0.024630797  0.012795121 -0.0182322119 -0.0131416590  1.0000000000
## Leptin       0.005058408 -0.011482224 -0.0064606033  0.0056783744  0.0067512331
## Adiponectin -0.013493173 -0.002348453  0.0163631580 -0.0145596916  0.0034330222
## Resistin    -0.013437974  0.005303273  0.0141791937  0.0361359879  0.0002720846
## MCP.1       -0.019420181 -0.023411886  0.0018688834 -0.0114135546  0.0099671791
##                   Leptin  Adiponectin      Resistin        MCP.1
## Age          0.005058408 -0.013493173 -0.0134379745 -0.019420181
## BMI         -0.011482224 -0.002348453  0.0053032733 -0.023411886
## Glucose     -0.006460603  0.016363158  0.0141791937  0.001868883
## Insulin      0.005678374 -0.014559692  0.0361359879 -0.011413555
## HOMA         0.006751233  0.003433022  0.0002720846  0.009967179
## Leptin       1.000000000  0.024053842  0.0142167959 -0.003354728
## Adiponectin  0.024053842  1.000000000 -0.0189448380 -0.018312770
## Resistin     0.014216796 -0.018944838  1.0000000000 -0.007619783
## MCP.1       -0.003354728 -0.018312770 -0.0076197833  1.000000000
```

```r
pairs(data)
```

```r
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```r
corrplot(correlation_matrix, type="full", method ="color", title = "Breast Cancer correlation plot", ma
```

There does not seem to be high correlations between the features.

```
# View the dimension
dim(org_data)
```

Original Dataset

```
## [1] 116  10
```

What is the class of the features in the dataset?

```
str(org_data)
```

```
## spc_tbl_ [116 x 10] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Age           : num [1:116] 48 83 82 68 86 49 89 76 73 75 ...
##  $ BMI           : num [1:116] 23.5 20.7 23.1 21.4 21.1 ...
##  $ Glucose       : num [1:116] 70 92 91 77 92 92 77 118 97 83 ...
##  $ Insulin       : num [1:116] 2.71 3.12 4.5 3.23 3.55 ...
##  $ HOMA          : num [1:116] 0.467 0.707 1.01 0.613 0.805 ...
##  $ Leptin        : num [1:116] 8.81 8.84 17.94 9.88 6.7 ...
##  $ Adiponectin   : num [1:116] 9.7 5.43 22.43 7.17 4.82 ...
##  $ Resistin      : num [1:116] 8 4.06 9.28 12.77 10.58 ...
```

```
##  $ MCP.1         : num [1:116] 417 469 555 928 774 ...
##  $ Classification: num [1:116] 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Age = col_double(),
##   ..   BMI = col_double(),
##   ..   Glucose = col_double(),
##   ..   Insulin = col_double(),
##   ..   HOMA = col_double(),
##   ..   Leptin = col_double(),
##   ..   Adiponectin = col_double(),
##   ..   Resistin = col_double(),
##   ..   MCP.1 = col_double(),
##   ..   Classification = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

Convert features into their appropriate datatypes. Features with values without decimal points are converted into integers and features with classes are converted into factors. Features with decimal points will remain as numeric. - `Age`: `integer` - `BMI`: `numeric` - `Glucose`: `integer` - `Insulin`: `numeric` - `HOMA`: `numeric` - `Leptin`: `numeric` - `Adiponectin`: `numeric` - `Resistin`: `numeric` - `MCP.1`: `integer` - `Classification`: `factor`

```r
# Convert features datatype from numeric to integer
org_data$Age <- as.integer(org_data$Age)
org_data$Glucose <- as.integer(org_data$Glucose)
org_data$MCP.1 <- as.integer(org_data$MCP.1)

# Convert target feature datatype from numeric to factor
org_data$Classification <- as.factor(org_data$Classification)
```

```r
str(org_data)
```

```
## spc_tbl_ [116 x 10] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Age           : int [1:116] 48 83 82 68 86 49 89 76 73 75 ...
##  $ BMI           : num [1:116] 23.5 20.7 23.1 21.4 21.1 ...
##  $ Glucose       : int [1:116] 70 92 91 77 92 92 77 118 97 83 ...
##  $ Insulin       : num [1:116] 2.71 3.12 4.5 3.23 3.55 ...
##  $ HOMA          : num [1:116] 0.467 0.707 1.01 0.613 0.805 ...
##  $ Leptin        : num [1:116] 8.81 8.84 17.94 9.88 6.7 ...
##  $ Adiponectin   : num [1:116] 9.7 5.43 22.43 7.17 4.82 ...
##  $ Resistin      : num [1:116] 8 4.06 9.28 12.77 10.58 ...
##  $ MCP.1         : int [1:116] 417 468 554 928 773 530 1256 280 136 318 ...
##  $ Classification: Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Age = col_double(),
##   ..   BMI = col_double(),
##   ..   Glucose = col_double(),
##   ..   Insulin = col_double(),
##   ..   HOMA = col_double(),
##   ..   Leptin = col_double(),
##   ..   Adiponectin = col_double(),
##   ..   Resistin = col_double(),
```

```
##   ..   MCP.1 = col_double(),
##   ..   Classification = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```r
# View count of null data
sum(is.na(org_data))
```

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

Since there are no null values, there is no need to replace or remove observations and/or features.

Display the summary statistics for the numeric variables (which includes both integer and floats):

```r
summary(org_data[, -which(colnames(org_data) == "Classification")])
```

```
##       Age            BMI           Glucose          Insulin
##  Min.   :24.0   Min.   :18.37   Min.   : 60.00   Min.   : 2.432
##  1st Qu.:45.0   1st Qu.:22.97   1st Qu.: 85.75   1st Qu.: 4.359
##  Median :56.0   Median :27.66   Median : 92.00   Median : 5.925
##  Mean   :57.3   Mean   :27.58   Mean   : 97.79   Mean   :10.012
##  3rd Qu.:71.0   3rd Qu.:31.24   3rd Qu.:102.00   3rd Qu.:11.189
##  Max.   :89.0   Max.   :38.58   Max.   :201.00   Max.   :58.460
##       HOMA             Leptin        Adiponectin        Resistin
##  Min.   : 0.4674   Min.   : 4.311   Min.   : 1.656   Min.   : 3.210
##  1st Qu.: 0.9180   1st Qu.:12.314   1st Qu.: 5.474   1st Qu.: 6.882
##  Median : 1.3809   Median :20.271   Median : 8.353   Median :10.828
##  Mean   : 2.6950   Mean   :26.615   Mean   :10.181   Mean   :14.726
##  3rd Qu.: 2.8578   3rd Qu.:37.378   3rd Qu.:11.816   3rd Qu.:17.755
##  Max.   :25.0503   Max.   :90.280   Max.   :38.040   Max.   :82.100
##      MCP.1
##  Min.   :  45.0
##  1st Qu.: 269.8
##  Median : 470.5
##  Mean   : 534.1
##  3rd Qu.: 699.2
##  Max.   :1698.0
```

Histograms of numeric features:

```r
# Setting up the layout for multiple histograms
par(mfrow = c(3, 3))

# Looping through each numeric variable to create histograms
for (i in 1:ncol(data[, -which(colnames(org_data) == "Classification")])) {
    hist(org_data[[i]], main = colnames(org_data)[i], xlab = colnames(org_data)[i])
}
```

Find the count of each class in target feature.

```
org_counts <- table(org_data$Classification)

org_counts
```

```
##
##  1  2
## 52 64
```

```
barplot(org_counts, main="Target Variable Value Counts", xlab="target")
```

Calculate the correlation matrix between the different features.

```
pairs(org_data)
```

```r
org_cor_matrix <- cor(org_data[, -which(colnames(org_data) == "Classification")])

org_cor_matrix
```

```
##                      Age          BMI    Glucose     Insulin        HOMA
## Age          1.000000000  0.008529857  0.2301056  0.03249535  0.12703259
## BMI          0.008529857  1.000000000  0.1388452  0.14529526  0.11448013
## Glucose      0.230105617  0.138845189  1.0000000  0.50465307  0.69621182
## Insulin      0.032495353  0.145295260  0.5046531  1.00000000  0.93219777
## HOMA         0.127032593  0.114480131  0.6962118  0.93219777  1.00000000
## Leptin       0.102626049  0.569592606  0.3050799  0.30146162  0.32720986
## Adiponectin -0.219812891 -0.302734758 -0.1221213 -0.03129608 -0.05633712
## Resistin     0.002741708  0.195350206  0.2913275  0.14673099  0.23110123
## MCP.1        0.013495741  0.224158822  0.2649158  0.17445601  0.25962496
##                  Leptin Adiponectin     Resistin       MCP.1
## Age          0.10262605 -0.21981289  0.002741708  0.01349574
## BMI          0.56959261 -0.30273476  0.195350206  0.22415882
## Glucose      0.30507994 -0.12212131  0.291327462  0.26491580
## Insulin      0.30146162 -0.03129608  0.146730986  0.17445601
## HOMA         0.32720986 -0.05633712  0.231101229  0.25962496
## Leptin       1.00000000 -0.09538874  0.256233522  0.01406955
## Adiponectin -0.09538874  1.00000000 -0.252363303 -0.20068796
## Resistin     0.25623352 -0.25236330  1.000000000  0.36650168
## MCP.1        0.01406955 -0.20068796  0.366501681  1.00000000
```

```r
corrplot(org_cor_matrix, type="full", method ="color", title = "Breast Cancer correlation plot", mar=c(
```

5. Modelling

Once the data has been split into training and test sets, the data will be fit into different machine learning models: - Logistic Regression - K-Nearest Neighbors Classifier (KNN) - Linear Discriminant Analysis (LDA) - Quadratic Discriminant Analysis (QDA)

The model will be trained to find the patterns on the training set and tested using the patterns it found on the test set. Since two different datasets will be used, the datasets will be split into train-val-test sets: - train: 70% of the synthetic data (kaggle data) - validation: 30% of the synthetic data (kaggle data) - test: 100% of the original data (UCI ML data)

Split the dataset using 70% for training and 30% for validation and prepare test set:

```r
set.seed(1)

train_index <- sample(1:nrow(data), 0.7 * nrow(data))
training_data <- data[train_index, ]
val_data <- data[-train_index, ]
test_data <- org_data
```

```r
# Ensure reproducibility of results
set.seed(1)
```

```r
# cbind(): column bind
train.X <- training_data[, -which(colnames(data) == "Classification")]
val.X <- val_data[, -which(colnames(data) == "Classification")]
test.X <- test_data[, -which(colnames(data) == "Classification")]
train.y <- training_data$Classification
val.y <- val_data$Classification
test.y <- test_data$Classification
```

Principal Component Analysis (PCA)

We will use PCA to check for homogeneity among the datasets. This is important for future generalization. Since we aim to build a predictive model that can generalize to unseen data, it is important to check that the training and testing data comes from similar distributions. If not, the model will not generalize well and have a poor performance.

```r
# standardizing synthetic data
trainX_scaled <- scale(train.X)
valX_scaled <- scale(val.X)

# combining the train and validation features for PCA
combined_features <- rbind(trainX_scaled, valX_scaled)

# performing PCA
pca_result <- prcomp(combined_features, center = TRUE, scale. = TRUE)

# applying PCA transformation to train and validation features
train_pca <- predict(pca_result, trainX_scaled)
validation_pca <- predict(pca_result, valX_scaled)
```

```r
# standardizing original data
testX_scaled <- scale(test.X)

# applying PCA transformation to original features using the PCA model from synthetic data
original_pca <- predict(pca_result, testX_scaled)
```

We will check for homogeneity by comparing the distributions of principal components (PCs) between the synthetic data and the original data.

By plotting, we can visually inspect to see if there are any noticeable patterns or clusters. If the datasets are homogeneous, you would expect the points from both datasets to be distributed similarly and not form distinct clusters.

```r
pca_scatterplot <- function(train_pca, original_pca) {

  par(mfrow = c(2, 2),  mar = c(5, 5, 5, 5))

  options(repr.plot.width = 30, repr.plot.height = 30)

  for (i in 1:4) {
    plot(train_pca[, i], train_pca[, i+1],
         xlab = paste("Synthetic PC", i), ylab = paste("Synthetic PC", i+1),
         main = paste("Synthetic PC", i, "vs", "Synthetic PC", i+1),
         pch = 16, col = "blue", cex = 1.5)
```

```
    points(original_pca[, i], original_pca[, i+1], pch = 16, col = "red", cex = 1.5)
  }
  legend("bottomright", legend = c("Synthetic Data", "Original Data"),
         pch = 16, col = c("blue", "red"), cex = 0.8)
}

pca_scatterplot(train_pca, original_pca)
```

In the scatterplots above, we can see that the datasets are homogenous because there are no clusters and the points from both the synthetic and original data are spread out.

```
plot_pca <- function(train_pca, original_pca) {
  par(mfrow = c(2, 4))

  for (i in 1:4) {
    hist(train_pca[, i],
         main = paste("Synthetic PC", i),
         xlab = "Principal Component Value",
         col = "lightblue",
         border = "black")

    hist(original_pca[, i],
         main = paste("Original PC", i),
         xlab = "Principal Component Value",
         col = "lightgreen",
```

```
        border = "black")
  }
}

plot_pca(train_pca, original_pca)
```

Looking at the histograms above, we can see that the shapes for both the synthetic and original data are very similar. They exhibit similar distribution shapes with centers that are close or aligned. Also, the spread of the histograms are similar, further indicating homogeneity.

We can also find the variance explained by each principal component (PC).

```
# Calculate the variance explained by each principal component
variance_explained <- (pca_result$sdev)^2
variance_explained
```

```
## [1] 1.0554052 1.0518554 1.0336469 1.0165072 1.0005579 0.9842718 0.9646444
## [8] 0.9609623 0.9321489
```

```
# Calculate the total variance
total_variance <- sum(variance_explained)
total_variance
```

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

```r
# Calculate the proportion of variance explained by each principal component
proportion_variance_explained <- variance_explained / total_variance
proportion_variance_explained
```

```
## [1] 0.1172672 0.1168728 0.1148497 0.1129452 0.1111731 0.1093635 0.1071827
## [8] 0.1067736 0.1035721
```

```r
variance_explained_df <- data.frame(
  PC = 1:length(variance_explained),
  Variance_Explained = variance_explained,
  Proportion_Variance_Explained = proportion_variance_explained
)

variance_explained_df
```

```
##   PC Variance_Explained Proportion_Variance_Explained
## 1  1          1.0554052                     0.1172672
## 2  2          1.0518554                     0.1168728
## 3  3          1.0336469                     0.1148497
## 4  4          1.0165072                     0.1129452
## 5  5          1.0005579                     0.1111731
## 6  6          0.9842718                     0.1093635
## 7  7          0.9646444                     0.1071827
## 8  8          0.9609623                     0.1067736
## 9  9          0.9321489                     0.1035721
```

The variance explained by each PC can reveal how much of the total variability in the dataset is captured by each PC. This information can be useful to understand the underlying structure of the data because components that explain a large amount of variance may represent important patterns or relationships. By better understanding the data, we can make better decisions in terms of feature selection and model building.

Logistic Regression Model

```r
glm.model <- glm(Classification ~ ., data = training_data, family = binomial)

summary(glm.model)
```

```
##
## Call:
## glm(formula = Classification ~ ., family = binomial, data = training_data)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  5.048e-01  3.595e-01   1.404    0.160
## Age         -8.929e-04  2.142e-03  -0.417    0.677
## BMI         -6.768e-03  8.669e-03  -0.781    0.435
## Glucose     -1.259e-03  1.467e-03  -0.858    0.391
## Insulin     -4.904e-03  5.851e-03  -0.838    0.402
## HOMA        -3.847e-03  2.364e-02  -0.163    0.871
## Leptin      -9.507e-04  2.483e-03  -0.383    0.702
```

```
## Adiponectin  3.898e-03  8.106e-03   0.481    0.631
## Resistin     6.663e-03  5.207e-03   1.280    0.201
## MCP.1        4.693e-05  1.514e-04   0.310    0.757
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3854.4  on 2799  degrees of freedom
## Residual deviance: 3850.1  on 2790  degrees of freedom
## AIC: 3870.1
##
## Number of Fisher Scoring iterations: 3
```

```r
# Predict on validation data
glm.probs <- predict(glm.model, val.X, type = "response")

head(glm.probs)
```

```
##         1         2         3         4         5         6
## 0.5617140 0.5608973 0.5588253 0.5694397 0.5606022 0.5261208
```

```r
# Get performance measures
num_test <- length(glm.probs)
glm.pred <- rep(0, num_test)
glm.pred[glm.probs > 0.5] <- 1

# Confusion matrix
table(glm.pred, val.y)
```

```
##         val.y
## glm.pred   1   2
##        0   0   2
##        1 522 676
```

```r
# Validation error
mean(glm.pred != val.y)
```

```
## [1] 0.565
```

```r
# Accuracy score
1 - mean(glm.pred != val.y)
```

```
## [1] 0.435
```

```r
# Predict on test data
glm.probs_test <- predict(glm.model, test.X, type = "response")

head(glm.probs_test)
```

```
##         1         2         3         4         5         6
## 0.5749235 0.5544364 0.5726687 0.5821229 0.5665884 0.5774556
```

```r
# Get performance measures
num_test <- length(glm.probs_test)
glm.pred_test <- rep(0, num_test)
glm.pred_test[glm.probs_test > 0.5] <- 1

# Confusion matrix
confusion_matrix <- table(glm.pred_test, test.y)
confusion_matrix
```

```
##              test.y
## glm.pred_test  1  2
##             0  0  7
##             1 52 57
```

Test error:

```r
mean(glm.pred_test != test.y)
```

```
## [1] 0.5517241
```

Validation error:

```r
mean(glm.pred != val.y)
```

```
## [1] 0.565
```

```r
accuracy_score <- 1 - mean(glm.pred != val.y)
accuracy_score
```

```
## [1] 0.435
```

```r
precision <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
precision
```

```
## [1] 0.890625
```

```r
recall <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
recall
```

```
## [1] 0.5229358
```

```r
F1_score <- (2*precision*recall)/(precision+recall)
F1_score
```

```
## [1] 0.6589595
```

KNN Model

```r
# Define N
N <- nrow(training_data)

# Define initial k
k_initial <- ceiling(sqrt(N))
k_initial
```

```
## [1] 53
```

```r
# Load library
library(class)
```

```r
# Ensure reproducibility of results
set.seed(1)

# Standardize X
standardized_X <- scale(data[, -which(colnames(data) == "Classification")])

# Split train test X standardized
X_train_std <- standardized_X[train_index, ]
X_val_std <- standardized_X[-train_index, ]

# Train and test KNN with K=k_initial
knn.pred <- knn(X_train_std, X_val_std, train.y, k = k_initial)
```

Confusion matrix:

```r
table(knn.pred, val.y)
```

```
##         val.y
## knn.pred   1   2
##        1 120 143
##        2 402 535
```

Validation error:

```r
val_error <- mean(knn.pred != val.y)
val_error
```

```
## [1] 0.4541667
```

```r
knn.pred_test <- knn(X_train_std, test.X, train.y, k = k_initial)

confusion_matrix <- table(knn.pred_test, test.y)
confusion_matrix
```

```
##              test.y
## knn.pred_test  1  2
##             1  2  3
##             2 50 61
```

Test error:

```
test_error <- mean(knn.pred_test != test.y)
test_error
```

```
## [1] 0.4568966
```

```
accuracy_score <- 1 - val_error
accuracy_score
```

```
## [1] 0.5458333
```

```
precision <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
precision
```

```
## [1] 0.953125
```

```
recall <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
recall
```

```
## [1] 0.5495495
```

```
F1_score <- (2*precision*recall)/(precision+recall)
F1_score
```

```
## [1] 0.6971429
```

LDA Model

```
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
lda.fit <- lda(Classification ~ ., data=training_data)

lda.fit
```

```
## Call:
## lda(Classification ~ ., data = training_data)
##
## Prior probabilities of groups:
##         1         2
```

```
## 0.4507143 0.5492857
##
## Group means:
##        Age      BMI  Glucose  Insulin     HOMA   Leptin Adiponectin Resistin
## 1 56.62044 27.48681 114.1426 8.881905 2.018529 25.43152    9.380178 12.79048
## 2 56.31079 27.35600 113.3505 8.679793 2.010116 25.21195    9.461455 13.13116
##      MCP.1
## 1 510.3891
## 2 513.6105
##
## Coefficients of linear discriminants:
##                      LD1
## Age         -0.011410056
## BMI         -0.086456034
## Glucose     -0.016065716
## Insulin     -0.062768974
## HOMA        -0.049015265
## Leptin      -0.012148581
## Adiponectin  0.049752176
## Resistin     0.084909832
## MCP.1        0.000599353
```

Plotting the linear discriminants:

```r
plot(lda.fit)
```

```r
# Predict output for test set
lda.pred <- predict(lda.fit, val.X)

# predict() returns the class, posterior probabilities and the linear discriminants
names(lda.pred)
```

```
## [1] "class"     "posterior" "x"
```

Confusion matrix:

```r
lda.class <- lda.pred$class

table(lda.class, val.y)
```

```
##          val.y
## lda.class   1   2
##         1   0   2
##         2 522 676
```

Validation error:

```r
val_error <- mean(lda.class != val.y)
val_error
```

```
## [1] 0.4366667
```

```r
# Predict output for test set
lda.pred_test <- predict(lda.fit, test.X)

lda.class_test <- lda.pred_test$class

confusion_matrix <- table(lda.class_test, test.y)
confusion_matrix
```

```
##               test.y
## lda.class_test  1  2
##              1  0  7
##              2 52 57
```

Test Error:

```r
test_error <- mean(lda.class_test != test.y)
test_error
```

```
## [1] 0.5086207
```

```r
accuracy_score <- 1 - val_error
accuracy_score
```

```
## [1] 0.5633333
```

```r
precision <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
precision
```

```
## [1] 0.890625
```

```r
recall <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
recall
```

```
## [1] 0.5229358
```

```r
F1_score <- (2*precision*recall)/(precision+recall)
F1_score
```

```
## [1] 0.6589595
```

QDA Model

```r
qda.fit <- qda(Classification ~ ., data=training_data)
qda.fit
```

```
## Call:
## qda(Classification ~ ., data = training_data)
##
## Prior probabilities of groups:
##         1         2
## 0.4507143 0.5492857
##
## Group means:
##        Age      BMI  Glucose  Insulin     HOMA   Leptin Adiponectin Resistin
## 1 56.62044 27.48681 114.1426 8.881905 2.018529 25.43152    9.380178 12.79048
## 2 56.31079 27.35600 113.3505 8.679793 2.010116 25.21195    9.461455 13.13116
##      MCP.1
## 1 510.3891
## 2 513.6105
```

Confusion matrix:

```r
qda.class <- predict(qda.fit, val.X)$class

table(qda.class, val.y)
```

```
##          val.y
## qda.class   1   2
##         1 100 129
##         2 422 549
```

Valdiation error:

```r
val_error <- mean(qda.class != val.y)
val_error
```

```
## [1] 0.4591667
```

```r
qda.class_test <- predict(qda.fit, test.X)$class

confusion_matrix <- table(qda.class_test, test.y)
confusion_matrix
```

```
##               test.y
## qda.class_test  1  2
##              1  7 16
##              2 45 48
```

Test error:

```r
test_error <- mean(qda.class_test != test.y)
test_error
```

```
## [1] 0.5258621
```

```r
accuracy_score <- 1 - val_error
accuracy_score
```

```
## [1] 0.5408333
```

```r
precision <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
precision
```

```
## [1] 0.75
```

```r
recall <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
recall
```

```
## [1] 0.516129
```

```r
F1_score <- (2*precision*recall)/(precision+recall)
F1_score
```

```
## [1] 0.611465
```

6. Experimentation

At this stage of the project, various experimentation will be conducted to optimize the baseline model. In addition, this optimized model is evaluated based on multiple metrics. This section covers the following: - Hyperparameter tuning - Feature importance - Confusion matrix - Cross-validation - Precision - Recall - F1 score - Classification report - ROC curve - Area under the curve (AUC)

Hyperparameter Tuning for Logistic Regression Model

```r
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

Define the parameter grid:

```r
param_grid <- expand.grid(
  alpha = seq(0, 1, by = 0.1),
  lambda = c(0.01, 0.1, 1)
)
```

```r
ctrl <- trainControl(method = "cv", number = 5)

glm_tuned <- train(
  Classification ~ .,
  data = training_data,
  method = "glmnet",
  tuneGrid = param_grid,
  family = "binomial",
  trControl = ctrl,
  verbose = FALSE
)

print(glm_tuned)
```

```
## glmnet
##
## 2800 samples
##    9 predictor
##    2 classes: '1', '2'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2240, 2240, 2240, 2240, 2240
## Resampling results across tuning parameters:
##
##   alpha  lambda  Accuracy   Kappa
##   0.0    0.01    0.5492857   0.000000000
##   0.0    0.10    0.5492857   0.000000000
##   0.0    1.00    0.5492857   0.000000000
##   0.1    0.01    0.5496429   0.001940389
##   0.1    0.10    0.5492857   0.000000000
##   0.1    1.00    0.5492857   0.000000000
##   0.2    0.01    0.5485714  -0.001122608
```

```
##   0.2    0.10    0.5492857   0.000000000
##   0.2    1.00    0.5492857   0.000000000
##   0.3    0.01    0.5492857   0.000000000
##   0.3    0.10    0.5492857   0.000000000
##   0.3    1.00    0.5492857   0.000000000
##   0.4    0.01    0.5492857   0.000000000
##   0.4    0.10    0.5492857   0.000000000
##   0.4    1.00    0.5492857   0.000000000
##   0.5    0.01    0.5492857   0.000000000
##   0.5    0.10    0.5492857   0.000000000
##   0.5    1.00    0.5492857   0.000000000
##   0.6    0.01    0.5492857   0.000000000
##   0.6    0.10    0.5492857   0.000000000
##   0.6    1.00    0.5492857   0.000000000
##   0.7    0.01    0.5492857   0.000000000
##   0.7    0.10    0.5492857   0.000000000
##   0.7    1.00    0.5492857   0.000000000
##   0.8    0.01    0.5492857   0.000000000
##   0.8    0.10    0.5492857   0.000000000
##   0.8    1.00    0.5492857   0.000000000
##   0.9    0.01    0.5492857   0.000000000
##   0.9    0.10    0.5492857   0.000000000
##   0.9    1.00    0.5492857   0.000000000
##   1.0    0.01    0.5492857   0.000000000
##   1.0    0.10    0.5492857   0.000000000
##   1.0    1.00    0.5492857   0.000000000
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.1 and lambda = 0.01.
```

```r
best_alpha <- glm_tuned$bestTune$alpha
best_lambda <- glm_tuned$bestTune$lambda

final_glm_model <- glmnet::glmnet(x = model.matrix(~ ., data = train.X)[, -1],
                                  y = as.factor(train.y),
                                  alpha = best_alpha, lambda = best_lambda, family = "binomial")
```

Make predictions on valdiation data and convert probabilities to class predictions:

```r
glm_probs <- predict(final_glm_model, newx = model.matrix(~ ., data = val.X)[, -1],
                     type = "response")

glm_pred <- ifelse(glm_probs > 0.5, 1, 0)
```

Evaluate the final model:

```r
confusion_matrix_glm <- table(glm_pred, val.y)
confusion_matrix_glm
```

```
##         val.y
## glm_pred   1   2
##        0   0   2
##        1 522 676
```

```r
validation_error <- mean(glm_pred != val.y)
validation_error
```

```
## [1] 0.565
```

```r
accuracy_score <- 1 - validation_error
accuracy_score
```

```
## [1] 0.435
```

```r
precision <- confusion_matrix_glm[2, 2] / sum(confusion_matrix_glm[, 2])
precision
```

```
## [1] 0.9970501
```

```r
recall <- confusion_matrix_glm[2, 2] / sum(confusion_matrix_glm[2, ])
recall
```

```
## [1] 0.5642738
```

```r
F1_score <- (2*precision*recall)/(precision+recall)
F1_score
```

```
## [1] 0.7206823
```

Make predictions on the test set.

```r
glm_probs_test <- predict(final_glm_model, newx = model.matrix(~ ., data = test.X)[, -1],
                      type = "response")

glm_pred_test <- ifelse(glm_probs_test > 0.5, 1, 0)
```

Evaluate the final model on the test set:

```r
confusion_matrix_glm_test <- table(glm_pred_test, test.y)
confusion_matrix_glm_test
```

```
##              test.y
## glm_pred_test  1  2
##             0  0  1
##             1 52 63
```

```r
validation_error_test <- mean(glm_pred_test != test.y)
validation_error_test
```

```
## [1] 0.5517241
```

```r
accuracy_score_test <- 1 - validation_error_test
accuracy_score_test
```

```
## [1] 0.4482759
```

```r
precision_test <- confusion_matrix_glm_test[2, 2] / sum(confusion_matrix_glm_test[, 2])
precision_test
```

```
## [1] 0.984375
```

```r
recall_test <- confusion_matrix_glm_test[2, 2] / sum(confusion_matrix_glm_test[2, ])
recall_test
```

```
## [1] 0.5478261
```

```r
F1_score_test <- (2*precision_test*recall_test)/(precision_test+recall_test)
F1_score_test
```

```
## [1] 0.7039106
```

Hyperparameter Tuning for KNN Model

Define the parameter grid:

```r
param_grid <- expand.grid(
  k = seq(1, 20, by = 1)
)
```

```r
ctrl <- trainControl(method = "cv", number = 5)

knn_tuned <- train(
  Classification ~ .,
  data = training_data,
  method = "knn",
  trControl = ctrl,
  tuneGrid = param_grid
)

print(knn_tuned)
```

```
## k-Nearest Neighbors
##
## 2800 samples
##    9 predictor
##    2 classes: '1', '2'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2240, 2240, 2239, 2241, 2240
## Resampling results across tuning parameters:
```

```
##
##   k   Accuracy   Kappa
##    1  0.5064241   0.001447358
##    2  0.5110644   0.011458282
##    3  0.5042813  -0.007529484
##    4  0.5064254  -0.002346290
##    5  0.4996320  -0.019051147
##    6  0.4971410  -0.022275439
##    7  0.5071493  -0.002628096
##    8  0.5035683  -0.008474682
##    9  0.5267909   0.036325799
##   10  0.5182137   0.018410897
##   11  0.5235689   0.028489149
##   12  0.5192889   0.018818458
##   13  0.5250071   0.027846722
##   14  0.5214222   0.020571473
##   15  0.5253610   0.023987570
##   16  0.5246455   0.023241965
##   17  0.5207137   0.011829939
##   18  0.5232118   0.016603147
##   19  0.5210791   0.010485084
##   20  0.5121505  -0.007891532
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

Get the best hyperparameters:

```r
best_k <- knn_tuned$bestTune$k

final_knn_model <- knn(
  train = model.matrix(~ ., data = training_data)[, -1],
  test = model.matrix(~ ., data = val_data)[, -1],
  cl = train.y,
  k = best_k
)
```

Make predictions on validation data and evaluate model:

```r
knn_pred <- final_knn_model

accuracy <- mean(knn_pred == val.y)
accuracy
```

```
## [1] 0.51
```

```r
confusion_matrix_knn <- table(knn_pred, val.y)
confusion_matrix_knn
```

```
##         val.y
## knn_pred   1   2
##        1 214 280
##        2 308 398
```

```r
validation_error <- mean(knn_pred != val.y)
validation_error
```

```
## [1] 0.49
```

```r
precision <- confusion_matrix_knn[2, 2] / sum(confusion_matrix_knn[, 2])
precision
```

```
## [1] 0.5870206
```

```r
recall <- confusion_matrix_knn[2, 2] / sum(confusion_matrix_knn[2, ])
recall
```

```
## [1] 0.5637394
```

```r
F1_score <- (2*precision*recall)/(precision+recall)
F1_score
```

```
## [1] 0.5751445
```

Evaluation on test set

```r
final_knn_model_test <- knn(
  train = model.matrix(~ ., data = training_data)[, -1],
  test = model.matrix(~ ., data = test_data)[, -1],
  cl = train.y,
  k = best_k
)
```

```r
knn_pred_test <- final_knn_model_test

accuracy_test <- mean(knn_pred_test == test.y)
accuracy_test
```

```
## [1] 0.5172414
```

```r
confusion_matrix_knn_test <- table(knn_pred_test, test.y)
confusion_matrix_knn_test
```

```
##               test.y
## knn_pred_test  1  2
##              1 19 23
##              2 33 41
```

```r
validation_error_test <- mean(knn_pred_test != test.y)
validation_error_test
```

```
## [1] 0.4827586
```

```r
precision_test <- confusion_matrix_knn_test[2, 2] / sum(confusion_matrix_knn_test[, 2])
precision_test
```

```
## [1] 0.640625
```

```r
recall_test <- confusion_matrix_knn_test[2, 2] / sum(confusion_matrix_knn_test[2, ])
recall_test
```

```
## [1] 0.5540541
```

```r
F1_score_test <- (2*precision_test*recall_test)/(precision_test+recall_test)
F1_score_test
```

```
## [1] 0.5942029
```

Hyperparameter Tuning for LDA Model

With LDA, there are typically no tuning parameters to optimize using hyperparameter tuning in the same way as some other models like KNN. LDA mainly relies on estimates of class means and covariance matrices. However, we can experiment with different values of the shrinkage parameter in LDA to see if it improves the performance of the model. Below we will train multiple LDA models with different values of the shrinkage parameter and evaluate their performance.

```r
shrinkage_values <- seq(0, 1, by = 0.1)
```

Train multiple LDA models with different shrinkage values:

```r
validation_errors <- numeric(length(shrinkage_values))
accuracies <- numeric(length(shrinkage_values))
recalls <- numeric(length(shrinkage_values))
precisions <- numeric(length(shrinkage_values))
F1_scores <- numeric(length(shrinkage_values))
confusion_matrices <- list()
```

```r
for (i in seq_along(shrinkage_values)) {
  shrinkage <- shrinkage_values[i]

  lda_fit <- lda(Classification ~ ., data = training_data, shrinkage = shrinkage)
  lda_pred <- predict(lda_fit, newdata = val.X)$class

  validation_errors[i] <- mean(lda_pred != val.y)
  accuracies[i] <- mean(lda_pred == val.y)

  confusion_matrix <- table(lda_pred, val.y)

  precision <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
  precisions[i] <- precision

  recall <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
  recalls[i] <- recall
```

```
  F1_score <- (2*precision*recall)/(precision+recall)
  F1_scores[i] <- F1_score

  confusion_matrices[[i]] <- confusion_matrix

}
```

Evaluate model at each shrinkage value:

```
for (i in seq_along(shrinkage_values)) {
  cat("Shrinkage value:", shrinkage_values[i], "\n")
  cat("Validation Error:", validation_errors[i], "\n")
  cat("Accuracy:", accuracies[i], "\n")
  cat("Recall:", recalls[i], "\n")
  cat("Precision:", precisions[i], "\n")
  cat("F1 Score:", F1_scores[i], "\n")
  cat("Confusion Matrix:\n")
  print(confusion_matrices[[i]])
}
```

```
## Shrinkage value: 0
## Validation Error: 0.4366667
## Accuracy: 0.5633333
## Recall: 0.5642738
## Precision: 0.9970501
## F1 Score: 0.7206823
## Confusion Matrix:
##         val.y
## lda_pred   1   2
##        1   0   2
##        2 522 676
## Shrinkage value: 0.1
## Validation Error: 0.4366667
## Accuracy: 0.5633333
## Recall: 0.5642738
## Precision: 0.9970501
## F1 Score: 0.7206823
## Confusion Matrix:
##         val.y
## lda_pred   1   2
##        1   0   2
##        2 522 676
## Shrinkage value: 0.2
## Validation Error: 0.4366667
## Accuracy: 0.5633333
## Recall: 0.5642738
## Precision: 0.9970501
## F1 Score: 0.7206823
## Confusion Matrix:
##         val.y
## lda_pred   1   2
##        1   0   2
##        2 522 676
```

```
## Shrinkage value: 0.3
## Validation Error: 0.4366667
## Accuracy: 0.5633333
## Recall: 0.5642738
## Precision: 0.9970501
## F1 Score: 0.7206823
## Confusion Matrix:
##         val.y
## lda_pred   1   2
##        1   0   2
##        2 522 676
## Shrinkage value: 0.4
## Validation Error: 0.4366667
## Accuracy: 0.5633333
## Recall: 0.5642738
## Precision: 0.9970501
## F1 Score: 0.7206823
## Confusion Matrix:
##         val.y
## lda_pred   1   2
##        1   0   2
##        2 522 676
## Shrinkage value: 0.5
## Validation Error: 0.4366667
## Accuracy: 0.5633333
## Recall: 0.5642738
## Precision: 0.9970501
## F1 Score: 0.7206823
## Confusion Matrix:
##         val.y
## lda_pred   1   2
##        1   0   2
##        2 522 676
## Shrinkage value: 0.6
## Validation Error: 0.4366667
## Accuracy: 0.5633333
## Recall: 0.5642738
## Precision: 0.9970501
## F1 Score: 0.7206823
## Confusion Matrix:
##         val.y
## lda_pred   1   2
##        1   0   2
##        2 522 676
## Shrinkage value: 0.7
## Validation Error: 0.4366667
## Accuracy: 0.5633333
## Recall: 0.5642738
## Precision: 0.9970501
## F1 Score: 0.7206823
## Confusion Matrix:
##         val.y
## lda_pred   1   2
##        1   0   2
```

```
##        2 522 676
## Shrinkage value: 0.8
## Validation Error: 0.4366667
## Accuracy: 0.5633333
## Recall: 0.5642738
## Precision: 0.9970501
## F1 Score: 0.7206823
## Confusion Matrix:
##         val.y
## lda_pred   1   2
##        1   0   2
##        2 522 676
## Shrinkage value: 0.9
## Validation Error: 0.4366667
## Accuracy: 0.5633333
## Recall: 0.5642738
## Precision: 0.9970501
## F1 Score: 0.7206823
## Confusion Matrix:
##         val.y
## lda_pred   1   2
##        1   0   2
##        2 522 676
## Shrinkage value: 1
## Validation Error: 0.4366667
## Accuracy: 0.5633333
## Recall: 0.5642738
## Precision: 0.9970501
## F1 Score: 0.7206823
## Confusion Matrix:
##         val.y
## lda_pred   1   2
##        1   0   2
##        2 522 676
```

The varying shrinkage valeus did not affect the resulting validation error, accuracy, recall, or precision. Therefore, all the LDA models have a similar performance.

Train multiple LDA models with different shrinkage values:

```r
validation_errors_test <- numeric(length(shrinkage_values))
accuracies_test <- numeric(length(shrinkage_values))
recalls_test <- numeric(length(shrinkage_values))
precisions_test <- numeric(length(shrinkage_values))
F1_scores_test <- numeric(length(shrinkage_values))
confusion_matrices_test <- list()
```

```r
for (i in seq_along(shrinkage_values)) {
  shrinkage <- shrinkage_values[i]

  lda_fit <- lda(Classification ~ ., data = training_data, shrinkage = shrinkage)
  lda_pred <- predict(lda_fit, newdata = test.X)$class

  validation_errors_test[i] <- mean(lda_pred != test.y)
```

```r
  accuracies_test[i] <- mean(lda_pred == test.y)

  confusion_matrix <- table(lda_pred, test.y)

  precision <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
  precisions_test[i] <- precision

  recall <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
  recalls_test[i] <- recall

  F1_score <- (2*precision*recall)/(precision+recall)
  F1_scores_test[i] <- F1_score

  confusion_matrices_test[[i]] <- confusion_matrix

}
```

```r
for (i in seq_along(shrinkage_values)) {
  cat("Shrinkage value:", shrinkage_values[i], "\n")
  cat("Validation Error:", validation_errors_test[i], "\n")
  cat("Accuracy:", accuracies_test[i], "\n")
  cat("Recall:", recalls_test[i], "\n")
  cat("Precision:", precisions_test[i], "\n")
  cat("F1 Score:", F1_scores_test[i], "\n")
  cat("Confusion Matrix:\n")
  print(confusion_matrices_test[[i]])
}
```

```
## Shrinkage value: 0
## Validation Error: 0.5086207
## Accuracy: 0.4913793
## Recall: 0.5229358
## Precision: 0.890625
## F1 Score: 0.6589595
## Confusion Matrix:
##         test.y
## lda_pred  1  2
##        1  0  7
##        2 52 57
## Shrinkage value: 0.1
## Validation Error: 0.5086207
## Accuracy: 0.4913793
## Recall: 0.5229358
## Precision: 0.890625
## F1 Score: 0.6589595
## Confusion Matrix:
##         test.y
## lda_pred  1  2
##        1  0  7
##        2 52 57
## Shrinkage value: 0.2
## Validation Error: 0.5086207
## Accuracy: 0.4913793
```

```
## Recall: 0.5229358
## Precision: 0.890625
## F1 Score: 0.6589595
## Confusion Matrix:
##         test.y
## lda_pred  1  2
##        1  0  7
##        2 52 57
## Shrinkage value: 0.3
## Validation Error: 0.5086207
## Accuracy: 0.4913793
## Recall: 0.5229358
## Precision: 0.890625
## F1 Score: 0.6589595
## Confusion Matrix:
##         test.y
## lda_pred  1  2
##        1  0  7
##        2 52 57
## Shrinkage value: 0.4
## Validation Error: 0.5086207
## Accuracy: 0.4913793
## Recall: 0.5229358
## Precision: 0.890625
## F1 Score: 0.6589595
## Confusion Matrix:
##         test.y
## lda_pred  1  2
##        1  0  7
##        2 52 57
## Shrinkage value: 0.5
## Validation Error: 0.5086207
## Accuracy: 0.4913793
## Recall: 0.5229358
## Precision: 0.890625
## F1 Score: 0.6589595
## Confusion Matrix:
##         test.y
## lda_pred  1  2
##        1  0  7
##        2 52 57
## Shrinkage value: 0.6
## Validation Error: 0.5086207
## Accuracy: 0.4913793
## Recall: 0.5229358
## Precision: 0.890625
## F1 Score: 0.6589595
## Confusion Matrix:
##         test.y
## lda_pred  1  2
##        1  0  7
##        2 52 57
## Shrinkage value: 0.7
## Validation Error: 0.5086207
```

```
## Accuracy: 0.4913793
## Recall: 0.5229358
## Precision: 0.890625
## F1 Score: 0.6589595
## Confusion Matrix:
##         test.y
## lda_pred  1  2
##        1  0  7
##        2 52 57
## Shrinkage value: 0.8
## Validation Error: 0.5086207
## Accuracy: 0.4913793
## Recall: 0.5229358
## Precision: 0.890625
## F1 Score: 0.6589595
## Confusion Matrix:
##         test.y
## lda_pred  1  2
##        1  0  7
##        2 52 57
## Shrinkage value: 0.9
## Validation Error: 0.5086207
## Accuracy: 0.4913793
## Recall: 0.5229358
## Precision: 0.890625
## F1 Score: 0.6589595
## Confusion Matrix:
##         test.y
## lda_pred  1  2
##        1  0  7
##        2 52 57
## Shrinkage value: 1
## Validation Error: 0.5086207
## Accuracy: 0.4913793
## Recall: 0.5229358
## Precision: 0.890625
## F1 Score: 0.6589595
## Confusion Matrix:
##         test.y
## lda_pred  1  2
##        1  0  7
##        2 52 57
```

Hyperparameter Tuning for QDA Model

Similar to LDA, QDA does not have the usual hyperparameters to tune in the same way as other models such as KNN. But, we can perform CV to evaluate the model's performance.

```r
ctrl_qda <- trainControl(method = "cv", number = 5)
```

```r
qda_cv <- train(
  Classification ~ .,
  data = training_data,
  method = "qda",
```

```r
  trControl = ctrl_qda
)

print(qda_cv)
```

```
## Quadratic Discriminant Analysis
##
## 2800 samples
##    9 predictor
##    2 classes: '1', '2'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2240, 2240, 2240, 2240, 2240
## Resampling results:
##
##   Accuracy   Kappa
##   0.5078571  -0.0382788
```

```r
qda_pred <- predict(qda_cv, newdata = val.X)

confusion_matrix_qda <- table(qda_pred, val.y)
confusion_matrix_qda
```

```
##         val.y
## qda_pred   1   2
##        1 100 129
##        2 422 549
```

```r
accuracy <- mean(qda_pred == val.y)
accuracy
```

```
## [1] 0.5408333
```

```r
validation_error <- mean(qda_pred != val.y)
validation_error
```

```
## [1] 0.4591667
```

```r
precision <- confusion_matrix_qda[2, 2] / sum(confusion_matrix_qda[, 2])
precision
```

```
## [1] 0.8097345
```

```r
recall <- confusion_matrix_qda[2, 2] / sum(confusion_matrix_qda[2, ])
recall
```

```
## [1] 0.5653965
```

```r
F1_score <- (2*precision*recall)/(precision+recall)
F1_score
```

```
## [1] 0.6658581
```

Evaluation on test set

```r
qda_pred_test <- predict(qda_cv, newdata = test.X)

confusion_matrix_qda_test <- table(qda_pred_test, test.y)
confusion_matrix_qda_test
```

```
##              test.y
## qda_pred_test  1  2
##             1  7 16
##             2 45 48
```

```r
accuracy_test <- mean(qda_pred_test == test.y)
accuracy_test
```

```
## [1] 0.4741379
```

```r
validation_error_test <- mean(qda_pred_test != test.y)
validation_error_test
```

```
## [1] 0.5258621
```

```r
precision_test <- confusion_matrix_qda_test[2, 2] / sum(confusion_matrix_qda_test[, 2])
precision_test
```

```
## [1] 0.75
```

```r
recall_test <- confusion_matrix_qda_test[2, 2] / sum(confusion_matrix_qda_test[2, ])
recall_test
```

```
## [1] 0.516129
```

```r
F1_score_test <- (2*precision_test*recall_test)/(precision_test+recall_test)
F1_score_test
```

```
## [1] 0.611465
```

Experimenting on Original Dataset for Both Train and Test

```r
set.seed(1)

train_index1 <- sample(1:nrow(org_data), 0.8 * nrow(org_data))
training_data1 <- org_data[train_index, ]
test_data1 <- org_data[-train_index, ]
```

```r
# Ensure reproducibility of results
set.seed(1)

# cbind(): column bind
train.X1 <- training_data1[, -which(colnames(org_data) == "Classification")]
test.X1 <- test_data1[, -which(colnames(org_data) == "Classification")]
train.y1 <- training_data1$Classification
test.y1 <- test_data1$Classification
```

```r
glm.model1 <- glm(Classification ~ ., data = training_data1, family = binomial)

summary(glm.model1)
```

Logistic Regression

```
##
## Call:
## glm(formula = Classification ~ ., family = binomial, data = training_data1)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -8.9286394  4.2585370  -2.097   0.0360 *
## Age         -0.0152227  0.0178949  -0.851   0.3949
## BMI         -0.0682257  0.0812608  -0.840   0.4011
## Glucose      0.1120642  0.0414024   2.707   0.0068 **
## Insulin      0.2497516  0.2930546   0.852   0.3941
## HOMA        -0.7520302  1.2261410  -0.613   0.5397
## Leptin      -0.0206268  0.0216091  -0.955   0.3398
## Adiponectin  0.0200163  0.0426998   0.469   0.6392
## Resistin     0.0543487  0.0303062   1.793   0.0729 .
## MCP.1        0.0006809  0.0008491   0.802   0.4226
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 117.541  on 85  degrees of freedom
## Residual deviance:  82.591  on 76  degrees of freedom
##   (2714 observations deleted due to missingness)
## AIC: 102.59
##
## Number of Fisher Scoring iterations: 7
```

```r
# Predict on validation data
glm.probs1 <- predict(glm.model1, test.X1, type = "response")

head(glm.probs1)
```

```
##          1          2          3          4          5          6
## 0.08607501 0.35854447 0.91359034 0.42893696 0.37652292 0.33725995
```

```r
# Get performance measures
num_test1 <- length(glm.probs1)
glm.pred1 <- rep(1, num_test1)
glm.pred1[glm.probs1 > 0.5] <- 2

# Confusion matrix
conf_mat_glm_org <- table(glm.pred1, test.y1)

conf_mat_glm_org
```

```
##          test.y1
## glm.pred1  1  2
##         1 12  3
##         2  3 12
```

```r
# Validation error
mean(glm.pred1 != test.y1)
```

```
## [1] 0.2
```

```r
# Accuracy score
1 - mean(glm.pred1 != test.y1)
```

```
## [1] 0.8
```

```r
conf_mat_glm_org[2, 2] / sum(conf_mat_glm_org[, 2])
```

```
## [1] 0.8
```

```r
conf_mat_glm_org[2, 2] / sum(conf_mat_glm_org[2, ])
```

```
## [1] 0.8
```

```r
(2*table(glm.pred1, test.y1)[2, 2] / sum(table(glm.pred1, test.y1)[, 2])*table(glm.pred1, test.y1)[2, 2
```

```
## [1] 0.8
```