Valérie BLANCH

2614867B

Discriminating Between Healthy Individuals and Patients with Pulmonary Arterial Hypertension

1. INTRODUCTION

This project's objective is to detect individuals with Pulmonary Arterial Hypertension (PAH) from gene expressions. Several predictive models will be fitted, to determine which genes have the most influence on the disease.

The data set used contains 1000 gene expressions belonging to 20 individuals. Through observation, each individual has been marked as "healthy" or "hypertensive".
The gene expressions are identified by numbers, ranging from 601 to 1600.

The dimensions of the original dataset are 1000 rows by 22 columns, the first two columns are meant to identify the individuals (and will be removed for this analysis).
The following study assumes genetics are indeed correlated to PAH, among other causes.

2. EXPLORATORY ANALYSIS

To be able to predict hypertension in individuals, the gene expressions must be formatted as variables, and thus the data set has been transposed to obtain the following:

- 20 rows corresponding to the 20 observations.
- 1000 columns corresponding to the gene expressions.

The original dataset, called *genes*, contains no missing values.

```
> anyNA(genes)
[1] FALSE
```

The two columns corresponding to the observations' IDs have been removed.

A new column has been added to record which patient is healthy (0) or hypertensive (1). As a result, we obtain the following dimensions:

```
> dim(genes)
[1]   20 1001
```

Thus, the data set created has a high dimensionality and very few observations: consequently, it has been decided to split the data into two sets of equal size (10 observations each), to be able to compute predictions and performance metrics without too much variability.

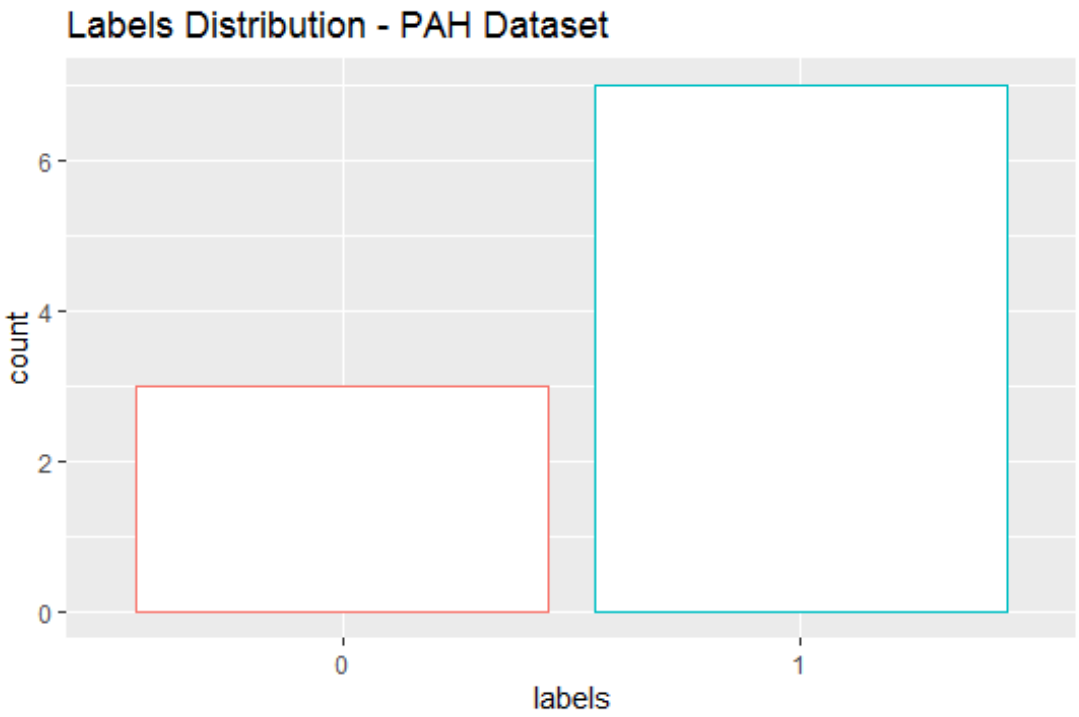
Cross-validation techniques like LOOCV have been used when possible.

Training set exploration:

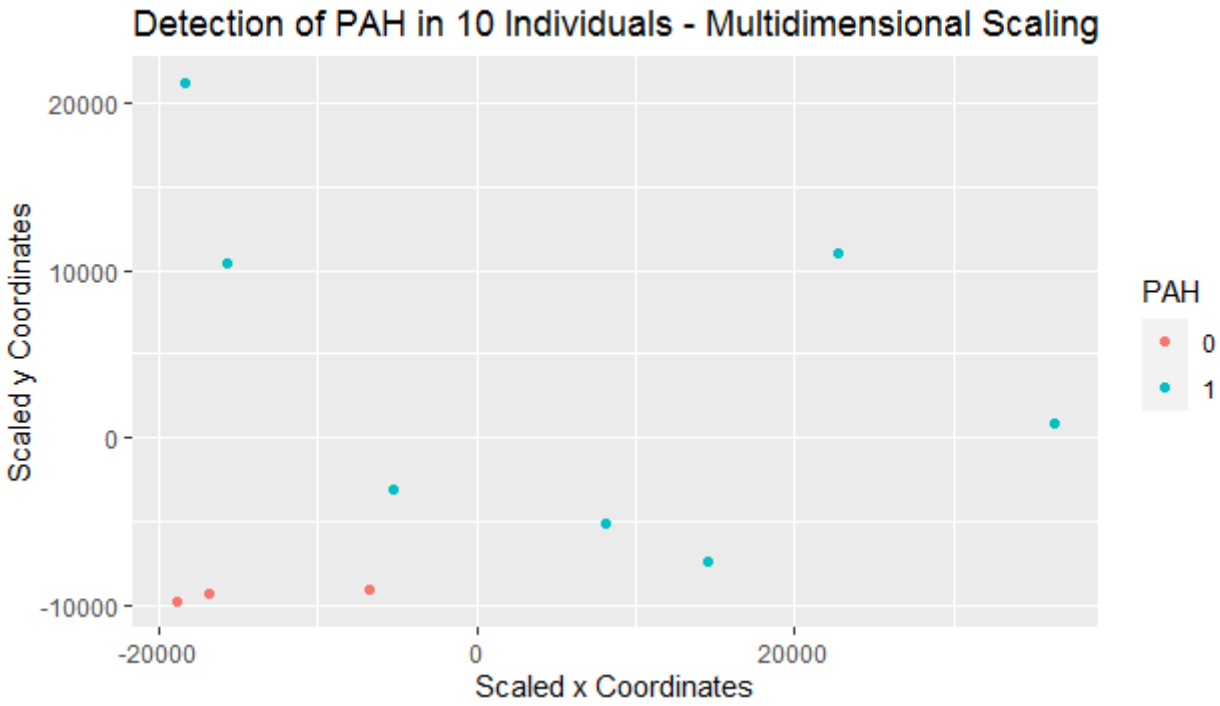
The training set is made of 10 rows and 1001 variables, the last one contains the labels "0" and "1", to recognise patients with or without PAH. Here are the dimensions of *train.set:*

```
> dim(train.set)
[1]   10 1001
```

The distribution of the response variable has been represented on a bar plot, which shows that in the training set 7 individuals are hypertensive and 3 are not. This variable has been formatted as a 2-levels factor in R.



To get a better overview of the predictors, and their relationship with each other and the response variable, a Multidimensional Scaling plot has been computed.



To draw this plot, a distance matrix then scaled coordinates have been calculated; on the plot, the individuals with PAH and those without seem to form two separated clusters.

Multicollinearity:

```
> highly.correlated <- findCorrelation(cor(train.set[,1:1000]), cutoff=0.75)

> length(highly.correlated)
[1] 956
```

Multicollinearity has been tested on the training set, and 956 predictors are highly correlated with one another. This might be an issue for modelling, as redundant information can make models instable. Furthermore, many of these variables may be irrelevant in predicting the target in the first place.

To simplify the dataset, the rfe() function from the *caret* package implements a random forest model that filters correlated and/or irrelevant features.

```
> rfe.pah <- rfe(train.set[,1:1000], train.set[,1001], sizes=10:50,
+                rfeControl=rfeCon .... [TRUNCATED]

> predictors(rfe.pah)
 [1] "X1464" "X1589" "X1541" "X1543" "X1547" "X1533" "X874"  "X1088" "X1330"
[10] "X1310" "X1223" "X1160" "X1319" "X1429" "X1595" "X1360" "X732"  "X1237"
[19] "X1238" "X1459" "X995"  "X1535" "X1116" "X1386" "X1233" "X895"  "X1513"
[28] "X1582" "X737"  "X1266" "X1471" "X898"  "X1154" "X1272"

> length(predictors(rfe.pah))
[1] 34
```

Using LOOCV and several subsets of variables (10 to 50), the training and test sets have been updated to contain the 34 genes printed above.

3. RESULTS

Three models have been trained for this analysis:

- k-Nearest Neighbours (R package *caret*)
- Random Forest (R package *caret*)
- Support-Vector Machine (R package *e1071*)

k- Nearest Neighbours:

The kNN algorithm calculates Euclidian distance between observations and classifies them according to a number *k* of nearest points.

```
> knn.model <- train(labels ~., data = train.set, method = "knn",
+                    trControl=trainControl('LOOCV'),
+  .... [TRUNCATED]

> knn.model
k-Nearest Neighbors

10 samples
34 predictors
 2 classes: '0', '1'

No pre-processing
Resampling: Leave-One-Out Cross-Validation
Summary of sample sizes: 9, 9, 9, 9, 9, 9, ...
Resampling results across tuning parameters:

  k  Accuracy  Kappa
  3  0.9        0.7826087
  5  0.6       -0.1764706

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 3.
```

Using the train() function from the *caret* package, a kNN model has been fitted, with several arguments: a Leave-One-Out Cross-Validation option to compensate for the sample size (by repeatedly training the model on 9 observations and testing on 1), and two different values for the parameter *k,* for fine-tuning. 3 and 5 are odd numbers, to avoid ties with binary outcomes. By using accuracy as a metric, the algorithm has determined that k=3 was an optimum.

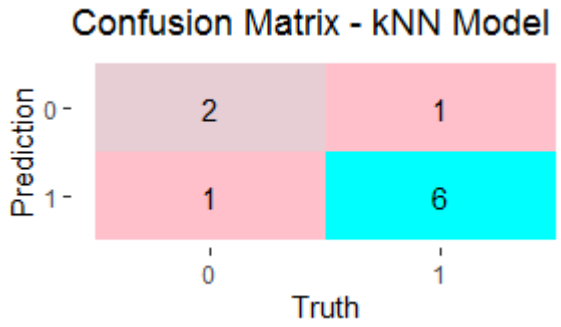
The table below classifies the variables by importance; for kNN models, the varImp() function computes a ROC curve analysis on each predictor. The scores are scaled from 0 to 100.

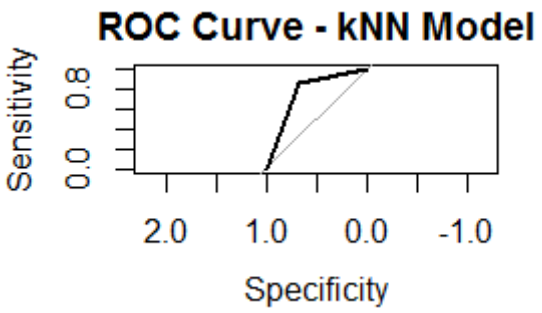
```
> knn_features
    X1464 X1589 X1541 X1543 X1547 X1533 X874 X1330 X1310 X1223 X1160 X1319 X1429
ROC   100   100   100   100   100   100  100   100   100   100   100   100   100
    X1595 X1360 X1237 X1238 X1459 X1535 X1116 X1386 X1233 X1582 X737 X732 X1266
ROC   100   100   100   100   100   100   100   100   100   100  100   75    75
    X1088 X895 X1272 X995 X1154 X1471 X898 X1513
ROC    50   50    50   25    25    25   25     0
```

As for predictions, the kNN model has predicted the 10 observations of the test set with an accuracy of 0.8 ((true positives + true negatives) / total), a sensitivity of 0.86 (true positives / (true positives + false negatives)), and a specificity of 0.67 (true negatives / (true negatives + false positives)).

```
> confusionMatrix(pred.knn, test.set$labels, positive='1')
Confusion Matrix and Statistics

          Reference
Prediction 0 1
         0 2 1
         1 1 6

               Accuracy : 0.8
                 95% CI : (0.4439, 0.9748)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : 0.3828

                  Kappa : 0.5238

 Mcnemar's Test P-Value : 1.0000

            Sensitivity : 0.8571
            Specificity : 0.6667
         Pos Pred Value : 0.8571
         Neg Pred Value : 0.6667
             Prevalence : 0.7000
         Detection Rate : 0.6000
   Detection Prevalence : 0.7000
      Balanced Accuracy : 0.7619

       'Positive' Class : 1
```

Those results can be visualised on a confusion matrix and a ROC curve (packages *yardstick* and *pROC*). The model has made 1 type I error (false positive) and 1 type II error (false negative).





Random Forest:

The Random Forest algorithm builds a chosen number of decision trees on different subsets of the training set and averages their results.

```
> rf.model
Random Forest

10 samples
34 predictors
 2 classes: '0', '1'

No pre-processing
Resampling: Leave-One-Out Cross-Validation
Summary of sample sizes: 9, 9, 9, 9, 9, 9, ...
Resampling results across tuning parameters:

  mtry  Accuracy  Kappa
   2    1         1
  18    1         1
  34    1         1

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 2.
```

By default, 500 trees have been built, and the *train* function has determined that mtry=2 was an optimum: this value is the number of variables selected at each split.

```
> varImp(rf.model)
rf variable importance

  only 20 most important variables shown (out of 34)

      Importance
X1223     100.00
X1360      93.47
X1116      92.70
X1459      88.73
X1330      88.45
X1310      87.39
X1547      86.28
X1541      84.02
X1237      83.61
X874       82.78
X1589      81.20
X1429      78.28
X1464      78.11
X1233      76.50
X737       72.55
X1160      69.85
X1543      67.87
X1582      67.52
X1533      67.40
X1319      66.99
```

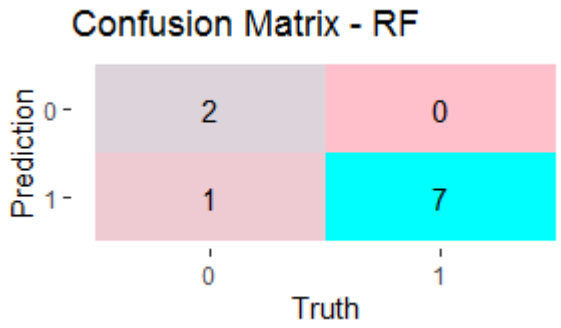
Using the varImp() function again, the most important features have been printed, and genes 1223, 1360, 1116 dominate the list, as they did with kNN.

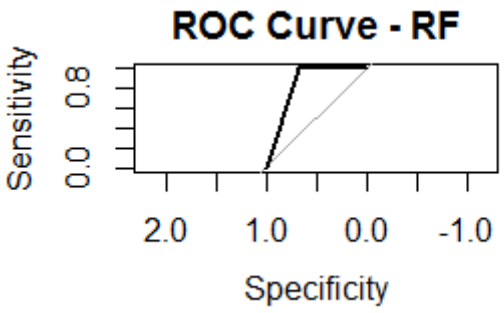
The random forest model provided more accurate predictions than the kNN model, as shown below:

```
> confusionMatrix(pred.rf, test.set$labels, positive='1')
Confusion Matrix and Statistics

          Reference
Prediction 0 1
         0 2 0
         1 1 7

               Accuracy : 0.9
                 95% CI : (0.555, 0.9975)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : 0.1493

                  Kappa : 0.7368

 Mcnemar's Test P-Value : 1.0000

            Sensitivity : 1.0000
            Specificity : 0.6667
         Pos Pred Value : 0.8750
         Neg Pred Value : 1.0000
             Prevalence : 0.7000
         Detection Rate : 0.7000
   Detection Prevalence : 0.8000
      Balanced Accuracy : 0.8333

       'Positive' Class : 1
```

With an accuracy of 0.9, a sensitivity of 1 and a specificity of 0.67, the model has made only one type II error.

Therefore, its AUC is larger than the kNN model's, as shown on the ROC curve graph:





Support-Vector Machine:

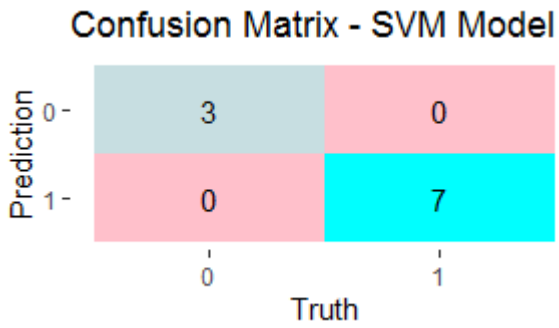
The objective of the SVM algorithm is to find the optimal decision boundary that separates the two classes, in other words to compute a hyperplane with minimized margins and loss.

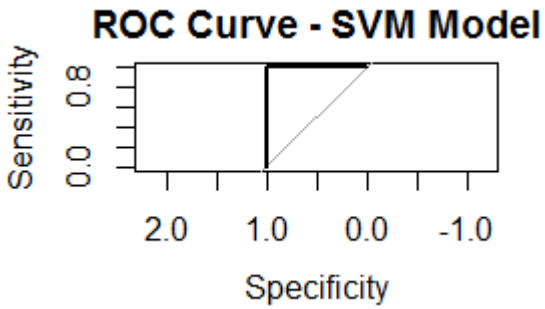
```
Call:
svm(formula = labels ~ ., data = train.set, type = "C-classification")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  10
```

By default, the algorithm has set the cost metric to 1 and the kernel to “radial”.

The SVM model has not made any mistake during the testing process. The AUC, accuracy, sensitivity, and specificity are thus all equal to 1.

```
> confusionMatrix(pred.svm, test.set$labels, positive='1')
Confusion Matrix and Statistics

          Reference
Prediction 0 1
         0 3 0
         1 0 7

               Accuracy : 1
                 95% CI : (0.6915, 1)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : 0.02825

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0
            Specificity : 1.0
         Pos Pred Value : 1.0
         Neg Pred Value : 1.0
             Prevalence : 0.7
         Detection Rate : 0.7
   Detection Prevalence : 0.7
      Balanced Accuracy : 1.0

       'Positive' Class : 1
```





4. DISCUSSION

The results of the three models have been summarised in a table for comparison.

	Accuracy	Specificity	Sensitivity
k-Nearest Neighbours	0.8	0.67	0.86
Random Forest	0.9	0.67	1
Support-Vector Machine	1	1	1

Therefore, using accuracy as a determining factor, the SVM is the “best” model for prediction.

However, the sample size was a major issue during training and testing: the models might be overfitted. A larger sample size would provide consistent models and the ability to test them thoroughly. A validation set could be added for fine-tuning parameters.