Practical Data Science with R
Final Report

James Godwin

Dataset 1: biomedical data

Introduction
A rare genetic disorder requires a screening method to identify carriers of the disease. Previously just one measurement was obtained, but now three additional measurements have been taken. This statistical analysis report is, therefore, analysing the best way to use this new data to create a more effective and robust screening method, while looking at its potential limitations.

Data
The dataset is two smaller datasets combined, one for 'normals' (non-carriers) with 127 individuals and another for 'carriers' with 67 individuals. To make the dataset more flexible, I combined them and created a binary variable 'carrier' with 1 to denote a carrier and 0 to denote a 'normal'. The data was collected across a period from 1978 to 1979, with measurements (m1, m2, m3 and m4) taken from the blood. M1 is the measurement that was previously used in a screening procedure to identify carriers. The individual's age, and date of measurement, was also noted for each observation. The date of measurement changed to just the digits for the year. A duplicate dataset was then produced for this to analyse whether there is a systematic drift in the measurement process. Furthermore, the combined dataset became two independent sets: a random training dataset (70%) and a test dataset (30%).

Methods
I first looked at the correlation between variables and saw there was some correlation between a few of the measurements. Therefore principal component analysis (PCA) was first used to produce a set of uncorrelated variables which maximise variance. PCA works by creating components which are a linear combination of the original variables in the dataset in such a way that maximises variance. These new components are uncorrelated with each other.

Discriminant analyses were best to maximise separation between classes in classification problems, ensuring the output of such models is highly accurate. Discriminant analyses are also usually rigorous in classification since the assumption that the likelihood of classes forming a Gaussian distribution holds in many applications (Ghojogh & Crowley, 2019, p. 8). LDA essentially creates a lower-dimensional space (similar to PCA) that maximises the between-class variance and minimising the within-class variance. LDA is the best classifier when its assumptions that the covariance matrices are equal and the likelihood has a Gaussian distribution are upheld. Therefore QDA should be considered in the case that the covariance matrices are not identical, or Naive Bayes if neither assumptions hold.

Similar to the pruning of ensemble classifiers, a stepwise classification can determine whether there exists a subset of the variables that perform better than the complete set of variables with LDA and QDA.

Other considerations included ensemble classifiers; however, there was usually a large discrepancy between the accuracy of classification of the test and training datasets. For example, even after pruning the decision tree, there was an 11% reduction in accuracy when predicting classes based on the test dataset. Therefore the problem of overfitting was persistent after simplifying the decision tree. Furthermore, when using a random forest, there was a misclassification rate of 17% on the test data which is much higher than some other methods used in the following project - despite the fact a random forest should overcome overfitting with the train/test datasets.

Initial analysis
The heatmap in figure 1, showing the correlation between all of the variables, illustrates there is not much correlation between being a carrier or not and most variables. Just the age of an individual and m4 correlate greater than 0.5, and it is also interesting to note that the original measurement, m1, is not strongly correlated. Therefore the screening procedure should ideally use more correlated variables, and perhaps consider if m1 is a useful measurement. Furthermore, m1, m3 and m4 all have correlations between each other higher than 0.5 and so as mentioned before PCA analysis will produce new uncorrelated components that will simplify the dataset.

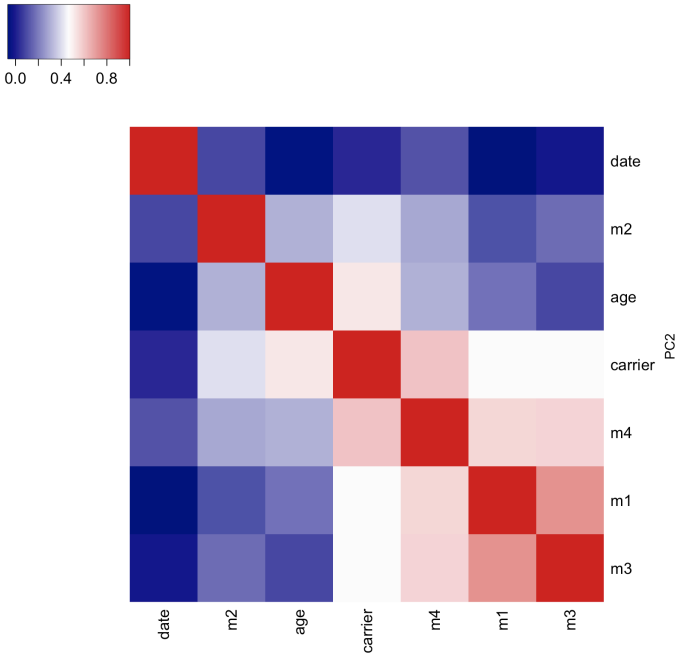

Figure 1 - heat map of correlation between variables

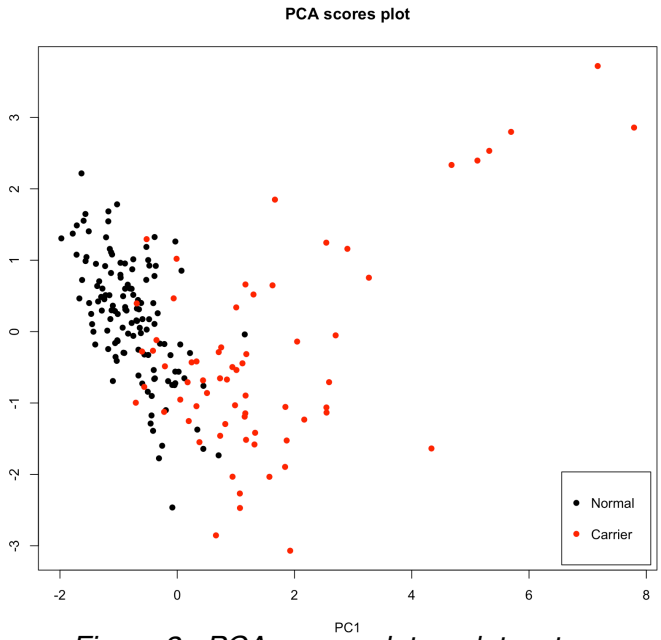

Figure 2 - PCA scores plot on dataset

The data has been scaled for PCA since it was difficult to conduct any kind of analysis on the principal components whereas with figure 2 we can see there is a possible decision boundary between being a 'normal' or carrier. However, there is a significant overlap between the two classes, meaning there is a region in the plot which could lead to misclassification. Additionally, there are a handful of observations which are carriers that are quite far away (in terms of PC1) from other observations of the same class. It could be just anomalies in the data; however, since there are more than only a few, there could be some measurement problems. One of the critical issues with the data collected is the potential that there has been drift over time in the measurements, this can be checked by looking at PCA scores plots by comparing 1978 observations and 1979 observations.

The first thing to note is that classification is more difficult when using just the 1979 and the scores for the carriers are much more varied in 1979 - suggesting the hypothesis that there was a drift in measurements overtime is true. Therefore PCA may not be the best classification method; however, it has verified one of the initial concerns using the dataset. For the best screening

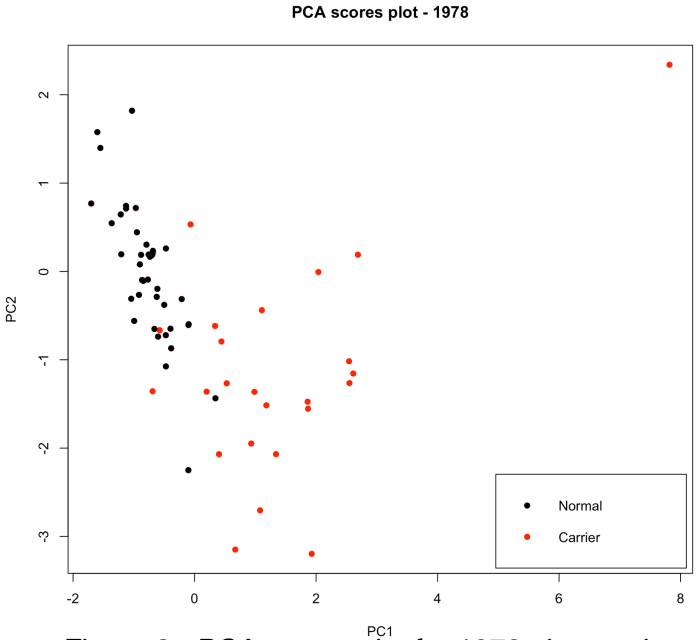

Figure 3 - PCA scores plot for 1978 observations

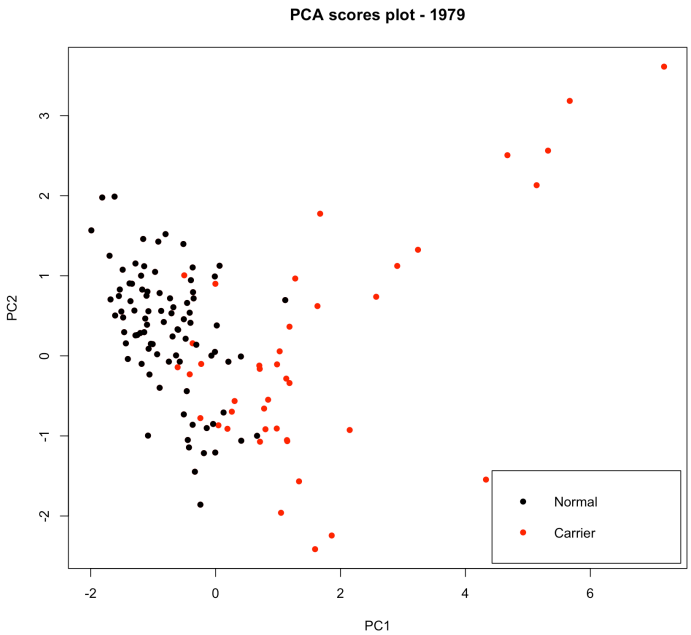

Figure 4 - PCA scores plot for 1979 observations

procedure for the data available, other methods will follow. For further analyses, the full dataset was used since creating independent test and training datasets of the separate years would give small datasets of not much use.

Linear Discriminant Analysis (LDA)

As mentioned before, LDA looks to maximise separation between classes to classify observations given a set of variables. After producing an LDA model using the training data we get the coefficients of linear discriminants in figure 5, and 'age' has the highest absolute value meaning it is the most influential in separating observations between being a carrier or 'normal'. The least prominent for classification is the variable for the date, and again interestingly M1 (used for prior classification) is the second least influential.

Variable	Coefficient
Age	7.421469E-02
Date	-2.370538E-06
M1	8.263962E-04
M2	3.895571E-02
M3	3.906667E-02
M4	7.980713E-03

Figure 5 - LDA coefficients

The results of LDA are in the confusion matrices below. For the training data, an accuracy rating of ~0.88 is obtained, which is quite good. However, over 10% of the data is misclassified, which is a significant amount. Furthermore, when introducing new data, via the independent test set, the accuracy decrease to just ~0.845, meaning the LDA model created is possibly overfitting the data. Hence even more new data introduced to the model will continue to have approximately 15% of the observations misclassified.

		Actual	
		Normal	Carrier
Predicted	Normal	80	11
	Carrier	5	40

Figure 6 - confusion matrix for training set

		Actual	
		Normal	Carrier
Predicted	Normal	35	2
	Carrier	7	14

Figure 7 - confusion matrix for test set

At the beginning of LDA analysis, I mentioned the variable significance for separation - this can be taken further by looking at whether there exists a subset of the variables together than produces a better model for classification. Stepwise variable selection makes the classification algorithm more efficient and providing the correct subset of variables are selected - it can reduce overfitting, which is the main issue with the current LDA model produced (Luo, Ding, & Huang, 2011, p. 420). After the forward stepwise selection, the best subset of variables found is using the variables m2, m3, m4 and age. Therefore after a new cross-validated (CV) LDA model with this subset of variables, the accuracy rating increases to ~0.88, which is a much better result. Since the Leave-One-Out (LOO) CV was used, the accuracy validation is much more reliable since it uses the most extreme form of a repeated CV.

		Actual	
		Normal	Carrier
Predicted	Normal	123	18
	Carrier	4	49

Figure 8 - confusion matrix for LOO CV

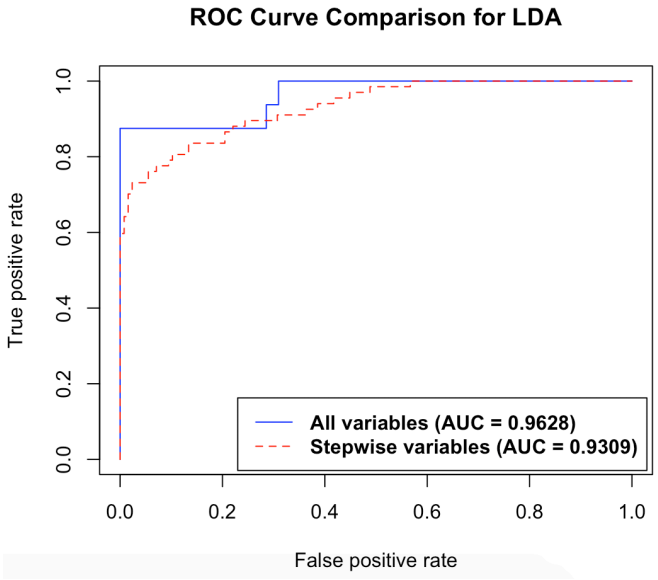

Figure 9 - ROC curves for both LDA models

While we have found the LDA model with the stepwise subset of variables gives the highest accuracy rating, even after CV, this can be analysed further through looking at the area under the ROC curve for a model. It is a more robust method for model validation in that since 65% of the dataset are normals, and it should have a reasonably high accuracy rating regardless. Thus this metric is more general in that it looks at how well the model performs for a variety of thresholds of classification (Bradley, 1997, pp. 1145-1146). The best model will have a ROC curve that has an area of one under the curve, while neither model has this the closest is the LDA with all variables. We find that including all variables is the best LDA model.

Quadratic Discriminant Analysis (QDA)

A more flexible approach to classification is QDA, which relaxes the assumption that the covariance matrices between carriers and 'normals' are equal. Potentially making the screening method more robust, but at the expense of possible overfitting due to more estimated parameters.

		Actual	
		Normal	Carrier
Predicted	Normal	124	15
	Carrier	3	52

Figure 10 - confusion matrix for QDA

Similarly, with LDA, I created a QDA model with the training dataset and independent test dataset. For the reasons described previously, only cross-validation was used since the accuracy is preferable to alternatives. Using CV we find, an accuracy rating of ~0.91, which is the best so far. The results are in the confusion matrix in figure 10.

		Actual	
		Normal	Carrier
Predicted	Normal	125	15
	Carrier	2	52

Figure 11 - confusion matrix for QDA with stepwise variables

Again to see whether there exists a combination of variables that provides a more efficient and robust classification is using the stepwise variable selection. Interestingly using QDA, it seems to suggest that only one of the variables - date - needs to be removed. Therefore the original measurement m1 may play a more critical role in the classification. With one of the variables removed, it should still mitigate against potential overfitting as a result of using QDA. The resulting accuracy with CV is again ~0.91; however, because this new model has few dependent variables, it is more likely to generalise better to new data. The results are in the confusion matrix in figure 11.

A ROC curve comparison shows that using the stepwise variables yields a greater area under the curve, and so is the preferable QDA model.

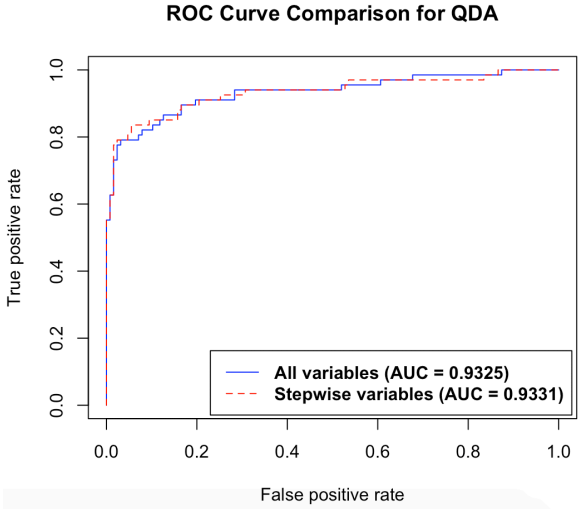

Figure 12 - ROC curve comparison for QDA

Naïve Bayes (NB) analysis

NB is a complementary approach to the previous two in that it relaxes both of the key assumptions for LDA, i.e. that the covariance matrices of the classes are equal and that the likelihood forms a Gaussian distribution. While some criticise this approach, ironically due to its assumption of independence (deemed not suitable by many), it has been found that there are various previous examples of its success and therefore should be considered (Al-Aidaroos, Bakar, & Othman, 2012, p. 6). Furthermore, it is a very efficient approach with its independence assumption and can perform well with little training data, which is convenient in our case as we have fewer carrier observations.

		Actual	
		Normal	Carrier
Predicted	Normal	125	16
	Carrier	2	51

Figure 13 - confusion matrix for NB model

After adjusting the parameters of the NB model, such as whether to scale the data or use a kernel, the accuracy rating obtained is ~0.91 which is just as good as QDA with the stepwise variable selection. Other models with fewer variables yielded the same area under the ROC curve and therefore were of little use considering. The AUC for this NB model was 0.8727, and since this is quite low, other models identified generalise better to different classification thresholds. Hence this dataset may be one of the few perhaps that NB does not classify well.

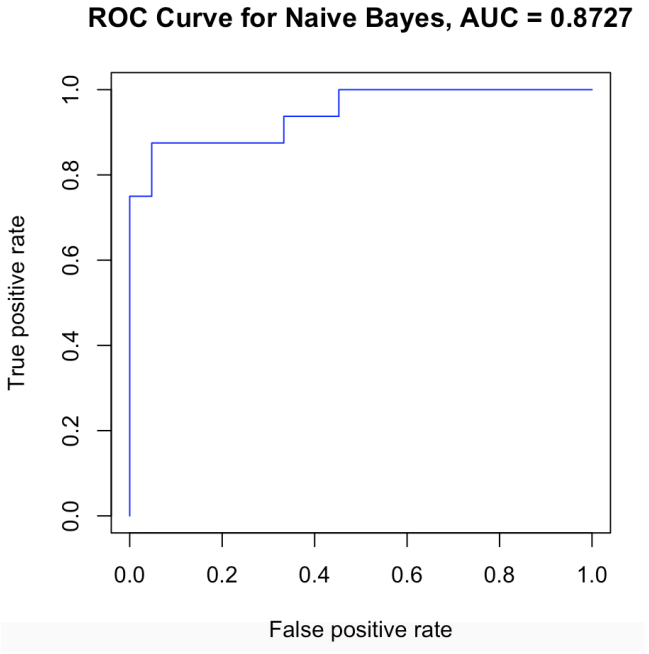

Figure 14 - ROC curve comparison for NB

Comparison

Since confusion matrices show just a single point on a ROC curve, it is best to compare these ROC curves rather than the matrices. The first observation in figure 15 is that the ROC for LDA has the highest AUC, therefore in terms of this metric LDA is the best classifier for this dataset. However, the optimal point on this curve to maximise the TPR which minimising the FPR gives a point which is less than with QDA. Hence why the confusion matrix illustrated in this report for QDA is better than LDA, but LDA is best overall for all classification thresholds. Secondly, using the AUC as a metric does not give a clear-cut analysis of these ROC curves - especially as these ROC curves frequently cross over each other and at varying sensitivity and specificity levels. Therefore another way to compare and analyse these curves is with cost curves, i.e. looking at the expected cost of misclassification for the specific classification

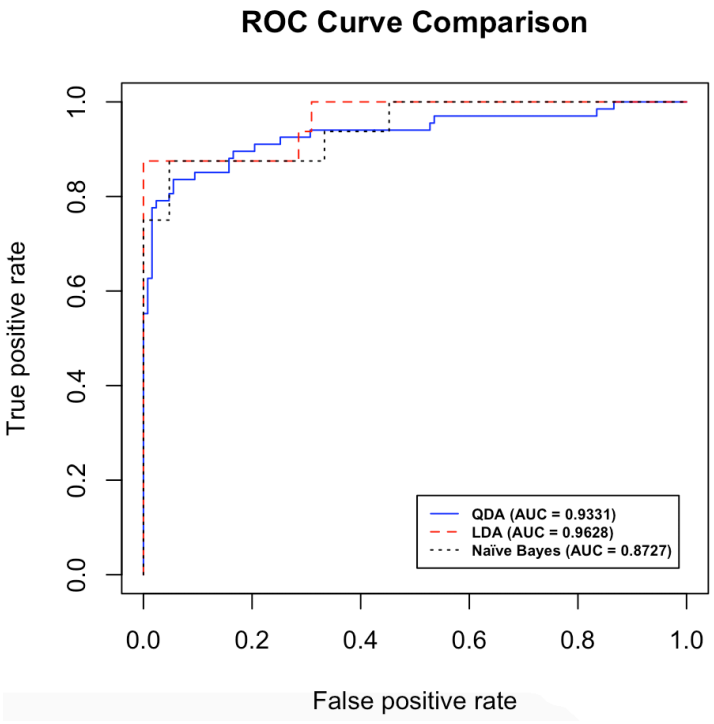

Figure 15 - ROC curve comparison of methods

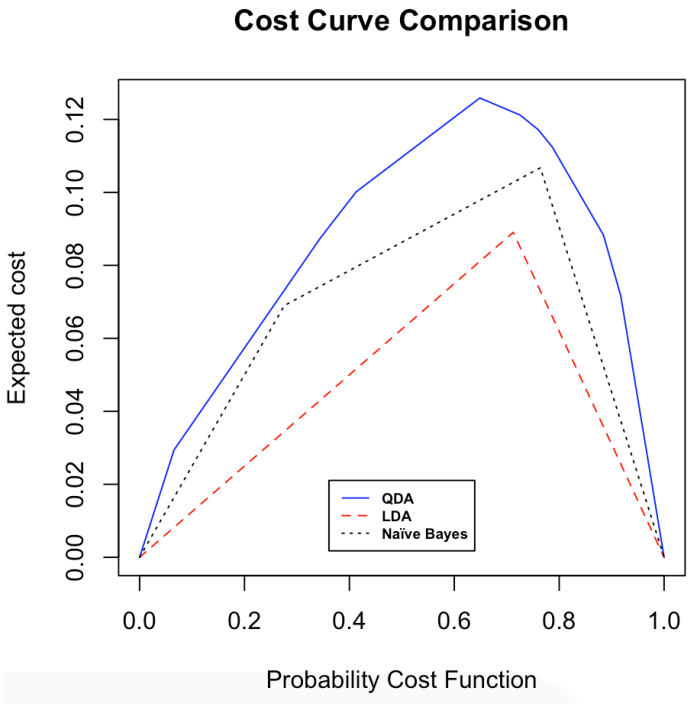

Figure 16 - cost curve comparison of methods

models. A cost curve has the expected cost as the dependent variable and the value of the probability cost function (PCF) which is essentially the slope of the ROC curve but at varying points along with it and is scaled to a value between 0 and 1 (Drummond & Holte, 2004, p. 1).

We find that the classifier with the lowest expected cost is LDA, but more importantly, for all values of the PCF since the cost curves do not cross over. Also, the cost curve illustrates just how much better LDA is in comparison to other methods. From PCF values between ~0 and ~0.7, LDA outperforms QDA and NB significantly more than when the PCF is greater than ~0.7. Therefore despite the varying performances for different thresholds using ROC curves, the cost of misclassification is significantly and consistently lower than using QDA or NB.

Conclusion

This project has sought to develop a screening procedure, using a collection of measurements including blood samples and age, with a variety of classification techniques. LDA is the most robust model for the given data with a cross-validated accuracy rate of 87% with specific sensitivity and specificity levels. From ROC analysis, the highest AUC came from the LDA model, therefore with different sensitivity and specificity levels, LDA was preferred. However given various ROC curves crossed, so LDA was never consistently preferred, a cost curve analysis found that LDA yielded the lowest expected cost. Given it is essential to reduce the loss of life and improve patient outcomes, it is preferred to choose a screening procedure that minimises cost, and so LDA is the preferred method.

In developing the LDA model, age was the most influential variable in determining whether someone was a carrier while the original measurement m1 was the second least influential in the model. Newer measurements were, therefore, better at classifying carriers. Additionally, a stepwise variable selection determined which measurements to omit, and while excluding m1 and date yielded a better accuracy rate, the full LDA model better performed in classification for different classification levels.

In terms of data issues, there was drift over time in measurements using PCA scores plots in figures 2 and 3. So the reason this model does not perform as well as one would hope is that if you put bad data into a model, you will end up with a bad model. Therefore this explains why some methods used to classify did not perform as expected, for example, with NB. Since younger people tended to have higher measurements and that the measurement process drifted over time, other measurements could exist that are more robust to drift over time. Also repeat measurements, especially with the newer measurements, should be taken to observe anomalies better.

Works Cited

Al-Aidaroos, K. M., Bakar, A. A., & Othman, Z. (2012). *Medical Data Classification with Naive Bayes Approach.* Selangor: Centre for Artificial Intelligence, Faculty of Information and Science Technology, Universiti Kebangsaan Malaysia.

Bradley, A. P. (1997, July). The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition, Volume 3 Issue 7*, pp. 1145-1159.

Drummond, C., & Holte, R. C. (2004). *What ROC Curves Can't Do (and Cost Curves Can).* Ontario: Institute for Information Technology, National Research Council Canada.

Ghojogh, B., & Crowley, M. (2019). *Linear and Quadratic Discriminant Analysis: Tutorial .* University of Waterloo .

Luo, D., Ding, C., & Huang, H. (2011). Linear Discriminat Analysis: New Formulations and Overfit Analysis. *25th AAAI Conference on Artificial Intelligence* (pp. 417-422). Association for the Advancement of Artificial Intelligence (AAAI).

Code from R

```
normalsdata = read.table("normals.txt", header = T)
normalsdata$carrier <- rep(0, nrow(normalsdata))
carrierdata = read.table("carriers.txt", header = T)
carrierdata$carrier <- rep(1, nrow(carrierdata))
data = rbind(normalsdata, carrierdata)
data$carrier = as.factor(data$carrier)
#creating a new dataframe, merging the two datasets, and adding a new factor variable whether the individual is a carrier or not

datapca = prcomp(data[,1:6], scale = TRUE)
biplot(datapca)
plot(datapca$x[,1], datapca$x[,2], xlab = "PC1", ylab = "PC2", main = "PCA scores plot", pch = 16, col = data$carrier)
legend("bottomright", inset = 0.01, legend = c("Normal", "Carrier"), col = unique(data$carrier), pch = 16)
#scaled PCA analysis of the dataset, however there is not much separation, maybe not best method?
#we also see there are probably no outliers

corrvariables = cor(data[,1:7], use = "all.obs", method = "pearson")
heatmap3(corrvariables, rowV = NULL, colv = NULL, showColDendro = F, showRowDendro = F, scale = "none", revC = T, key.title = "Correlation coefficient")
#heatmap shows most correlation is between carrier and m4, additonally m1 and m3 are quite correlated
#there is not much correlation between the date or any other variables
#there is however some correlation between age and carrier

m = nrow(data)
trainIndex = sample(1:m, size = round(0.7*m), replace = FALSE)
traindata = data[trainIndex ,]
testdata = data[-trainIndex ,]
#splitting the data into independent training and test data sets

summary(datapca)
#hence we should keep 5 PCs, since the cumulative variance is >0.9

pcascores = datapca$x[,1:5]
pcaldamodel = lda(pcascores, data[,7], CV = T)
table(real = data[,7], predicted = pcaldamodel$class)
#PCA-LDA model has an accuracy of ~0.874, good but not perfect, also will this generalise?
```

```
group = traindata[,7]
ldamodel = lda(traindata[,1:6], group)
ldamodel
#age has the greatest absolute value, and so is the most influential in distinguishing between
        the groups
#m2 and m3 are also very good

ldamodelcv = lda(data[,1:6], data[,7], CV = T)
table(predicted = ldamodelcv$class, real = data[,7])
#0.87 accuracy rate for CV lda model

predtest = predict(ldamodel, testdata[,1:6])
table(predicted = predtest$class, real = testdata[,7])
#~0.845 accuracy rating, less than the PCA-LDA model

predtrain = predict(ldamodel, traindata[,1:6])
table(predicted = predtrain$class, real = traindata[,7])
#~0.88 accuracy rating, hence the model is possibly overfitted

nb = train((carrier)~., method = "naive_bayes", data = data, trControl = trainControl(method =
        "LOOCV"))
nb$results
#we see that using a kernel is more accurate
#look at practical 5 to help with this

grid = expand.grid(laplace = 0, usekernel = F, adjust = 1)
nb = train((carrier)~., method = "naive_bayes", data = data, trControl = trainControl(method =
        "LOOCV"), preProcess=c("scale"), tuneGrid = grid)
nb$results
#creation of a nb with required and wanted parameters

table(predicted = nb$pred[,1] , true = nb$pred[,2])
#we obtain a ~0.89 accuracy rating, our highest yet, but could be better

real = as.numeric(data[,7])
auc_nb = roc(real, as.numeric(nb$pred[,1]))
auc_nb

nb$pred

newvarsnb = c("age", "m1", "m2", "m3", "m4", "carrier")
newdatanb = data[newvarsnb]

newnb  = train((carrier)~., method = "naive_bayes", data = newdatanb, trControl =
        trainControl(method = "LOOCV"), tuneGrid = grid)

pred_newnb = predict(newnb, newdatanb, type = "prob")
rates_newnb = prediction(pred_newnb[,2], newdatanb[,6])
perf_newnb = performance(rates_newnb, "tpr", "fpr")
plot(perf_newnb, col = "red", main = "ROC Curve for Naive Bayes")

model_nb = naive_bayes(traindata[,1:6], traindata[,7], usekernel = T)
pred_nb = predict(model_nb, testdata[,1:6])
real = testdata[,7]
table(true = real, predicted = pred_nb)
#~0.879 accuracy for using test data
```

```
pred2_nb = predict(model_nb, traindata[,1:6])
real2 = traindata[,7]
table(true = real2, predicted = pred2_nb)
#using the training data we have an accuracy rating of ~0.92, hence poss overfit


real = as.numeric(testdata[,7])
auc_nb = roc(real, as.numeric(pred_nb))
auc_nb
#we find that the area under the ROC curve for the test data is 0.878

pred_nb = predict(model_nb, testdata[,1:6], type = "prob")
library(ROCR)
pred_nb
rates_nb = prediction(pred_nb[,2], testdata[,7])
perf_nb = performance(rates_nb, "tpr", "fpr")
plot(perf_nb, col = "red", main = "ROC Curve for Naive Bayes")
#plotting ROC curve for NB

plot(perf_nb, col = "blue", lty = 1, main = "ROC Curve for Naive Bayes, AUC = 0.8727")
plot(perf_newnb, col = "red", add = TRUE, lty = 2)

pred_lda = predict(ldamodel, testdata[,1:6], type = "prob")
rates_lda = prediction(pred_lda$posterior[,2], testdata[,7])
perf_lda = performance(rates_lda, "tpr", "fpr")
plot(perf_lda, col = "blue", lty = 1, main = "ROC Curve Comparison of LDA and Naive Bayes")
plot(perf_nb, col = "red", add = TRUE, lty = 2)
legend("bottomright", inset = 0.05, legend = c("LDA", "Naive Bayes"), col = c("blue", "red"), lty =
       1:2, text.font = 2, cex = 0.6)
#here we find that LDA is a better classifier for all sensistiivity settings and specificity settings

disease.tree = tree(carrier~., data = traindata)
plot(disease.tree)
text(disease.tree, cex = 0.7)
#creation of decision tree
tree.pred = predict(disease.tree, traindata[,-7], type = "class")
table(predicted = tree.pred, true = traindata[,7])
#training data has 0.948 accuracy
tree.pred = predict(disease.tree, testdata[,-7], type = "class")
table(predicted = tree.pred, true = testdata[,7])
#test data has 0.81 accuracy, poss overfitted

cv.disease = cv.tree(disease.tree, FUN = prune.misclass)
plot(cv.disease)
#hence we find we can prune the decision tree to 5

prune.disease = prune.misclass(disease.tree, best = 5)
prune.disease
plot(prune.disease)
text(prune.disease, cex = 0.7)
#plotting new decision tree

prunetree.pred = predict(prune.disease, traindata[,-7], type = "class")
table(predicted = prunetree.pred, true = traindata[,7])
#accuracy rate of 0.94 accuracy
```

```
prunetree.pred = predict(prune.disease, testdata[,-7], type = "class")
table(predicted = prunetree.pred, true = testdata[,7])
#accuracy rate of 0.83, an improvement but not large

rf = randomForest(carrier~., data = traindata, mtry = 6, importance = TRUE)
rf
#random forest model with OOB estimate of 9.56%

rf.pred = predict(rf, testdata[,-7], type = "class")
table(predicted = rf.pred, true = testdata[,7])
#again an accuracy rate of 0.83, and so the random forest classifier is not good

stepmodel = stepclass(carrier~., data = data, method = "lda", direction = "forward", criterion =
        "AC", improvement = 0.01)
#we can create a model that has m2, m3, m4, age
#this gives an accuracy of 0.67

stepmodel
newvars = c("age", "m2", "m3", "m4")
newdata = data[newvars]
limmodel = lda(newdata, data[,7], CV = T)
table(predicted = limmodel$class, real = data[,7])
#0.88 accuracy rating in classification, hence using lda with LOO the accuracy is higher than
        using stepclass

pred_lda2 = limmodel$posterior[,2]
rates_lda2 = prediction(limmodel$posterior[,2], data[,7])
perf_lda2 = performance(rates_lda2, "tpr", "fpr")
plot(perf_lda2, col = "blue", lty = 1, main = "ROC Curve of LDA with stepwise")
#roc curve for lda with stepwise variables

stepmodelback = stepclass(carrier~., data = data, method = "lda", direction = "backward",
        criterion = "AC", improvement = 0.01)
#backwards model
newvars2 = c("age", "m1", "m2", "m3", "m4")
newdata2 = data[newvars2]
limmodel2 = lda(newdata2, data[,7], CV = T)
table(predicted = limmodel2$class, real = data[,7])
#0.88 accuracy rating, more complex model so we prefer the forwards classification

qdamodel1 = qda(data[,-7], data[,7], CV = T)
table(predicted = qdamodel1$class, real = data[,7])
#normal qda, all variables and a ~0.91 accuracy

rates_qda1 = prediction(qdamodel1$posterior[,2], data[,7])
pred_qda1 = qdamodel1$posterior[,2]
perf_qda1 = performance(rates_qda1, "tpr", "fpr")
plot(perf_qda2, col = "blue", lty = 1, main = "ROC Curve of QDA")
#roc curve for QDA LOO CV

stepmodel2backward = stepclass(carrier~., data = data, method = "qda", direction = "forward",
        criterion = "AC", improvement = 0.01)
newvarsqda = c("age", "m1", "m2", "m3", "m4")
newdataqda = data[newvarsqda]
#best set of variables for using with qda, both directions produce the same combination
```

```
qdamodel = qda(newdataqda, data[,7], CV = T)
table(predicted = qdamodel$class, real = data[,7])
#0.91 accruacy rating, some improvement using a qda model and LOO cross validation

pred_qda2 = qdamodel$posterior[,2]
rates_qda2 = prediction(qdamodel$posterior[,2], data[,7])
perf_qda2 = performance(rates_qda2, "tpr", "fpr")
plot(perf_qda2, col = "blue", lty = 1, main = "ROC Curve of QDA with stepwise")
#ROC curve for QDA with special subset of variables from stepwise

plot(perf_lda, col = "blue", lty = 1, main = "ROC Curve Comparison of QDA, LDA and Naive
        Bayes")
plot(perf_nb, col = "red", add = TRUE, lty = 2)
plot(perf_qda, col = "green", add = TRUE, lty = 3)
legend("bottomright", inset = 0.05, legend = c("LDA", "Naive Bayes", "QDA"), col = c("blue",
        "red", "green"), lty = 1:3, text.font = 2, cex = 0.6)
#comparison of ROC curve for QDA, LDA and Naive Bayes, and we see that LDA is best

plot(perf_qda1, col = "blue", lty = 1, main = "ROC Curve Comparison for QDA")
plot(perf_qda2, col = "red", add = TRUE, lty = 2)
legend("bottomright", inset = 0.01, legend = c("All variables (AUC = 0.9325)", "Stepwise
        variables (AUC = 0.9331)"), col = c("blue", "red"), lty = 1:2, text.font = 2, cex = 1)
#comparison of using qda for without/with stepwise collection of variables

realqda = as.numeric(data[,7])
auc_qda = roc(realqda, as.numeric(pred_qda1)) #all variables
auc_qda
reallqa2 = as.numeric(data[,7])
auc_qda2 = roc(realqda, as.numeric(pred_qda2)) #new variables
auc_qda2

plot(perf_lda, col = "blue", lty = 1, main = "ROC Curve Comparison for LDA")
plot(perf_lda2, col = "red", add = TRUE, lty = 2)
legend("bottomright", inset = 0.01, legend = c("All variables (AUC = 0.9628)", "Stepwise
        variables (AUC = 0.9309)"), col = c("blue", "red"), lty = 1:2, text.font = 2, cex = 1)
#comparison of using lda with/without stepwise collection of variables

reallda = as.numeric(data[,7])
auc_lda = roc(reallda, as.numeric(pred_lda2))
auc_lda
reallda2 = as.numeric(testdata[,7])
auc_lda2 = roc(reallda2, as.numeric(pred_lda$posterior[,2]))
auc_lda2
#calculation of area under the curve for LDA

datanewdate = data
datanewdate$date = sub('.*(\\d{2}).*', '\\1', datanewdate$date)
#extracting last 2 digits of date ie the year

datanewdate = datanewdate[order(datanewdate$date),]
rownames(datanewdate) <- 1:nrow(datanewdate)
#ordering dataset by new date and then changing row index

datanewdate$date = as.numeric(datanewdate$date)
datanewdate$carrier = as.factor(datanewdate$carrier)
```

```
corrvariables2 = cor(datanewdate[,1:7], use = "all.obs", method = "pearson")
heatmap3(corrvariables2, rowV = NULL, colv = NULL, showColDendro = F, showRowDendro =
        F, scale = "none", revC = T)
#heat map of variables

datapca78 = prcomp(datanewdate[,1:6], scale = TRUE)

plot(datapca78$x[1:64,1], datapca78$x[1:64,2], xlab = "PC1", ylab = "PC2", main = "PCA scores
        plot - 1978", pch = 16, col = "red")
points(datapca78$x[1:39,1], datapca78$x[1:39, 2], pch = 16, col = "black")
legend("bottomright", inset = 0.01, legend = c("Normal", "Carrier"), col =
        unique(datanewdate$carrier), pch = 16)
#plot of 1978

plot(datapca78$x[65:194,1], datapca78$x[65:194,2], xlab = "PC1", ylab = "PC2", main = "PCA
        scores plot - 1979", pch = 16, col = "red")
points(datapca78$x[65:152,1], datapca78$x[65:152, 2], pch = 16, col = "black")
legend("bottomright", inset = 0.01, legend = c("Normal", "Carrier"), col =
        unique(datanewdate$carrier), pch = 16)
#plot for 1979 observations

plot(perf_qda2, col = "blue", lty = 1, main = "ROC Curve Comparison")
plot(perf_lda, col = "red", add = TRUE, lty = 2)
plot(perf_nb, col = "black", add = TRUE, lty = 3)
legend("bottomright", inset = 0.05, legend = c("QDA (AUC = 0.9331)", "LDA (AUC = 0.9628)",
        "Naïve Bayes (AUC = 0.8727)"), col = c("blue", "red", "black"), lty = 1:3, text.font = 2, cex
        = 0.6)
#roc curve comparison

costqda2 = performance(rates_qda2, 'ecost')
plot(costqda2, col = "blue", lty = 1, main = "Cost Curve Comparison")
costlda = performance(rates_lda, 'ecost')
plot(costlda)
costnb = performance(rates_nb, 'ecost')
plot(costnb)
#cost curve analysis

plot(costqda2, col = "blue", lty = 1, main = "Cost Curve Comparison", xlab = "Probability Cost
        Function")
plot(costlda, col = "red", add = TRUE, lty = 2)
plot(costnb, col = "black", add = TRUE, lty = 3)
legend("bottom", inset = 0.05, legend = c("QDA", "LDA", "Naïve Bayes"), col = c("blue", "red",
        "black"), lty = 1:3, text.font = 2, cex = 0.6)
#cost curve comparison
```

Dataset 2: DNA data

Introduction

A phage is a virus that infects bacteria and affects bacterial populations which can enter humans. However, their effect on humans' immune system is not known, and studies on these phages require distinguishing between human and phage DNA sequences. Therefore this analysis is primarily concerned with being able to take a DNA sequence and analyse it to decide whether it is from a human or phage DNA sequence. The primary aim is to compare a random forest classifier and other potential classifiers.

Data

The dataset contains 600 observations, equally split between human DNA sequences and phage DNA. Each observation contains a sequence of 100 acid bases (A, C, G, T) and a variable to denote whether it is human ("pos") or a phage ("neg") sample. To make the dataset more flexible and to analyse the new variables (such as the number of A bases), to be discussed in the method section, will form a new dataset for each observation. The new dataset combines an independent training and test sets containing 70% and 30% of all observations, respectively.

Methods

The first, and most important, task was to extract data from this sequence dataset to make it easier (and more successful) to analyse and then classify whether a sequence or not is from human or phage DNA. Loops counted the number of the respective bases in each sequence, and all of the potential pairs of bases (AA, AT, AC, AG,...). In the new dataset, each observation formed of the counts for the single bases and pairs, instead of the actual DNA sequence. There were no triplets (ATC, GCA, CCA,...) for two reasons. Firstly scientific evidence seems to suggest a key indicator is the GC-content, or the amount of G and C in the genome sequence (Birdsell, 2002, p. 1182). Due to some variation in the GC content across DNA sequences - for example, the phage ϕX174 has a GC content of 44% whereas for humans it is around 40% (Bansal, 2003, p. 1557). Secondly, when looking at variable importance via ensemble method boosting, in figure 1, it is clear that the number of CGs present is the most important and with stepwise variable selection the same variable and the proportion of CGs come out on top.

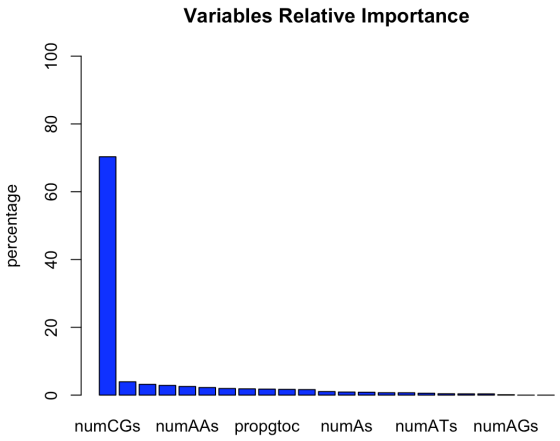

Figure 1 - importance of variables

An initial statistical analysis identified any correlations between variables, and produce a set of principal components which further reduces the dataset. While the feature extraction reduced the dataset from 101 variables to just 24, it was still worth to see whether the dimensionality could be reduced further and with a decision region to observe to classify observations.

The two main methods for statistical analysis used were discriminant analyses and decision trees. Discriminant analyses are excellent for classification problems since they seek the maximise separation between observations to produce a decision region, to then classify - in this case, whether a sequence is from a human or a phage. Linear Discriminant Analysis (LDA) is a more robust classifier than Quadratic Discriminant Analysis (QDA) since there are fewer parameters to be estimated. However, QDA is worth considering since there are few assumptions and is much more flexible if the decision boundary is some quadratic. Furthermore, in this context, LDA/QDA are regularly used in such pattern recognition and classification problems concerning DNA sequencing (Zhang, 2000, p. 4). Finally, decision trees are relatively simple to compute in that they take an observation and apply several rules to dictate which rule it should be tested against, to reach an eventual classification. Thus it is almost like a 'sorting hat' for observations, and since there are a few - but essential variables - to consider here a decision tree should be beneficial in classification (Siddiquee & Tasnim, 2018, p. 3).

Lastly, neural networks and partial least squares (PLS) models were not deemed valid classifiers in comparison to other models and therefore omitted from the overall analysis. In the appendix is its code and output for interest.

Initial analysis

There could be some correlation between variables; for example, the number of CGs may be correlated with the number of Gs and the number of Cs. In order to prevent this correlation, making classification difficult, Principal Component Analysis (PCA) was used to produce instead a set of uncorrelated variables which comprised of maximising the variance between variables to help with classification. Hence a PC is a linear combination of all variables in the dataset, with coefficients reflecting maximum variance.

The first two principal components are in figure 2 with a scores plot. There is a significant amount of overlap between -2 and 2 on PC1 - despite PC1 accounting for the most variance. Thus there is little use in PCA alone to produce a decision region to classify these DNA sequences. However, the PCs themselves are uncorrelated variables to be used in LDA/QDA.

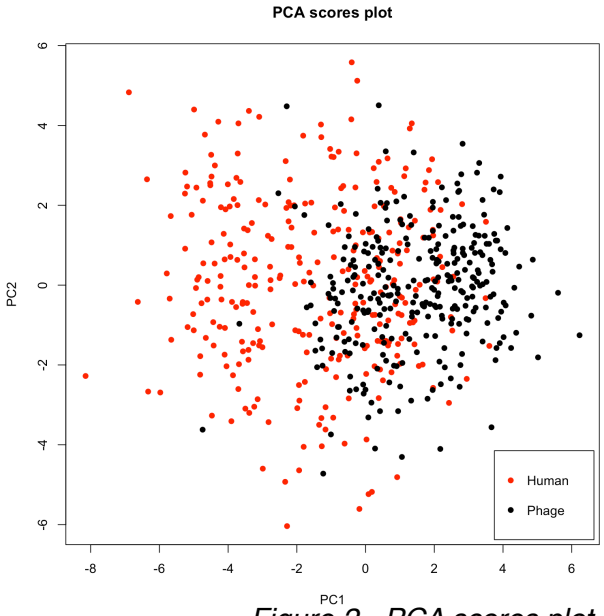

Figure 2 - PCA scores plot

Random forest classifier

Decision trees, in general, were explained in the methods section - however, there are additional methods of ensemble classification that overcome some of the critical problems they have. Decision trees tend to become very large to learn increasingly complex patterns in the data, which, as a result, makes the general model overfit the training data so it cannot generalise to new data inputs well. However, a random forest overcomes this problem by using numerous unrelated decision trees at once and averaging the results so that overfitting is less likely.

The resulting random forest model was tuned to have 200 trees with ten variables tried at each split. While increasing the number of trees generally produces a better model - this is at the expense of the model being less computationally efficient for a minimal gain in accuracy (Probst & Boulesteix, 2018, pp. 15-17). This model gives an Out-Of-Bag (OOB) estimate of error rate of 7.38%, which is a measure of how well the model can generalise, and so replaces the need for cross-validation. Using the random forest to classify the test data set we obtain an accuracy of 96% (or an error rate fo 6.66%), the result of this is in the confusion matrix in figure 3. Since the error for the test data is smaller than the OOB, the model is not overfitted and performs quite well in classifying whether a DNA sequence is from a human or phage. However, since there are other ensemble analyses available, these should be compared with the random forest classifier.

		Actual	
		Phage	Human
Predicted	Phage	85	5
	Human	7	83

Figure 3 - confusion matrix for random forest

Other ensemble classifiers

As mentioned previously, decision trees can overfit, but they can perform well if pruned (i.e. use an optimum number of branches). The first decision tree has 12 splits and with the confusion matrix in figure 4 has an accuracy rating of 95%, which is very strong and is better than the random forest. However, classifying the test set yields a 93% accuracy which possibly means the decision tree is

overfitted and is not a reliable model. A more robust model can prune a decision tree through cross-validation.

		Actual	
		Phage	Human
Predicted	Phage	201	13
	Human	7	199

Figure 4 - confusion matrix for normal decision tree with training data

		Actual	
		Phage	Human
Predicted	Phage	86	6
	Human	6	82

Figure 5 - confusion matrix for normal decision tree with test data

The resulting pruned decision tree has just two branches according to figure 6 since this has the lowest misclassification number with the lowest number of branches. However, because the model is so much more general, the accuracy for the training data is ~92%, and the test data is ~91%. Hence overfitting is still potentially present in conjunction with a decrease in accuracy, implying that the random forest classifier is better than both the standard and pruned decision trees.

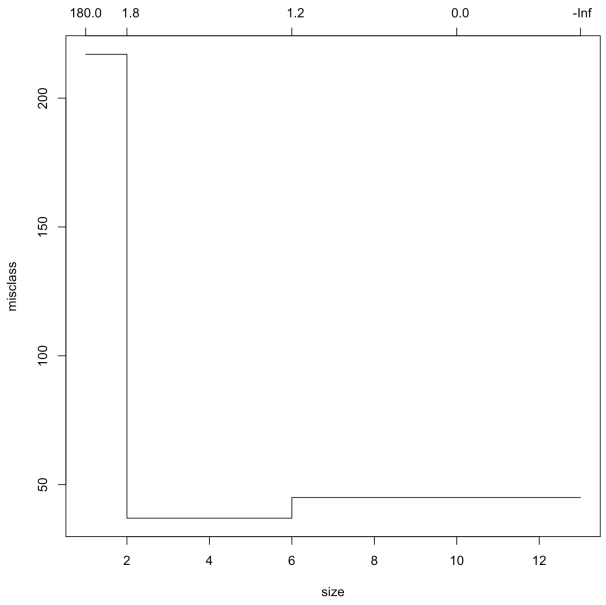

Figure 6 - misclassification v.s. size of tree

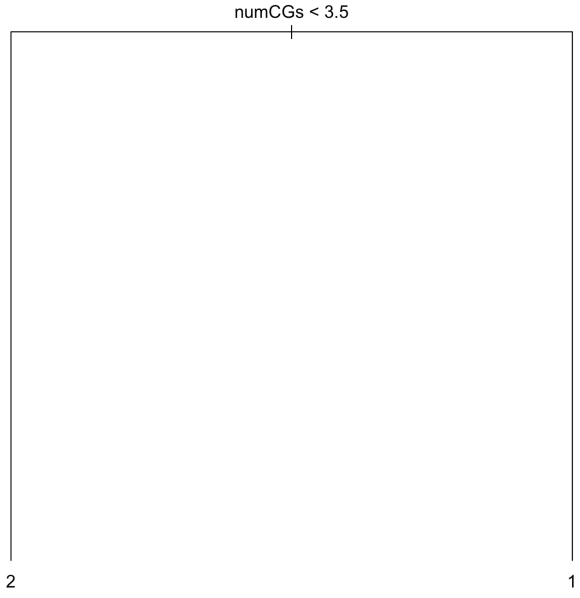

Figure 7 - pruned decision tree

		Actual	
		Phage	Human
Predicted	Phage	195	19
	Human	13	193

Figure 8 - confusion matrix for pruned tree with training data

		Actual	
		Phage	Human
Predicted	Phage	85	9
	Human	7	79

Figure 9 - confusion matrix for pruned tree with test data

Boosting is another ensemble technique that takes a collection of small decision trees to create a single and more accurate model than the trees separately. Boosting, however, is slightly different from random forests in that random forests use complete decision trees rather than smaller trees. This method, therefore, can be advantageous, providing the model has been tuned correctly - however, it is usually quite challenging to tune and therefore can lead to overfitting. The results from R suggest that boosting here does lead to overfitting since the training data has a classification accuracy of ~95% and the test data has an accuracy of ~94%. Since we have found the random forest is not overfitted and classifies the test data better, the random forest model remains the best classifier.

Linear Discriminant Analysis (LDA)

Due to the correlation between some of the new variables created, standard LDA was not possible. However, the uncorrelated principal components found from PCA can be analysed using discriminant analysis. Their cumulative variance determined the number of components to use, reaching above 95%, and this was the first 11 components.

The subsequent model for PCA-LDA had an accuracy of ~90.5%, which is much lower than other methods found. The variables in figure 1 may be interpreted as not particularly useful since numCGs dominates all other variables in terms of relative importance. Thus a stepwise variable selection may be better than using the PCs. These variables are numCG, numCC, numAC, propcg, numGs, numCs and numTAs. This subset using LDA gives an accuracy of 94%, which is much improved and similar to previous methods. Additionally, the area under the ROC curve (AUC) for LDA with stepwise is greater than with PCA-LDA and so is a more robust method.

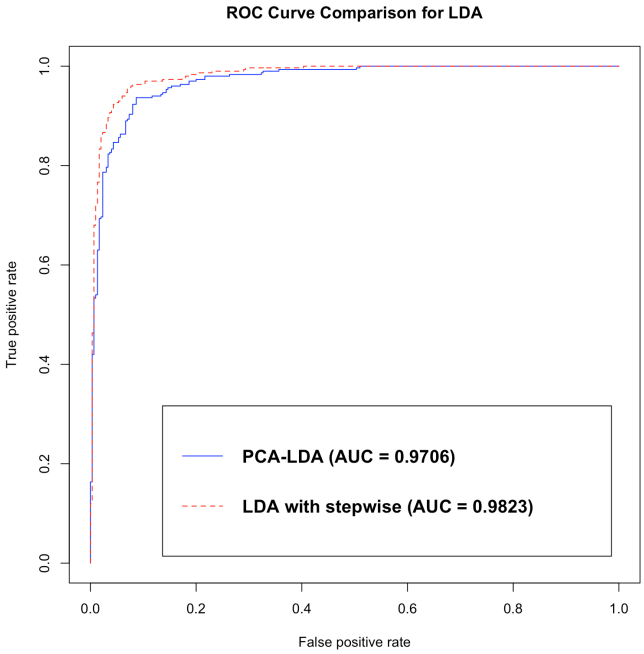

Figure 10 - ROC curves for PCA-LDA and LDA

Quadratic Discriminant Analysis (QDA)

Again standard QDA analysis was not possible with the complete dataset; so the analysis was on the 11 PCs which account for over 95% of the variance. However, PCA-QDA did not yield a particularly successful result, with just a 92% accuracy rate after LOO cross-validation on the whole dataset. A stepwise variable selection using the QDA method gave 11 variables to use instead of the seven before, and some variables increased in importance such as propaganda which was the second most crucial variable found in the step iterations. QDA analysis using the stepwise variable subset gave a new CV accuracy rate of 94%, which is an improvement. Comparing these two QDA approaches using ROC curves, and QDA with the subset of stepwise variables gives a higher AUC, and so is the preferred method.

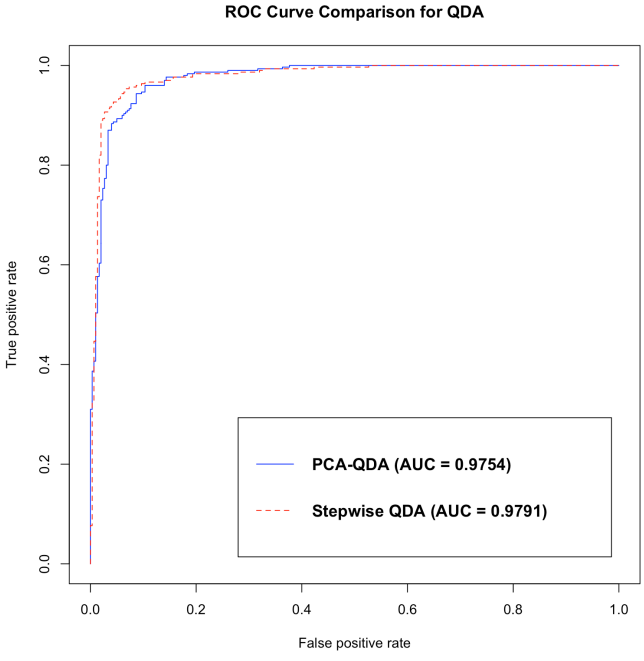

Figure 11 - ROC curves for QDA models

Comparison of methods

The best classifier from the discriminant analyses was the LDA with stepwise in figure 9, which has the highest AUC compared to QDA with stepwise, however. At the same time, this is quite a general measure, and it is clear that there are thresholds for when this does not hold. A comparison of their respective cost curves can illustrate the thresholds clearly for when one classifier outperforms the other (Drummond & Holte, 2004, p. 1). Figure 11 shows that QDA with stepwise has a lower expected cost for misclassification; however, when the probability cost function is greater than 0.5, LDA tends to outperform QDA. Thus comparison here is complicated, but either method can successfully classify between humans and phages.

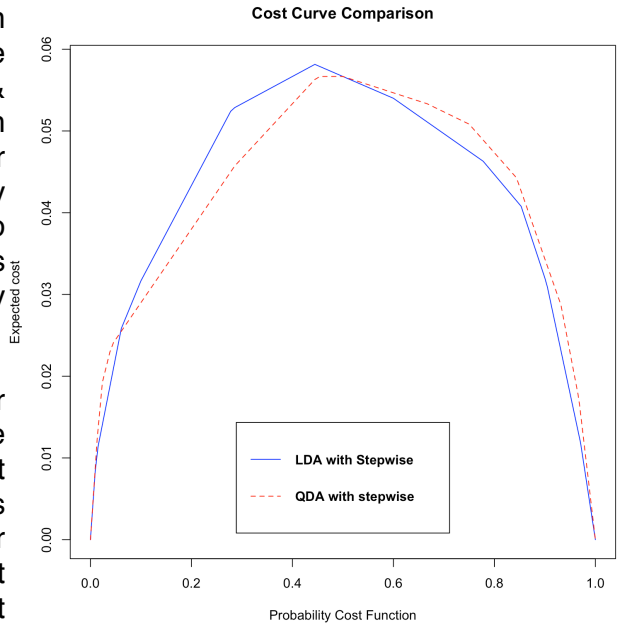

Figure 12 - cost curve comparison

As mentioned before, the random forest classifier was the most successful ensemble classifier since it was the only model that was least likely to overfit the data due to its error for test classification was less than the OOB. Contrasting the other ensemble classifiers, which have lower (albeit marginal) accuracy rates for their test set compared to their training set.

Conclusion

This project sought to classify DNA sequences as either human or phage for further study on their interactions with each other. To simplify sequences, certain features were extracted; for example, the proportion of CGs in the sample, which made the sequences much easier to use machine learning techniques on. This report focussed on a comparison of a random forest classifier, which gave an OOB of 7.38%. The OOB is roughly comparable to the error in the classification of LDA and QDA (both with stepwise). The LOO CV error for both LDA and QDA is ~6%; therefore, a random forest may not be the best classifier in distinguishing between humans and phages. QDA analysis with the stepwise selection of variables tends to produce a lower cost for misclassification. Overall, LDA does have a higher AUC and therefore is the best classifier for these sequences in that respect.

Works Cited

Bansal, M. (2003). DNA structure: Revisiting the Watson-Crick double helix. *Current Science, Volume 85*, 1556-1563.

Birdsell, J. A. (2002). Integrating Genomics, Bioinformatics and Classical Genetics to Study the Effects of Recombination on Genome Evolution. *Molecular Biology and Evolution, Volume 19*, 1181-1197.

Drummond, C., & Holte, R. C. (2004). *What ROC Curves Can't Do (and Cost Curves Can).* Ontario: Institute for Information Technology, National Research Council Canada.

Probst, P., & Boulesteix, A.-L. (2018). To Tune or Not to Tune the Number of Trees in Random Forest. *Journal of Machine Learning Research*, 1-18.

Siddiquee, M. A., & Tasnim, H. (2018). *A Comprehensive Study of Decision Trees to Classify DNA Sequence.* University of New Mexico.

Zhang, M. Q. (2000). Discriminant analysis and its application in DNA sequence motif recognition. *Briefings in Bioinformatics, Volume 1*, 331-342.

Code from R

```
phagedata = read.table("human-phage.txt")
phagedata[1:10, 1:10]

# ////////    ///////// CREATION OF NEW VARIABLES WITH LOOPS ///////////
numAs = vector(mode = "integer", length = nrow(phagedata))
for (i in 1:nrow(phagedata)){
  numAs[i] = length(which(phagedata[i,] == "A"))
}
numAs
#loop for counting all As in observations

numTs = vector(mode = "integer", length = nrow(phagedata))
for (i in 1:nrow(phagedata)){
  numTs[i] = length(which(phagedata[i,] == "T"))
}
numTs
#loop for counting all Ts in observations

numGs = vector(mode = "integer", length = nrow(phagedata))
for (i in 1:nrow(phagedata)){
  numGs[i] = length(which(phagedata[i,] == "G"))
}
numGs
#loop for counting all Gs in observations

numCs = vector(mode = "integer", length = nrow(phagedata))
for (i in 1:nrow(phagedata)){
  numCs[i] = length(which(phagedata[i,] == "C"))
}
numCs
#loop for counting all Gs in observations

numAAs = vector(mode = "integer", length = nrow(phagedata))
for (i in 1:nrow(phagedata)){
  for (j in 1:(ncol(phagedata)-1)){
    numAAs[i] = numAAs[i] + ((phagedata[i,j] == "A") & (phagedata[i,j+1] == "A"))
  } }
numAAs
```

```
numATs = vector(mode = "integer", length = nrow(phagedata))
for (i in 1:nrow(phagedata)){
  for (j in 1:(ncol(phagedata)-1)){
    numATs[i] = numATs[i] + ((phagedata[i,j] == "A") & (phagedata[i,j+1] == "T"))
  } }
numATs

numTAs = vector(mode = "integer", length = nrow(phagedata))
for (i in 1:nrow(phagedata)){
  for (j in 1:(ncol(phagedata)-1)){
    numTAs[i] = numTAs[i] + ((phagedata[i,j] == "T") & (phagedata[i,j+1] == "A"))
  } }
numTAs

numAGs = vector(mode = "integer", length = nrow(phagedata))
for (i in 1:nrow(phagedata)){
  for (j in 1:(ncol(phagedata)-1)){
    numAGs[i] = numAGs[i] + ((phagedata[i,j] == "A") & (phagedata[i,j+1] == "G"))
  } }
numAGs

numACs = vector(mode = "integer", length = nrow(phagedata))
for (i in 1:nrow(phagedata)){
  for (j in 1:(ncol(phagedata)-1)){
    numACs[i] = numACs[i] + ((phagedata[i,j] == "A") & (phagedata[i,j+1] == "C"))
  } }
numACs

numTTs = vector(mode = "integer", length = nrow(phagedata))
for (i in 1:nrow(phagedata)){
  for (j in 1:(ncol(phagedata)-1)){
    numTTs[i] = numTTs[i] + ((phagedata[i,j] == "T") & (phagedata[i,j+1] == "T"))
  } }
numTTs

numTGs = vector(mode = "integer", length = nrow(phagedata))
for (i in 1:nrow(phagedata)){
  for (j in 1:(ncol(phagedata)-1)){
    numTGs[i] = numTGs[i] + ((phagedata[i,j] == "T") & (phagedata[i,j+1] == "G"))
  } }
numTGs

numGTs = vector(mode = "integer", length = nrow(phagedata))
for (i in 1:nrow(phagedata)){
  for (j in 1:(ncol(phagedata)-1)){
    numGTs[i] = numGTs[i] + ((phagedata[i,j] == "G") & (phagedata[i,j+1] == "T"))
  } }
numGTs

numTCs = vector(mode = "integer", length = nrow(phagedata))
for (i in 1:nrow(phagedata)){
  for (j in 1:(ncol(phagedata)-1)){
    numTCs[i] = numTCs[i] + ((phagedata[i,j] == "T") & (phagedata[i,j+1] == "C"))
  } }
numTCs
```

```
numCTs = vector(mode = "integer", length = nrow(phagedata))
for (i in 1:nrow(phagedata)){
  for (j in 1:(ncol(phagedata)-1)){
    numCTs[i] = numCTs[i] + ((phagedata[i,j] == "C") & (phagedata[i,j+1] == "T"))
  } }
numCTs

numGCs = vector(mode = "integer", length = nrow(phagedata))
for (i in 1:nrow(phagedata)){
  for (j in 1:(ncol(phagedata)-1)){
    numGCs[i] = numGCs[i] + ((phagedata[i,j] == "G") & (phagedata[i,j+1] == "C"))
  } }
numGCs

numCGs = vector(mode = "integer", length = nrow(phagedata))
for (i in 1:nrow(phagedata)){
  for (j in 1:(ncol(phagedata)-1)){
    numCGs[i] = numCGs[i] + ((phagedata[i,j] == "C") & (phagedata[i,j+1] == "G"))
  } }
numCGs

numGGs = vector(mode = "integer", length = nrow(phagedata))
for (i in 1:nrow(phagedata)){
  for (j in 1:(ncol(phagedata)-1)){
    numGGs[i] = numGGs[i] + ((phagedata[i,j] == "G") & (phagedata[i,j+1] == "G"))
  } }
numGGs

numCCs = vector(mode = "integer", length = nrow(phagedata))
for (i in 1:nrow(phagedata)){
  for (j in 1:(ncol(phagedata)-1)){
    numCCs[i] = numCCs[i] + ((phagedata[i,j] == "C") & (phagedata[i,j+1] == "C"))
  } }
numCCs

#     ///////      ////////   DATA PREP ///////       /////////      ///////////////

output = phagedata$V1

singlephage = cbind(numAs, numCs, numGs, numTs)
doublephage = cbind(numAAs, numACs, numAGs, numATs, numCAs, numCCs, numCGs,
          numCTs, numGAs, numGGs, numGCs, numGTs, numTTs, numTAs, numTCs, numTGs)
#creating matrix with sequence counts
doublephageoutput = cbind(output, doublephage)
doublephageoutput = as.data.frame(doublephageoutput)
phageoutput = cbind(doublephageoutput, singlephage)
#creating data frames

corrvariables = cor(phageoutput[,1:21], use = "all.obs", method = "pearson")
heatmap3(corrvariables, rowV = NULL, colv = NULL, showColDendro = F, showRowDendro = F,
          scale = "none", revC = T, key.title = "Correlation coefficient")
#heatmap for variables

doublephagepca = prcomp(doublephageoutput[,2:17], scale = TRUE)
biplot(doublephagepca)
```

```
plot(doublephagepca$x[,1], doublephagepca$x[,2], xlab = "PC1", ylab = "PC2", main = "PCA
        scores plot", pch = 16, col = doublephageoutput$output)
#PCA with just double sequences
phageoutputpca = prcomp(phageoutput[,2:21], scale = TRUE)
plot(phageoutputpca$x[,1], phageoutputpca$x[,2], xlab = "PC1", ylab = "PC2", main = "PCA
        scores plot", pch = 16, col = phageoutput$output)
legend("bottomright", inset = 0.01, legend = c("Human", "Phage"), col =
        unique(phageoutput$output), pch = 16)

phageoutputpca$x[,1:2]
#PCA with double sequences and individual
#see step in the right direction, perhaps need some triples?

propcg = (numCs + numGs)/(numAs + numTs)
#proportion of CG/AT
propatot = numAs/numTs
#proportion of A/T
propgtoc = numGs/numCs
#proportion of G/C

newphagedata = cbind(phageoutput, propcg, propatot, propgtoc)
newphagedata$output = as.factor(newphagedata$output)

m = nrow(newphagedata)
trainIndex = sample(1:m, size = round(0.7*m), replace = FALSE)
traindata = newphagedata[trainIndex ,]
testdata = newphagedata[-trainIndex ,]
#splitting into independent training and test data sets

# //////////    //////////    ////////  PCA-LDA    ///////////    ///////////

newphagepca = prcomp(newphagedata[,2:24], scale = TRUE)
summary(newphagepca)
#since the variables are collinear we need to produce some PCs to perform PCA-LDA
#we find the first 11 PCs have 95% of the cumulative variance

ldamodel = lda(newphagepca$x[,1:11], newphagedata[,1], CV = T)
#PCA-LDA model
table(predicted = ldamodel$class, real = newphagedata[,1])
#accuracy rating of 0.905

pred_lda = ldamodel$posterior[,2]
rates_lda = prediction(pred_lda, newphagedata[,1])
perf_lda = performance(rates_lda, "tpr", "fpr")
plot(perf_lda, col = "blue", lty = 1, main = "ROC Curve of LDA")
#ROC curve for LDA

reallda = as.numeric(newphagedata[,1])
auc_lda = roc(reallda, as.numeric(pred_lda))
auc_lda
#AUC for LDA is 0.9706

stepmodel = stepclass(output~., data = newphagedata, method = "lda", direction = "forward",
        criterion = "AC", improvement = 0.001)
#stepwise variable selection
```

```
newvars = c("numCGs", "numCCs", "numACs", "propcg", "numGs", "numCs", "numTAs")
newdata = newphagedata[newvars]
#new dataset for stepwise variables

ldamodel2 = lda(newdata, newphagedata[,1], CV = T)
table(predicted = ldamodel2$class, real = newphagedata[,1])
#0.94 accuracy using the stepwise collection of variables

pred_lda2 = ldamodel2$posterior[,2]
rates_lda2 = prediction(pred_lda2, newphagedata[,1])
perf_lda2 = performance(rates_lda2, "tpr", "fpr")
plot(perf_lda2, col = "red", lty = 1, main = "ROC Curve for LDA with stepwise")
#ROC Curve for LDA with stepwise

reallda2 = as.numeric(newphagedata[,1])
auc_lda2 = roc(reallda2, as.numeric(pred_lda2))
auc_lda2
#AUC for LDA with stepwise is 0.9823

plot(perf_lda, col = "blue", lty = 1, main = "ROC Curve Comparison for LDA")
plot(perf_lda2, col = "red", add = TRUE, lty = 2)
legend("bottomright", inset = 0.05, legend = c("PCA-LDA (AUC = 0.9706)", "LDA with stepwise
       (AUC = 0.9823)"), col = c("blue", "red"), lty = 1:2, text.font = 2, cex = 1.3)
#ROC curve comparison for both LDA methods

# /////////    //////////    //////////  QDA  //////////    ///////////    //////////

qdamodel = qda(newphagepca$x[,1:11], newphagedata[,1], CV = T)
#qda model
table(predicted = qdamodel$class, real = newphagedata[,1])
#accuracy rating of 0.92

pred_qda = qdamodel$posterior[,2]
rates_qda = prediction(pred_qda, newphagedata[,1])
perf_qda = performance(rates_qda, "tpr", "fpr")
plot(perf_qda, col = "blue", lty = 1, main = "ROC Curve of QDA")
#ROC curve for QDA

realqda = as.numeric(newphagedata[,1])
auc_qda = roc(realqda, as.numeric(pred_qda))
auc_qda
#AUC for QDA is 0.9754

stepmodelqda = stepclass(output~., data = newphagedata, method = "qda", direction =
       "forward", criterion = "AC", improvement = 0.001)
#stepwise variable selection
newvars2 = c("numCGs", "propcg", "numGs", "numCs", "numTTs", "numGCs", "numAGs",
       "numGAs", "numACs", "numAs", "numCAs")
newdata2 = newphagedata[newvars2]
#new dataset

qdamodel2 = qda(newdata2, newphagedata[,1], CV = T)
table(predicted = qdamodel2$class, real = newphagedata[,1])
#accuracy 0.94

pred_qda2 = qdamodel2$posterior[,2]
```

```
rates_qda2 = prediction(pred_qda2, newphagedata[,1])
perf_qda2 = performance(rates_qda2, "tpr", "fpr")
plot(perf_qda2, col = "blue", lty = 1, main = "ROC Curve for QDA with stepwise")
#ROC curve for QDA with stepwise

realqda2 = as.numeric(newphagedata[,1])
auc_qda2 = roc(realqda2, as.numeric(pred_qda2))
auc_qda2
#AUC for QDA with stepwise is 0.9791

plot(perf_qda, col = "blue", lty = 1, main = "ROC Curve Comparison for QDA")
plot(perf_qda2, col = "red", add = TRUE, lty = 2)
legend("bottomright", inset = 0.05, legend = c("PCA-QDA (AUC = 0.9754)", "Stepwise QDA
        (AUC = 0.9791)"), col = c("blue", "red"), lty = 1:2, text.font = 2, cex = 1.2)
#ROC curve comparison for both QDA methods

costlda2 = performance(rates_lda2, 'ecost')
costqda2 = performance(rates_qda2, 'ecost')
plot(costlda2, col = "blue", lty = 1, main = "Cost Curve Comparison", xlab = "Probability Cost
        Function")
plot(costqda2, col = "red", add = TRUE, lty = 2)
legend("bottom", inset = 0.05, legend = c("LDA with Stepwise", "QDA with stepwise"), col =
        c("blue", "red"), lty = 1:2, text.font = 2, cex = 1)
#cost curve comparison for optimal LDA and QDA methods

# ///////    //////////    ////////// ENSEMBLE CLASSIFIERS /////////   /////////  ///////////

dna.tree = tree(output~., data = traindata)
plot(dna.tree)
text(dna.tree, cex = 0.75)

dna.tree

#creation of deciison tree
tree.pred = predict(dna.tree, traindata[,-1], type = "class")
table(predicted = tree.pred, true = traindata[,1])
#0.95 accuracy rating with training data
tree.pred = predict(dna.tree, testdata[,-1], type = "class")
table(predicted = tree.pred, true = testdata[,1])
#0.938 accuracy rating, possibly overfit so should prune

cv.dna = cv.tree(dna.tree, FUN = prune.misclass)
plot(cv.dna, cex = 1.2)
#the optimum number of branches is 4

prune.dna = prune.misclass(dna.tree, best = 2)
plot(prune.dna)
text(prune.dna, cex = 1.2)
#plot of new pruned decision tree

tree.pred = predict(prune.dna, traindata[,-1], type = "class")
table(predicted = tree.pred, true = traindata[,1])
#0.923 accuracy rating

tree.pred = predict(prune.dna, testdata[,-1], type = "class")
table(predicted = tree.pred, true = testdata[,1])
```

```
#0.911 accruacy, much less overfitted so is improved

rf = randomForest(output~., data = traindata, ntree = 200, mtry = 10, importance = TRUE)
rf
#random forest model, we get accuracy of 0.93
#OOB error for training set is 7.38%

rf.pred = predict(rf, testdata[,-1], type = "class")
table(predicted = rf.pred, true = testdata[,1])
#accuracy of 0.96 using the test data
#error for test set is ~6.66%, therefore the model is not overfitting

adaboost = boosting(output~., data = traindata, control = rpart.control(maxdepth = 1))
predboosting = predict.boosting(adaboost, newdata = testdata[,-1])
table(predicted = predboosting$class, true = testdata[,1])
#0.938 accuracy for test data

predboosting2 = predict.boosting(adaboost, newdata = traindata[,-1])
table(predicted = predboosting2$class, true = traindata[,1])
#0.947 accuracy rating for training data

#       ///////      /////  DISCARDED METHODS ////////  / /////////

#       ///////      ///// PLSR /////////      /////////      ///////
newphagedata.df = data.frame(X = I(as.matrix(newphagedata[,2:24])), Y =
        I(as.matrix(newphagedata[,1])))
#dataframe for PLS
plsrmodel = plsr(Y~X, data = newphagedata.df, ncomp = 20, scale = T, validation = "LOO",
        method = "kernelpls")
#PLSR model
plot(RMSEP(plsrmodel), legendpos = "topright")
selectNcomp(plsrmodel, plot = T)
#we find the min number of components is 10
plsrmodel = plsr(Y~X, data = newphagedata.df, ncomp = 10, scale = T, validation = "LOO",
        method = "kernelpls")
#PLSR model with optimal no. of components
biplot(plsrmodel, comps = 1:2, which = "scores", cex = 0.6)
#biplot of X scores and Y scores
biplot(plsrmodel, comps = 1:2, which = "y", cex = 0.6)
#biplot of Y scores and Y loadings

plot(plsrmodel$scores[,1], plsrmodel$scores[,2], xlab = "PLS1", ylab = "PLS2", pch = 20,
        col=as.factor(newphagedata[,1]))

n = nrow(newphagedata.df)
trainIndex2 = sample(1:n, size = round(0.7*n), replace = FALSE)
plstraindata = newphagedata.df[trainIndex2, ]
plstestdata = newphagedata.df[-trainIndex2,]
#training and test data sets using the df

plstrain = plsr(Y~X, data = plstraindata, ncomp = 10, scale = T, validation = "LOO", method =
        "kernelpls")
#pls model for training data
pred = predict(plstrain, ncomp = 10, newdata = plstestdata)
predclass = as.integer(pred + 0.5)
table(predicted = predclass, true = plstestdata$Y)
```

```
#for the test data we have accruacy of 92%

pred = predict(plstrain, ncomp = 10, newdata = plstraindata)
predclass = as.integer(pred + 0.5)
table(predicted = predclass, true = plstraindata$Y)
#for the training data we have accuracy of 93%, poss overfit but not drastic

# ////////      //////////    NEURAL NETWORKS /////////     ////////

nnmodel = nnet(output~., data = traindata, size = 6)
nnpred = predict(nnmodel, testdata[,-1], type = "class")
table(predicted = as.factor(nnpred), true = testdata[,1])
#standard neural network model

test.acc=double(10)
for(i in 1:10)
{
  nnmodel=nnet(output~., data=traindata, size=6)
  nnpred= predict(nnmodel, testdata[,-1], type = "class")
  cm=table(predicted = as.factor(nnpred), true = testdata[,1])
  test.acc[i]=(cm[1,1]+cm[2,2])/dim(testdata)[1]
  }
#loop for 10 neural network models
test.acc
#accuracy for each NN model
sd(test.acc)
#sd for the 10 models

numFolds = trainControl(method = 'cv', number = 10, savePredictions = T)
nnmodel2 = train(output~., data = traindata, method = 'nnet', preProcess = c('center', 'scale'),
          trControl = numFolds)
nnmodel2

grid = expand.grid(size = 5, decay = 0.1)
nnmodel3 = train(output~., data = traindata, method = 'nnet', maxit = 100, preProcess =
          c('center', 'scale'), trControl = numFolds, tuneGrid = grid)
nnmodel3$results
#NN model with cross validation
table(predicted = nnmodel3$pred[,1], real = nnmodel3$pred[,2])
#accuracy of 0.945 with train data

nnpred = predict(nnmodel3, testdata[,-1])
table(predicted = nnpred, real = testdata[,1])
#0.92 with test data, clearly overfitted
```