# Statistical Pattern Recognition M3S7: Project #3

**Aseem Sharma**

# Contents

# Section 1: A brief description of the data

| Dimension Number | Min | 1st Quartile | Median | Mean | 2nd Quartile | Max | NAs | Standard Deviation |
|---|---|---|---|---|---|---|---|---|
| 1 | -116.100 | -35.450 | -2.206 | -9.750 | 12.840 | 93.180 | 3 | 35.02 |
| 2 | -98.4400 | -35.5950 | -0.9513 | -9.4548 | 11.7650 | 88.7200 | 4 | 31.71 |
| 3 | -286.80 | -56.53 | -14.06 | -16.26 | 25.49 | 224.10 | 3 | 64.35 |
| 4 | -240.00 | -40.00 | 60.00 | 46.57 | 140.00 | 250.00 | 2 | 106.94 |
| 5 | -416.70 | -125.00 | -62.50 | -76.69 | -12.82 | 95.24 | 3 | 86.70 |
| 6 | -164.500 | -59.395 | -12.320 | -5.085 | 44.780 | 228.700 | 5 | 70.70 |
| 7 | -88.81 | -51.51 | -35.62 | -33.13 | -18.49 | 62.13 | 3 | 25.75 |
| 8 | -368.900 | -51.875 | 8.156 | 7.360 | 76.650 | 339.500 | 2 | 99.19 |
| 9 | -347.900 | -62.070 | 1.866 | -5.078 | 56.720 | 246.500 | 4 | 87.46 |
| 10 | -331.50 | -103.65 | -22.63 | -28.95 | 43.19 | 179.60 | 3 | 93.91 |
| 11 | -284.90 | -26.07 | 35.78 | 23.25 | 84.78 | 337.60 | 4 | 89.39 |
| 12 | -259.3000 | -38.4850 | 0.3725 | 34.6013 | 109.8000 | 345.6000 | 6 | 117.78 |
| 13 | -231.50 | -26.01 | 25.29 | 18.43 | 66.28 | 254.80 | 1 | 74.69 |
| 14 | -245.50 | -48.02 | 36.82 | 28.51 | 101.25 | 360.80 | 3 | 95.89 |
| 15 | -308.600 | -72.160 | -8.566 | -13.696 | 44.395 | 280.700 | 3 | 88.29 |
| 16 | -276.30 | -90.41 | -19.97 | -12.19 | 66.84 | 254.20 | 5 | 100.01 |
| 17 | -292.90 | -33.03 | 40.35 | 34.12 | 104.20 | 272.80 |  | 95.19 |
| 18 | -220.800 | -75.230 | -21.690 | -1.729 | 61.725 | 319.600 | 1 | 99.70 |
| 19 | -337.58 | -179.50 | -15.23 | -13.00 | 150.35 | 314.15 | 2 | 189.96 |
| 20 | -367.02 | -194.77 | -36.43 | -32.38 | 135.46 | 303.39 | 4 | 190.47 |
| 21 | -304.71 | -132.38 | 48.19 | 48.19 | 221.81 | 407.77 | 5 | 204.81 |
| 22 | -342.930 | -153.625 | 3.360 | 9.303 | 173.935 | 363.290 | 2 | 199.65 |
| 23 | -365.36 | -188.62 | -19.88 | -22.38 | 143.91 | 321.12 | 4 | 196.61 |
| 24 | -268.47 | -112.31 | 49.37 | 40.49 | 194.43 | 330.53 | 3 | 176.37 |
| 25 | -330.23 | -162.45 | -10.64 | -15.07 | 135.50 | 304.20 | 4 | 182.99 |
| 26 | -336.16 | -162.07 | 13.96 | 16.95 | 192.83 | 387.80 | 2 | 207.43 |
| 27 | -247.92 | -41.26 | 191.90 | 187.11 | 413.92 | 619.51 | 3 | 255.80 |
| 28 | -249.83 | -132.75 | -14.80 | -14.80 | 96.75 | 232.77 | 3 | 135.98 |

Table 1: summary table

The training data is summarised in Table 1. All the features had missing values with the exception of feature 17. However it is entirely possible that when the test data was separated from the entire data set at the start, that feature 17 too contained NA's within the test set. For some of the classifiers this poses a problem. Wherever it did cause a problem, the missing data was replaced by the mean of the other points belonging to the same class in the feature in question. In addition, the nature of the data varies quite significantly. Feature 7 takes values between the relatively narrow band of -88.81 to 62.13, when compared to feature 27 which has a range of over 800. Thus, it was necessary to scale the data for the KNN procedure, in order to improve performance, and remove any bias towards particular features. Here is the R code which was used to extract the data:

```
t(as.table(summary(data.df[,2:29])))
(summary(data.df2[,2:29]))
apply(data.df2, 2, sd)
```

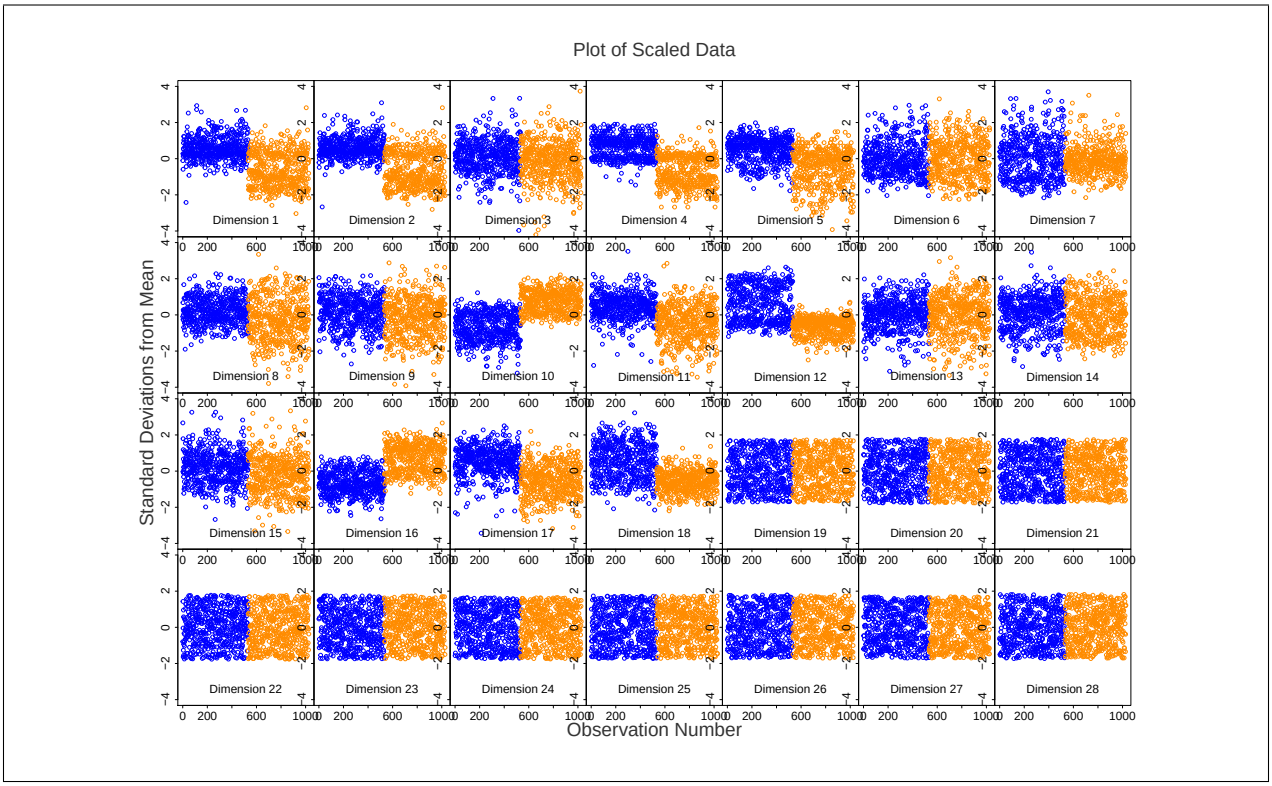



Figure 1: Scaled Data Plot

Figure 1 was constructed using the following R code:

```
par(mfrow=c(4,7))
par(cex = 0.6)
par(mar = c(0, 0, 0,0), oma = c(4, 4, 4, 0.5))
par(tcl = -0.25)
par(mgp = c(2, 0.6, 0))
for (i in 2:29){
  plot(c(1:1027),data.df3[,i],col = (ifelse(data.df3[,1]==0,"blue","dark
     orange")),xlab=i-1,ylim=range(-4,4))
  title(sub = bquote(paste("Dimension ", .(i-1))), line = -2)
}
mtext("Observation Number", side = 1, outer = TRUE, cex = 1, line = 2.2,
      col = "grey20")
mtext("Standard Deviations from Mean", side = 2, outer = TRUE, cex = 1, line =
    2.2,
      col = "grey20")
mtext("Plot of Scaled Data", side = 3, outer = TRUE, cex = 1, line = 2.2,
      col = "grey20")
```

In figure 1 we can visualise the data in a way that is not immediately apparent when simply examining all the critical values of each dimension. It is also relatively easy to make comparisons between different dimensions, as the data has been scaled to have a mean of 0, and standard deviation 1. Dimensions 1, 2, 4, 10, and 12 are particularly notable for the amount of separation between classes. This notion of separability will be explored further in later figures. Most features which have a degree of separability also seem to have

values lying outside of 2 standard deviations from the mean. In contrast, whilst it is impossible to make a classification based on any of dimensions 19-28 in isolation, they all seem to be spread fairly uniformly within 2 standard deviations from the mean, with no values lying outside of this range.
Figure 2 combines elements of both the initial table and figure 1 to reveal some interesting aspects about the data. While figure 1 was scaled so as to see how much the data deviated from the mean, here leaving the data unscaled is beneficial to graphically see the differing ranges and clustering of the different features. This plot also reveals the fact that there are many outliers in the data, especially prevalent in dimensions 7, 13 and 17. This is an important aspect of the data and can be crucial to determining which classifiers work the best. A classifier such as KNN for example, is fairly robust to outliers, whereas the multi-layered perceptron is more sensitive to them. We note that features 1, 2, 4 and 10 have distinctively different interquartile ranges dependent on class. Figure 3 ranks all the dimensions based on their ability to discern between classes by using a single one dimensional decision boundary. As suspected, features 1,2,4 and 10 are four of the most highly ranked features. This gives us great intuition into which features will be particularly useful to classifiers like KNN and CART, and serve as a crude measure against which to see how the wrapper performs. The code used to construct figure 2:

```
######## Box Plot
class.data.list <- list()
kcount <- 2
for (i in seq(1,56,2))
{
  class.data.list[[i]] <- data.df2.0[,kcount]
  class.data.list[[i+1]] <- data.df2.1[,kcount]
  kcount <- kcount + 1
}

par(mfrow=c(1,1))
boxplot(class.data.list, ylab = "Values",xlab = "Dimension Number", notch =
    TRUE, col = c("dark orange","blue"), xaxt='n')
axis(side=1, at=seq(1,56,2),labels=c(1:28))
legend(x=1,y=600, ncol=1,  cex=1, c("Class 1","Class 2"),text.width=5,lty=c
    (1,1),lwd=c(6,6),col=c("dark orange","blue"))
#boxplot(data.df2[,2:29], ylab = "Values", notch=TRUE)
title(main="All Box Plots", line = 2,xlab="Dimension Number",  ylab="Values",
    outer=TRUE)
```

The code used to construct figure 3:

```
# Functions for separability Plot

boundary.select <- function(class.data, sensitivity)
{
  c <- cbind(1:(sensitivity+1),(1:(sensitivity+1)))
  k = 1
  for (i in seq(as.integer(min(class.data[,2])), as.integer(max(class.data
      [,2])),(abs(as.integer(min(class.data[,2])))+ abs(as.integer(max(class.
      data[,2])))))/sensitivity))
{
    class.data2 = class.data
   class.data2[,2] <- replace(class.data2[,2], class.data2[,2] > i, 1 )
   class.data2[,2] <- replace(class.data2[,2], class.data2[,2] != 1, 0)
```

```
12
13    c[k,2]= 0.5-abs(0.5 - (sum(class.data2[,1]==class.data2[,2])/nrow(class.
          data)))
14    c[k,1]= i
15    k=k+1
16   }
17   print(min(c[,2]))
18   min(c[,2])
19 }
20 boundary.select.all <- function(sensitivity)
21 {
22   error.rate <- c(1:28)
23   for (i in 2:29)
24   {
25     error.rate[i-1] <- boundary.select(as.matrix(data.df2[,c(1,i)]),
           sensitivity)
26   }
27   error.rate
28 }
29 error.rate.aseem <- boundary.select.all(100)
30 c.e<-cbind(c(1:28),error.rate.aseem[c(1:28)])
31
32 class.error <- c.e[order(c.e[,2]),]
33
34
35 par(mfrow=c(1,1))
36 plot(c(1:28),class.error[,2],  cex = 4,xlab="Dimension Number", ylab="Error
       Rate")
37 title(main="Separability of Classes based on one dimensional classification",
       outer=TRUE, xlab="Dimension Rank", ylab="Error Rate")
38 text(c(1:28), class.error[,2], class.error[,1], cex=1,  col=(ifelse(class.
       error[,2]<0.3,"dark green","red")))
```

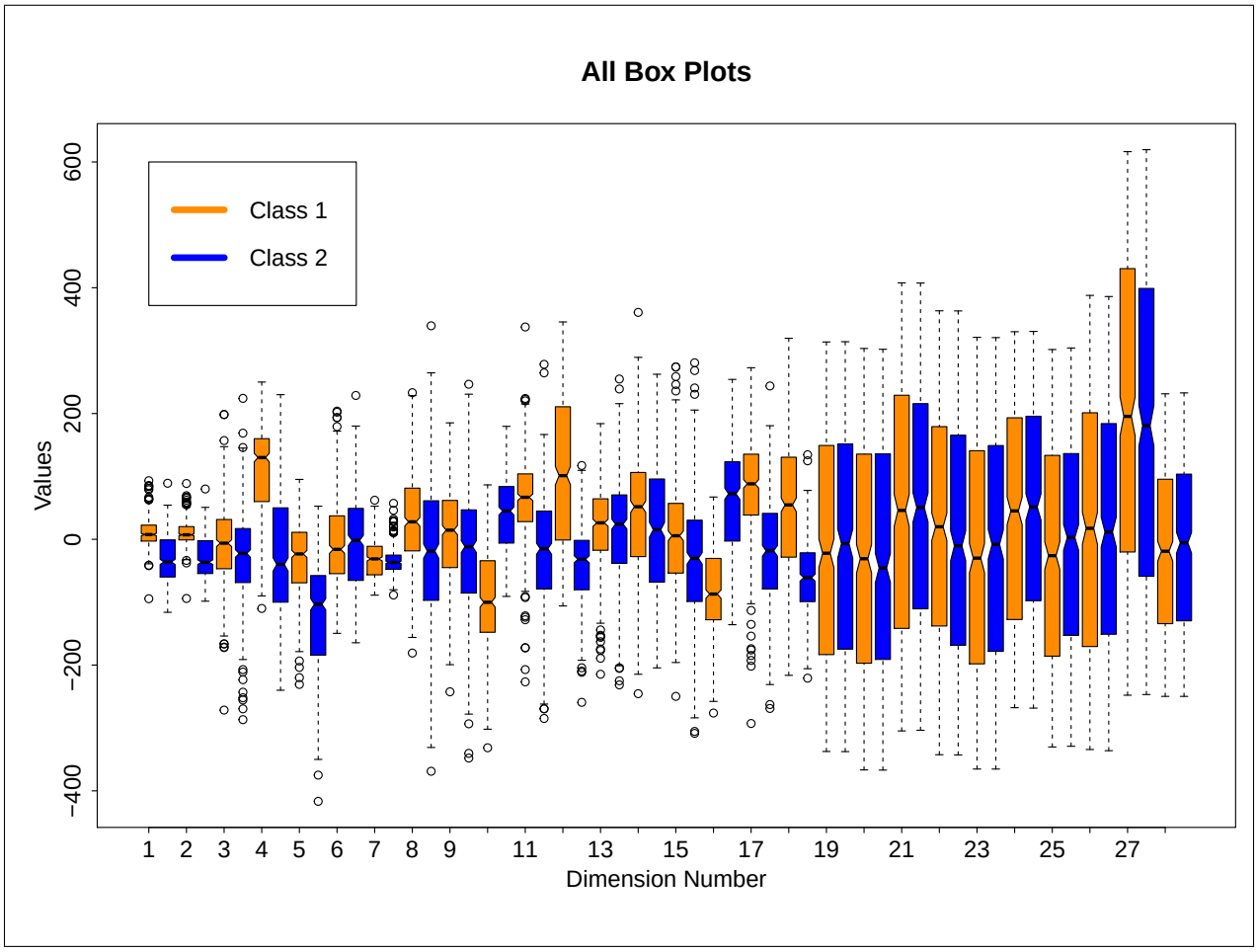



Figure 2: Box Plots

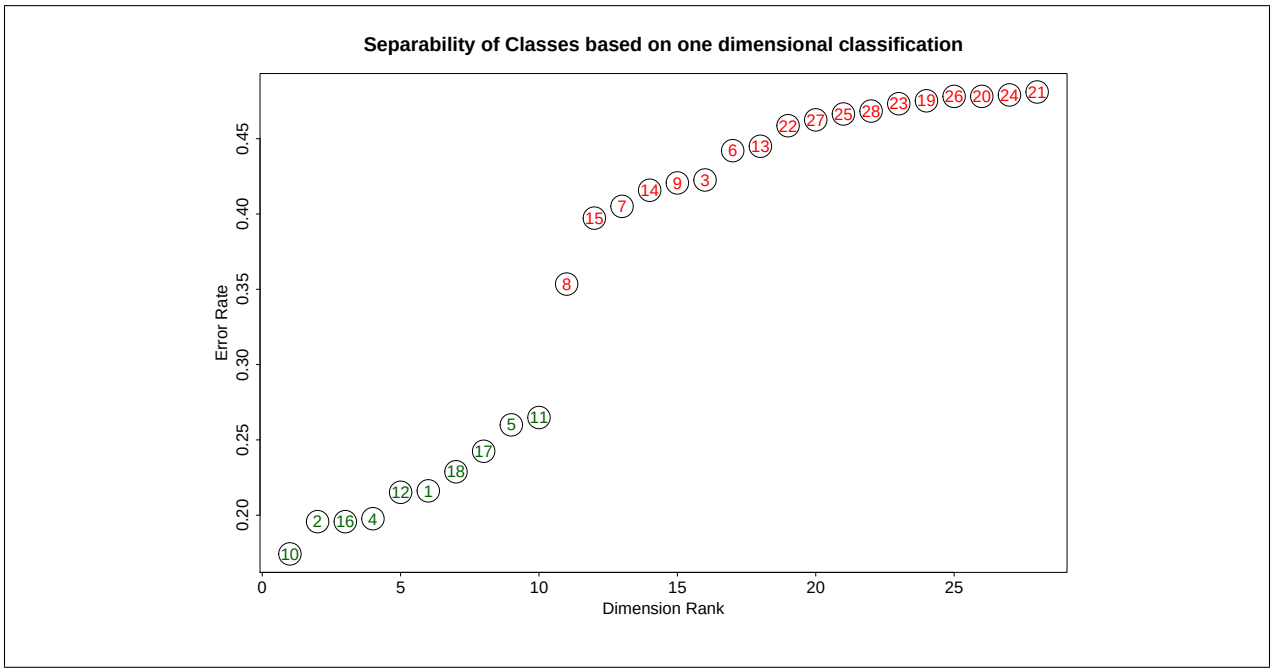



Figure 3: Separability

Finally, we conclude our brief discussion of the data by examining the correlation between the different dimensions, dimension 1 up to dimension 18, courtesy of figure 4. Features 19-28 were not shown in the plot as they displayed no significant correlation with any other feature. This fact, allied with the fact that graphically there was no easy way to classify those features, leads us to believe that they are effectively white noise.

Of note here is that there is very strong positive correlation between features 1 and 2. Figure 2 also depicted very similar box plots for the two features, indicating that they both may be modelled by a similar distribution, and selecting both of them may not add much benefit to a classifier compared to just selecting 1 as part of a reduced feature set. In addition, dimensions 10 and 16 seem to be negatively correlated with most other dimensions, apart from each other. This gives us inclination to believe that they will be particularly useful dimensions, especially considering that they were also two of the most separable dimensions in Figure 3.

```r
###### correlation plot

par(mfrow=c(1,1))
correlation_matrix <- cor(as.matrix(data.df2[2:29]))
correlation_matrix.selected <- cor(as.matrix(data.df2[2:19]))
corrplot(correlation_matrix.selected,  tl.pos = "d")
title(main="Correlation between selected Data Sets", outer = TRUE, line = 0)
```

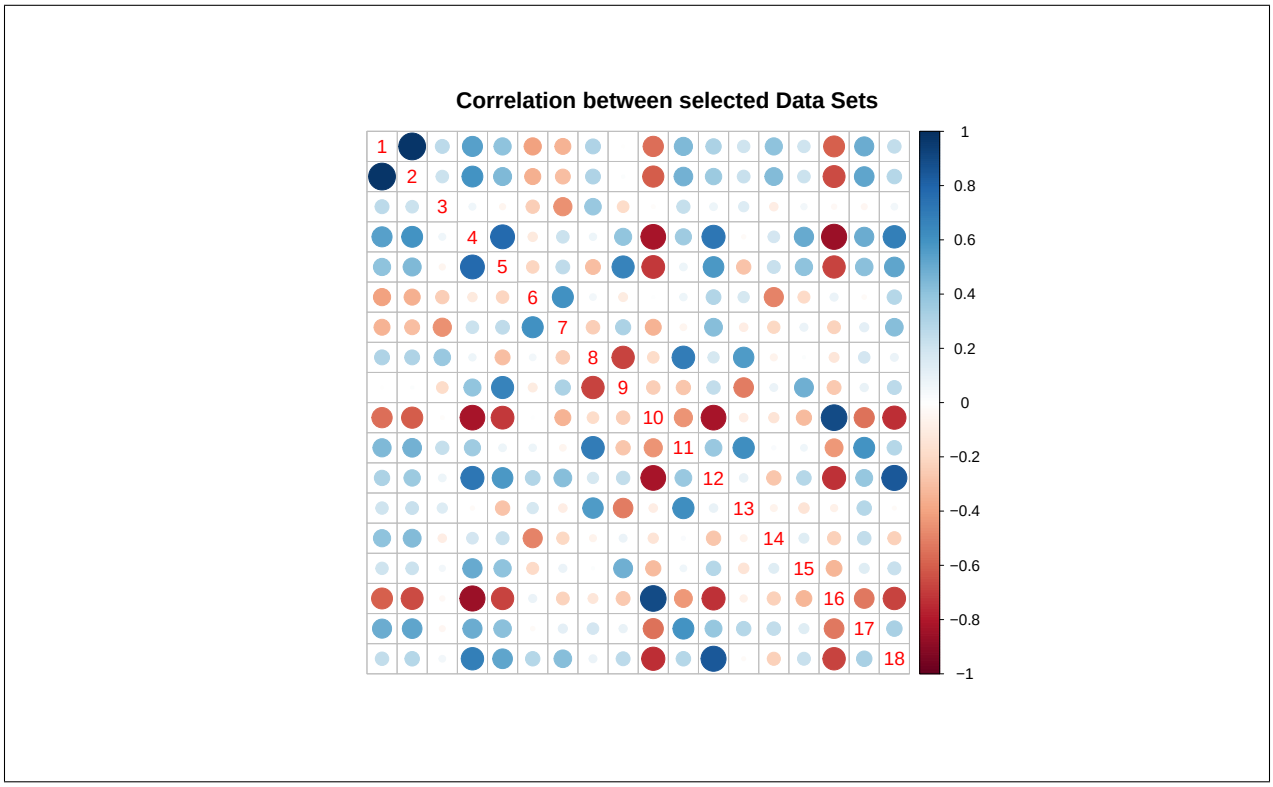



Figure 4: Correlation Graphic

# Section 2: Classifiers Used, Performance Assessment

## Wrapper

As the title of this section suggests, the method of choosing a suboptimal wrapper was preferred to both a suboptimal filter, and any optimal method. Sequential forward selection was preferred to backward selection, given the fact that in many of the classifiers, dimensions 19-28 reveal little about the class of the data. Choosing an optimal method would be beyond the scope of the project, given its computational expense outlined in the lecture notes. In any case, suboptimal methods are often very effective, in terms of classifier performance.
The code for the wrapper used during this project is as follows:

```
wrapper <- function (x, k = 7, size = 10, classifier)
# wrapper function

{
  set.seed(1)

  sub.wrapper <- function(x, a, k,classifier)
  # sub wrapper

  {

    c <- c(1:(ncol(x)-1))
    c <- cbind(c,c)
    # prepares a matrix to store dimension number and associated error

    for (i in 1:(ncol(x)-1))
    # loops across all dimensions
    {

      # binds the remaining data with features selected so far
      c[i,2] <- my.ten.cv(x=cbind(as.matrix(x[-1,1]),as.matrix(x[-1,i+1]),a),
          kay=k,classifier=classifier)
    }

    c[,1] <- as.matrix(x[1,2:ncol(x)])
    print(which.min(c[,2]))

    # returns dimension which had lowest error
    c[which.min(c[,2]),]
  }
########
  # creates a matrix of data with dimensions labelled
  y <- rbind(c(0:28),x)

  print(y[1,])

  # prepares vector storage
  a <- cbind(c(1:size),rep(NA,size))
```

```
  b <- vector()
  #b <- rep(0,nrow(x))
  print(length(b))
  print(nrow(as.matrix(x)))
  for (i in 1:size)
  {

  # calls the sub wrapper
  z <- sub.wrapper(y[,], k=k, b, classifier)

  # removes dimension chosen by sub wrapper
  y <- y[,y[1,]!=z[1]]


  # stores best dimensions in a vector
  a[i,] <- z

  # adds dimension chosen to current matrix of chosen dimensions
  b <- cbind(b,x[,(z+1)[1]])
  }
  a
}
```

## Cross Validation

10-fold cross validation was chosen as the desired performance assessment in this project. Leave-one-out cross validation would be more reliable, but also far more computationally expensive. Similarly choosing less than 10 folds would be less computationally expensive, but also less reliable. In practice, 10-fold cross validation offers a good tradeoff between computational cost and reliability. The function created to cross validate the procedures is outlined as follows:

```
###################################
# Cross Validation Funtion
##########################
my.ten.cv <- function( Hwd=c(3,0.001),x, kay, folds=10, classifier)
{
  set.seed(1)
  a <- rep(NA, folds)
  x.r <- x[sample(nrow(as.matrix(x))),]
  folds2 <- cut(seq(1,nrow(x.r)),breaks=folds,labels=FALSE)

  if (classifier == "KNN")
  {
    for (i in 1:folds)
    {
      testIndices <- which(folds2==i,arr.ind=TRUE)
      x.te <- x.r[testIndices,]
      x.tr <- x.r[-testIndices,]
      cl <- as.numeric(knn(as.matrix(x.tr[,-1]),as.matrix(x.te[,-1]),x.tr[,1],
```

```
          k=kay)) -1
      a[i] <- sum(cl != x.r[testIndices,1])/nrow(x.te)
    }
  }
  else if (classifier == "QDA")
  {
    for (i in 1:folds)
    {
      testIndices <- which(folds2==i,arr.ind=TRUE)
      x.te <- x.r[testIndices,]
      x.tr <- x.r[-testIndices,]
      qda.l <- qda(as.matrix(x.tr[,-1]),as.matrix(x.tr[,1]))
      qda.pred <- predict(qda.l,as.matrix(x.te[,-1]))
      a[i] <- sum(qda.pred$class != x.r[testIndices,1])/nrow(x.te)
    }

  }
  else if (classifier == "LD")
  {
    for (i in 1:folds)
    {
      testIndices <- which(folds2==i,arr.ind=TRUE)
      x.te <- as.data.frame(x.r[testIndices,])
      x.tr <- as.data.frame(x.r[-testIndices,])
      names(x.te)[1]<-"c"
      names(x.tr)[1]<-"c"
      ld.1 <- glm(c~.,data=x.tr,family="binomial")
      ld.pred <- as.numeric(predict(ld.1, x.te, type="response") >0.5)
      a[i] <- sum(ld.pred != x.r[testIndices,1])/nrow(x.te)
    }
  }
  else if (classifier == "CART")
  {
    for (i in 1:folds)
    {
      testIndices <- which(folds2==i,arr.ind=TRUE)
      x.te <- x.r[testIndices,]
      x.tr <- x.r[-testIndices,]
      gini.tree <- rpart(as.factor(x.tr[,1])~., data = x.tr[,-1])
      prune.gini.tree<- prune(gini.tree, 0.017 ) # 0.017 determined from plot
      gini.class <- predict(prune.gini.tree, x.te[,-1], type='vector')-1
      a[i] <- sum(gini.class != x.r[testIndices,1])/nrow(x.te)
    }
  }
  else if (classifier == "MLP")
  {
    for (i in 1: folds)
    {
      H <- Hwd[1]
```

```
      wd <- Hwd[2]
      testIndices <- which(folds2==i,arr.ind=TRUE)
      x.te <- as.data.frame(x.r[testIndices,])
      x.tr <- as.data.frame(x.r[-testIndices,])
      error.rate <- rep(NA, kay)
      names(x.te)[1]<-"c"
      names(x.tr)[1]<-"c"
        count <- 0
      iteration <- 0

      while(count < kay)
      {
        iteration = iteration + 1
        set.seed(iteration)

        mlp <- nnet( as.factor(c)~. , data = x.tr, size = H,decay = wd, maxit
            = 200, trace=FALSE)
        evals <- eigen( nnetHess(mlp, x.tr[, -1], x.tr[, 1]), T)$values
        if(min(evals) > 0.00001)
          # ensure a minimum is found
        {

          if (ncol(as.matrix(x.tr[,-1]))==1)
            # avoid Data Frame naming issues
          {
            x.te.1 <-as.data.frame(x.te[,-1])
            names(x.te.1)[1] <- "V2"
            pred.mlp <- predict(mlp,x.te.1)
            count <- count + 1
          }
          else
          {
          pred.mlp <- predict(mlp, x.te[,-1])
          count <- count + 1
          }
          error.rate[count] = sum((pred.mlp > 0.5) != x.r[testIndices,1])/nrow
              (x.te)
        }
      }
      a[i] <- sum(error.rate)/kay
      # average error rates
    }

        }

  sum(a)/folds # error rate
}
```

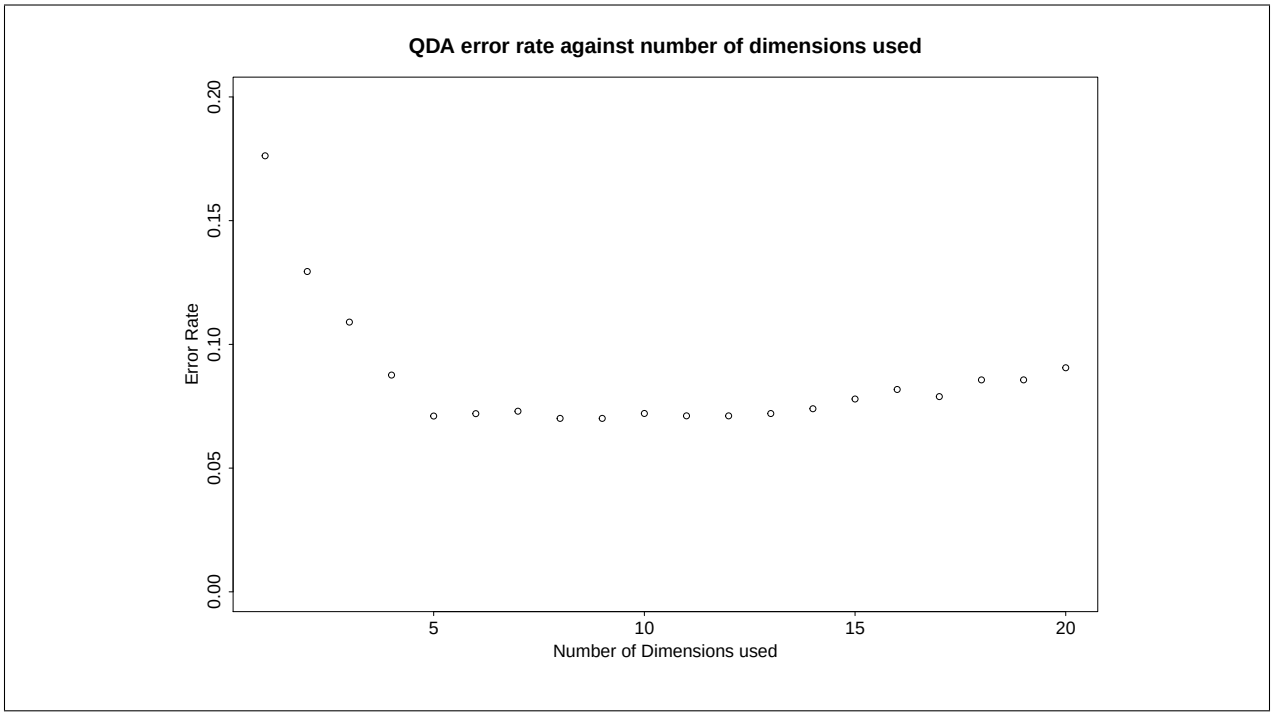



Figure 5: QDA Error Rate against number of features used

## QDA

The first method I chose to classify the data with was Quadratic Discriminant Analysis. It is straightforward to implement, does not require any pre-selection of parameters, and thus there is little scope for things to go awry. The assumption it makes about the data, that it is multivaraite normal, is a fairly standard assumption to make given how often the normal distribution can be used to adequately model large quantities of observed data. QDA is also preferred to LDA, since the likeliness for all 28 dimensions to share a covariance matrix is extremely low, and practically impossible given the analysis of the data in the previous section. QDA is not without its faults however; its simplicity is both a strength and a weakness. It will not deal very well with data that is not approximately normally distributed by definition, and there are no parameters which can be adjusted to compensate for this obvious weakness. In addition, it cannot deal with missing data. For all the classifiers bar CART, in this project we elected to replace missing values in the training set with class means as a fairly simple solution suggested by Webb[1]. A caveat of this approach is that we create spurious observations, for example, dimension 4 takes exclusively discrete values, however the mean is not discrete. However, this does not affect the classification methods used in any significant way, and since the number of NAs in the data is fairly small compared to the entire observation set, this is a fairly safe approach. Using the wrapper and the cross validation procedure simultaneously, the following plot was obtained. The lowest error rate was 0.701 to 3 s.f. for a feature set of size 8 or 9. Since adding the 9th dimension didn't improve the error rate, the feature set selected was {10,2,11,12,7,26,19,3}.

```
wrapper.qda <- wrapper(data.df2, size = 20, classifier="QDA")
plot(c(1:20),wrapper.qda[,2],ylim=range(0,0.2))
title(xlab="Number of Dimensions used",
      ylab ="Error Rate",
      main="QDA error rate against number of dimensions used" , outer=TRUE)
```

## KNN

The second classifier chosen was K-nearest neighbours. Webb notes that "Nearest-neighbour methods are easy to implement and are recommended as a starting point for a nonparametric approach."[1]. The nearest-neighbour approach is also preferred to other nonparametric approaches such as kernels in this particular problem since "nearest-neighbour methods are superior to fixed kernel approaches to density estimation beyond four dimensions"[1], when we have, in fact, 28 different features. However, KNN is a computationally expensive task relative to many other classifiers, and requires preselection of a single parameter. Thus, effective dimension reduction in the data is crucial not only to ensure a low error rate, but to make the use of the classifier viable from the perspective of computational time. In addition, the data had to be scaled so that the mean was 0 and the standard deviation 1, in order to improve KNN performance. Since we have established that there is no great correlation between the different dimensions, effective dimension reduction is eminently possible in this problem. Again, the wrapper and the cross validation were used simultaneously, but this time over a range of 11 k values from 1 to 21. The lowest error rate obtained was in a dimension set of size 16, and a k value of 21. Then, for this particular feature, the cross validation procedure was run over 51 different values of k, but the k value of 21 produced the smallest error rate, of 0.0448 to 3 s.f. . The dimension set chosen was size 16, {10,7,11,4,1,12,14,8,5,18,15,16,23,2,3,9}. All dimensions bar 17 from the green dimensions on figure 3 were chosen, lending credence to the idea that separability in an individual dimension is a property that is linked with usefulness in a nearest neighbours approach.

```
find.k <- function(x)
{
  c <- c(1:51)
  c <- cbind(c(1:51),seq(1,101,2))
  for (i in seq(1,101,2))
  {
    knncv <- my.ten.cv(x=cbind(as.matrix(data.df3[,1]),as.matrix(data.df3[,x +
        1])),k=i,classifier="KNN")

    c[(i+1)/2,1]= knncv
  }
  c
}


# Run wrapper for different values of k. Establish that k = 21 produces the
   lowest error rates
wrapper.select <- matrix(NA,20,22)
for (i in seq(1,21,2))
{
a <- wrapper(data.df3, size = 20, k=i,classifier="KNN")
wrapper.select[,i] <- a[,1]
wrapper.select[,i+1] <- a[,2]
}
find.k(wrapper.select[1:16,21])
wrapper.KNN <- wrapper.select[,21:22]
plot(c(1:20),wrapper.KNN[,2],ylim=range(0,0.2))
title(xlab="Number of Dimensions used",
     ylab ="Error Rate",
     main="KNN error rate against number of dimensions used" , outer=TRUE)
```

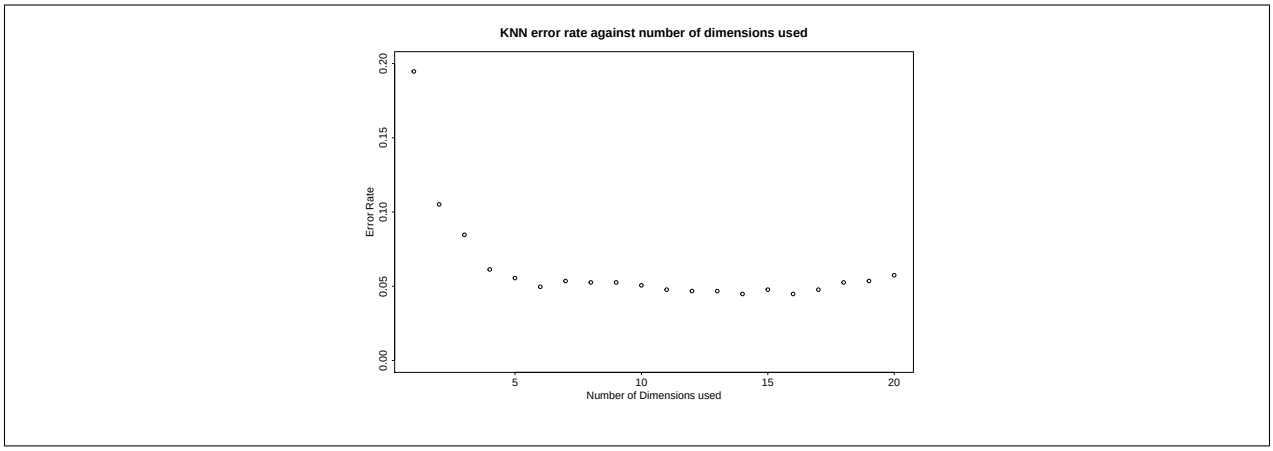



Figure 6: KNN Error Rate against number of features used

## Logistic Discrimination

The third method used to classify the data was Logistic Discrimination. Logistic Discrimination has many virtues, some of which directly relate to the data set in the problem. It is easy to use, does not require any preselection of parameters, and specifically to this problem, "it is applicable over a wide range of distributions"[1] and "appropriate for both continuous and discrete-valued variables"[1]. While there is only one discrete valued variable in the data set, dimension 4, we have already noted a key useful property of that particular dimension, of its class separability, and thus using a method which can handle its discrete values may prove worthwhile. The varying nature of the data also hinted that there may be a range of different distributions at play, further justifying the use of Logistic Discrimination as a valid classification approach in this problem. It is also useful as a simple starting point to provide comparison with the more complex multi-layered perceptron which is investigated later on, and is in some ways a special case of logistic regression or discrimination. The approach here was very similar to the one employed with QDA, given there were no parameters to be preselected. The best error rate was given by a dimension set of {10,11,1,12,7,13,17,14,5,8,24,3,15,9,21,4,25,19,6,20,28,27,2,16}, and the error rate was 0.0682 to 3 s.f. . The interesting features of this classification method was that a clear minimum could not be established after running the wrapper through just 20 dimensions, and it had a very similar error rate from using a 5 dimension set all the way to a 28 dimension set to classify observations. It did justice to the theory, proving quite versatile, and consistent.

```
wrapper.ld <- wrapper(data.df2, size =28, classifier="LD")
plot(c(1:28),wrapper.ld[,2],ylim=range(0,0.2))
title(xlab="Number of Dimensions used",
      ylab ="Error Rate",
      main="LD error rate against number of dimensions used" , outer=TRUE)
```

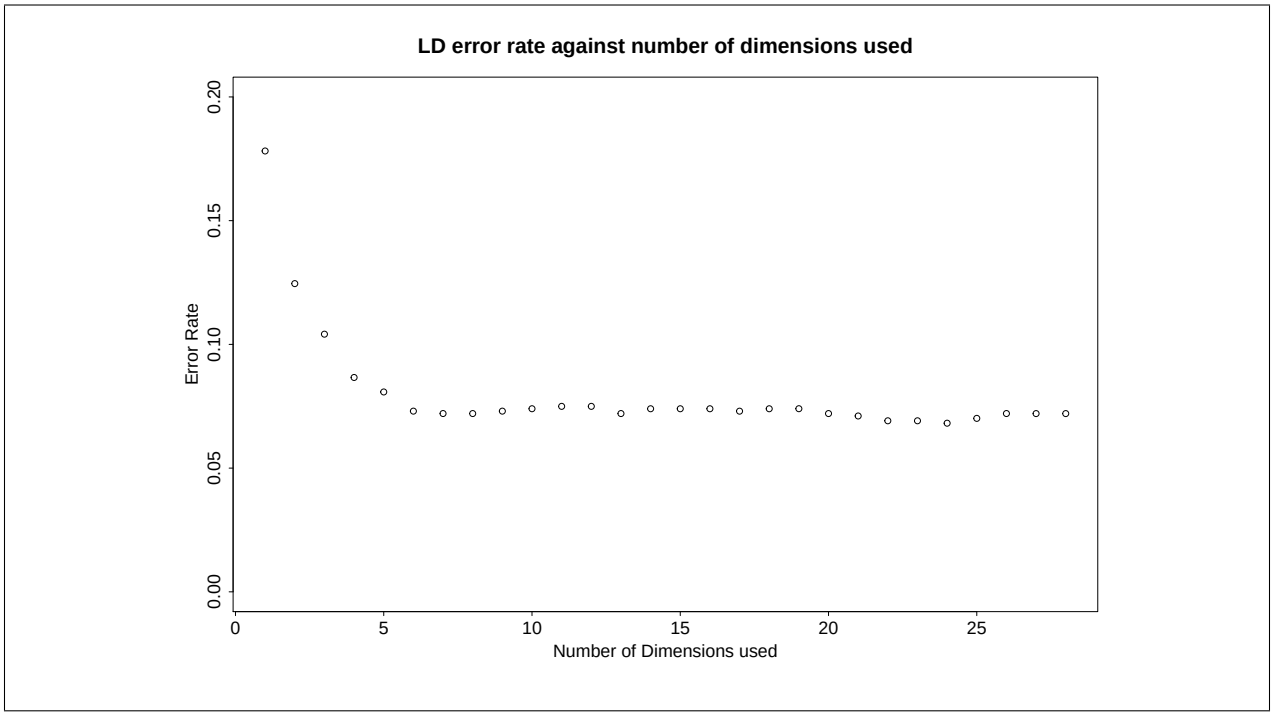


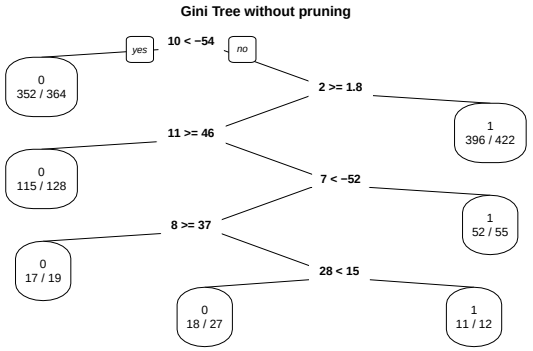

Figure 7: Logistic Discrimination Error Rate against number of features used

## CART

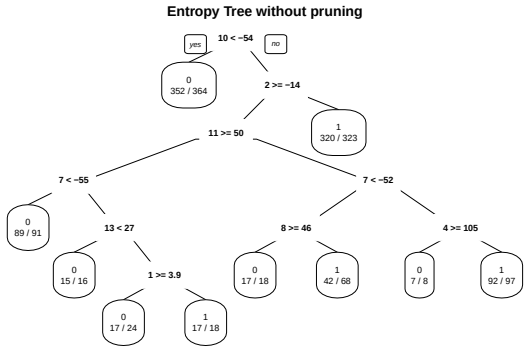
The fourth method used to classify the data was CART. Once again, the running theme of simplicity applies here as well. Conceptually, it is very easy to understand, like the nearest neighbour method. However, in contrast to the KNN approach, it is also readily interpretable, and can handle missing data, key attributes not present in most of the other classifiers chosen. Specifically, for this problem, CART may achieve good performance, given the numerous separable dimensions discovered and visually displayed in figure 3. It will be interesting to note if the classes chosen at the nodes of the tree correspond to the dimensions highlighted in green in figure 3. The approach to CART was markedly different from the previous three classifiers. Firstly, by its very nature CART does not require a wrapper. The lecture notes suggested that the Gini index and entropy are most often used to construct trees. In this project, both were investigated, and the results can be seen graphically on the following page. Both approaches involved constructing a tree, then pruning it based on the leftmost pruning point below the horizontal dotted line in the CP plot. Interestingly, both pruned trees were very similar, however, the Gini tree proved superior, correctly allocating 14 more points with its one extra node, and thus it was this one that was examined with cross validation. The error rate produced was 0.110 to 3 s.f. . The dimensions of the first two splits, 10 and 2, were also ranked as the dimensions with the highest degree of separability in the earlier graphic, again backing up the link between separability in an individual dimension and a nonparametric method.

```
###### Cart

# Gini Tree
set.seed(1)
gini.tree <- rpart(as.factor(data.df[,1])~., data = data.df[,-1])
par(mfrow=c(1,1))
prp(gini.tree,extra=2, main="Gini Tree without pruning")
plotcp(gini.tree)
```

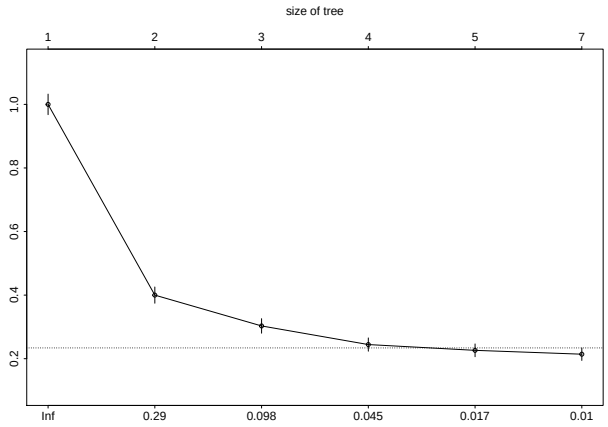
```
gini.tree$cptable
prune.gini.tree<- prune(gini.tree, 0.017 )
prp(prune.gini.tree,extra=2, main="Pruned Gini Tree")

# Information Tree
set.seed(1)
information.tree <- rpart(as.factor(data.df[,1])~., data = data.df[,-1],parms=
    list(split="information"))
prp(information.tree,extra=2, main="Entropy Tree without pruning")
plotcp(information.tree)
gini.tree$cptable
prune.information.tree<- prune(gini.tree, 0.039 )
prp(prune.information.tree,extra=2, main ="Pruned Entropy Tree")

# Establish Gini Tree is better than Information Tree

wrapper.CART <- my.ten.cv(x=data.df, kay=1, classifier="CART")
```



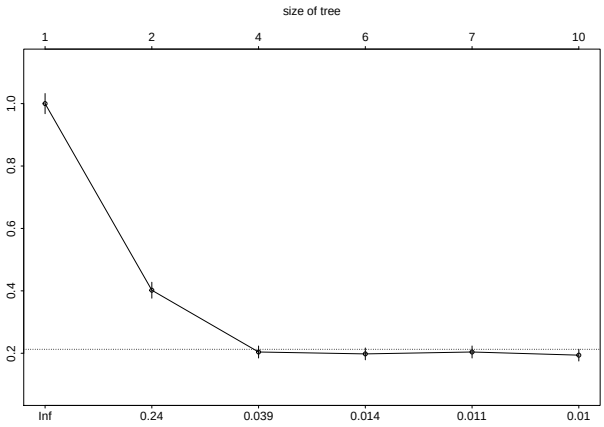

(a) Gini without pruning



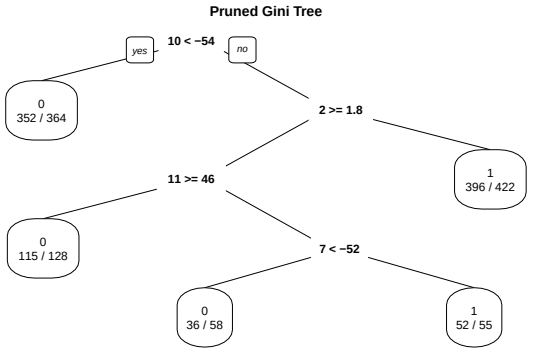

(b) Entropy without pruning



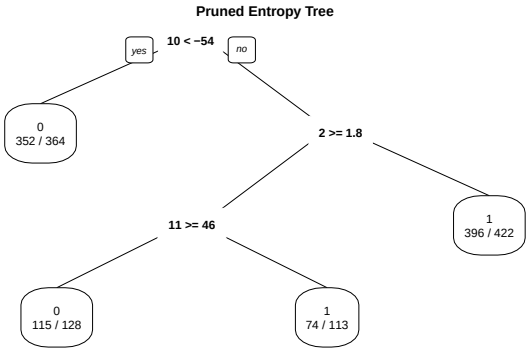

(c) Gini CP plot, with left most CP of 0.017




(d) Entropy CP plot, with left most CP of 0.039




(e) Gini tree pruned by complexity parameter 0.017




(f) Entropy tree pruned by complexity parameter 0.039

## MLP

Webb suggests that "a simple pattern recognition technique is implemented as a baseline before considering neural network methods"[1], and so we leave the relatively complex MLP until last. The main difficulty of implementing MLP successfully stems from the fact that there are 2 parameters which need to be preselected. Secondly, the process is extremely time consuming. Furthermore, MLP cannot handle missing data, and is sensitive to outliers. Subsequently, one may question the selection of such a laborious classifier in the first place. However, its strength lies in its predictive power. In practice, merely using a wrapper in conjunction with cross validation to obtain a dimension set upon which to further investigate the MLP took over 15 minutes, at least an order of magnitude higher than any other procedure outlined in this project. MLP also required the data to be scaled between 0 and 1 before it could perform optimally, as suggested in the literature. The general procedure employed with MLP comprised of first building a reduced dimension set, while not averaging over multiple minima in the cross validation step. Once the wrapper had determined the dimension set with the lowest error, this dimension set was subject to 10-fold cross validation, varying the parameters, and averaging the predictions over the collection of 3 minima to provide a final prediction as outlined in the notes. The dimension set chosen was {10,7,11,18,1,8,4}, and the best parameters were 3 hidden nodes and a decay of 0.01. This predicted an error rate of 0.0601 to 3 s.f. The error rate against the number of dimensions used can be seen in Figure 8.

```
mlp.check <- wrapper(data.df4, size = 20, classifier ="MLP", k=1)
mlp.check
H <- c(1,3,5)
wd <- c(0.01,0.001,0.0001)
gr <- as.matrix(expand.grid(H,wd))


wrapper.MLP <- apply(gr,1,my.ten.cv, kay=3,classifier="MLP",x=cbind(data.df4
    [,1],data.df4[,mlp.check[1:9,1]+1]),folds=10)
plot(c(1:20),mlp.check[,2],ylim=range(0,0.2))
title(xlab="Number of Dimensions used",
      ylab ="Error Rate",
      main="MLP error rate against number of dimensions used" , outer=TRUE)
```

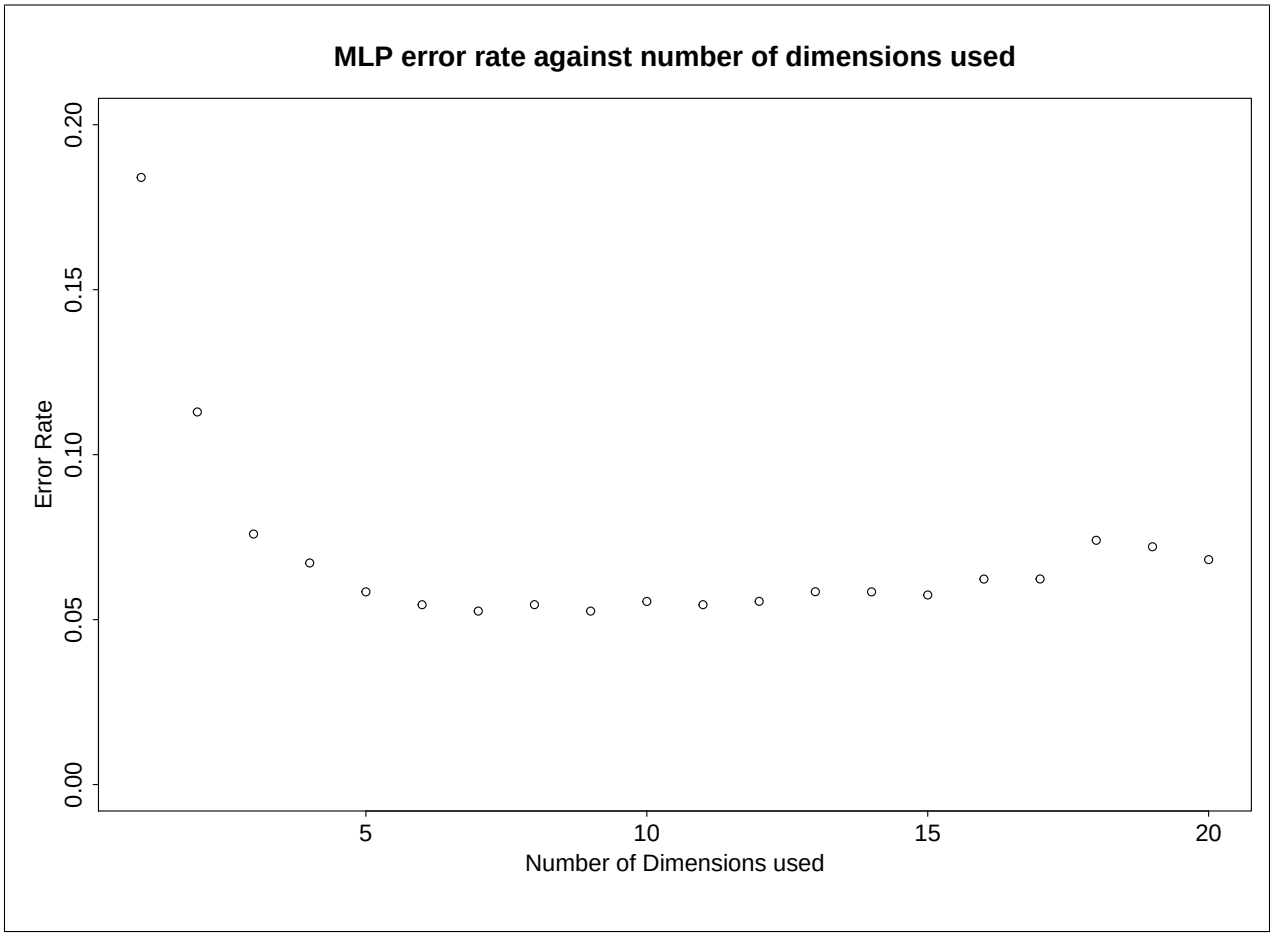



Figure 9: MLP Error Rate against number of features used

## Section 3: Peformance Assessment

Having discussed the methods used to establish the performance rate of each of the classifiers individually, we can see a comparitive summary in table 2. CART performed markedly worse than the other four classifiers. KNN and MLP performed the best, but not by much. Given the separability of the data as noticed in the first section, the performance of KNN was not surprising. The QDA and LD classifiers performed adequately too, and there was not much to distinguish between the two methods. The performance of QDA suggests that perhaps some dimensions did roughly approximate a multivariate normal distribution.

| | **QDA** | **CART** | **LD** | **KNN** | **MLP** |
|---|---|---|---|---|---|
| Error Rate | 0.0701 | 0.110 | 0.0682 | 0.0448 | 0.0601 |
| Parameters | 0 | 0 | 0 | 1 | 2 |
| Speed | Very Fast | VeryFast | Fast | Medium | Very Slow |

Table 2: Performance Assessment Table

## Section 4: McNemar's Test

The two best classifiers in the previous section were KNN and MLP. Consequently, the McNemar's test was applied to these two functions, using the dimension sets and parameters which had obtained the lowest error rates. The z score of the McNemar's test was below the critical value, so we accept $H_0$ and conclude that there is no difference between the successful classification rate of KNN and MLP.
The code to perform the McNemar's Test is as follows:

```
1
2  "McNemar"<-function(x,y, true, sig = 0.01)
3  {
4    x.wrong <- x != true
5    y.wrong <- y != true
6    n.1 <- sum((x.wrong == T) & (y.wrong==F))
7    n.2 <- sum((y.wrong == T) & (x.wrong==F))
8    z <- (abs(n.1 - n.2) - 1)/(sqrt(n.1+n.2))
9    cr <- qnorm(1 - sig/2, 0, 1)
10   list(z=z, cr = cr)
11 }
12 data.te.na <- data.te
13 for (i in 1:29)
14 {
15   data.te.na[is.na(data.te.na[,i]), i] <- mean(data.df2[,i], na.rm = TRUE) #
       replace NAs with class mean
16 }
17
18 data.te.KNN <- cbind(data.te.na[,1],scale(data.te.na[,-1]))
19 data.te.MLP <- cbind(data.te.na[,1],apply(data.te.na[,2:29],2,function(a){(a-
     min(a))/max(a-min(a))}))
20
21 #KNN
22 knn.pred<- as.numeric( knn( data.df3[,wrapper.KNN[1:16,1]+1], data.te.KNN[,
     wrapper.KNN[1:16,1]+1], data.df3[,1],k=11 ))-1
23 #MLP
24   data.df.MLP.2 <- as.data.frame(cbind(data.df4[,1],data.df4[,mlp.check
       [1:9,1]+1]))
25   data.te.MLP.2 <- as.data.frame(cbind(data.te.KNN[,1],data.te.KNN[,mlp.check
       [1:9,1]+1]))
26   names(data.te.MLP.2)[2] <- "1"
27   names(data.te.MLP.2)[3] <- "2"
28   names(data.te.MLP.2)[4] <- "3"
29   names(data.te.MLP.2)[5] <- "4"
30   names(data.te.MLP.2)[6] <- "5"
31   names(data.te.MLP.2)[7] <- "6"
32   names(data.te.MLP.2)[8] <- "7"
33   names(data.te.MLP.2)[9] <- "8"
34   names(data.te.MLP.2)[10] <- "9"
35
36   names(data.df.MLP.2)[2] <- "1"
37   names(data.df.MLP.2)[3] <- "2"
```

Section 4: McNemar's Test continued on next page... 

```
  names(data.df.MLP.2)[4] <- "3"
  names(data.df.MLP.2)[5] <- "4"
  names(data.df.MLP.2)[6] <- "5"
  names(data.df.MLP.2)[7] <- "6"
  names(data.df.MLP.2)[8] <- "7"
  names(data.df.MLP.2)[9] <- "8"
  names(data.df.MLP.2)[10] <- "9"

mlp <- nnet( as.factor(V1)~. , data = data.df.MLP.2, size = 3,decay = 0.001,
    maxit = 1000, trace=FALSE)
evals <- eigen( nnetHess(mlp, data.df.MLP.2[,-1], data.df.MLP.2[,1]), T)$
    values
print(min(evals))
pred.mlp <- round(predict(mlp, data.te.MLP.2[,-1]))

McNemar(knn.pred, pred.mlp, data.te[,1])
```

## Section 5: Conclusion

Overall, KNN is the clear winner. While computationally expensive, it is still much faster than the MLP, which was the only other classifier which came close in terms of performance, as indicated by the McNemar's test. It was not particularly sensitive to it's one preselected parameter, and performed very well for a wide range of k, meaning that it was quite forgiving in this problem even if we had used a worse approach. Given that the value of k did not impact its performance too much, its potential drawback in having a preselected parameter was mitigated somewhat. The biggest disadvantage of KNN was the interpretability of the model. However, out of the classifiers, only CART was considerably more interpretable, but the trade off with its large misclassification rate compared to the rest of the classifiers ensured it was not viable. Arguably, QDA and Logistic Discrimination were both reasonably quick, and slightly more interpretable, however even their best error rates were at least 40% greater than that of KNN. In terms of speeding up KNN, perhaps data editting could be explored. However, as the lecture note suggest, in practice multiedit does not perform well for higher dimensions.

## Section 6: Miscellaneous Code Appendix

```r
require(ggplot2)

# Load libraries


Sys.setenv(JAVA_HOME='C:\\Program Files\\Java\\jre7\\')
library(MASS)
library(corrplot)
library(FSelector)
library(class)
library(rpart)
library(nnet)
library(rpart.plot)


set.seed(1)
data <- read.table("http://www2.imperial.ac.uk/~eakc07/S7/data4.dat",quote="\"")
data.r <- sort(sample(dim(data)[1],100,FALSE))
data.te <- as.data.frame(data[data.r,])

data.df <- as.data.frame(data[-data.r,])
colnames(data.df) <- c(0:28)
par(mfrow=c(4,7))
```

```r
data.df[532:533,] # Check where the split between 0 and 1 is

data.df2.0 <- data.df[1:532,] # split data into classes
data.df2.1 <- data.df[533:nrow(data.df),] # split data into classes
for (i in 1:29)
{
data.df2.0[is.na(data.df2.0[,i]), i] <- mean(data.df2.0[,i], na.rm = TRUE) # replace NAs with class mean
}
for (i in 1:29)
{
data.df2.1[is.na(data.df2.1[,i]), i] <- mean(data.df2.1[,i], na.rm = TRUE) # replace NAs with class mean
}

data.df2 <- rbind(data.df2.0,data.df2.1) # rebuild training with NAs replaced, used for QDA, KNN
data.df3 <- cbind(data.df2[,1],scale(data.df2[,-1])) # normalised data, used for KNN
data.df4 <- cbind(data.df2[,1],apply(data.df2[,2:29],2,function(a){(a-min(a))/max(a-min(a))}))
# data scaled between 0 and 1, used for MLP
```

# References

[1] Andrew R. Webb, *Statistical Pattern Recognition* John Wiley & Sons, 2nd edition, 2002.