STAT6020_PredictiveAnalytics_Assignment2

Prepared by Tim Virga

12/09/2021

We are interested in developing a tool that can classify wines into one of the three types based on the other variables provided.

Linear and Quadratic Discriminant Analysis

```r
library (MASS)
# reading in the data
wine = read.csv("wine.csv")
# selecting numeric variables
wine_numpred = c("ash", "ash_alkalinity", "nonflav_phenols","proanth")
class = wine[,"vintage"]
# numeric variables and class
pre_and_class = wine[, c(wine_numpred,"vintage")]
```

1. LDA

```r
# Training and prediction
lda.fit = lda(vintage~., data=pre_and_class, CV=TRUE)
ldapred_class = lda.fit$class
# Confusion matrix
cfmatrix = table(ldapred_class, class)
# Accuracy divides TN and TP against sum of samples
accuracy = sum(diag(cfmatrix))/sum(cfmatrix)
error = 1-accuracy
print(error)
```

```
## [1] 0.2542373
```

```r
print(accuracy)
```

```
## [1] 0.7457627
```

```r
show(cfmatrix)
```

```
##               class
## ldapred_class  A  B  C
##              A 31  1 16
##              B  1 53  7
##              C 16  4 48
```

After fitting a LDA model using numerical variables as predictors for class=vintage we observe a 25.4% error rate and 74.6% accuracy rate. Noting in the confusion matrix what class values the LDA model has incorrectly classified.

2. QDA

```
# Training and prediction
qda.fit = qda(vintage~., data=pre_and_class, CV=TRUE)
qdapred_class = qda.fit$class
accuracy = sum(diag(cfmatrix))/sum(cfmatrix)
error = 1-accuracy
print(error)
```

```
## [1] 0.2542373
```

```
print(accuracy)
```

```
## [1] 0.7457627
```

```
cfmatrix = table(qdapred_class, class)
show(cfmatrix)
```

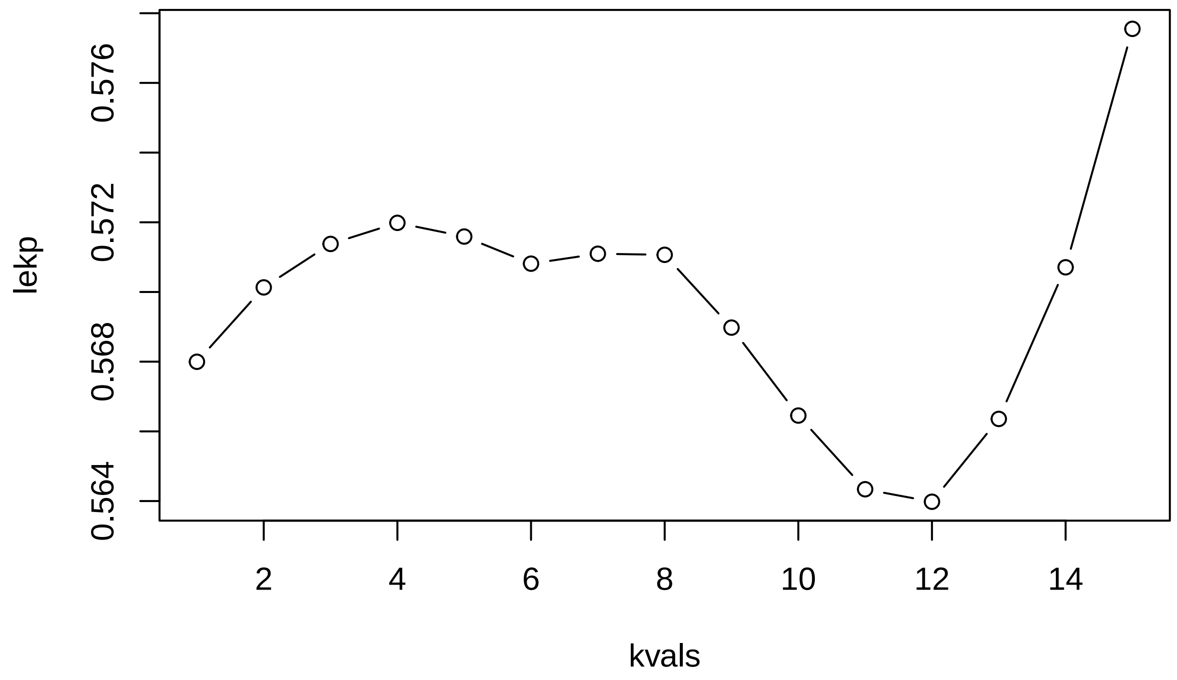
```
##               class
## qdapred_class  A  B  C
##              A 36  0 17
##              B  1 50  8
##              C 11  8 46
```

After fitting a QDA model using numerical variables as predictors for class=vintage we observe a 25.4% error rate and 74.6% accuracy rate. Noting in the confusion matrix what class values the LDA model has incorrectly classified. Interestingly, the accuracy of the model appears to be identical however, it's obvious from the confusion matrix that the QDA model has predicted different different values for classification than LDA. It's unclear whether the data set exhibits a more favourable linear, or quadratic spread of data- where the analyst may want to then consider the simpler model.

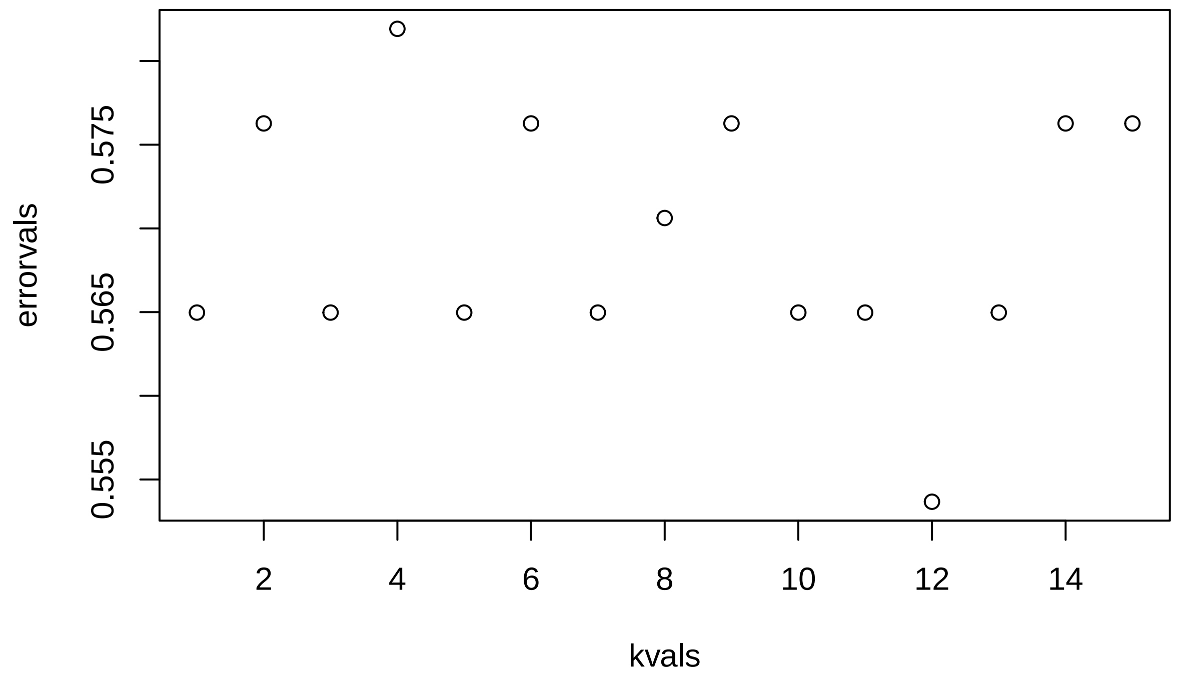
K Nearest Neighbours

3.

```r
library(class)

# predictors (numeric only):
numeric_predictors = c("ash", "ash_alkalinity", "nonflav_phenols","proanth")

# class labels
class = wine[,"vintage"]
Predictors = wine[, numeric_predictors]

# storing some values from the loop iterations
kvals = c()
errorvals = c()
accuracyvals = c()

# performing a loop to evaluate K=n models
for (K in c(1:15)) {
  predicted_class = knn.cv(train=Predictors, cl=class, k=K) #associate the k=K class
  cm = table(predicted_class, class)
  accuracy = (cm[1,1]+cm[2,2])/sum(cm)
  error = 1 - accuracy
  kvals[K] = K
  errorvals[K] = error
  accuracyvals[K] = accuracy

  }
lek = loess(errorvals~kvals)
lekp = predict(lek)
plot(kvals, lekp, type="b")
```



```
plot(errorvals~kvals)
```



```
# specify the k=n best performing K value to predict with most accuracy
predicted_class = knn.cv(train=Predictors, cl=class, k=12)
  cm = table(predicted_class, class)
  accuracy = (cm[1,1]+cm[2,2])/sum(cm)
  error = 1 - accuracy
  print(error)
```

```
## [1] 0.5649718
```

```
  print(accuracy)
```

```
## [1] 0.4350282
```

```
  print(cm)
```

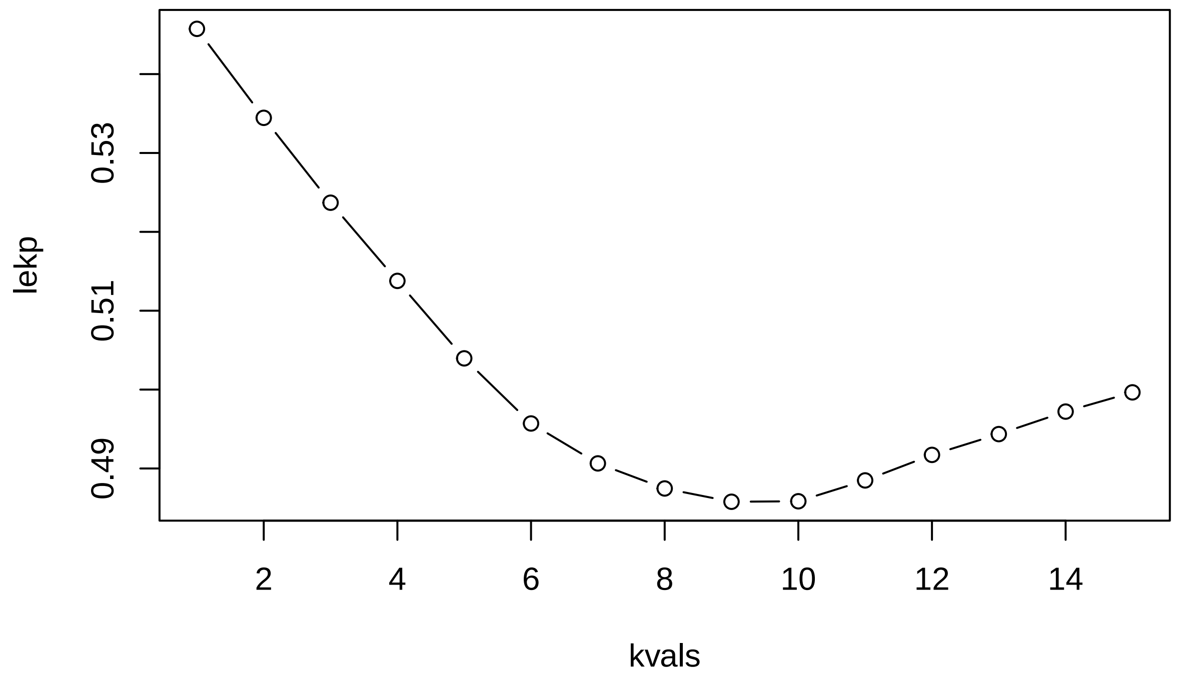
```
##                  class
## predicted_class  A  B  C
##                A 33  2 19
##                B  2 44 16
##                C 13 12 36
```

We have designed a KNN (LOOCV) model with a loop of n=15. After fitting the model using numerical variables as predictors for class=vintage we observe that an optimal value for k is k=12 in the first chunk.

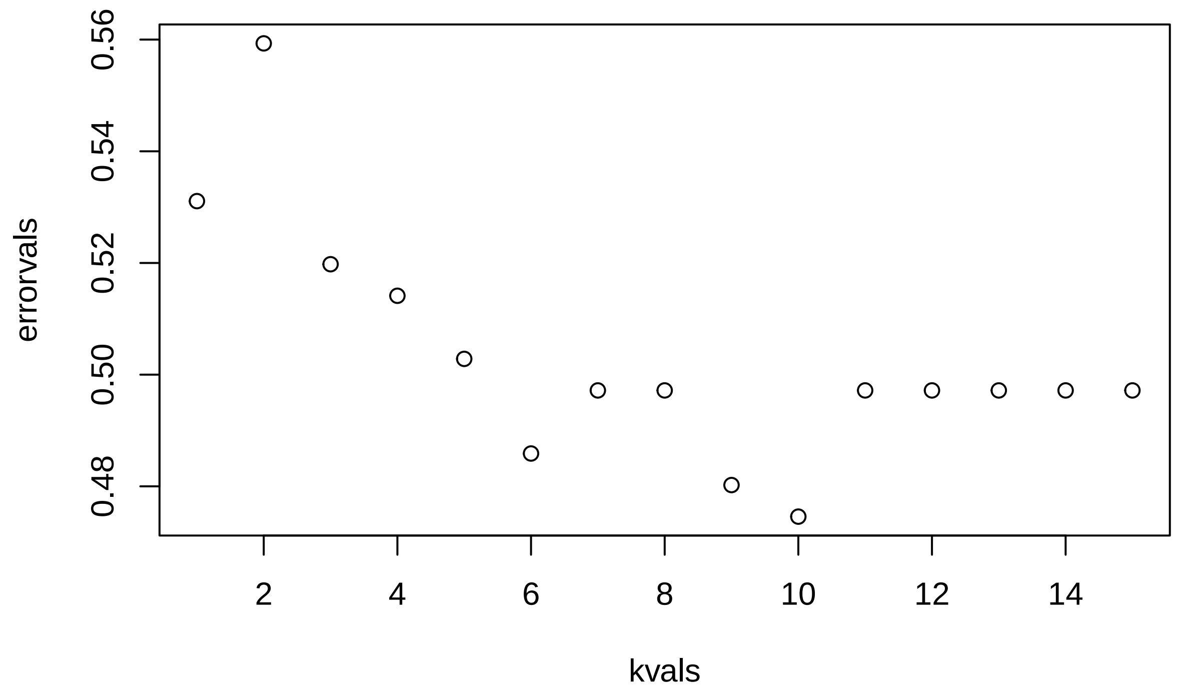
Separately, in the following chunk we recomputed the code with an optimal kvalue k=12 and observed an error rate of approximately 55% and an accuracy rate of approximately 45%. Noting in the confusion matrix what class values the LDA model has incorrectly classified. The KNN model is a non-parametric approach to classification that performs well for complex decision boundaries. Given the KNN (LOOCV) model has performed significantly worse than the QDA and LDA models, we can speculate that the true decision boundaries may be more linear.

4.

```r
# predictors (numeric only):
numeric_predictors = c("ash", "ash_alkalinity", "nonflav_phenols","proanth")

# class labels
class = wine[,"vintage"]
Predictors = wine[, numeric_predictors]

# scaling the predictors
Predictors = scale(Predictors)

kvals = c()
errorvals = c()
accuracyvals = c()

for (K in c(1:15)) {
  predicted_class = knn.cv(train=Predictors, cl=class, k=K) #associate the k=K class
  cm = table(predicted_class, class)
  accuracy = (cm[1,1]+cm[2,2])/sum(cm)
  error = 1 - accuracy
  kvals[K] = K
  errorvals[K] = error
  accuracyvals[K] = accuracy

  }
lek = loess(errorvals~kvals)
lekp = predict(lek)
plot(kvals, lekp, type="b")
```



```
plot(errorvals~kvals)
```



```
predicted_class = knn.cv(train=Predictors, cl=class, k=10)
cm = table(predicted_class, class)
accuracy = (cm[1,1]+cm[2,2])/sum(cm)
error = 1 - accuracy
print(error)
```

```
## [1] 0.4858757
```

```
print(accuracy)
```

```
## [1] 0.5141243
```

```
print(cm)
```

```
##                  class
## predicted_class  A  B  C
##               A 36  2 17
##               B  2 55 14
##               C 10  1 40
```

We have designed a KNN (LOOCV) model with a loop of n=15. After fitting the model using numerical variables as predictors for class=vintage and scaling the predictor variables, we observe that an optimal value for k is k=10. This is a value lower than the previous KNN (LOOCV) model without scaling, suggesting the data without predictors scaled featured skewed predictor variable values. In the process of standardisation, we observe that the scale of values for "ash" and "proanth" to be similar. We can expect scaling to have introduced an increased influence on the model for predictor "nonflav_phenols" which was structured on a lower value scale and a reduced influence on the model for "ash_alkalinity" which was originally structured on a higher value scale.

Naive Bayes

5.

```
library(naivebayes)
```

```
## naivebayes 0.9.7 loaded
```

```
no_obs <- dim(pre_and_class)[1] # dataset size
Pred_class <- rep("", no_obs)

# LOOCV using loop less the one observation per loop:
for (i in 1:no_obs){
# fitting the naive bayes model
NB.fit <- naive_bayes(vintage ~., data = pre_and_class[-i, ])

# Predict the leftover:
Pred_class_test <- predict(NB.fit, newdata = pre_and_class[i,], type = "class")
Pred_class[i] <- as.character(Pred_class_test)
}
nbcm <- table(Pred_class, class)
accuracy <- sum(diag(nbcm))/sum(nbcm)
error <- 1 - accuracy
```

```
print(error)
```

```
## [1] 0.2824859
```

```
print(accuracy)
```

```
## [1] 0.7175141
```

```
print(nbcm)
```

```
##            class
## Pred_class  A  B  C
##          A 38  1 21
##          B  0 50 11
##          C 10  7 39
```

After fitting a Naive Bayes Classifier model with a LOOCV loop across the total number of observations, using numerical variables as predictors for class=vintage, we observe a 28.2% error rate and 71.8% accuracy rate. Noting in the confusion matrix what class values the LDA model has incorrectly classified. Naive Bayes Classifer models the probability of a class using Baye's Rule making the assumption the features are independent. For our numeric predictors, this has not proven to be more effective than QDA and LDA models.

6.

```
library(naivebayes)

pre_and_class_col = wine[, c(wine_numpred,"colour","vintage")]

no_obs <- dim(pre_and_class_col)[1]
Pred_class <- rep("", no_obs)
for (i in 1:no_obs){
# set laplace smoothing to 1
NB.fit <- naive_bayes(vintage ~., data = pre_and_class_col[-i, ], laplace=1)
Pred_class_test <- predict(NB.fit, newdata = pre_and_class_col[i, ], type = "class")
Pred_class[i] <- as.character(Pred_class_test)
}
nbcm <- table(Pred_class, class)
accuracy <- sum(diag(nbcm))/sum(nbcm)
error <- 1 - accuracy
```

```
print(error)
```

```
## [1] 0.180791
```

```
print(accuracy)
```

```
## [1] 0.819209
```

```
print(nbcm)
```

```
##           class
## Pred_class  A  B  C
##          A 43  2  8
##          B  0 46  7
##          C  5 10 56
```

After fitting a Naive Bayes Classifier model with a LOOCV loop across the total number of observations, using numeric variables and categorical variable "Colour" as predictors for class=vintage, we observe an 18.1% error rate an 81.9% accuracy rate. The Naive Bayes Classifier model different to LDA & QDA can be modeled using categorical variables because of the assumptions made for independent variables. We observe the inclusion of just 1 categorical predictor "Colour" improve the performance of our model to perform better than QDA and LDA.

7.

```r
library(naivebayes)

no_obs <- dim(wine)[1]
Pred_class <- rep("", no_obs)
for (i in 1:no_obs){
# fit all wine data with laplace smoothing
NB.fit <- naive_bayes(vintage ~., data = wine[-i, ], laplace=1)
Pred_class_test <- predict(NB.fit, newdata = wine[i, ], type = "class")
Pred_class[i] <- as.character(Pred_class_test)
}
nbcm <- table(Pred_class, class)
accuracy <- sum(diag(nbcm))/sum(nbcm)
error <- 1 - accuracy
```

```r
print(error)
```

```
## [1] 0.1242938
```

```r
print(accuracy)
```

```
## [1] 0.8757062
```

```r
print(nbcm)
```

```
##             class
## Pred_class   A  B  C
##            A 43  1  6
##            B  0 50  3
##            C  5  7 62
```

After fitting a Naive Bayes Classifier model with a LOOCV loop across the total number of observations, using all variables as predictors for class=vintage and a Laplace smoothing laplace=1, we observe a 12.4% error rate an 87.6% accuracy rate. Concluding this Naive Bayes (LOOCV) model to be the most accurate model out of those computed.

```r
NB.fit$call
```

```
## naive_bayes.formula(formula = vintage ~ ., data = wine[-i, ],
##     laplace = 1)
```

```r
NB.fit$prior
```

```
##
##         A         B         C
## 0.2670455 0.3295455 0.4034091
```

```
tables(NB.fit,)
```

```
##
## ---------------------------------------------------------------------------------
##  ::: ash (Gaussian)
## ---------------------------------------------------------------------------------
##
## ash            A         B         C
##   mean 2.4306383 2.4560345 2.2447887
##   sd   0.1811486 0.2291245 0.3154673
##
## ---------------------------------------------------------------------------------
##  ::: ash_alkalinity (Gaussian)
## ---------------------------------------------------------------------------------
##
## ash_alkalinity         A         B         C
##          mean 21.351064 17.062069 20.238028
##          sd    2.235861  2.561375  3.349770
##
## ---------------------------------------------------------------------------------
##  ::: magnesium (Gaussian)
## ---------------------------------------------------------------------------------
##
## magnesium         A         B         C
##      mean  99.38298 105.98276  94.54930
##      sd    10.99714  10.22465  16.75350
##
## ---------------------------------------------------------------------------------
##  ::: nonflav_phenols (Gaussian)
## ---------------------------------------------------------------------------------
##
## nonflav_phenols          A          B          C
##           mean 0.44510638 0.29017241 0.36366197
##           sd   0.12435700 0.07064841 0.12396128
##
## ---------------------------------------------------------------------------------
##  ::: proanth (Gaussian)
## ---------------------------------------------------------------------------------
##
## proanth         A         B         C
##    mean 1.1493617 1.8925862 1.6302817
##    sd   0.4122178 0.4124193 0.6020678
##
## ---------------------------------------------------------------------------------
##  ::: colour (Categorical)
## ---------------------------------------------------------------------------------
##
## colour          A          B          C
##   col1 0.21568627 0.06451613 0.42666667
##   col2 0.74509804 0.20967742 0.04000000
##   col3 0.01960784 0.43548387 0.02666667
##   col4 0.01960784 0.29032258 0.50666667
##
```

```
## ----------------------------------------------------------------------------------
```

$$P(Y = A, B, C)$$

$$P(X_{colour} = col1|Y)$$

```
# no_obs of class A
col1_pred_A = nrow(wine[which (wine$colour=='col1' & wine$vintage=='A'),])
# priors
vin_A = nrow(wine[which (wine$vintage=='A'),])
col1_pred_A_P = col1_pred_A / vin_A

# no_obs of class B
col1_pred_B = nrow(wine[which (wine$colour=='col1' & wine$vintage=='B'),])
# priors
vin_B = nrow(wine[which (wine$vintage=='B'),])
col1_pred_B_P = col1_pred_B / vin_B

# no_obs of class C
col1_pred_C = nrow(wine[which (wine$colour=='col1' & wine$vintage=='C'),])
# priors
vin_C = nrow(wine[which (wine$vintage=='C'),])
col1_pred_C_P = col1_pred_C / vin_C

print(col1_pred_A_P)
```

```
## [1] 0.2083333
```

```
print(col1_pred_B_P)
```

```
## [1] 0.05172414
```

```
print(col1_pred_C_P)
```

```
## [1] 0.4366197
```

By taking the total observations where colour=col1 and vintage classes were either A, B or C respectively, then dividing colour+class observations by the total number of observations for each respective class, we are able to obtain the estimated probabilities for each of the classes given that colour is col1.