Homework 2

Logistic regression, linear discriminant analysis, and KNN classification

John Brandt

2/10/2018

Conceptual questions

Question 1

$$R(f) = E[I(Y \neq h(X))|X = x]$$
$$h(x) = \arg \max_{c \in \{1,\dots,K\}} \quad P(y = c|X = x)$$

$h(x)$ minimizes $R(f)$ because it maximizes the chance that $y \in \{1, \dots, K\}$

$$E[I(y \neq \arg \max_{c \in \{1,\dots,K\}} P(y = c|X = x)]$$

$R(f)$ is minimized when $P(y \neq h(x))$ is maximized where $h(x)$ is some classifier. If $h(x) = argmax_c \;\; P(y = c|X = x)$, then $R(f)$ is minimized because the expected value of $Y \neq c$, one of the classifiers, given the $argmax P(y = c|X = x)$ is zero.

Question 2

```
flowers <- iris
flowers$setosa <- 0
flowers$setosa[flowers$Species == "setosa"] <- 1
```

a)

```
flowers_m1 <- glm(setosa ~ Petal.Length, data = flowers, family = binomial)
```

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

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

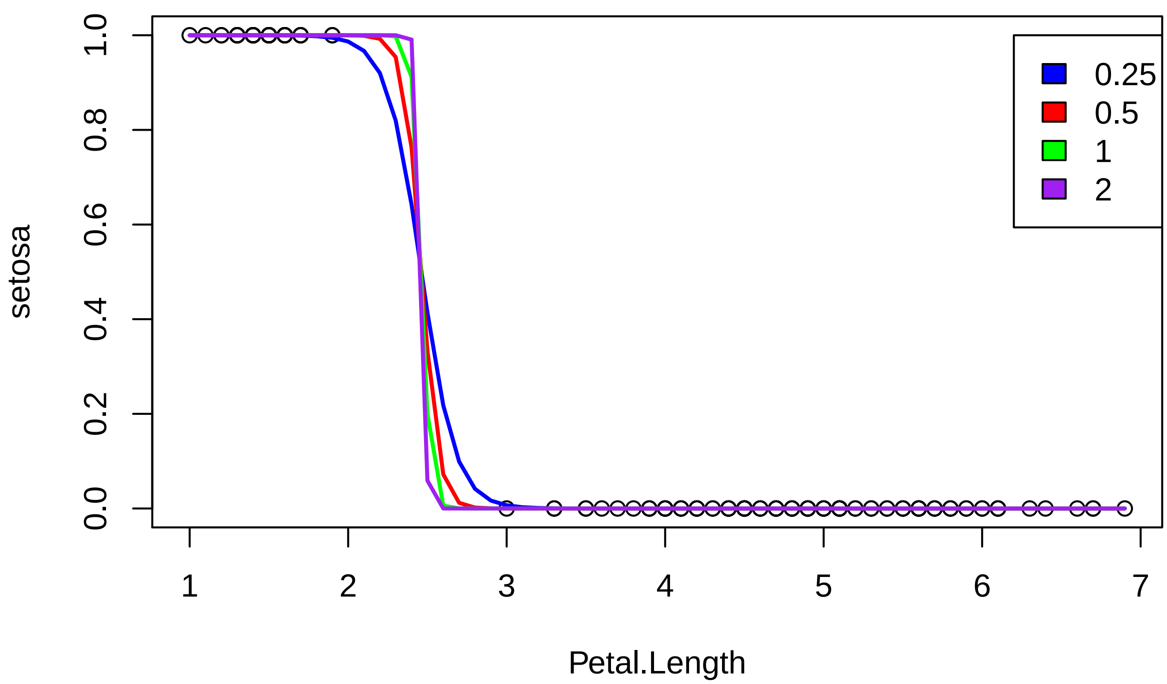
b)

```
coef(flowers_m1)
```

```
##  (Intercept) Petal.Length
##     91.67164    -37.22296
```

```
tmp <- data.frame(Petal.Length=seq(min(flowers$Petal.Length),
                                   max(flowers$Petal.Length), by = 0.1))
pred <- predict(flowers_m1, newdata=tmp, type="response")
```

```r
predict_prob <- function(multiplier){
  odds <- exp((coef(flowers_m1)[1]*multiplier) + (coef(flowers_m1)[2]*multiplier*tmp))
  prob <- odds / (1 + odds)
  return(prob)
}

predict_0.25 <- predict_prob(0.25)
predict_0.5 <- predict_prob(0.5)
predict_1 <- predict_prob(1)
predict_2 <- predict_prob(2)

plot(setosa ~ Petal.Length, data = flowers)
lines(tmp$Petal.Length, predict_0.25$Petal.Length, lwd=2, col = "blue")
lines(tmp$Petal.Length, predict_0.5$Petal.Length, lwd=2, col="red")
lines(tmp$Petal.Length, predict_1$Petal.Length, lwd=2, col="green")
lines(tmp$Petal.Length, predict_2$Petal.Length, lwd=2, col="purple")
legend(6.2,1, c("0.25", "0.5", "1", "2"), fill = c("blue", "red", "green", "purple"))
```



As β_c increases in size, the decision boundary becomes more linear and vertical, likely increasing the seperability of any test dataset.

c)

As β_c increases from 0.25 to 1, the log likelihood approaches but does not reach zero, thus failing to converge before identifying the β_0 and β_1

To derive a simpler equation for log likelihood, I apply log rules to the likelihood equation, and then construct the matrix algebra in R.

$$L(\beta) = \prod_{i=1}^{n} L_i(\beta) = \prod_{i=1}^{n} \left(\frac{e^{x_i^t \beta}}{1 + e^{x_i^t \beta}} \right)^{y_i} \cdot \left(\frac{1}{1 + e^{x_i^t \beta}} \right)^{1-y_i}$$

$$log\Big(L(\beta)\Big) = \sum_{i=1}^{n} y_i\Big[log\Big(e^{x_i^t\beta}\Big) - log\Big(1+e^{x_i^t\beta}\Big)\Big] + (1-y_i)log\Big(\frac{1}{1+e^{x_i^t\beta}}\Big)$$

$$log\Big(L(\beta)\Big) = \sum_{i=1}^{n}\Big[y_i x_i^t\beta - y_i log\Big(1+e^{x_i^t\beta}\Big) - (1-y_i)log\Big(1+e^{x_i^t\beta}\Big)\Big]$$

$$log\Big(L(\beta)\Big) = \sum_{i=1}^{n} y_i x_i^t\beta - log\Big(1+e^{x_i^t\beta}\Big)$$

```
betas0.25 <- c(coef(flowers_m1)[1]*0.25, coef(flowers_m1)[2]*0.25)
betas0.5 <- c(coef(flowers_m1)[1]*0.5, coef(flowers_m1)[2]*0.5)
betas1 <- c(coef(flowers_m1)[1], coef(flowers_m1)[2])
betas2 <- c(coef(flowers_m1)[1]*2, coef(flowers_m1)[2]*2)

#yi <- flowers$setosa
#xi <- flowers$Petal.Length

likelihood <- rep(NA, 150)

loglik <- function(input) {
  betas_matrix <<- matrix(data = input, nrow = 2, ncol = 1)
  for (i in 1:150) {
      xi_transp <- matrix(data = c(as.numeric(1),
                    as.numeric(flowers$Petal.Length[i])), nrow = 1, ncol = 2)
      xi_beta <- xi_transp %*% betas_matrix
      likelihood[i] <<- (flowers$setosa[i] * xi_beta) - log(1 + exp(xi_beta))
  }
  return(sum(likelihood))
}

loglik(betas0.25)
```

```
## [1] -0.02640396
```

```
loglik(betas0.5)
```

```
## [1] -0.0001060629
```

```
loglik(betas1)
```

```
## [1] -0.000000003666313
```

```
loglik(betas2)
```

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

d) The maximum likelihood parameter estimates become infinite when classes are linearly seperable. This essentially means that in the estimate $\frac{e^{\beta x}}{1+e^{\beta x}}$, any increase of β will improve the likelihood. So, $\frac{e^{10x}}{1+e^{10x}} \sim 0.999$ vs. $\frac{e^{\inf x}}{1+e^{\inf x}} \sim \overline{0.999}$

Applied Problem: Spam

Data pre-processing

```
data <- read.csv("spam.csv")
```

Correct column names

```
colnames(data)[colnames(data) == "char_freq_."] <-"char_freq_semicolon"
colnames(data)[colnames(data) == "char_freq_..1"] <-"char_freq_l_parens"
colnames(data)[colnames(data) == "char_freq_..2"] <-"char_freq_sl_brac"
colnames(data)[colnames(data) == "char_freq_..3"] <-"char_freq_exclam"
colnames(data)[colnames(data) == "char_freq_..4"] <-"char_freq_dol_sign"
colnames(data)[colnames(data) == "char_freq_..5"] <-"char_freq_hashtag"
```

Variables expected to be highly associated with spam

In order to determine which predictors to include in the spam classifier, I decided to simply consider which predictors were more likely to be associated with spam and with not-spam. To do so, I calculate a ratio between the average value of each predictor for spam and for not-spam.

For instance, a ratio of 165 means that the predictor was 165 times more common in spam e-mails than in non-spam e-mails. A ratio of 0.002 means that the predictor was 500 times more common in non-spam e-mails than in spam-emails.

```
# Group by spam column and calculate means for each predictor
data_means <- data %>%
  group_by(spam) %>%
  summarise_all(funs(mean))

# Create a dataframe that has columns "variable" and the average values
# for "not spam" and "spam"
data_means <- as.data.frame(t(data_means[,-1]))
data_means <- round(data_means, 3)
data_means <- tibble::rownames_to_column(data_means, "variable")
colnames(data_means) <- c("variable", "not_spam", "spam")

# Calculate the ratio between the average spam and not spam values
ratio <- round((data_means$spam/data_means$not_spam), 3)
data_means$ratio <- ratio

# Arrange columns by descending ratio and print the top 5 words associated with spam
# and the bottom 5 words associated with spam
data_means <- data_means %>%
  arrange(desc(ratio))

print(data_means[c(1:5,53:58),c(1,4)])
```

```
##              variable   ratio
## 1         word_freq_3d 165.000
## 2        word_freq_000  35.286
```

```
## 3     word_freq_remove  30.556
## 4     word_freq_credit  25.750
## 5   char_freq_dol_sign  14.500
## 53    word_freq_telnet   0.009
## 54   word_freq_meeting   0.009
## 55       word_freq_lab   0.006
## 56    word_freq_george   0.002
## 57        word_freq_cs   0.000
## NA                <NA>      NA
```

Test and train dataset

I create 80% train and 20% test subsets for model validation.

```r
set.seed(123)

s <- sample(1:nrow(data), nrow(data)/5, replace=FALSE)
test <- data[s,]
train <- data[-s,]
```

Logistic regression

The model formula was determined by starting with the predictors most and least associated with spam, and iteratively adding one more predictor until the addition of a new predictor did not decrease the residual deviance by more than 2%.

```r
m3 <- glm(spam ~ word_freq_3d + word_freq_000 + word_freq_remove + word_freq_credit +
            capital_run_length_longest + char_freq_dol_sign + word_freq_money +
            word_freq_free + word_freq_george + word_freq_lab + word_freq_meeting +
            word_freq_telnet + word_freq_project + word_freq_edu + word_freq_hp +
            char_freq_exclam, data=train, family=binomial)
```

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

```r
summary(m3)
```

```
##
## Call:
## glm(formula = spam ~ word_freq_3d + word_freq_000 + word_freq_remove +
##     word_freq_credit + capital_run_length_longest + char_freq_dol_sign +
##     word_freq_money + word_freq_free + word_freq_george + word_freq_lab +
##     word_freq_meeting + word_freq_telnet + word_freq_project +
##     word_freq_edu + word_freq_hp + char_freq_exclam, family = binomial,
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -5.2255  -0.3398   0.0000   0.0903   3.6273
##
## Coefficients:
##                             Estimate Std. Error z value Pr(>|z|)
## (Intercept)                -1.346073   0.089266 -15.079  < 2e-16 ***
## word_freq_3d                3.287185   1.803352   1.823  0.06833 .
```

```
## word_freq_000                 4.360734   0.702355   6.209 5.34e-10 ***
## word_freq_remove              3.444041   0.425855   8.087 6.10e-16 ***
## word_freq_credit              2.105833   0.716326   2.940  0.00328 **
## capital_run_length_longest    0.015921   0.001888   8.434  < 2e-16 ***
## char_freq_dol_sign            7.305691   0.916863   7.968 1.61e-15 ***
## word_freq_money               0.562917   0.202065   2.786  0.00534 **
## word_freq_free                1.120866   0.148877   7.529 5.12e-14 ***
## word_freq_george            -16.616383   2.187175  -7.597 3.03e-14 ***
## word_freq_lab                -3.160540   1.727773  -1.829  0.06736 .
## word_freq_meeting            -2.441604   0.769704  -3.172  0.00151 **
## word_freq_telnet             -3.747759   2.220731  -1.688  0.09148 .
## word_freq_project            -2.034798   0.720139  -2.826  0.00472 **
## word_freq_edu                -2.044488   0.326853  -6.255 3.97e-10 ***
## word_freq_hp                 -2.273681   0.306300  -7.423 1.14e-13 ***
## char_freq_exclam              0.626683   0.120018   5.222 1.77e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4934.3  on 3680  degrees of freedom
## Residual deviance: 1726.7  on 3664  degrees of freedom
## AIC: 1760.7
##
## Number of Fisher Scoring iterations: 12
```

GLM - training error

```r
# confusion matrix
table(fitted=(fitted(m3) >=0.5)*1, actual=train$spam)
```

```
##       actual
## fitted    0    1
##      0 2148  221
##      1   85 1227
```

```r
#training error rate
round(mean((fitted(m3) >= 0.5)*1 != train$spam),3)
```

```
## [1] 0.083
```

GLM - test error

```r
# predict GLM
glm_predict <- predict(m3, newdata = test)

# conusion matrix
table(fitted = (glm_predict >=0.5), actual=test$spam)
```

```
##        actual
## fitted    0   1
##   FALSE 538  69
##   TRUE   17 296
```

```r
# test error rate
round(mean((glm_predict >0.5) != test$spam),3)
```

```
## [1] 0.093
```

LDA

The LDA uses the same predictors as the GLM.

```r
lda1 <- lda(spam ~ word_freq_3d + word_freq_000 + word_freq_remove + word_freq_credit +
              capital_run_length_longest + char_freq_dol_sign + word_freq_money +
              word_freq_free + word_freq_george + word_freq_lab +
              word_freq_meeting + word_freq_telnet + word_freq_project +
              word_freq_edu + word_freq_hp + char_freq_exclam, data=train)
```

```r
lda1_pred <- predict(lda1)$class
```

Training error rate

```r
# Confusion matrix
table(LDA_predicted=lda1_pred, actual=train$spam)
```

```
##                actual
## LDA_predicted    0    1
##               0 2192  533
##               1   41  915
```

```r
# Training error rate
round(mean(lda1_pred != train$spam),3)
```

```
## [1] 0.156
```

Test error rate

```r
lda_test <- predict(lda1, newdata = test)$class

# confusion matrix
table(lda_test, test$spam)
```

```
##
## lda_test   0   1
##        0 539 142
##        1  16 223
```

```r
round(mean(lda_test != test$spam), 3)
```

```
## [1] 0.172
```

KNN

The KNN uses the same predictors as the LDA and GLM models.

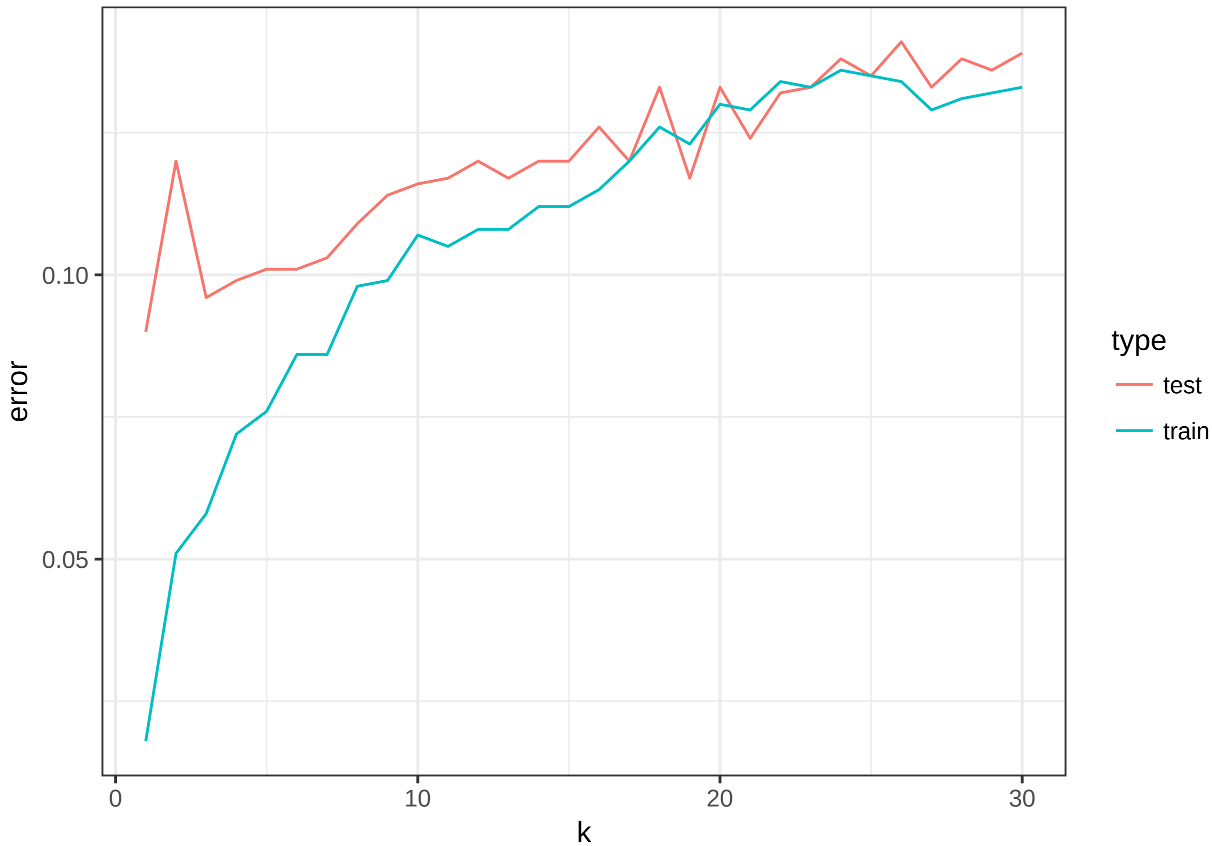
```r
train = train[,1:58]

train_knn <- subset(train, select=c("word_freq_3d", "word_freq_000", "word_freq_remove",
                              "word_freq_credit", "capital_run_length_longest",
                              "char_freq_dol_sign", "word_freq_money", "word_freq_free",
                              "word_freq_george", "word_freq_lab", "word_freq_meeting",
                              "word_freq_telnet", "word_freq_project", "word_freq_edu",
                              "word_freq_hp", "char_freq_exclam"))

test_knn <- subset(test, select=c("word_freq_3d", "word_freq_000", "word_freq_remove",
                              "word_freq_credit", "capital_run_length_longest",
                              "char_freq_dol_sign", "word_freq_money", "word_freq_free",
                              "word_freq_george", "word_freq_lab", "word_freq_meeting",
                              "word_freq_telnet", "word_freq_project", "word_freq_edu",
                              "word_freq_hp", "char_freq_exclam"))
```

```r
knn_df <- as.data.frame(matrix(0, 60, 3))
colnames(knn_df) <- c("error", "type", "k")

knn_df[31:nrow(knn_df), 2] <- "test"
knn_df[1:30, 2] <- "train"

for (i in 1:30) {
  model_train <- knn(train=train_knn, test=train_knn, k=i, cl=train[,58])
  model_test <- knn(train=train_knn, test=test_knn, k=i, cl=train[,58])
  knn_df[i,1] <- round(mean(model_train != train$spam), 3)
  knn_df[i + 30,1] <- round(mean(model_test != test$spam), 3)
  knn_df[i,3] <- i
  knn_df[i+30, 3] <- i
}
```

```r
require(ggplot2)
```

```
## Loading required package: ggplot2
```

```r
plot <- ggplot(aes(x=k, y=error), data=knn_df)+
  geom_line(aes(color=type))+
  theme_bw()

print(plot)
```



KNN - Training error rate

```
knn_train_model <- knn(train=train_knn, test=train_knn, k=3, cl=train[,58])

# confusion matrix
table(Predicted=knn_train_model, actual=train$spam)
```

```
##          actual
## Predicted    0    1
##         0 2134  116
##         1   99 1332
```

```
# error rate
round(mean(knn_train_model != train$spam),3)
```

```
## [1] 0.058
```

KNN - Test error rate

```
knn_test <- knn(train=train_knn, test=test_knn, k=3, cl=train[,58])

# confusion matrix
table(Predicted=knn_test, actual=test$spam)
```

```
##          actual
```

```
## Predicted   0   1
##         0 514  47
##         1  41 318
```

```r
# error rate
round(mean(knn_test != test$spam),3)
```

```
## [1] 0.096
```

Model selection

Linear discriminant analysis performed noticeably worse than did the other approaches. This is likely because, with 16 predictors, drawing a linear boundary in 16 dimensional space is not possible without some margin of error. KNN and logistic regression had very similar training and test error rates. KNN forgoes a linear decision boundary in favor of a nearest neighbors approach. This works well because the number of observations is large enough that points are still somewhate dense in 16 dimensional space. The logistic regression likely performed better than the LDA because it does not require multivariate normality and is generally a more flexible model.

KNN slightly outperformed logistic regression for this dataset. With 6 and 10% training and test errors, respectively, versus 8 and 9%.