Data Analysis Project

Jacob Passfield

The following report analyses two different data sets: the first being biomedical data and the latter being DNA data. Each data set is analysed individually and independently.

Data Set 1: Biomedical Data

Introduction

Correctly classifying whether a patient carries a rare genetic disorder or not is paramount for the patient's well-being and livelihood. It also ensures the cost of supporting a carrier, treatment for example, is not wasted on a healthy individual.

The data set used in this analysis arose from a study to develop screening methods where four measurements (m1, m2, m3 and m4) were taken on a blood sample. The current industry standard is to only use m1 to identify carriers of the disease. Thus the aim of this analysis is to develop a new screening procedure to detect carriers and to describe its effectiveness. The analysis assesses whether using another measurement or combining measurements would be most effective at identifying carriers. The methods used to do this are through linear discriminant analysis and neural networks, both with leave-one-out cross validation.

In addition, this analysis addresses the concerns that experts have regarding whether younger people tend to have higher measurements and the concerns that the laboratory has regarding whether there may be a systematic drift over time in their measurement process.

Data and methods

The original biomedical data is contained in two separate files, one file for carriers of the disease and the other file for "normals", those that are not carriers. The two files consist of 67 and 127 samples respectively. I combined these files together to form one data set. This data set has 7 columns. The first column, called class, contains a label of 0 or 1, 1 if the patient is a carrier and 0 if the patient is not. The second column contains the age of the patient, the third contains the date the blood sample was taken (yyyy-mm-dd) and the fourth to seventh columns contain the four measurements m1, m2, m3 and m4, respectively. The date column was processed to display the date in the correct format. The observations in the data set are arranged by class, age and date.

Three methods of classification are considered in the analysis, linear discriminant analysis (LDA), artificial neural networks (ANN) and naïve Bayes, and only the best method found is presented in this analysis. For further details of such methods, refer to *Appendix A*. Note that naïve Bayes was not found to be an optimal classifier and so its details are omitted from this report. Each classifier is trained using leave-one-out cross validation. Cross validation holds some of the observations in the data in order to test the classifier. LOOCV holds only one observation to test in each round until all observations are tested. Note that k-fold cross validation was considered but with the data set being relatively small LOOCV was found more appropriate.

To describe the effectiveness of different screening procedures, the data set is split into training and test sets. This is because classifiers have a problem with overfitting, where a model is effective at classifying the observations used to build said model but fails to classify new and unseen data with

the same effectiveness. The training set includes 136 samples and the test set includes 58 samples, a 70/30 split. Now since the test set informs us whether the trained classifier generalises to new data well, only the results of the test sets are presented in this report.

Results

Since the current industry standard is to only use m1 to identify carriers of the disease, I started with finding which classifier led to the best test accuracy using m1 only. A neural network with one hidden node, a decay of zero and with the data centered and scaled led to the best test accuracy. *Tables 1.1* and *1.2* show the confusion matrices and their corresponding accuracy obtained using the neural network classifier on the training and test data set respectively.

ANN on m1	**PREDICTED CLASS**	
TRUE CLASS	**0**	**1**
0	86	3
1	18	29
ACCURACY		0.8455882

Table 1.1 *Confusion matrix of m1 obtained from the neural network classifier with one hidden node and a decay of zero using training data that was centered and scaled. The data was scaled and centered.*

ANN on m1	**PREDICTED CLASS**	
TRUE CLASS	**0**	**1**
0	36	2
1	9	11
ACCURACY		0.8103448

Table 1.2 *Confusion matrix of m1 obtained from the neural network classifier with one hidden node and a decay of zero using test data. The data was scaled and centered.*

The confusion matrix obtained from using the training data is included here to illustrate the difference between the training accuracy and test accuracy. This reiterates the importance of splitting the data into training and test sets to assess how well the model classifies unseen data. With the training data at an accuracy of 84.55% and the test accuracy at 81.03% the model classifies unseen data relatively well.

With the aim of developing a new screening procedure to detect carriers and its corresponding effectiveness, I created a biplot to identify whether any other measurements may be useful in classifying between a carrier and non-carrier of the disease. *Figure 1* shows the biplot obtained from principal component analysis performed on the biomedical data set without scaling.

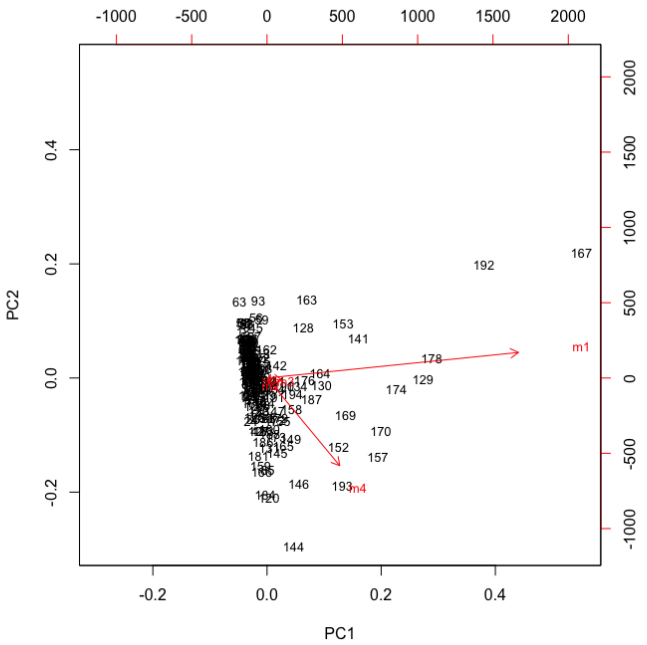

Figure 1.1 *Biplot obtained from principal component analysis performed on the biomedical data set without scaling.*

In principal component analysis, the principal components are ordered by how much variation of a data set they explain. The first principal component accounts for most of the variance in a data set, the second accounts for the second most variance and so on. A biplot can reveal which variables are the biggest causes of variation within a data set and so which variables influence the principal components.

Figure 1.1 clearly shows that m1 dominates the analysis with a large and positive loading (the red line for m1 is long and points in a positive direction) and that it heavily influences the first principal component (the red line being somewhat horizontal). Since *Figure 1.1* reveals that m1 accounts for most variation across the first principal component, and so the data, this must be why m1 is used as a current industry standard.

The scores corresponding to the observations 167 and 192 could be potential outliers causing the great variation for m1. These observations are carriers of the disease, removing them may be detrimental to the analysis and so I decided to leave observations alone.

Figure 1.1 also shows that m4 is significant, whilst m2 and m3 are not. It may be useful to test the effectiveness of m4 but it would not be helpful to test m2 and m3. With the loading of m4 revealing that m4 is most responsible for the variance along the second principal component (the red line being somewhat vertical), I tested the effectiveness of m4 at classifying the data.

Using m4 only, linear discriminant analysis led to the best test accuracy. *Table 1.3* shows the confusion matrix and the corresponding accuracy obtained using LDA classification on the test data set.

LDA on m4	**PREDICTED CLASS**	
TRUE CLASS	**0**	**1**
0	32	6
1	8	12
ACCURACY		0.7586207

Table 4 *Confusion matrix of m4 obtained from LDA using test data.*

Table 1.3 shows that the test accuracy using LDA with m4 only is 75.86%. This is much lower than the test accuracy of m1 at 81.03%, thus m4 should not be used instead of m1 to identify carriers of the disease, it is not more effective.

The lower test accuracy is not a surprise since *Figure 1.1* shows that m1 accounts for more variation than m4. With the loadings of m1 and m4 at almost a right angle, both measurements are not correlated and so combining these measurements may lead to better classification since they explain different variations within the data.

Combining m1 and m4, a neural network with one hidden and a decay of zero and with the data centered and scaled led to the best test accuracy. *Table 1.4* shows the confusion matrix and corresponding accuracy obtained using the neural network model on the test data set.

ANN on m1 and m4	**PREDICTED CLASS**	
TRUE CLASS	**0**	**1**
0	36	2
1	9	11
ACCURACY		0.8275862

Table 1.4 *Confusion matrix obtained using m1 and m4 in a neural network classifier with one hidden node and a decay of zero using test data. The data was scaled and centered.*

The test accuracy of 82.76% using m1 and m4 is higher than the test accuracy of 81.03% just using m1, it is more effective. This is expected because m1 and m4 influence PC1 and PC2 the most respectively so using both includes the different variations they account for.

Since combining m1 and m4 led to better accuracy, I decided to use all measurements to classify the data. I found that using a neural network with one hidden node and a 0.0001 decay with the data scaled and centered led to the best test accuracy, the resulting confusion matrix and accuracy shown in *Table 1.5*.

ANN on all m	**PREDICTED CLASS**	
TRUE CLASS	**0**	**1**
0	36	2
1	9	11
ACCURACY		0.862069

Table 1.5 *Confusion matrix obtained using all the measurements in a neural network classifier with one hidden node and a decay of zero using test data that was scaled and centered. The data was scaled and centered.*

The test accuracy using all the measurements is 86.21%, higher than using m1 and m4 (82.26%) and using m1 only (81.03%). Again this is no surprise, using all the measurements in the data set is bound to lead to a higher test accuracy as all information available and variation within measurements is accounted for. All the test accuracies obtained in this analysis are summarised in *Table 1.6* showing that using m1, m2, m3 and m4 is the most effective screening procedure, whilst using m4 only is the worst.

Measurement(s) used in classification	**Test accuracy in %**
m1	81.03
m4	75.86
m1 and m4	82.76
m1, m2, m3 and m4	86.21

Table 1.6 *A summary of the variables used in each classification tried in this analysis and the corresponding test accuracy.*

Figure 1.2 is a scatter plot to identify whether young people tend to have higher measurements. The red dots correspond to the mean measurement value of each age recorded in the data set.

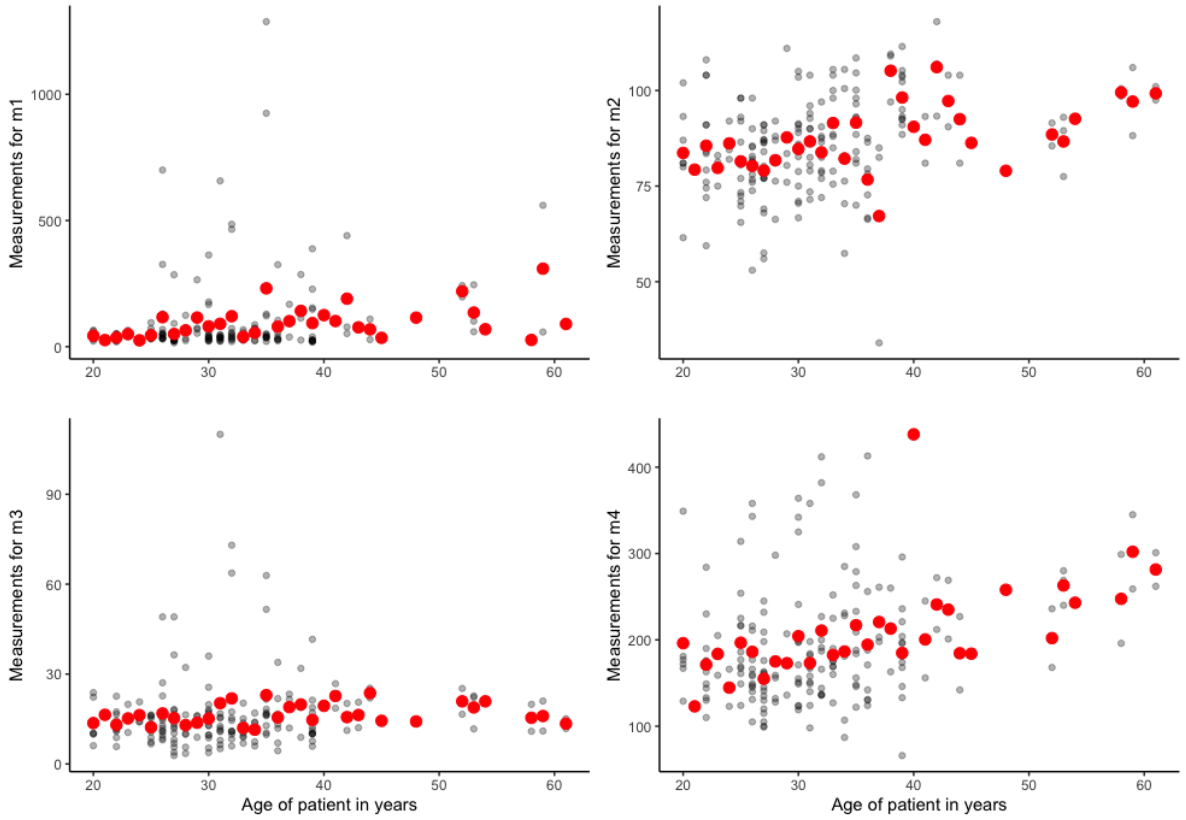

Figure 2 *A scatter plot of the age of every patient recorded in the data set against their measurement value. Black dots correspond to the raw measurement values. Darker grey dots indicate many patients of the same age have the same measurement value. The red dots are the mean measurement value for each age recorded.*
1.2(a) *top left - m1.* ***1.2(b)*** *top right - m2.* ***1.2(c)*** *bottom left - m3.* ***1.2(d)*** *bottom right - m4.*

Figure 1.2 disagrees with the experts noting that young people tend to have higher measurements. For example, with the red dots increasing along the horizontal axis in *1.2(d)* this clearly shows that older people have higher measurements of m4, not young people. This pattern is repeated in *1.2(a)* and *1.2(b)* too, the red dots are lower to the left of the horizontal axis than to the right. In *1.2(c)* the value between the old and young are similar. Now this may be because more blood samples of young people were recorded than older people. In *1.2(b)* grey dots are more dense for younger people than older. Overall I do not think young people tend to have higher measurements.

Figure 1.3 is a scatter plot to identify whether there is a systematic drift over time in the laboratory's measurement process. The red dots correspond to the mean measurement value for each date recorded in the data set.

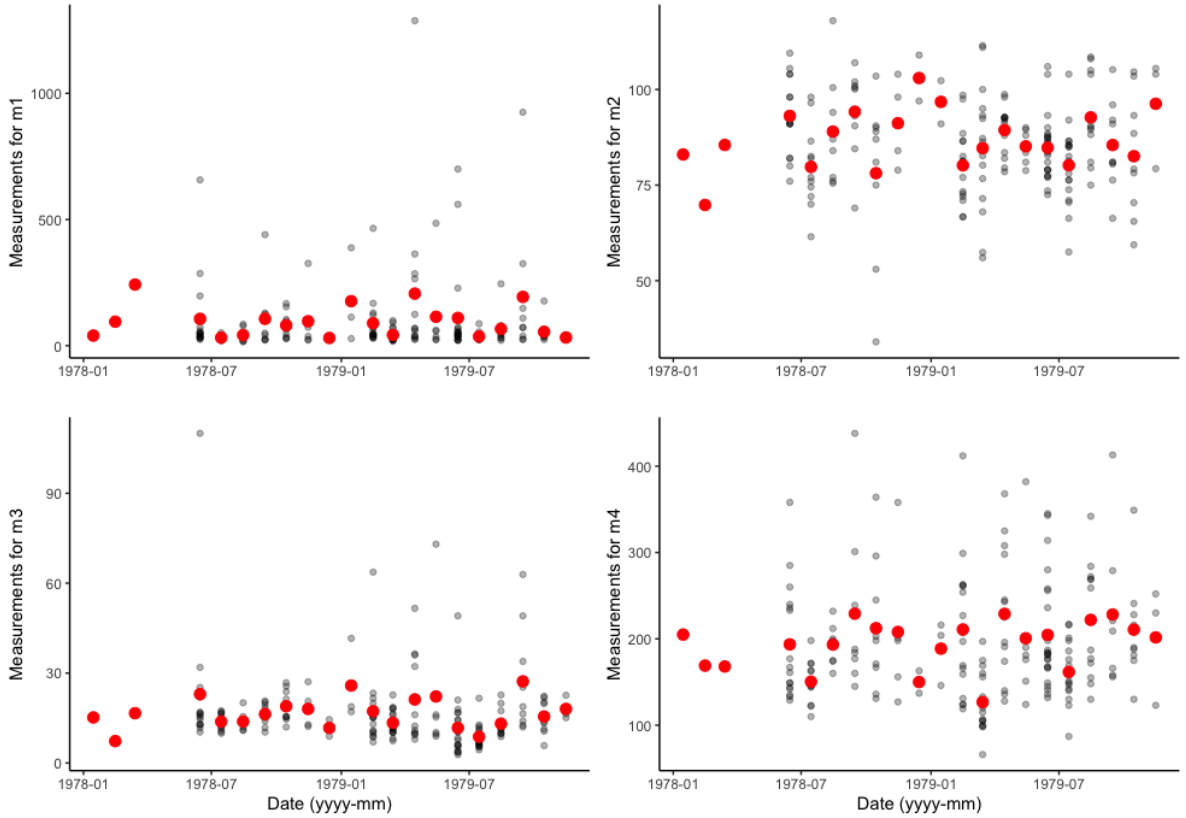

Figure 1.3 *A scatter plot of the date of blood samples taken against the corresponding measurement value. Black dots correspond to the raw measurement values. Darker grey dots indicate that many of the same measurement values were recorded on the same day. The red dots are the mean measurement value for each day the blood sample was taken.* ***1.3(a)*** *top left - m1. 1.****3(b)*** *top right - m2.* ***1.3(c)*** *bottom left - m3.* ***1.3(d)*** *bottom right - m4.*

Figure 1.3 shows that measurements between dates are roughly similar. In each subfigure the red dots are all in the same range of each other, there is no distinct increase or decrease. With a few high values of m1 in *1.3(a)* I understand the concern that newer measurements are different from older measurements but perhaps more carriers were sampled at a more recent date. Also, in all subfigures the first three dots indicate that only a few samples were taken at an older date and only 194 samples were taken over a brief period of two years therefore it is difficult to draw meaningful conclusions. Overall I do not believe there needs to be worry of a systematic drift.

Conclusion

To conclude the analysis found that the optimal screening procedure to detect carriers would be to use all four measurements in the data set since it had the highest test accuracy at 86.21%. The analysis found the next best procedure was to use m1 and m4 for classification at a test accuracy of 82.76%, followed by the current industry standard of using only m1 at an accuracy of 81.03%, using m2, m3 and m4 only was not found to be useful.

The analysis also found that young people do not tend to have higher measurements and there is no systematic drift in the laboratory's measurement procedure. Although being confident in such a conclusion is difficult due to potential sampling issues.

Future work

Since the four measurements are not equally easy to obtain due to cost and accuracy for example, it is important to take these issues into consideration. For example, if it is too difficult to obtain all four measurements, the results of the analysis indicate to next combine m1 and m4. However with the accuracy of m1 and m4 being only 1.73% higher than the accuracy obtained from using the current standard of m1 only, it is more important to develop techniques to obtain such measurements easily and cost-effectively.

Increasing the sampling size of data would make this analysis much more reliable. Alternatively it is important to take more blood samples from older people and over a longer period of time to be able to assess whether time and age have an effect on measurement values.

Data Set 2: DNA Data

Introduction

A sequence of DNA consists of four chemical bases: adenine, thymine, cytosine and guanine (A, T, C or G). These bases are also known as nucleotides. In a DNA double helix adenine will always pair with thymine and cytosine will always pair with guanine, and vice versa, which is important for DNA replication [1]. A codon is a sequence of three nucleotides and the order of which corresponds to a specific amino acid [2]. A chain of amino acids is called a protein and proteins are pivotal for the function of an organism [3].

Bacteriophages or simply phages are viruses that infect bacteria and regulate bacterial populations in natural ecosystems. Phages however invade the human body and contribute to the evolution of bacterial cells in the human body by acquiring and spreading DNA. The immune system reacts to them to an unknown extent and their impact on human health is not also unknown. Thus studies suggest more attention must be paid to their interference.

The aim of this analysis is to distinguish between human and phage DNA sequences. The methods used to do this are through the use of adaptive boosting and random forests and the effectiveness of each method is compared. The data set used to conduct the analysis is made up of human and phage DNA sequences.

Data and methods

The DNA data contains 300 human DNA sequences and 300 phage DNA sequences, labelled pos and neg respectively. There are 101 columns, the first containing such labels and the remaining 100 specifying a specific nucleotide. With the aim of distinguishing between human and phage DNA, I extracted various features of the sequence and created a new extracted features data set. This data set has 16 columns, the first being the sequence label and the rest being the 15 features I extracted which are: the number of As, Ts, Gs and Cs and the number of consecutive As, Ts, Gs and Cs; the number of occurrences of AT, TA, GC and CG, in light of how bases pair in a DNA double helix; the number of GAGs and AGAs and finally the proportion of C/G to A/T, since it is known that such a proportion can be used to help discriminate between organisms.

The methods of classification considered in this analysis are adaptive boosting and random forests. For further details of such methods refer to *Appendix A*. The effectiveness of both classifiers are compared using the accuracy of the test data set, obtained by splitting the data by 70% for a training set and 30% for the test set or 420 and 180 samples respectively.. This overcomes problems of overfitting.

Results

Adaptive boosting is used first to classify the DNA data set. *Table 2.1* shows the resulting confusion matrix and accuracy of boosting on the test data set.

BOOSTING	**PREDICTED CLASS**	
TRUE CLASS	**neg**	**pos**
neg	65	25
pos	33	57
ACCURACY		0.6777778

Table 2.1 *Confusion matrix obtained from the boosting classifier using the DNA test data.*

Due to randomness of the classifier and evaluation procedure changing between models the test accuracy results vary each time the model is trained, thus I ran the boosting model ten times and calculated the mean test accuracy to be 65.83%. Although the confusion matrix is omitted from the report, the training accuracy of one boosting model was found to be 88.10% indicating serious violations of overfitting. The low test accuracy score indicates that the model does not distinguish between human and phage DNA well. As a result I used random forest classification to see if the test accuracy improves. *Table 2.2* shows the resulting confusion matrix and accuracy on the test data set.

RANDOM FOREST	**PREDICTED CLASS**	
TRUE CLASS	**neg**	**pos**
neg	63	28
pos	27	62
ACCURACY		0.6944444

Table 2.2 *Confusion matrix obtained from the random forest classifier using the DNA test data.*

Again I ran the model ten times and calculated the test accuracy to be 68.61% making the random forest classifier better than boosting by roughly 3%. Note the random forest classifier calculates an out-of-bag error rate. One random forest model calculated this rate to be 26.43% meaning the training accuracy is 73.57%. With roughly only a 5% difference between training and test accuracies, overfitting is less of a problem than it was with the boosting classifier. Thus the random forest classifier is certainly better than boosting on the DNA data set.

I repeated the same analysis with the extracted features data set. *Table 2.3* shows the resulting confusion matrix and accuracy of boosting on the extracted features test data set.

BOOSTING	**PREDICTED CLASS**	
TRUE CLASS	**neg**	**pos**
neg	86	6
pos	10	80
ACCURACY		0.9222222

Table 2.3 *Confusion matrix obtained from the boosting classifier using the extracted features test data.*

Running the model ten times achieves a mean test accuracy of 91.39% and with one boosting model resulting in a training accuracy of 96.67% this model classifies unseen data well and much better than the original DNA sequence. A test accuracy of 91.39% is significantly higher than both test accuracies obtained from both the classifiers that used the original DNA data set by roughly 20%. With the extracted features resulting in better classification results, random forest classification was tested to see if it is still a better classifier than boosting. *Table 2.4* shows the resulting confusion matrix and accuracy on the test data set.

RANDOM FOREST	**PREDICTED CLASS**	
TRUE CLASS	**neg**	**pos**
neg	80	8
pos	10	82
ACCURACY		0.9

Table 2.4 *Confusion matrix obtained from the random forest classifier using the extracted features test data.*

The mean test accuracy of ten different random forest classifiers was calculated to be 90.04%, roughly 1% worse than the boosting classifier used on the extracted features data set. This may be the result of, again, the randomness of each classifier or how the training and test sets were split. Having created a new data set containing only extracted features, a sequence once in the training set could now be in the test set. One training accuracy was found to be 95%, which again is roughly a difference of 5%, indicating that random forests do not have a significant problem with overfitting, regardless of the data presented to it. Boosting uses the data directly to find its optimal classification rate whereas random forests do not. Instead forests combine the prediction of many decision trees to find its optimal classification rate. Thus it is no surprise that random forests do not have the same problem with overfitting than boosting does. Even still, this analysis shows that the boosting classifier is better than the random forest for the extracted features data set.

With these results proving that selecting features of DNA sequences better distinguish between human and phage DNA, I created a plot to identify which features were most responsible for the better classification results. *Figure 2.1* shows a variable importance plot obtained using the optimal classifier (boosting) on the extracted features data set and plots the importance of each variable used in boosting from most significant to least significant.

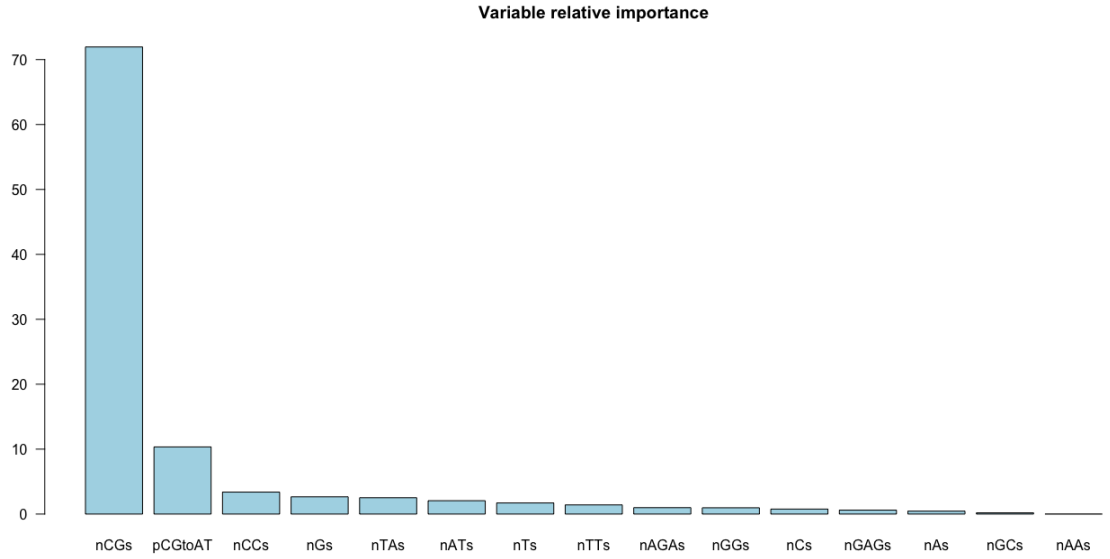

Figure 2.1 *Variable importance plot obtained from using the boosting classifier on the extracted features data.*

Figure 2.1 identifies that the number of Cs followed by Gs is very significant in distinguishing between human and phage DNA whilst the number of consecutive As is negligent. The number of CGs is so significant that perhaps only using this feature would be enough to yield excellent classification results, showing that perhaps the proportion of C/G to A/T is not the only feature to help discriminate between organisms.

Conclusion

In summary the best way to distinguish between human and phage DNA is by extracting specific features within each sequence. Between boosting and random forest classification, a random forest was a better classifier for the original DNA data set, whilst boosting was better for the extracted features data set, with a test accuracy of 68.61% and 91.39% respectively.

The differences in training and test accuracies was 5% for both the DNA and extracted features data sets when using a random forest. This concludes that random forest classification has generally less problems with overfitting, especially when compared to boosting, regardless of the data set used.

Since extracted features are better at identifying between human and phage DNA, research into which specific features is paramount. The analysis showed that the number of cytosine followed by guanine was most important in classification and so further research into why this may be is advised.

Future work

Further analysis into the specific features that distinguish between human DNA and phage DNA should be researched, especially the number of CGs in a sequence. Then with distinguished features known the interaction of phage DNA can be studied better and their impact on the human body can be investigated.

Final remarks

The code for the analysis can be found in the appropriate appendices. However if you would like to run the code for yourself, or understand why I came to certain conclusions, for example why LDA was used to classify m4 instead of ANN or naïve Bayes or why I used scaled and centered data, then please explore my GitHub repository linked here: https://github.com/jacobpassfield/data-analysis-project.

And finally, if you have any queries regarding either analysis then do not hesitate to get in touch.

References

[1] National Human Genome Research Institute, DNA sequencing Fact Sheet, available at https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Fact-Sheet (accessed 06/05/21)
[2] National Human Genome Research Institute, Codon, available at https://www.genome.gov/genetics-glossary/Codon (accessed 06/05/21)
[3] Medline Plus, What are proteins and what do they do?, https://medlineplus.gov/genetics/understanding/howgeneswork/protein/#:~:text=Proteins%20are%20made%20up%20of%20hundreds%20or%20thousands,protein's%20unique%203-dimensional%20structure%20and%20its%20specific%20function (accessed 06/05/21)

APPENDIX A: Description of methods used

Scaling and centering

Scaling subtracts the variable column mean $\bar{x}_i$ from the variable $\bar{x}_{ij}$ of row j and centering divides by the standard deviation of the variable column s_j such as:

$$\tilde{x}_{ij} = \frac{x_{ij} - \bar{x}_i}{s_i}.$$

Linear discriminant analysis

The aim of linear discriminant analysis is to find the direction of maximum separation of two different classes. Observations are predicted by estimating the probability that it belongs to each class. The observation is classified in the class with the highest probability.

https://machinelearningmastery.com/linear-discriminant-analysis-for-machine-learning/

Artificial neural networks

Artificial neurons are similar to real neurons. They consist of inputs that are multiplied by weights from which some function determines the activation of the neuron and an output is made. A combination of neurons form an artificial neural network. As well as the inner and output layer, neural networks may have hidden layers and so the output of the first layer is the input of the hidden layer and the output of the hidden layer form the final output. The decay argument prevents the output growing too large to help with model convergence.

Random forest

Random forests combine the prediction results of multiple decision trees. Decisions trees are a classifier that use multiple levels to obtain a final decision. Each level questions the data until all the data can be classified as accurately as possible. Combining predictions is optimal if the predictions from each tree are uncorrelated. A random forest tries to ensure the predictions from each tree have less correlation to learn more about the data. In this report the random forest consists of 500 decision trees and 3 variables are tried at each split. Trying a specified number of variables at each split is the way random forest decorrelates the decision trees.

https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/

Adaptive boosting

Adaptive boosting combines multiple weak classifiers to form one strong classifier. Decision stumps are the name of weak classifiers which are decision trees with a single split. Data that is difficult to classify is penalised whilst data that is opposite is rewarded. The penalty and reward is numeric and the total sum represents the final output of the classifier.

https://towardsdatascience.com/understanding-adaboost-2f94f22d5bfe

APPENDIX B: R commands and output for the biomedical data set

Data

```
carriers <- read.table("biomedical-data/data/carriers.txt", header = T)
normals <- read.table("biomedical-data/data/normals.txt", header = T)

library(tidyverse)
data <- bind_rows("0" = normals, "1" = carriers, .id = "class")
data$class <- as.factor(data$class)

library(stringr)
data$date <- sprintf("%06d", data$date)
data$date <- str_replace_all(data$date, "000", "015")
data$date <- str_replace_all(data$date, "00", "15")
data$date <- as.Date(data$date, "%m%d%y")

data <- arrange(data, class, date)

# SAVE
save(data, file = "biomedical-data/data/biomedical-data.RData")
```

Results

```
load(file = "biomedical-data/data/biomedical-data.RData")

set.seed(1234)
trainIndex = createDataPartition(data$class, p = 0.7, list = F)
train = data[trainIndex,]
test = data[-trainIndex,]
```

m1

```
nn = train(class ~ ., data = train[,c(1,4)], method = "nnet", trControl =
trainControl(method = "LOOCV"))
nn$results
```

```
#   size decay  Accuracy     Kappa
# 1    1 0e+00 0.7058824 0.1964549
# 2    1 1e-04 0.7279412 0.2613036
# 3    1 1e-01 0.7941176 0.4764916
# 4    3 0e+00 0.7205882 0.2459877
# 5    3 1e-04 0.7500000 0.3492823
# 6    3 1e-01 0.8382353 0.6111256
# 7    5 0e+00 0.8088235 0.5353482
# 8    5 1e-04 0.8088235 0.5353482
```

```
# 9    5 1e-01 0.8455882 0.6308170 < < <

nnS = train(class ~ ., data = train[,c(1,4)], method = "nnet", trControl =
trainControl(method = "LOOCV"), preProcess = c("center", "scale"))
nnS$results

#   size decay  Accuracy     Kappa
# 1    1 0e+00 0.8455882 0.6308170 < < <
# 2    1 1e-04 0.8455882 0.6308170
# 3    1 1e-01 0.8308824 0.5912180
# 4    3 0e+00 0.8308824 0.5956567
# 5    3 1e-04 0.8382353 0.6111256
# 6    3 1e-01 0.8308824 0.5912180
# 7    5 0e+00 0.8308824 0.5956567
# 8    5 1e-04 0.8308824 0.5912180
# 9    5 1e-01 0.8308824 0.5912180

# Scaled data wins

nngrid = expand.grid(size = 1, decay = 0)
nn = train(class ~ ., data = train[,c(1,4)], method = "nnet", trControl =
trainControl(method = "LOOCV"), preProcess = c("center", "scale"), tuneGrid =
nngrid)
nn$results
# size decay  Accuracy    Kappa
#    1     0 0.8455882 0.630817

table(true = nn$pred[,2], predicted = nn$pred[,1])
#      predicted
# true   0   1
#    0  86   3
#    1  18  29
(86+29)/dim(train)[1] # 0.8455882

nngrid = expand.grid(size = 1, decay = 0)
nn = train(class ~ ., data = train[,c(1,4)], method = "nnet", trControl =
trainControl(method = "LOOCV"), preProcess = c("center", "scale"), tuneGrid =
nngrid)
cm = table(true = nn$pred[,2], predicted = nn$pred[,1])
cm
(cm[1,1] + cm[2,2]) / dim(train)[1] # 0.8455882

test.pred = predict(nn, test[,c(1,4)])
cm = table(true = test[,1], predicted = as.factor(test.pred))
cm
(cm[1,1] + cm[2,2]) / dim(test)[1] # 0.8103448

nngrid = expand.grid(size = 1, decay = 0)
nn.test.acc = double(10)
for(i in 1:10){
  nn = train(class ~ ., data = train[,c(1,4)], method = "nnet", trControl =
trainControl(method = "LOOCV"), preProcess = c("center", "scale"), tuneGrid =
nngrid)
```

```
  test.pred = predict(nn, test[,c(1,4)])
  cm = table(true = test[,1], predicted = test.pred)
  nn.test.acc[i] = (cm[1,1] + cm[2,2]) / dim(test)[1]
}
mean(nn.test.acc) # 0.8103448
```

PCA biplot

```
pca = prcomp(data[,4:7])

biplot(pca, cex = 0.8)
```

m4

```
lda = train(class ~ ., data = train[,c(1,7)], method = "lda", trControl =
trainControl(method = "LOOCV"))
lda$results # 0.8382353

lda = train(class ~ ., data = train[,c(1,7)], method = "lda", trControl =
trainControl(method = "LOOCV"))
cm = table(true = lda$pred[,2], predicted = lda$pred[,1])
cm
(cm[1,1] + cm[2,2]) / dim(train)[1] # 0.8382353
test.pred = predict(lda, test[,c(1,7)])
cm = table(true = test[,1], predicted = test.pred)
cm
(cm[1,1] + cm[2,2]) / dim(test)[1] # 0.7586207
```

m1 and m4

```
library(caret)

nn = train(class ~ ., data = train[,c(1,4,7)], method = "nnet", trControl =
trainControl(method = "LOOCV"))
nn$results

#   size decay  Accuracy      Kappa
# 1    1 0e+00 0.6544118 0.03792896
# 2    1 1e-04 0.6911765 0.17670799
# 3    1 1e-01 0.7867647 0.46090760
# 4    3 0e+00 0.6985294 0.20114613
# 5    3 1e-04 0.7500000 0.36431125
# 6    3 1e-01 0.8382353 0.63503293
# 7    5 0e+00 0.7867647 0.47300909
# 8    5 1e-04 0.7794118 0.47544356
# 9    5 1e-01 0.8602941 0.67988107

nnS = train(class ~ ., data = train[,c(1,4,7)], method = "nnet", trControl =
trainControl(method = "LOOCV"), preProcess = c("center", "scale"))
nnS$results

#   size decay  Accuracy     Kappa
```

```
# 1    1 0e+00 0.8602941 0.6798811 < < <
# 2    1 1e-04 0.8602941 0.6798811
# 3    1 1e-01 0.8529412 0.6612702
# 4    3 0e+00 0.8308824 0.6042510
# 5    3 1e-04 0.8382353 0.6153253
# 6    3 1e-01 0.8529412 0.6612702
# 7    5 0e+00 0.8529412 0.6540321
# 8    5 1e-04 0.8455882 0.6424637
# 9    5 1e-01 0.8529412 0.6612702

# Scaled data wins

nngrid = expand.grid(size = 1, decay = 0)
nn = train(class ~ ., data = train[,c(1,4,7)], method = "nnet", trControl =
trainControl(method = "LOOCV"), preProcess = c("center", "scale"), tuneGrid =
nngrid)
cm = table(true = nn$pred[,2], predicted = nn$pred[,1])
cm
(cm[1,1] + cm[2,2]) / dim(train)[1] # 0.8602941

test.pred = predict(nn, test[,c(1,4,7)])
cm = table(true = test[,1], predicted = test.pred)
cm
(cm[1,1] + cm[2,2]) / dim(test)[1] # 0.8275862
```

m1, m2, m3 and m4

```
library(caret)

nn = train(class ~ ., data = train[,c(1,4:7)], method = "nnet", trControl =
trainControl(method = "LOOCV"))
nn$results

#   size decay  Accuracy     Kappa
# 1    1 0e+00 0.7352941 0.2943211
# 2    1 1e-04 0.6764706 0.1161004
# 3    1 1e-01 0.8235294 0.5614082
# 4    3 0e+00 0.7500000 0.3786617
# 5    3 1e-04 0.7426471 0.3567568
# 6    3 1e-01 0.8455882 0.6386640
# 7    5 0e+00 0.7794118 0.4697167
# 8    5 1e-04 0.7647059 0.4281209
# 9    5 1e-01 0.8235294 0.5935243

nnS = train(class ~ ., data = train[,c(1,4:7)], method = "nnet", trControl =
trainControl(method = "LOOCV"), preProcess = c("center", "scale"))
nnS$results

#   size decay  Accuracy     Kappa
# 1    1 0e+00 0.9191176 0.8165768
# 2    1 1e-04 0.9264706 0.8341059 < < <
# 3    1 1e-01 0.9191176 0.8165768
# 4    3 0e+00 0.8750000 0.7135778
```

```
# 5    3 1e-04 0.9044118 0.7832271
# 6    3 1e-01 0.9264706 0.8341059
# 7    5 0e+00 0.8455882 0.6603235
# 8    5 1e-04 0.8676471 0.7102959
# 9    5 1e-01 0.9264706 0.8341059

# Scaled data wins

nngrid = expand.grid(size = 1, decay = 0.0001)
nn = train(class ~ ., data = train[,c(1,4:7)], method = "nnet", trControl =
trainControl(method = "LOOCV"), preProcess = c("center", "scale"), tuneGrid =
nngrid)
cm = table(true = nn$pred[,2], predicted = nn$pred[,1])
cm
(cm[1,1] + cm[2,2]) / dim(train)[1] # 0.9264706

test.pred = predict(nn, test[,c(1,4:7)])
cm = table(true = test[,1], predicted = test.pred)
cm
(cm[1,1] + cm[2,2]) / dim(test)[1] # 0.862069
```

Age

```
library(tidyverse)
library(ggplot2)

age_mean_m1 <- data %>%
  group_by(age) %>%
  summarise(m1 = mean(m1))

age_mean_m2 <- data %>%
  group_by(age) %>%
  summarise(m2 = mean(m2))

age_mean_m3 <- data %>%
  group_by(age) %>%
  summarise(m3 = mean(m3))

age_mean_m4 <- data %>%
  group_by(age) %>%
  summarise(m4 = mean(m4))

library(patchwork)

age_m1 <- ggplot(data, aes(x = age, y = m1)) +
  geom_point(alpha = 0.3) +
  geom_point(data = age_mean_m1, size = 3, colour = "red") +
  labs(x = "", y = "Measurements for m1") +
  theme_classic()

age_m2 <- ggplot(data, aes(x = age, y = m2)) +
  geom_point(alpha = 0.3) +
  geom_point(data = age_mean_m2, size = 3, colour = "red") +
```

```
  labs(x = "", y = "Measurements for m2") +
  theme_classic()

age_m3 <- ggplot(data, aes(x = age, y = m3)) +
  geom_point(alpha = 0.3) +
  geom_point(data = age_mean_m3, size = 3, colour = "red") +
  labs(x = "Age of patient in years", y = "Measurements for m3") +
  theme_classic()

age_m4 <- ggplot(data, aes(x = age, y = m4)) +
  geom_point(alpha = 0.3) +
  geom_point(data = age_mean_m4, size = 3, colour = "red") +
  labs(x = "Age of patient in years", y = "Measurements for m4") +
  theme_classic()

(age_m1 + age_m2) / (age_m3 + age_m4)
```

Systematic drift

```
library(tidyverse)
library(ggplot2)

date_mean_m1 <- data %>%
  group_by(date) %>%
  summarise(m1 = mean(m1))

date_mean_m2 <- data %>%
  group_by(date) %>%
  summarise(m2 = mean(m2))

date_mean_m3 <- data %>%
  group_by(date) %>%
  summarise(m3 = mean(m3))

date_mean_m4 <- data %>%
  group_by(date) %>%
  summarise(m4 = mean(m4))

library(patchwork)

date_m1 <- ggplot(data, aes(x = date, y = m1)) +
  geom_point(alpha = 0.3) +
  geom_point(data = date_mean_m1, size = 3, colour = "red") +
  labs(x = "", y = "Measurements for m1") +
  theme_classic()

date_m2 <- ggplot(data, aes(x = date, y = m2)) +
  geom_point(alpha = 0.3) +
  geom_point(data = date_mean_m2, size = 3, colour = "red") +
  labs(x = "", y = "Measurements for m2") +
  theme_classic()
```

```
date_m3 <- ggplot(data, aes(x = date, y = m3)) +
  geom_point(alpha = 0.3) +
  geom_point(data = date_mean_m3, size = 3, colour = "red") +
  labs(x = "Date (yyyy-mm)", y = "Measurements for m3") +
  theme_classic()

date_m4 <- ggplot(data, aes(x = date, y = m4)) +
  geom_point(alpha = 0.3) +
  geom_point(data = date_mean_m4, size = 3, colour = "red") +
  labs(x = "Date (yyyy-mm)", y = "Measurements for m4") +
  theme_classic()

(date_m1 + date_m2) / (date_m3 + date_m4)
```

APPENDIX C: R commands and output for the DNA data set

Data

```
data = read.table("dna-data/data/human-phage.txt")

# NUMBER OF As
nAs = vector(mode = "integer", length = nrow(data))
for (i in 1:nrow(data)){
  nAs[i] = length(which(data[i,] == "A")) }

# NUMBER OF Ts
nTs = vector(mode = "integer", length = nrow(data))
for (i in 1:nrow(data)){
  nTs[i] = length(which(data[i,] == "T")) }

# NUMBER OF Gs
nGs = vector(mode = "integer", length = nrow(data))
for (i in 1:nrow(data)){
  nGs[i] = length(which(data[i,] == "G")) }

# NUMBER OF Cs
nCs = vector(mode = "integer", length = nrow(data))
for (i in 1:nrow(data)){
  nCs[i] = length(which(data[i,] == "C")) }

# NUMBER OF CONSECUTIVE As
nAAs = vector(mode = "integer", length = nrow(data))
for (i in 1:nrow(data)){
  for (j in 1:(ncol(data)-1)){
    nAAs[i] = nAAs[i] + ((data[i,j] == "A") & (data[i,j+1] == "A"))} }

# NUMBER OF CONSECUTIVE Ts
nTTs = vector(mode = "integer", length = nrow(data))
for (i in 1:nrow(data)){
```

```
  for (j in 1:(ncol(data)-1)){
    nTTs[i] = nTTs[i] + ((data[i,j] == "T") & (data[i,j+1] == "T"))} }

# NUMBER OF CONSECUTIVE Gs
nGGs = vector(mode = "integer", length = nrow(data))
for (i in 1:nrow(data)){
  for (j in 1:(ncol(data)-1)){
    nGGs[i] = nGGs[i] + ((data[i,j] == "G") & (data[i,j+1] == "G"))} }

# NUMBER OF CONSECUTIVE Cs
nCCs = vector(mode = "integer", length = nrow(data))
for (i in 1:nrow(data)){
  for (j in 1:(ncol(data)-1)){
    nCCs[i] = nCCs[i] + ((data[i,j] == "C") & (data[i,j+1] == "C"))} }

# NUMBER OF ATs
nATs = vector(mode = "integer", length = nrow(data))
for (i in 1:nrow(data)){
  for (j in 1:(ncol(data)-1)){
    nATs[i] = nATs[i] + ((data[i,j] == "A") & (data[i,j+1] == "T"))} }

# NUMBER OF TAs
nTAs = vector(mode = "integer", length = nrow(data))
for (i in 1:nrow(data)){
  for (j in 1:(ncol(data)-1)){
    nTAs[i] = nTAs[i] + ((data[i,j] == "T") & (data[i,j+1] == "A"))} }

# NUMBER OF CGs
nCGs = vector(mode = "integer", length = nrow(data))
for (i in 1:nrow(data)){
  for (j in 1:(ncol(data)-1)){
    nCGs[i] = nCGs[i] + ((data[i,j] == "C") & (data[i,j+1] == "G"))} }

# NUMBER OF GCs
nGCs = vector(mode = "integer", length = nrow(data))
for (i in 1:nrow(data)){
  for (j in 1:(ncol(data)-1)){
    nGCs[i] = nGCs[i] + ((data[i,j] == "G") & (data[i,j+1] == "C"))} }

# NUMBER OF GAGs
nGAGs = vector(mode = "integer", length = nrow(data))
for (i in 1:nrow(data)){
  for (j in 1:(ncol(data)-2)){
    nGAGs[i] = nGAGs[i] + ((data[i,j] == "G") & (data[i,j+1] == "A") &
(data[i,j+2] == "G"))} }

# NUMBER OF AGAs
nAGAs = vector(mode = "integer", length = nrow(data))
for (i in 1:nrow(data)){
  for (j in 1:(ncol(data)-2)){
    nAGAs[i] = nAGAs[i] + ((data[i,j] == "A") & (data[i,j+1] == "G") &
(data[i,j+2] == "A"))} }
```

```
# PROPORTION OF C/G to A/T
pCGtoAT = vector(mode = "integer", length = nrow(data))
for (i in 1:nrow(data)){
  pCGtoAT[i] = ( nCs[i] / nGs[i] ) / ( nAs[i] / nTs[i] )}

# Adding to data
EFdata = data
EFdata$nAs = nAs
EFdata$nTs = nTs
EFdata$nGs = nGs
EFdata$nCs = nCs
EFdata$nAAs = nAAs
EFdata$nTTs = nTTs
EFdata$nGGs = nGGs
EFdata$nCCs = nCCs
EFdata$nATs = nATs
EFdata$nTAs = nTAs
EFdata$nCGs = nCGs
EFdata$nGCs = nGCs
EFdata$nGAGs = nGAGs
EFdata$nAGAs = nAGAs
EFdata$pCGtoAT = pCGtoAT

EFdata = EFdata[,c(1,102:116)]

# SAVE
save(EFdata, file = "dna-data/data/ef-data.RData")
```

Results

Full DNA sequence

```
data = read.table("dna-data/data/human-phage.txt")

library(caret)
set.seed(1234)
trainIndex = createDataPartition(data$V1, p = 0.7, list = F)
train = data[trainIndex,]
test = data[-trainIndex,]

# # # ADABOOSTING # # #

library(adabag)

boost = boosting(V1 ~ ., data = train, control = rpart.control(maxdepth = 1))

# Training set

boost.pred = predict.boosting(boost, newdata = train[,-1])
cm = table(true = train[,1], predicted = boost.pred$class)
(cm[1,1] + cm[2,2]) / dim(train)[1] # 0.8809524
```

```
# Test set

boost1.pred = predict.boosting(boost, newdata = test[,-1])
cm = table(true = test[,1], predicted = boost1.pred$class)
cm
#       predicted
# true  neg pos
#  neg  68  22
#  pos  32  58
(cm[1,1]+cm[2,2])/dim(test)[1] # 0.6777778

boost.test.acc = double(10)
for(i in 1:10){
  boost2 = boosting(V1 ~ ., data = train, control = rpart.control(maxdepth =
1))
  boost2.pred = predict.boosting(boost2, newdata = test[,-1])
  cm = table(true = test[,1], predicted = boost2.pred$class)
  boost.test.acc[i] = (cm[1,1] + cm[2,2]) / dim(test)[1]
}
mean(boost.test.acc) # 0.6583333

# # # RANDOM FOREST # # #

library(randomForest)

rf = randomForest(V1 ~ ., data = train, importance = TRUE)
rf # OOB error rate 26.43%

rf.pred = predict(rf, test[,-1], type = "class")
cm = table(predicted = rf.pred, true = test[,1])
cm
#           true
# predicted neg pos
#       neg  63  28
#       pos  27  62
(cm[1,1]+cm[2,2])/dim(test)[1] # 0.6944444

rf.test.acc = double(10)
for(i in 1:10){
  rf1 = randomForest(V1 ~ ., data = train, importance = TRUE)
  rf1.pred = predict(rf1, test[,-1], type = "class")
  cm = table(true = test[,1], predicted = rf1.pred)
  rf.test.acc[i] = (cm[1,1] + cm[2,2]) / dim(test)[1]
}
mean(rf.test.acc) # 0.6861111

# # # COMPARING CLASSIFIERS # # #

# Adaboosting:   0.6583333
# Random forest: 0.6861111 < < <
```

Extracted features of the DNA sequence

```
load(file = "dna-data/data/e-f-data.RData")

library(caret)

set.seed(1234)
trainIndex = createDataPartition(EFdata$V1, p = 0.7, list = F)
trainEF = EFdata[trainIndex,]
testEF = EFdata[-trainIndex,]

# # # ADABOOSTING # # #

library(adabag)

boost = boosting(V1 ~ ., data = trainEF, control = rpart.control(maxdepth =
1)) # 100 iterations default

# Training data

boost.pred = predict.boosting(boost, newdata = trainEF[,-1])
cm = table(true = trainEF[,1], predicted = boost.pred$class)
(cm[1,1]+cm[2,2])/dim(trainEF)[1] # 0.9666667

# Test data

boost1.pred = predict.boosting(boost, newdata = testEF[,-1])
cm = table(true = testEF[,1], predicted = boost1.pred$class)
cm
#       predicted
# true  neg pos
#  neg   86   6
#  pos   10  80
(cm[1,1]+cm[2,2])/dim(testEF)[1] # 0.9222222

boost.test.acc = double(10)
for(i in 1:10){
  boost2 = boosting(V1 ~ ., data = trainEF, control = rpart.control(maxdepth
= 1))
  boost2.pred = predict.boosting(boost2, newdata = testEF[,-1])
  cm = table(true = testEF[,1], predicted = boost2.pred$class)
  boost.test.acc[i] = (cm[1,1] + cm[2,2]) / dim(testEF)[1]
}
mean(boost.test.acc) # 0.9138889

# # # RANDOM FOREST # # #

library(randomForest)

rf = randomForest(V1 ~ ., data = trainEF, importance = TRUE)
rf # OOB error rate 5%

rf.pred = predict(rf, testEF[,-1], type = "class")
cm = table(predicted = rf.pred, true = testEF[,1])
cm
```

```
#           true
# predicted neg pos
#       neg  80   8
#       pos  10  82
(cm[1,1]+cm[2,2])/dim(testEF)[1] # 0.9

rf.test.acc = double(10)
for(i in 1:10){
  rf1 = randomForest(V1 ~ ., data = trainEF, importance = TRUE)
  rf1.pred = predict(rf1, testEF[,-1], type = "class")
  cm = table(true = testEF[,1], predicted = rf1.pred)
  rf.test.acc[i] = (cm[1,1] + cm[2,2]) / dim(testEF)[1]
}
mean(rf.test.acc) # 0.9044444

# # # COMPARING CLASSIFIERS # # #

# Adaboosting:   0.9138889 < < <
# Random forest: 0.9044444 < < <

# # # VARIABLE IMPORTANCE # # #

importanceplot(boost)
```