Classification_2

Khanin Sisaengsuwanchai

2022-06-07

Thinking Through Nearest Neighbors.

(a) What would happen if the number of neighbors equals the number of cases in the training sample?

- If we choose the number of K (neighbors) to be the number of training sample, the predictions of each testing datapoint will always be a class that has the largest number of data in the training sample. According to the ISLR, this is called a null model because it gives the same result every time we make predictions.

(b) What would happen if the number of neighbors is 1? Does this even make sense? Explain.

- If the number of neighbors (K) is 1, the class of testing data will always be assigned to the class of closest training data, which is overly flexible. I don't believe using this K=1 makes sense because the closeness between training and testing data might occur by chance and thus leads to extremely overfitting.

(c) Suggest a method for selecting a reasonable number of neighbors.

- To select an appropriate number of neighbors(K), I would plot testing error rates with 1/K. I would expect that the testing errors will result a U-shape. I would pick a reasonable number of neighbors(K) at the minimum of the U-shape in the testing errors. If the testing errors are not U-shape, I would pick K where the errors drop significantly.

```
# (d)  Apply the method you proposed in the previous section to the crab data in the MASS package:

library(MASS)
head(crabs) # Top 6 rows
```

```
##   sp sex index   FL  RW   CL   CW  BD
## 1  B   M     1  8.1 6.7 16.1 19.0 7.0
## 2  B   M     2  8.8 7.7 18.1 20.8 7.4
## 3  B   M     3  9.2 7.8 19.0 22.4 7.7
## 4  B   M     4  9.6 7.9 20.1 23.1 8.2
## 5  B   M     5  9.8 8.0 20.3 23.0 8.2
## 6  B   M     6 10.8 9.0 23.0 26.5 9.8
```

```r
dim(crabs) # Dimension of the data
```

```
## [1] 200   8
```

```r
summary(crabs)
```

```
##  sp      sex         index            FL              RW              CL
##  B:100   F:100   Min.   : 1.0   Min.   : 7.20   Min.   : 6.50   Min.   :14.70
##  O:100   M:100   1st Qu.:13.0   1st Qu.:12.90   1st Qu.:11.00   1st Qu.:27.27
##                  Median :25.5   Median :15.55   Median :12.80   Median :32.10
##                  Mean   :25.5   Mean   :15.58   Mean   :12.74   Mean   :32.11
##                  3rd Qu.:38.0   3rd Qu.:18.05   3rd Qu.:14.30   3rd Qu.:37.23
##                  Max.   :50.0   Max.   :23.10   Max.   :20.20   Max.   :47.60
##        CW              BD
##  Min.   :17.10   Min.   : 6.10
##  1st Qu.:31.50   1st Qu.:11.40
##  Median :36.80   Median :13.90
##  Mean   :36.41   Mean   :14.03
##  3rd Qu.:42.00   3rd Qu.:16.60
##  Max.   :54.60   Max.   :21.60
```

```r
# Observation of crabs:
# No missing values, data are all numerical, and no obvious outliers

# Create crab's type based on sp and sex
crabs.class = rep("1", 200) # Create an empty vector with 200 indices
crabs.class[crabs$sp == "B" & crabs$sex == "F"] = "2" # Assign class
crabs.class[crabs$sp == "O" & crabs$sex == "M"] = "3"
crabs.class[crabs$sp == "O" & crabs$sex == "F"] = "4"
# Assign classes to crabs dataframe
# B,M = "1"; B,F = "2"; O,M = "3"; O,F = "4"
crabs$class = crabs.class

head(crabs) # Show top 6 rows to confirm the assigned crab class
```

```
##   sp sex index   FL  RW   CL   CW  BD class
## 1  B   M     1  8.1 6.7 16.1 19.0 7.0     1
## 2  B   M     2  8.8 7.7 18.1 20.8 7.4     1
## 3  B   M     3  9.2 7.8 19.0 22.4 7.7     1
## 4  B   M     4  9.6 7.9 20.1 23.1 8.2     1
## 5  B   M     5  9.8 8.0 20.3 23.0 8.2     1
## 6  B   M     6 10.8 9.0 23.0 26.5 9.8     1
```

```r
# Split train and test to prepare for KNN with 80:20 rules by class.
# In other words, for each class, the number of training data is 40 and testing data is 10;

crab.class1 = crabs[crabs$class == "1", ]  # Select rows with class = "1", all columns
crab.class1_train = crab.class1[1:40,] # Select the first 40 rows as training observations
crab.class1_test = crab.class1[41:50,] # Select the last 10 rows as testing observations

crab.class2 = crabs[crabs$class == "2", ]  # Select rows with class = "2", all columns
```

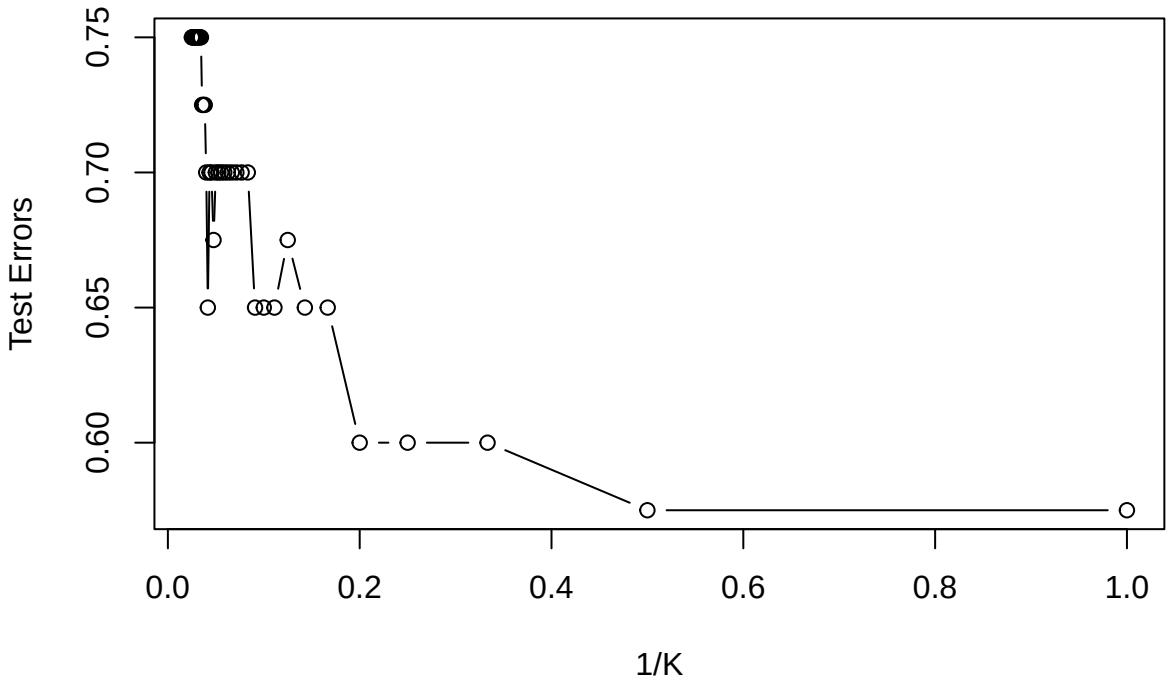
```r
crab.class2_train = crab.class2[1:40,] # Select the first 40 rows as training observations
crab.class2_test = crab.class2[41:50,] # Select the last 10 rows as testing observations

crab.class3 = crabs[crabs$class == "3", ]  # Select rows with class = "3", all columns
crab.class3_train = crab.class3[1:40,] # Select the first 40 rows as training observations
crab.class3_test = crab.class3[41:50,] # Select the last 10 rows as testing observations

crab.class4 = crabs[crabs$class == "4", ]  # Select rows with class = "4", all columns
crab.class4_train = crab.class4[1:40,] # Select the first 40 rows as training observations
crab.class4_test = crab.class4[41:50,] # Select the last 10 rows as testing observations

# Append training data
crab.train = rbind(crab.class1_train, crab.class2_train)
crab.train = rbind(crab.train, crab.class3_train)
crab.train = rbind(crab.train, crab.class4_train)
dim(crab.train) # check dimensions of training data; results are as expected
```

```
## [1] 160   9
```

```r
# Append testing data
crab.test = rbind(crab.class1_test, crab.class2_test)
crab.test = rbind(crab.test, crab.class3_test)
crab.test = rbind(crab.test, crab.class4_test)
dim(crab.test) # check dimensions of testing data; results are as expected
```

```
## [1] 40  9
```

```r
## Perform KNN
library(class)

set.seed(1) # a seed must be set in order to ensure reproducibility of results.

max_k = 40 # Define the maximum k
error.testing  = vector(length = length(max_k)) # create a blank testing errors vector
for (i in 1:max_k) {
  knn.pred = knn(crab.train[, c(4:8)], crab.test[, c(4:8)], crab.train$class, k = i)
  error.testing[i]  = mean(knn.pred != crab.test$class)   # Fill testing errors
}


# Plot a graph as suggested in (c)
plot(1/1:max_k, error.testing, type = "b", xlab='1/K', ylab='Test Errors')
```




```
# According to this plot, I would choose 1/K equal to 0.2 (k=5) because this is
# where the test errors drop significantly even though this plot might not be the
# U-shape as I expected.

# Build a model with k=5
knn.pred <- knn(crab.train[, c(4:8)], crab.test[, c(4:8)], crab.train$class, k = 5)
# For each point of test dataset, knn finds the closest 5 training points, and choose a
# class based on maximum voting.
table(knn.pred, crab.test$class)
```

```
##
## knn.pred  1  2  3  4
##        1  1  0  0  0
##        2  0  3  0  0
##        3  4  0  1  0
##        4  5  7  9 10
```

```
# e) Comment on your findings
mean(knn.pred == crab.test$class) # Accuracy
```

```
## [1] 0.375
```

```
mean(knn.pred != crab.test$class) # Error rate
```

```
## [1] 0.625
```

```
# Precision of class 1 is 1/1  = 100%
# Recall for class 1   is 1/10 = 10%
# Precision of class 2 is 3/3  = 100%
# Recall for class 2   is 3/10 = 30%
# Precision of class 3 is 1/5  = 20%
# Recall for class 3   is 1/10 = 10%
# Precision of class 4 is 10/31 = 32.6%
# Recall for class 4   is 10/10 = 100%

# According to the results of KNN with k=5, the model is doing a terrible job trying to
# classify the crab types. We can see that the total testing accuracy is only 37.5% which
# performs worsen than randomly guess the class. Moreover, precision and recall for each class
# are also poor and not reliable. For these reasons, I conclude that we should not
# use this model, and should find other classification methods to predict the class using
# 5 morphological measurements. Because of completely non-parametric approach, KNN
# performs poorly when we have a small number of training data or a lot of independent variables(p).
# If we would like to improve the accuracy, I would try other methods such as NaiveBeyes
# that work well with small number of data, which depends on independent variables assumption.
```

Logistic Regression vs Linear Discriminant Analysis.

```
## Loading the flea dataset
flea = read.csv(file = "flea.csv", header = TRUE)

head(flea) # Print the top 6 rows
```

```
##            X tars1 tars2 head aede1 aede2 aede3 species
## 1 1 Concinna   191   131   53   150    15   104       1
## 2 2 Concinna   185   134   50   147    13   105       1
## 3 3 Concinna   200   137   52   144    14   102       1
## 4 4 Concinna   173   127   50   144    16    97       1
## 5 5 Concinna   171   118   49   153    13   106       1
## 6 6 Concinna   160   118   47   140    15    99       1
```

```
dim(flea)  # Print dimensions of the data
```

```
## [1] 74  8
```

```
table(flea$species) # Count distinct class
```

```
##
##  1  2  3
## 21 22 31
```

```
summary(flea[, c(-1)]) # Understand how the data behave
```

```
##      tars1           tars2            head           aede1
##  Min.   :122.0   Min.   :107.0   Min.   :43.00   Min.   :116.0
##  1st Qu.:148.0   1st Qu.:118.2   1st Qu.:49.00   1st Qu.:125.5
##  Median :185.5   Median :123.0   Median :50.50   Median :136.5
##  Mean   :177.3   Mean   :124.0   Mean   :50.35   Mean   :134.8
##  3rd Qu.:198.2   3rd Qu.:130.0   3rd Qu.:52.00   3rd Qu.:142.8
##  Max.   :242.0   Max.   :146.0   Max.   :58.00   Max.   :157.0
##      aede2           aede3          species
##  Min.   : 8.00   Min.   : 55.00   Min.   :1.000
##  1st Qu.:11.00   1st Qu.: 85.25   1st Qu.:1.000
##  Median :14.00   Median : 98.50   Median :2.000
##  Mean   :12.99   Mean   : 95.38   Mean   :2.135
##  3rd Qu.:15.00   3rd Qu.:106.00   3rd Qu.:3.000
##  Max.   :16.00   Max.   :123.00   Max.   :3.000
```

```
# Observation of flea:
# No missing values, data are all numerical, and no obvious outliers


# Since we have a very small sample for each class (21 22 31), I choose to not split the data
# into training and test dataset. Therefore, I would compare the accuracy of these methods using
# training accuracy and errors.

# Normally, the Logistic Regression in glm cannot handle multiple classes.
# Thus, I will separate the data into 3 datasets as the following.
flea.1 = flea
flea.1[flea.1$species != 1,]$species = 0

flea.2 = flea
flea.2[flea.2$species != 2,]$species = 0

flea.3 = flea
flea.3[flea.3$species != 3,]$species = 0

# Each dataset contains only its class and 0
```

```
## Fit Logistic regression on each dataset

# Flea1
# Put all variables into the logistic regression model
glm1.fits <- glm(
  species ~ tars1 + tars2 + head + aede1 + aede2 + aede3, data = flea.1, family = binomial
)
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(glm1.fits) #
```

```
##
## Call:
```

```
## glm(formula = species ~ tars1 + tars2 + head + aede1 + aede2 +
##     aede3, family = binomial, data = flea.1)
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -5.366e-05  -2.100e-08  -2.100e-08   2.100e-08   6.797e-05
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.433e+03  1.538e+06  -0.001    0.999
## tars1        8.718e-02  6.188e+03   0.000    1.000
## tars2        9.549e+00  5.770e+03   0.002    0.999
## head        -3.802e+01  4.447e+04  -0.001    0.999
## aede1        1.104e+01  1.536e+04   0.001    0.999
## aede2        3.334e+01  9.488e+04   0.000    1.000
## aede3        1.541e+00  1.306e+04   0.000    1.000
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 8.8281e+01  on 73  degrees of freedom
## Residual deviance: 1.0171e-08  on 67  degrees of freedom
## AIC: 14
##
## Number of Fisher Scoring iterations: 25
```

```
# However, I got the warning that " glm.fit: algorithm did not converge"
# and p-values of all variables are extremely large. Therefore, I cannot use this model
# and this approach.
```

```
# Another approach is the Logistic Regression that can handle multiclass
# using multinom function

library(nnet)

logistic.model  = multinom(species ~ tars1 + tars2 + head + aede1 + aede2 + aede3, data = flea)
```

```
## # weights:  24 (14 variable)
## initial  value 81.297309
## iter  10 value 25.530412
## iter  20 value 0.496925
## iter  30 value 0.000679
## final  value 0.000058
## converged
```

```
logistic.output = summary(logistic.model)
logistic.output
```

```
## Call:
## multinom(formula = species ~ tars1 + tars2 + head + aede1 + aede2 +
##     aede3, data = flea)
##
## Coefficients:
```

```
##   (Intercept)      tars1      tars2     head      aede1     aede2      aede3
## 2    23.01142 -0.5990977 -0.1519661 3.929830  0.3426858 -9.926010 -0.3510001
## 3    31.10532  0.8430506 -1.7775656 5.223387 -1.2609775  2.191904 -0.9293456
##
## Std. Errors:
##   (Intercept)    tars1    tars2     head    aede1     aede2    aede3
## 2    10.50209 107.0001 602.8599 1110.278 511.1321  410.6496 518.9124
## 3    60.92961 142.3054 415.1722 1310.560 251.1677 1557.1287 176.8007
##
## Residual Deviance: 0.0001156031
## AIC: 28.00012
```

With multinomial logistic regression, the model treats class 1 as the baseline.
Note: different choices of baseline give different coefficients but the log adds will stay the same.
Logistic regression is originated by linear regression, but used logistic function to give smooth curves. To fit the model, the method uses maximum likelihood to estimate betas hat.
The following equations demonstrate the predicted betas(coefficients) of the estimated logistic regression.

```
knitr::include_graphics("Logistic_equations.png")
```

$$\ln\left(\frac{P(species=2)}{P(species=1)}\right) = 23.01 - 0.60(tars1) - 0.15(tars2) + 3.93(head) + 0.34(aede1) - 9.93(aded2) - 0.35(aded3)$$

$$\ln\left(\frac{P(species=3)}{P(species=1)}\right) = 31.11 + 0.84(tars1) - 1.78(tars2) + 5.22(head) - 1.26(aede1) + 2.19(aded2) - 0.93(aded3)$$

To interpret the coefficient, for example, a one-unit increase in tars1 would increase the log odds of species=3 over species=1 by 0.84.

```
# Since multinom does not provide and p-values , I will compute them using
# the following command
z = logistic.output$coefficients/logistic.output$standard.errors # Calculate Z-value
p = (1 - pnorm(abs(z), 0, 1)) * 2 # Two tail z-test
print("P-values of the model")
```

```
## [1] "P-values of the model"
```

```
p
```

```
##   (Intercept)     tars1     tars2      head     aede1     aede2     aede3
## 2  0.02844253 0.9955326 0.9997989 0.9971759 0.9994651 0.9807158 0.9994603
## 3  0.60969254 0.9952732 0.9965839 0.9968199 0.9959943 0.9988769 0.9958060
```

```
# Since all the p-values for each variable are much larger than 0.05, indicating that
# the association between these variables and dependent variables occur by chance,
# I believe that something might go wrong with the model, but I cannot further investigate
# because of the time constants.

# Predict using training observations
logistic.pred <- predict(logistic.model)

table(logistic.pred, flea$species)   # Count predict values with Y
```

```
##
## logistic.pred  1  2  3
##             1 21  0  0
##             2  0 22  0
##             3  0  0 31
```

```
mean(logistic.pred == flea$species)  # Accuracy
```

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

```
mean(logistic.pred != flea$species)  # Error rate
```

```
## [1] 0
```

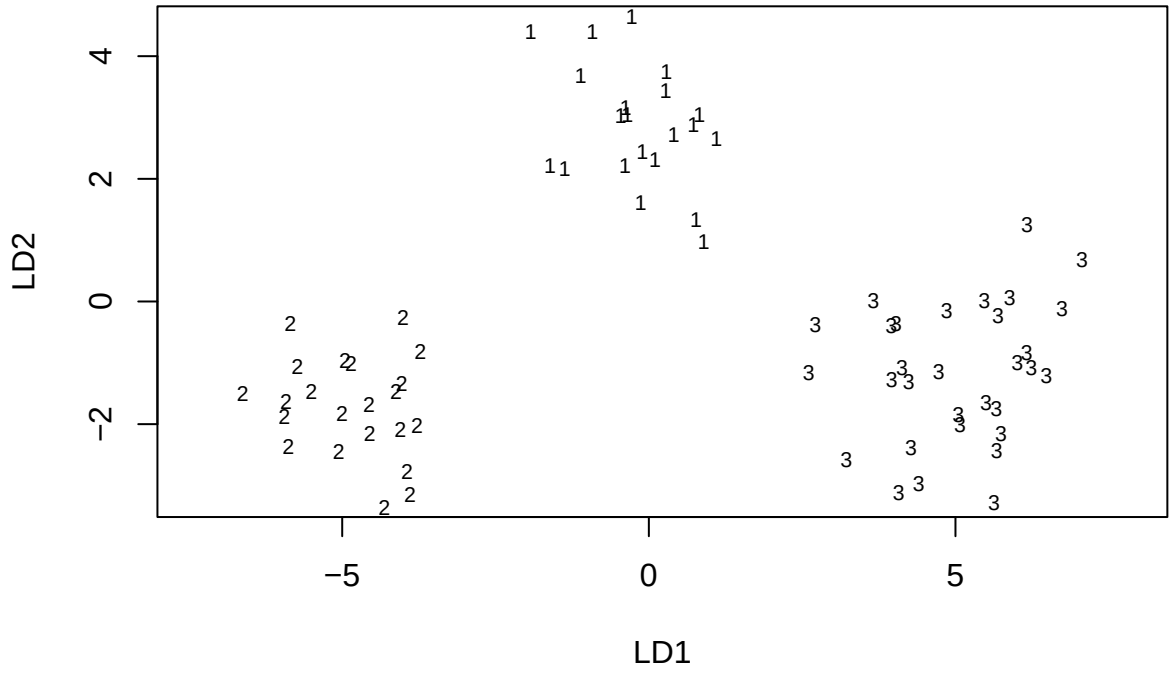
```
## Interpretation of the accuracy
# According to the model performance based on training observations,
# the model perfectly classify each class, indicating by accuracy of 100% and
# the error rate of 0%. However, we need to keep in mind that we use training
# observations, not test observations. We might get a poor performance
# when we use test dataset because of overfitting. When we get 100% accuracy, I believe
# something might go wrong with our model because we cannot reduce irreducible errors.
# I am of the opinion that we need to get more data to further verify this model.
# With the available data, I cannot make conclusion on how this model actually performs.
```

```
## LDA
library(MASS)
lda.fit <- lda(species ~ tars1 + tars2 + head + aede1 + aede2 + aede3, data = flea)

# LDA is a type of generative models where it makes assumptions about the data.
# Specifically, LDA assumes that f(x) of k is normal distribution, and each class
# has the constant variance. QDA, on the other hands, assumes that each class has
# its own variance.
lda.fit
```

```
## Call:
## lda(species ~ tars1 + tars2 + head + aede1 + aede2 + aede3, data = flea)
##
## Prior probabilities of groups:
##         1         2         3
## 0.2837838 0.2972973 0.4189189
##
## Group means:
##      tars1    tars2     head    aede1    aede2    aede3
## 1 183.0952 129.6190 51.23810 146.1905 14.09524 104.8571
## 2 138.2273 125.0909 51.59091 138.2727 10.09091 106.5909
## 3 201.0000 119.3226 48.87097 124.6452 14.29032  81.0000
##
## Coefficients of linear discriminants:
##               LD1         LD2
## tars1  0.09271011  0.01362951
## tars2 -0.05923153  0.04156024
## head  -0.13727991 -0.29038542
```

```
## aede1 -0.06511195  0.20210548
## aede2  0.28758616  0.52263544
## aede3 -0.04704291  0.01145841
##
## Proportion of trace:
##    LD1    LD2
## 0.8207 0.1793
```

```
# LDA output provides prior probability of each class that corresponds to
# the number of datapoint in each class. It also provides group means that will be
# used in to estimate the population means in unknowable world of each class.
# The coefficients of linear discriminants of each variable provide are used to form decision
# boundary using equation 4.24 in ISLR.

plot(lda.fit)
```




```
# This plot is computed by multiplying the coefficients of
# linear discriminants with each training observations.
# We can see that the model accurately separate each class.


# Predict using training observations
lda.pred <- predict(lda.fit)

lda.class <- lda.pred$class # Obtain the class from prediction
table(lda.class, flea$species)
```

```
##
## lda.class  1  2  3
##         1 21  0  0
##         2  0 22  0
##         3  0  0 31
```

```
mean(lda.class == flea$species)  # Accuracy
```

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

```
mean(lda.class != flea$species)  # Error rate
```

```
## [1] 0
```

```
## Interpretation of the accuracy
# According to the model performance based on training observations,
# the model perfectly classify each class, indicating by accuracy of 100% and
# the error rate of 0%. However, we need to keep in mind that we use training
# observations, not test observations. Again, we might get a poor performance
# when we use test dataset because of overfitting.
# Without the additional data, I cannot make conclusion on how this model actually performs.
```

Since both models give the same results, additional testing data are required to compare and choose the best model.