A Comparison of Classification Methods on the Avila Bible Dataset

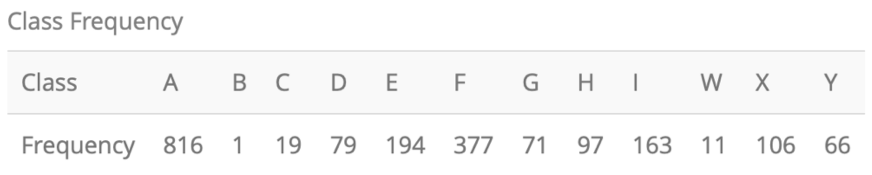
Robert Edwards
2416963E

1. INTRODUCTION

Palaeography is a field that studies analyses ancient and medieval texts to infer metadata such as when and where the text was written, determine the number of scribes, and the workload of each. Traditionally, Palaeographers manually analysed the texts to identify the unique writing patterns of a scribe or group of scribes, see [1] and references included. Modern techniques use computers to extract quantitative information from digitalized versions of ancient texts. The "Avila" Bible is a Latin copy of the Bible dating back to the 12th century that has been digitized via imaging. A dataset has been created from 800 images of the "Avila" Bible with 10 measurable features that are used to distinguish one of 12 copyists to each unique pattern, see [1] for further details. This project analyses a subset of 2000 observations created from "Avila" dataset to classify a pattern, or observation, to a copyist using classification methods covered in the course.

2. DATA

Before fitting models an attempt is made to understand the data through summary statistics and visualisation. The frequency of each class is shown in *Table 1*. There are three classes (A, B, C) that have very low frequencies and may need to be excluded from some models depending on the random sampling of the training, validation, and test subsets. Only class A and class F have a large number of observations. The low observations in the other classes may affect the variability of some methods.



Class Frequency

Class	A	B	C	D	E	F	G	H	I	W	X	Y
Frequency	816	1	19	79	194	377	71	97	163	11	106	66

Table 1: Frequencies for each class. Class B, class C, and class W are of note because they have low counts. These classes may need to be exempted from some models.

The features extracted from the "Avila" Bible are visualised in *Figure 1*. At first glance, all the features seem distributed around 0 and have similar interquartile ranges, with the exception of feature F5. The first set (F1, F2, F3) has many outliers and are heavily right-skewed. The features in the first set do not appear normal and have high variability, a more flexible method may have higher accuracy classifying these data. The second set (F4, F5, F6, F7) has less variability than the first set but there are a number of outliers. Features F4 and F6 seem normally distributed whereas features F5 and F7 are skewed left. The third set of features (F8, F9, F10) all have a number of outliers but are symmetric and appear normally distributed. All the features are distributed around 0 and seem to be standardized. According to the source of this dataset [2], the data have been normalized using Z-normalization. No further normalizing or standardizing of the data is done for the purposes of this report.

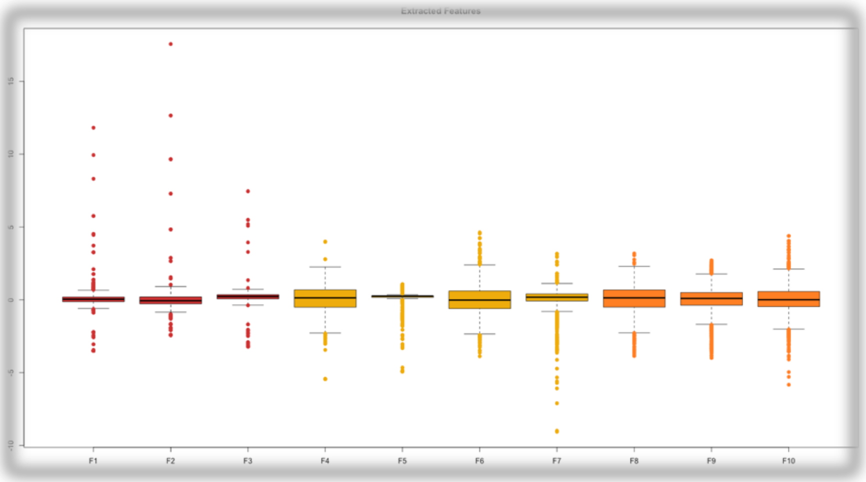


Figure 1: Boxplot of extracted features from the Avila Bible.

Another important way to consider the data is via a pairs plot showing the data as described by a feature against each other feature. Pairs plots are good for finding interactions between features; which are highly correlated and which are uncorrelated. The pairs plot in *Figure 2* shows a strong positive correlation between features F6 and F10 and also between features F8 and F9. The interactions between the other features look to be either uncorrelated or have a complex interactions. The diagonal of the pairs plot shows histograms for each feature. Features F4, F6, F8, F9, and F10 seem to have a normal or close to normal distribution. Methods that assume normality may have a higher accuracy with these features.

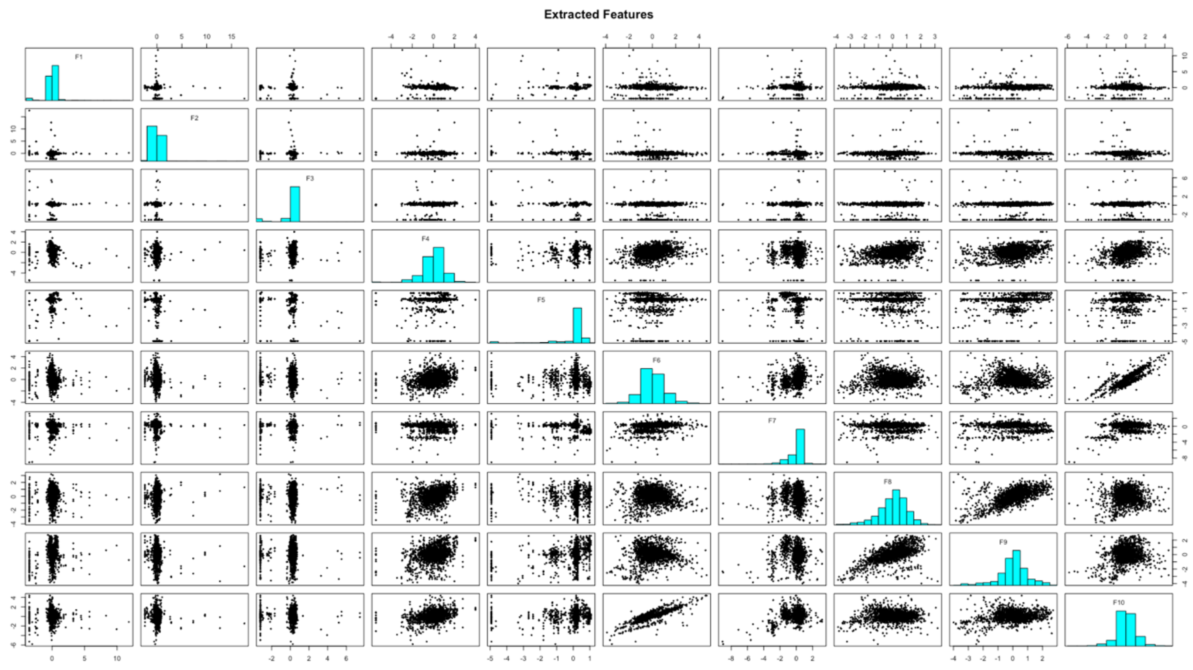


Figure 2: Pairs plot of extracted features from "Avila" Bible

3. PROCEDURE

For each of the seven methods explored the data are randomly split into three subsets: training (50%), validation (25%), and testing (25%). A classification model is created using the training data and a prediction of the validation data is created using the models. A cross-classification table is created from the predicted classifications and the true classifications of the validation data and the error rate is calculated for each model. Before model selection, the training and validation data are randomly sampled 100 times and the overall error rates for each model are averaged. The best model is then chosen by the smallest overall average error rate of the seven methods. The trained models are then tested on the test set and the expected future error rates are calculated. Error rates for each class are also calculated in an attempt to identify the model that most accurately predicts each class.

4. METHODS

Seven methods were explored for classifying the Avila Bible data: Linear Regression, K Nearest Neighbours, Canonical Variate Analysis, Linear Discriminant Analysis, Quadratic Discriminant Analysis, Classification Trees, and Random Forests. A brief description of each method is given including the strengths, weaknesses, limitations, and implementation.

Linear Regression

Linear regression is a widely used approach for classification of a quantitative response because it is simple and results are interpretable. Linear Regression is a parametric model that assumes the regression function parameters are linear, statistical independence of the error, homoscedasticity, and normality of the error distribution. Many of the assumptions for linear regression are not valid for real world datasets but linear regression serves as a good baseline classification method. Many of the other methods explored in this report and in the course text [3,4] are extensions of linear regression. To use linear regression to classify the "Avila" Bible features to one of the 12 copyists, categorical response variables, each class is assigned to an integer; the copyists are assigned 1 for copyist A through 12 for copyist Y. Prediction values are then rounded to the closest whole integer in the range of classifiers.

KNN

A more flexible technique for classifying data is K-Nearest Neighbors (KNN). The method estimates the class of an observation by looking at the surrounding observations. More formally, KNN is a nonparametric approach that identifies the K observations in the training set that are closest to the observation being classified. An estimate for the conditional probability of each class is calculated and the observation is assigned to the class with the highest probability, see the class text [3,4]. KNN is a nonparametric method that works well for even small numbers of observations. Since the distance between observations is calculated, the data should be scaled. There are a number of tuning parameters for KNN. An attempt is made to optimize KNN by training models with a range of K values, various types of distance, and various kernel types. The kknn library is used for modelling because it has options for tuning the three parameters being considered. With some classes having very low numbers of observations, KNN with high

K values are not expected to perform well. Additionally, knowing that KNN performs worse as the number of dimensions increase, see class text [3,4]. KNN models may not perform well if all 10 features are considered for modelling.

Discriminant Analysis

Discriminant analysis involves slightly more abstract technique that classifies observations by first transforming the data in a way that maximizes the ratio of group means to group variance to find the optimal separation between classes. The three versions explored in this paper are Canonical Variate Analysis (CVA), Linear Discriminant Analysis (LDA), and Quadratic Discriminant Analysis (QDA). CVA, also known as Fisher's Discriminant Analysis, does not assume a distribution for the data but it does assume equal prior probabilities. CVA works by finding ratios of variance between classes to variance within classes and assigning the observation to the class with the highest ratio. If a normal distribution is assumed then LDA and QDA can be used and the prior probabilities can be estimated as the proportion of observations belonging to each class, whereas with CVA the prior probabilities were assumed equal, see [1,2] for further details. Using Bayes Discriminant Rule the posterior probabilities for each class and the observation is classified to the class with the highest posterior probability. CVA and LDA both assume equal covariance matrices and Bayes Discriminant Rule simplifies to a linear function. Since QDA assumes unequal covariance matrix the Bayes Discriminant Rule simplifies to a quadratic function. QDA must estimate separate covariance matrices for each group and must have more observations than predictors for it to be used, see [5] for more details. LDA and QDA are both parametric models as they assume a multivariate normal distribution for the data in each class. There is a tradeoff between bias and variance in these models as more is assumed about the data. Discriminant analysis works well as separation between classes increases. LDA and QDA also work well with a small number of observations.

Trees

Classification Trees are a nonparametric technique that work well when the data is nonlinear and not normal. Classification Trees are grown via recursive binary splitting using a measure of the entropy as the split criterion. An observation is assigned to the class of the most abundant class of training observations in the same region. If the data has been overfitted in training then the tree can be pruned to reduce variability at the cost of increased bias. Trees are simple to explain and interpret but are not robust, in that a variability in data can grow a very different tree.

Random Forests

One method of reducing variability in a tree is to use a technique called Bagging. Many different training sets are sampled, unpruned trees are grown from the trainings sets, and the resulting predictions are averaged. Bagging tries to reduce variability by averaging many different trees but if there are particularly strong predictors then the trees can be highly correlated. The strong predictors tend to be chosen as the criterion split and the trees are grown similarly. The Random Forests method attempts to decorrelate trees by limiting the number of predictors considered at each split allowing less strong predictors to be considered. If there are a number of correlated predictors, then only a small number of predictors should be considered at each split. The default setting of the square root of the number of parameters is used to build the Random Forests. There are many more methods that can increase the accuracy of classification trees, see [3,4].

5. RESULTS

Before comparing against the other methods some parameters of K Nearest Neighbours are tuned using the training data. Error rates were determined for a range of K values, types of distance, and kernel types. Odd K values from 1 to 19 are used to give a range of potential estimates that tradeoff between increased bias and decreased variability as the value of K increases. Four distances were evaluated using the Minkowski distance parameter in the kknn package: 1, 2, and 10 corresponding to Manhattan distance, Euclidean distance, and Chebyshev distance. Additionally, a distance parameter of 0.1 is considered. Lastly, four kernel types are considered: rectangular, triangular, gaussian, and optimal. The set of parameters with the smallest error rate were a K value of 11, a distance measure of 10, and a rectangular kernel with an error rate of 0.516. *Figure 3* shows the outputs for tuning the KNN model.

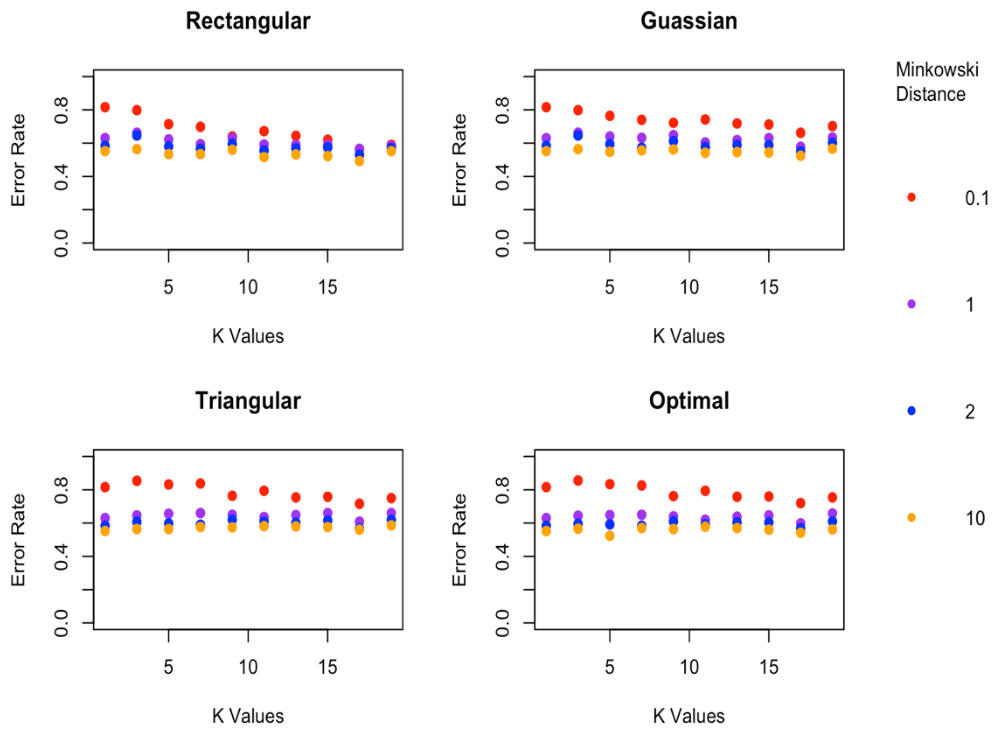


Figure 3: Optimization of K-Nearest Neighbours by finding the smallest error rate for various parameters. K values are odd integers ranging from 1 to 19. Minkowski distances are 0.1, 1, 2, and 10. The four kernel types are rectangular, triangular, gaussian, and optimal.

After tuning the KNN method, all the methods were trained and cross validated for 100 different random samplings of the "Avila" Bible dataset. The error rates for the 100 cross validations are shown in *Figure 4*. During this process classes were excluded from the training and validation sets if there were no observations or if the observations were less than 10 when training QDA.

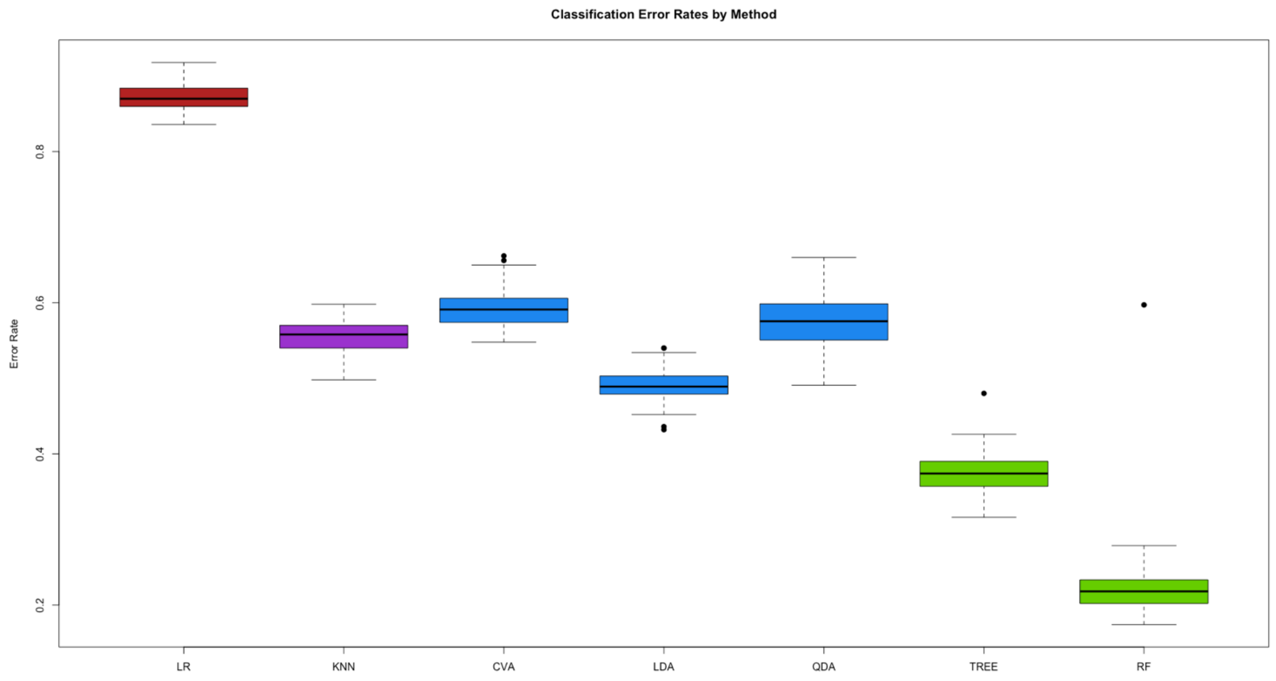


Figure 4: Error rates from 100 random samplings of training and validation data.

Methods such as Classification Trees and Random Forests trade off high variability for reduced bias. *Figure* 4 shows outliers in CVA, LDA, Classification Trees, and Random Forests. By averaging the error rates calculated from cross validating training data from 100 random samples, the variability of each method is reduced. Random Forests has the lowest overall average classification rate and is selected as the optimal model to use on the test data. All trained methods are tested on the test set to find the expected prediction error rate on future datasets. The overall classification rates are shown in *Table* 2 after the models were cross validated on the test set. Random Forests has the lowest error rate at 0.238 and should be used as the expected overall error rate for classification of future predictions.

LR	KNN	CVA	LDA	QDA	TREE	RF
0.844	0.538	0.586	0.48	0.6369	0.392	0.238

Table 2: Overall test error rates by class.

If patterns are to be classified on a class-by-class basis then the method with the lowest error rate for each class can be used instead of the method with the overall lowest error rate. Some interesting results can be seen in *Table 3*, namely that no method is particularly reliable at predicting classes B, C, and W due to low observations of those classes. Class G and class H are best predicted by QDA, suggesting the assumption of normality for those data is reasonable and can be modelled with a quadratic function. Linear Regression does not perform well on any of the classes suggesting complex nonlinear relationships between the predictors and the classes. This is not surprising from looking at boxplots and pairs plots in *Figure 1* and *Figure 2* of the data. Class I and Class Y are predicted very well by Random Forests with error rates under 10%. KNN was expected to perform better than more biased models but has about the same error rate as CVA, LDA, and QDA. The higher than expected error rate in KNN is likely due to the disproportionate number of classes in the observations and the relatively high number of predictors used in modelling.

	LR	KNN	CVA	LDA	QDA	TREE	RF
A	0.9845	0.4811	0.5440	0.1088	0.7306	0.1865	0.1088
B	1.0000	NaN	1.0000	NA	NA	1.0000	NA
C	0.6667	NaN	1.0000	1.0000	NA	1.0000	0.6667
D	0.6897	1.0000	0.8276	1.0000	0.7931	1.0000	0.7586
E	0.6383	0.5833	0.6596	0.9149	0.7234	0.3404	0.2553
F	0.9775	0.6163	0.6854	0.9213	0.7416	0.6966	0.4045
G	0.9412	1.0000	0.4118	0.9412	0.2353	1.0000	0.6471
H	1.0000	0.6500	0.5714	0.9643	0.2143	0.5714	0.3929
I	0.7826	0.1905	0.1739	0.1304	0.2174	0.0652	0.0217
W	1.0000	NaN	0.8000	1.0000	NA	1.0000	0.8000
X	1.0000	0.1000	0.3913	0.5652	0.5652	0.4783	0.2174
Y	1.0000	0.2500	0.2500	0.5000	0.1875	0.2500	0.0625

Table 3: Error rates by class for each method.

6. CONCLUSION

Seven classification methods were evaluated in the classification of copyists for a subset of the Avila Bible dataset. Due to the highly complex and non-normal relationships between predictors, more flexible methods performed better than parametric models. Random Forests performed the best with an overall classification rate of 0.238. If classification of a specific copyist were desired, then the method with the lowest error rate in *Table 3* can be used for prediction rather than Random Forests for increased accuracy. Namely, Classes G and H are best predicted by QDA. Classes B, C, and W were often taken out of the training and validation sets due to too few observations. Interestingly, KNN performs about average compared to other classification methods.

There are a number of techniques that are beyond the scope of this report that could further improve the prediction accuracy of the methods considered. Only a subset of 2000 observations were used in modelling. By using the full dataset of 20,000 observations available from the website [2] classes won't be removed as often when training the model and accuracy for those classes should increase. Another way to improve prediction accuracy is to perform dimension reduction on the predictors. Some classes could be best predicted by only a few predictors. By considering all the predictors, reduceable error is contained in the prediction estimates for the classes. Reducing the number of dimensions could improve accuracy of all models but particularly: Linear Regression, KNN, CVA, LDA, and QDA. The error rate of Classification Trees could be improved by boosting the trees. The methods considered in this report attempted to classify the observations to all classes. An attempt at reducing overfitting of more rigid models could be attained by predicting a singular class rather than all classes. In addition to finer tuning of the models, other methods such as Neural Networks, State Vector Machines, and Logistic Regression could be considered.

7. Bibliography

[1] De Stefano, C., Maniaci, M., Fontanella, F., Scotto di Freca, A. (2018) Reliable writer identification in medieval manuscripts through page layout features: The "Avila" Bible case. *Engineering Applications of Artificail Intelligence*, 72 pp. 99-110
[2] De Stefano, C., Maniaci, M., Fontanella, F., Scotto di Freca, A, "Avila Data Set", *UCI Machine Learning* Repository, archive.ics.uci.edu/ml/datasets/Avila
[3] James, G., Witten, D., Hastie, T., Tibshirani, R. (2013) *An Introduction to Statistical Learning* 1ed. Springer-Verlag New York.
[4] Hastie, T., Tibshirani, R., Friedman, J. (2009) *The Elements of Statistical Learning.* 2ed. Springer-Verlag New York.
[5] Hewson, P., (2009), *Multivariate Statistics with R*

```r
## Author: Robert Edwards
## University of Glasgow
## Multivariate Methods 2018
## Classification Project

setwd("~/OneDrive - University of Glasgow/University of Glasgow/Multivariate Methods/Project")
source("Methods.R")
library(class)
library(MASS)
library(e1071)
library(knitr)
library(kableExtra)

# Load data
avila <- Avila.48

## Creating a data frame with the Class outcome and the 12 explanatory variables
auth.lab <- avila[,11]
data.avila <- as.data.frame(cbind(auth.lab, avila[,-11]))
colnames(data.avila) <- c("Class", "F1", "F2", "F3", "F4", "F5", "F6", "F7", "F8", "F9", "F10")


##############################################
## Inspecting the Data
##############################################
colors1 <- c("brown3", "brown3", "brown3", "darkgoldenrod2", "darkgoldenrod2", "darkgoldenrod2", "darkgoldenrod2", "chocolate1", "chocolate1", "chocolate1")
boxplot(data.avila[-1], main="Extracted Features",
        col=colors1,
        pars=list(outcol=colors1))

kable(round(cov(data.avila[-1]), 4), caption="Covariance Matrix") %>%
  kable_styling(bootstrap_options = "striped",
                full_width = F,
                position = "center")

kable(round(cor(data.avila[-1]), 4), caption="Correlation Matrix") %>%
  kable_styling(bootstrap_options = "striped",
                full_width = F,
                position = "center")

class.count <- as.data.frame(table(data.avila[1]))
colnames(class.count) <- c("Class", "Frequency")
class.count <- class.count[c("Frequency", "Class")]

kable(t(class.count), caption="Class Frequency") %>%
  kable_styling(bootstrap_options = "striped",
                full_width = F,
                position = "center")

dddpanel.hist <- function(x, ...)
{
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(usr[1:2], 0, 1.5) )
  h <- hist(x, plot = FALSE)
  breaks <- h$breaks; nB <- length(breaks)
  y <- h$counts; y <- y/max(y)
  rect(breaks[-nB], 0, breaks[-1], y, col = "cyan", ...)
}
```

```r

pairs(data.avila[, -1], main="Extracted Features",
      diag.panel=panel.hist,
      pch=20)


##############################################
## Splitting the data frame into 3 sets: training, validation and test.
##############################################
n <- nrow(avila)
ind1 <- sample(c(1:n), round(n / 2))          # Training set
ind2 <- sample(c(1:n)[-ind1], round(n / 4))   # Validation set
ind3 <- setdiff(c(1:n), c(ind1, ind2))        # Test set
train.avila <- data.avila[ind1, ]
valid.avila <- data.avila[ind2, ]
test.avila <- data.avila[ind3, ]

table(train.avila$Class)
table(valid.avila$Class)
table(test.avila$Class)


##############################################
## Optimizing KNN
##############################################
rect.err.rates <- as.data.frame(matrix(data=NA, nrow=itr/2, ncol=5))
names(rect.err.rates) <- c("K", "dist=0.1", "dist=1", "dist=2", "dist=100")
tri.err.rates <- as.data.frame(matrix(data=NA, nrow=itr/2, ncol=5))
names(tri.err.rates) <- c("K", "dist=0.1", "dist=1", "dist=2", "dist=100")
gaus.err.rates <- as.data.frame(matrix(data=NA, nrow=itr/2, ncol=5))
names(gaus.err.rates) <- c("K", "dist=0.1", "dist=1", "dist=2", "dist=100")
opt.err.rates <- as.data.frame(matrix(data=NA, nrow=itr/2, ncol=5))
names(opt.err.rates) <- c("K", "dist=0.1", "dist=1", "dist=2", "dist=100")

itr <- 20
for (i in seq(1, itr, 2)) {
  set.seed(i)
  n <- nrow(avila)
  ind1 <- sample(c(1:n), round(n / 2))          # Training set
  ind2 <- sample(c(1:n)[-ind1], round(n / 4))   # Validation set
  ind3 <- setdiff(c(1:n), c(ind1, ind2))        # Test set
         train.avila <- data.avila[ind1, ]
         valid.avila <- data.avila[ind2, ]
         test.avila <- data.avila[ind3, ]

         rect.err.rates[ceiling(i/2),1] <- i
         rect.err.rates[ceiling(i/2),2] <- classify.knn(train.avila, valid.avila, k=i, dist=0.1, ker="rectangular") #
         rect.err.rates[ceiling(i/2),3] <- classify.knn(train.avila, valid.avila, k=i, dist=1, ker="rectangular") # Manhattan
         rect.err.rates[ceiling(i/2),4] <- classify.knn(train.avila, valid.avila, k=i, dist=2, ker="rectangular") # Euclidean
         rect.err.rates[ceiling(i/2),5] <- classify.knn(train.avila, valid.avila, k=i, dist=10, ker="rectangular") # Chebyshev's

         tri.err.rates[ceiling(i/2),1] <- i
         tri.err.rates[ceiling(i/2),2] <- classify.knn(train.avila, valid.avila, k=i, dist=0.1, ker="triangular") #
         tri.err.rates[ceiling(i/2),3] <- classify.knn(train.avila, valid.avila, k=i, dist=1, ker="triangular") # Manhattan
```

```
        tri.err.rates[ceiling(i/2),4] <- classify.knn(train.avila, valid.avila, k=i, dist=2, ker="triangular") # Euclidean
        tri.err.rates[ceiling(i/2),5] <- classify.knn(train.avila, valid.avila, k=i, dist=10, ker="triangular") # Chebyshev's

        gaus.err.rates[ceiling(i/2),1] <- i
        gaus.err.rates[ceiling(i/2),2] <- classify.knn(train.avila, valid.avila, k=i, dist=0.1, ker="gaussian") #
        gaus.err.rates[ceiling(i/2),3] <- classify.knn(train.avila, valid.avila, k=i, dist=1, ker="gaussian") # Manhattan
        gaus.err.rates[ceiling(i/2),4] <- classify.knn(train.avila, valid.avila, k=i, dist=2, ker="gaussian") # Euclidean
        gaus.err.rates[ceiling(i/2),5] <- classify.knn(train.avila, valid.avila, k=i, dist=10, ker="gaussian") # Chebyshev's

        opt.err.rates[ceiling(i/2),1] <- i
        opt.err.rates[ceiling(i/2),2] <- classify.knn(train.avila, valid.avila, k=i, dist=0.1, ker="optimal") #
        opt.err.rates[ceiling(i/2),3] <- classify.knn(train.avila, valid.avila, k=i, dist=1, ker="optimal") # Manhattan
        opt.err.rates[ceiling(i/2),4] <- classify.knn(train.avila, valid.avila, k=i, dist=2, ker="optimal") # Euclidean
        opt.err.rates[ceiling(i/2),5] <- classify.knn(train.avila, valid.avila, k=i, dist=10, ker="optimal") # Chebyshev's
      }

      par(mfcol=c(2,2), mar=c(4.1, 4.1, 4.1, 2), oma=c(0,0,0,6), xpd=NA)
      plot(rect.err.rates[,1], rect.err.rates[,2],
           main="Rectangular",
           xlab="K Values",
           ylab="Error Rate",
           ylim=c(0,1),
           col="red")
      points(rect.err.rates[,1], rect.err.rates[,3], col="purple")
      points(rect.err.rates[,1], rect.err.rates[,4], col="blue")
      points(rect.err.rates[,1], rect.err.rates[,5], col="orange")

      plot(tri.err.rates[,1], tri.err.rates[,2],
           main ="Triangular",
           xlab="K Values",
           ylab="Error Rate",
           ylim=c(0,1),
           col="red")
      points(tri.err.rates[,1], tri.err.rates[,3],col="purple")
      points(tri.err.rates[,1], tri.err.rates[,4],col="blue")
      points(tri.err.rates[,1], tri.err.rates[,5],col="orange")

      plot(gaus.err.rates[,1], gaus.err.rates[,2],
           main ="Guassian",
           xlab="K Values",
           ylab="Error Rate",
           ylim=c(0,1),
           col="red")
      points(gaus.err.rates[,1], gaus.err.rates[,3],col="purple")
      points(gaus.err.rates[,1], gaus.err.rates[,4],col="blue")
      points(gaus.err.rates[,1], gaus.err.rates[,5],col="orange")

      plot(opt.err.rates[,1], opt.err.rates[,2],
           main ="Optimal",
           xlab="K Values",
           ylab="Error Rate",
```

```r
            ylim=c(0,1),
            col="red")
      points(opt.err.rates[,1], opt.err.rates[,3],col="purple")
      points(opt.err.rates[,1], opt.err.rates[,4],col="blue")
      points(opt.err.rates[,1], opt.err.rates[,5],col="orange")
      leg <- c("0.1", "1", "2", "10")
      legend("topright", legend = leg, col=c("red","purple","blue","orange"), pch=16, inset=c(-0.75, -2), bty="n")
      #legend("topright", legend = leg, col=c("red","purple","blue","orange"),  pch=16, inset=c(-1.3, -4), bty="n")
      legend("topright", legend="Minkowski\nDistance", inset=c(-1.15, -3), bty="n")


      ############################################
      ## Model Selection
      ############################################
      set.seed(2)
      itr=1
      err.rates <- as.data.frame(matrix(data=NA, nrow=itr, ncol=8))
      names(err.rates) <- c("LR", "KNN", "CVA", "LDA", "QDA", "TREE", "RF")


      for (i in 1:itr) {
        n <- nrow(avila)
        ind1 <- sample(c(1:n), round(n / 2))          # Training set
        ind2 <- sample(c(1:n)[-ind1], round(n / 4))   # Validation set
        ind3 <- setdiff(c(1:n), c(ind1, ind2))        # Test set
        train.avila <- data.avila[ind1, ]
        valid.avila <- data.avila[ind2, ]
        test.avila <- data.avila[ind3, ]

        mcr <- classify.lr(train.avila, valid.avila)
        err.rates[i,1] <- mcr[1]
        err.rate.lr <- as.data.frame(unlist(mcr[-1]))
        mcr <- classify.knn(train.avila, valid.avila, k=11, dist=10, ker="rectangular")
        err.rates[i,2] <- mcr[1]
        err.rate.knn <- as.data.frame(unlist(mcr[-1]))
        mcr <- classify.cva(train.avila, valid.avila)
        err.rates[i,3] <- mcr[1]
        err.rate.cva <- as.data.frame(unlist(mcr[-1]))
        mcr <- classify.lda(train.avila, valid.avila)
        err.rates[i,4] <- mcr[1]
        err.rate.lda <- as.data.frame(unlist(mcr[-1]))
        mcr <- classify.qda(train.avila, valid.avila)
        err.rates[i,5] <- mcr[1]
        err.rate.qda <- as.data.frame(unlist(mcr[-1]))
        mcr <- classify.tree(train.avila, valid.avila)
        err.rates[i,6] <- mcr[1]
        err.rate.tree <- as.data.frame(unlist(mcr[-1]))
        mcr <- classify.rf(train.avila, valid.avila)
        err.rates[i,7] <- mcr[1]
        err.rate.rf <- as.data.frame(unlist(mcr[-1]))
      }

      err.rates
      boxplot(err.rates[-8], main="Classification Error Rates by Method",
              ylab="Error Rate",
              col=c("firebrick", "darkorchid3", "dodgerblue2", "dodgerblue2", "dodgerblue2", "chartreuse3", "chartreuse3"))

        kable(round(err.rates, 4)) %>%
          kable_styling(bootstrap_options = "striped", full_width = F, position = "cente
r")
          #row_spec(0, angle = -45)

        #####################################
        ## Model Selection
        ####################################
        test.err.rates <- as.data.frame(matrix(data=NA, nrow=itr, ncol=7))
        names(test.err.rates) <- c("LR", "KNN", "CVA", "LDA", "QDA", "TREE", "RF")
        mcr <- classify.lr(train.avila, test.avila)
        test.err.rates[1] <- mcr[1]
        mcr <- classify.knn(train.avila, test.avila, k=17, dist=10, ker="rectangular")
        test.err.rates[2] <- mcr[1]
        mcr <- classify.cva(train.avila, test.avila)
        test.err.rates[3] <- mcr[1]
        mcr <- classify.lda(train.avila, test.avila)
        test.err.rates[4] <- mcr[1]
        mcr <- classify.qda(train.avila, test.avila)
        test.err.rates[5] <- mcr[1]
        mcr <- classify.tree(train.avila, test.avila)
        test.err.rates[6] <- mcr[1]
        mcr <- classify.rf(train.avila, test.avila)
        test.err.rates[7] <- mcr[1]
        test.err.rates

        kable(round(test.err.rates, 4)) %>%
          kable_styling(bootstrap_options = "striped", full_width = F, position = "cente
r")

        #####################################
        ## Class Specific Error Rates
        ####################################
        err.rates.class <- cbind(err.rate.lr,
                                 err.rate.knn,
                                 err.rate.cva,
                                 err.rate.lda,
                                 err.rate.qda,
                                 err.rate.tree,
                                 err.rate.rf)
        rownames(err.rates.class) <- c("A","B","C","D","E","F","G","H","I","W","X","Y")

        colnames(err.rates.class) <- c("LR", "KNN", "CVA", "LDA", "QDA", "TREE", "RF")
        err.rates.class

        kable(round(err.rates.class, 4)) %>%
          kable_styling(bootstrap_options = "striped", full_width = F, position = "cente
r")
```

```
## Author: Robert Edwards
## University of Glasgow
## Multivariate Methods 2018
## All Classifying Methods

library(kknn)

#############################################
### Linear Regression
```

```r
##############################################
classify.lr <- function(train.data, valid.data) {

  # Change Class labels to numeric
  train.data <- as.data.frame(cbind(as.numeric(train.data[,1]), train.data[,-1]))
  valid.data <- as.data.frame(cbind(as.numeric(valid.data[,1]), valid.data[,-1]))

  print("as numeric train data")
  print(table(as.numeric(train.data[,1])))
  print(table(as.numeric(train.data[,1]))[1])
  colnames(train.data) <- c("Class", "F1", "F2", "F3", "F4", "F5", "F6", "F7", "F8", "F9", "F10")
  colnames(valid.data) <- c("Class", "F1", "F2", "F3", "F4", "F5", "F6", "F7", "F8", "F9", "F10")


  ## Fitting the linear model to the training data.
  res <- lm(Class~.,data=train.data)

  ## Using the predict function to predict for the validation data.
  pred.valid <- predict(res, valid.data[, -1])

  ## Changing the predictions to predited labels.
  pred.valid.label <- round(pred.valid)

  pred.valid.label[pred.valid.label < 1] <- 1
  pred.valid.label[pred.valid.label > 12] <- 12


  ## Cross-classification table.
  xtab <- table(valid.data[, 1], pred.valid.label)

  ## Total Correctly Classiffied
  corr.class.total <- sum(pred.valid.label == valid.data[, 1])
  ## Correct classification rate (CCR).
  corr.class.rate <- sum(pred.valid.label == valid.data[, 1]) / length(pred.valid.label)
  ## Class-specific CCRs.
  ccr.class <- diag(xtab) / rowSums(xtab)

  lr.err <- 1 - sum(diag(xtab)) / sum(xtab)

  errors <- list(lr.err, 1-ccr.class)
  return(errors)
} # End classify.lr


##############################################
### KNN
##############################################
classify.knn <- function(train.data, test.data, k=1, dist=2, ker="rectangular") {

  n.test <- nrow(test.data)

  k.nn.tr <- kknn(formula=formula(Class~.), train=train.data, test=train.data[-1], k=k, distance=dist, kernel=ker)
  k.nn.tst <- kknn(formula=formula(Class~.), train=train.data, test=test.data[-1], k=k, distance=dist, kernel=ker)

  ## Class-specific CCRs.
  xtab <- table(k.nn.tst$fitted.values, test.data[, 1])
  ccr.class <- diag(xtab) / rowSums(xtab)
  ## Calculate and store the test set correct classification rate
  knn.err <- sum(k.nn.tst$fitted.values == test.data[, 1]) / n.test

  errors <- list(knn.err, 1-ccr.class)
  return(errors)
} # end KNN


##############################################
### CVA
##############################################
classify.cva <- function(train.data, valid.data) {
  ## Set Prior probabilites equal
  data.lda <- lda(train.data[, -1], train.data[, 1], prior=rep(1/nlevels(train.data[, 1]), 12) )


  ## predict the labels based on the model.
  data.valid.ld <- predict(data.lda, newdata=valid.data[, -1])

  ## cross-classification table
  xtab <- table(valid.data[, 1], data.valid.ld$class)

  ## Total Correctly Classiffied
  corr.class.total <- sum(diag(xtab))
  ## Correct Classification Rate
  corr.class.rate <- sum(diag(xtab)) / sum(xtab)
  ## Misclassification rate =  1 - correct classifcation rate
  mcr <- 1 - sum(diag(xtab)) / sum(xtab)

  ## Class-specific CCRs.
  ccr.class <- diag(xtab) / rowSums(xtab)

  ## Error Rate
  cva.err <- 1 - sum(diag(xtab)) / sum(xtab)

  errors <- list(cva.err, 1-ccr.class)
  return(errors)
} #end classify.cva



##############################################
## LDA
##############################################
classify.lda <- function(train.data, valid.data) {

  ## Check if n > 0 and remove those classes
  if ( sum(table(train.data$Class) < 1 ) != 0) {
    too.few <- levels(train.data$Class)[table(train.data$Class) < 1]
    cat("Removing from train data: ", too.few, "\n")

    for (i in 1:length(too.few)) {
      train.data <- subset(train.data, Class != too.few[i]) # if n < 0 for each class
      valid.data <- subset(valid.data, Class != too.few[i]) # if n < 0 for each class
    }

    train.data$Class <- factor(train.data$Class)
    valid.data$Class <- factor(valid.data$Class)
  }
  if ( sum(table(valid.data$Class) < 1 ) != 0) {
    too.few <- levels(valid.data$Class)[table(valid.data$Class) < 1]
    cat("Removing from valid data: ", too.few, "\n")

    for (i in 1:length(too.few)) {
      train.data <- subset(train.data, Class != too.few[i]) # if n < 0 for each class
      valid.data <- subset(valid.data, Class != too.few[i]) # if n < 0 for each class
    }
    train.data$Class <- factor(train.data$Class)
    valid.data$Class <- factor(valid.data$Class)
  }


  ## Let lda set priors
  data.lda <- lda(Class~., data=train.data)

  ## Predicting the classes on the data.
  data.valid.pred.lda<- predict(data.lda, valid.data)

  ## Constructing a cross-classification table of the real versus predicted classifications.
  xtab<-table(valid.data$Class, data.valid.pred.lda$class)

  ## Class-specific CCRs.
  ccr.class <- diag(xtab) / rowSums(xtab)

  ## Calculating the misclassification rate.
  lda.err <- 1 - sum(diag(xtab)) / sum(xtab)

  errors <- list(lda.err, 1-ccr.class)
  return(errors)
} # end classify.lda


############################################
## QDA
############################################
classify.qda <- function(train.data, valid.data) {

  ## Check if n < p and remove those classes
  if ( sum(table(train.data$Class) <= ncol(train.data[-1]) ) != 0) {
    too.few.train <- levels(train.data$Class)[table(train.data$Class) <= ncol(train.data[-1])]
    cat("Removing from train data: ", too.few.train, "\n")
  }
  if ( sum(table(valid.data$Class) <= ncol(valid.data[-1]) ) != 0) {
    too.few.valid <- levels(valid.data$Class)[table(valid.data$Class) <= ncol(valid.data[-1])]
    cat("Removing from train data: ", too.few.valid, "\n")
  }

  too.few <- c(too.few.train, too.few.valid)

  if ( length(too.few) != 0) {
    for (i in 1:length(too.few)) {
      train.data <- subset(train.data, Class != too.few[i]) # if n < p for each class
      valid.data <- subset(valid.data, Class != too.few[i]) # if n < p for each class
    }
    train.data$Class <- factor(train.data$Class)
    valid.data$Class <- factor(valid.data$Class)
  }

  data.qda <- qda(Class~., data=train.data)

  ## Prediction
  data.valid.pred.qda <- predict(data.qda, valid.data)

  ## Constructing a cross-classification table of the real versus predicted classifications.
  xtab <- table(valid.data$Class, data.valid.pred.qda$class)

  ## Class-specific CCRs.
  ccr.class <- diag(xtab) / rowSums(xtab)

  ## Calculating the error rate.
  qda.err <- 1 - sum(diag(xtab)) / sum(xtab)

  errors <- list(qda.err, 1-ccr.class)
  return(errors)
} # end classify.qda


################################################
## TREES
################################################
classify.tree <- function(train.data, valid.data) {

  set.seed(126)
  library(rpart)
  ## Set"class" to ensure we get a classification
  data.rp <- rpart(Class~. , data=train.data, method="class")

  ## We predict the classifications based on the fitted tree using the predict command with the argument type set to "class", otherwise it returns a matrix of posterior probabilities of each obseration belonging to each class.
  data.valid.rp.pred <- predict(data.rp, valid.data, type="class")

  ## Constructing a cross-classification table of the real versus predicted classifications.
  xtab <- table(valid.data$Class, data.valid.rp.pred)

  ## Taking a look at the posterior probabilities for the observations we got the prediction wrong for.
  data.valid.rp.prob <- predict(data.rp, valid.data)
  data.valid.rp.prob[valid.data$Class != data.valid.rp.pred, ]

  ## Class-specific CCRs.
  ccr.class <- diag(xtab) / rowSums(xtab)
  ## Calculating the misclassification rate.
  tree.err <- 1 - sum(diag(xtab)) / sum(xtab)

  errors <- list(tree.err, 1-ccr.class)
  return(errors)
} # end classify.tree
```

```r
################################################
## RANDOM FORESTS
################################################
classify.rf <- function(train.data, valid.data) {
  library(randomForest)
  set.seed(126)

  too.few <- NA
  ## Remove empty classes
  if ( sum(table(train.data$Class) < 1 ) > 0) {
    too.few <- levels(train.data$Class)[table(train.data$Class) < 1]
    cat("Removing from train data: ", too.few, "\n")

    for (i in 1:length(too.few)) {
      train.data <- subset(train.data, Class != too.few[i]) # if n < p for each class
      valid.data <- subset(valid.data, Class != too.few[i]) # if n < p for each class
    }
    train.data$Class <- factor(train.data$Class)
    valid.data$Class <- factor(valid.data$Class)
  }
  if ( sum(table(valid.data$Class) < 1 ) > 0) {
    too.few <- NA
    too.few <- levels(valid.data$Class)[table(valid.data$Class) < 1]
    cat("Removing from valid data: ", too.few, "\n")

    for (i in 1:length(too.few)) {
      train.data <- subset(train.data, Class != too.few[i]) # if n < p for each class
      valid.data <- subset(valid.data, Class != too.few[i]) # if n < p for each class
    }
    train.data$Class <- factor(train.data$Class)
    valid.data$Class <- factor(valid.data$Class)
  }


  data.rf <- randomForest(Class~., data=train.data)

  ## Prediction
  data.valid.rf.pred <- predict(data.rf, newdata=valid.data)

  ## Cross classification
  xtab <- table(valid.data$Class, data.valid.rf.pred)

  ## Class-specific CCRs.
  ccr.class <- diag(xtab) / rowSums(xtab)
  ## Calculating the misclassification rate.
  rf.err <- 1 - sum(diag(xtab)) / sum(xtab)

  errors <- list(rf.err, 1-ccr.class)
  return(errors)
} # end classify.rf
```