

## COEN 281, Homework 2 - Linear Classifiers

Pratishtha Deep(W1285877), Yutong Li (W1182922)

November 6, 2018

```
require("MASS")
```

### #Que 1:

*#1)a. Use the read.table command to load this data into R.*

```
set.seed(123)
```

```
data <- read.table("C:/Users/pratishtha/Downloads/data  
mining/HW_export/HW2/data/az-5000.txt", header = TRUE)
```

```
head(data, 1)
```

```
## char      x1      y1      x2      y2 x3      y3      x4      y4  
## 1      n 0.1875 0.140625 0.09375 0.515625 0 0.8828125 0.1796875 0.5078125  
##      x5      y5      x6      y6      x7      y7      x8  
## 1 0.4609375 0.140625 0.640625 0.109375 0.515625 0.5390625 0.3828125  
##      y8      x9      y9  
## 1 0.8828125 0.671875 0.9765625
```

```
tail(data, 1)
```

```
## char      x1      y1      x2      y2      x3      y3      x4 y4  
## 5000      e 0.2265625 0.125 0.4921875 0.2890625 0.84375 0.2109375 0.84375 0  
##      x5      y5      x6      y6 x7      y7      x8      y8  
## 5000 0.390625 0.0234375 0.15625 0.203125 0 0.5859375 0.1328125 0.953125  
##      x9      y9  
## 5000 0.5234375 0.9921875
```

```
dim(data)
```

```
## [1] 5000  19
```

*#1)b. Use the sample command to randomly select 80% of the data for training.*

```
#train <- sample(1:5000, 4000)
```

```
train <- sample.int(n = nrow(data), size = floor(.8*nrow(data)), replace = F)
```

#1)c. Use the table command to show the number of cases per class in the training data.

```
table(data$char[train])
```

```
##
##  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r
## 141 154 153 158 149 153 156 144 149 149 144 163 158 134 172 171 155 183
##  s  t  u  v  w  x  y  z
## 167 145 167 151 171 138 152 123
```

## #Que:2 Linear Discriminant Analysis.

#2)a. Use the c() command to create a vector of prior probabilities equal to 1/26 for each class.

```
char.priors <- c(rep(1/26, 26))
```

#2)b. Use the lda command to run linear discriminant analysis on the training data with the equal priors above.

```
Char.lda <- lda(char ~., data, subset = train, prior = char.priors)
```

#2)c. Combine the functions table and predict to print a "confusion" matrix on the test data.

```
Char.confusion <- table(data[-train,]$char, predict(Char.lda,
data[-train,])$class)
Char.confusion
```

```
##
##      a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w
## a 21  0  0  6  0  0  1  0  0  0  0  1  0  0  0  0  0  0  0  1  0  0
## b  0 25  0  0  1  1  0  0  0  0  0  2  0  0  0  0  0  0  1  0  0  0
## c  0  0 41  0  2  0  0  2  0  0  0  1  0  0  0  0  0  0  0  0  0  0
## d  1  0  0 25  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  1  0  0
## e  0  0  1  0 44  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## f  0  0  0  1  0 27  0  0  0  0  0  2  0  0  0  2  0  1  0  1  0  0
## g  0  0  0  0  1  3 20  0  0  0  0  0  0  0  0  0  1  0  5  2  0  0
## h  0  1  0  0  1  0  0 24  0  0  1  1  0  3  0  0  0  0  0  2  0  0
## i  0  0  0  0  0  1  0  0 29  6  0  2  0  0  0  0  0  1  0  4  0  0
## j  0  1  0  0  0  1  0  0  1 34  0  0  0  0  1  0  0  0  1  3  0  0
## k  0  0  0  0  0  1  0  2  0  0 20  0  3  1  0  0  0  0  0  1  0  0
## l  0  0  0  2  2  0  0  3  0  0  0 31  0  0  0  0  0  0  0  0  0  0
## m  0  0  0  0  0  0  0  0  0  0  1  0 32  0  0  0  0  0  0  0  0  1
## n  0  0  0  0  0  0  0  1  0  0  1  0  3 25  0  0  0  0  0  0  0  0
## o  0  0  1  0  0  0  1  0  0  0  0  0  0  0 33  0  0  0  2  0  0  1
```

```
## p 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 36 0 0 1 0 0 0 0
## q 1 0 0 0 1 1 4 0 0 0 0 1 0 0 0 0 32 0 1 0 0 0 0
## r 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 29 0 0 1 1 0
## s 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 36 1 0 0 0
## t 0 2 0 0 1 2 0 0 0 0 0 0 0 0 1 0 0 2 0 35 0 1 0
## u 1 0 0 0 0 0 0 1 0 0 1 0 0 2 0 0 0 0 0 0 40 8 0
## v 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 2 43 2
## w 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 25
## x 2 1 3 1 1 0 0 0 0 0 0 0 1 0 4 0 0 2 0 0 0 5 0
## y 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 2 0 0 0
## z 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##
##      x y z
## a 3 1 3
## b 0 0 0
## c 0 0 0
## d 1 0 0
## e 0 0 0
## f 0 0 0
## g 0 4 0
## h 0 0 1
## i 0 0 0
## j 1 2 0
## k 0 1 0
## l 1 0 0
## m 0 0 0
## n 0 0 0
## o 1 0 0
## p 0 0 0
## q 0 1 0
## r 0 0 0
## s 0 0 0
## t 2 0 0
## u 0 0 0
## v 0 0 0
## w 0 0 0
## x 18 5 2
## y 0 34 0
## z 0 1 35
```

*#computing which character had the best/worst performance*

```
for (i in seq(1,26)) {
  cat (row.names(Char.confusion)[i], " ")
  cat (Char.confusion[i,i]/sum(Char.confusion[i,]), "\n")
}
```

```
## a 0.5675676
## b 0.8333333
## c 0.8913043
## d 0.862069
## e 0.9565217
## f 0.7941176
## g 0.5555556
## h 0.7058824
## i 0.6744186
## j 0.7555556
## k 0.6896552
## l 0.7948718
## m 0.9411765
## n 0.8333333
## o 0.8461538
## p 0.972973
## q 0.7619048
## r 0.8529412
## s 0.9
## t 0.7608696
## u 0.754717
## v 0.877551
## w 0.9259259
## x 0.4
## y 0.8717949
## z 0.9459459
```

*#Character "P" had the best performance. Character X has worst.*

*#2)d. What was the total accuracy on the test and train sets?*

```
length(which(predict(Char.lda, data[-train,])$class == data[-train,]$char))
```

```
## [1] 794
```

*#794/1000 #or 79.4%*

```
length(which(predict(Char.lda, data[train,])$class == data[train,]$char))
```

```
## [1] 3136
```

*#3136/4000 #or 78.9%*

## ***#Que:3 Logistic Regression.***

*#Loading credit data*

```
credit_data <- read.table("C:/Users/pratishtha/Downloads/data
```

```

mining/HW_export/HW2/data/credit_data.txt", header = TRUE)

dim(credit_data)

## [1] 885  15

set.seed(123)

#selecting 80% of data as training data

train <- sample(1:885, 0.8*885)

#showing the number of cases per class for both training and test data

table(credit_data$Fail[train])

##
##    0    1
## 547 161

table(credit_data$Fail[-train])

##
##    0    1
## 146   31

#3)a. Use the glm (with family=binomial) command to fit a logistic regression
to predict which firms will go bankrupt. Report the table of coefficients
from R with their p-values. What are the 4 most important predictor
variables?

#fitting logistic regression to training data and creating summary data

credit.glm <- glm(Fail~.-Id, family = binomial, data = credit_data[train,])
summary(credit.glm)

##
## Call:
## glm(formula = Fail ~ . - Id, family = binomial, data = credit_data[train,
##      ])
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -2.00600  -0.59629  -0.29571  -0.02315   2.42949
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -3.495673   0.854985  -4.089 4.34e-05 ***
## Leverage     -0.380287   0.253589  -1.500 0.133713

```

```
## CumulProfit -2.373825 0.725865 -3.270 0.001074 **
## Liquid -14.881906 3.528628 -4.217 2.47e-05 ***
## OverDueDebt 1.746010 0.312334 5.590 2.27e-08 ***
## WorkCap -1.642276 0.589250 -2.787 0.005319 **
## OperProfit -0.624649 0.926995 -0.674 0.500411
## ShortDebt 2.405079 0.646639 3.719 0.000200 ***
## GuarDebt -1.060947 0.454228 -2.336 0.019506 *
## StateLag 0.007791 0.002092 3.724 0.000196 ***
## Fiscallag -0.004302 0.003036 -1.417 0.156519
## InFinan -0.217202 0.497453 -0.437 0.662380
## Links 6.302690 2.497104 2.524 0.011603 *
## CapStruct 2.277517 0.769851 2.958 0.003093 **
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 759.14 on 707 degrees of freedom
## Residual deviance: 507.07 on 694 degrees of freedom
## AIC: 535.07
##
## Number of Fisher Scoring iterations: 7
```

OperProfit, InFinan, Capstruct and Fiscallag are the 4 most important predictor variables

(b) Do their signs appear to be what you'd expect?

Yes.

#3)c. Suppose that we predict a firm will go bankrupt if the predicted probability  $P(Y = 1 | X = x)$  of bankruptcy is 0.5 or greater. Find the confusion matrix for such predictions on the test data.

```
yHat <- predict(credit.glm , credit_data[-train, c(1,3:15)], type =
"response")
table(credit_data$Fail[-train], yHat >= 0.5)

##
##      FALSE TRUE
## 0      136   10
## 1       17   14
```

## #Que:4 Regularized Logistic Regression.

```
library(glmnet)
```

*#loading credit data*

```
credit_data <- read.table("C:/Users/pratishtha/Downloads/data  
mining/HW_export/HW2/data/credit_data.txt", header = TRUE)
```

```
dim(credit_data)
```

```
## [1] 885 15
```

```
set.seed(123)
```

*#selecting 80% of data as training data*

```
train <- sample(1:885, 0.8*885)
```

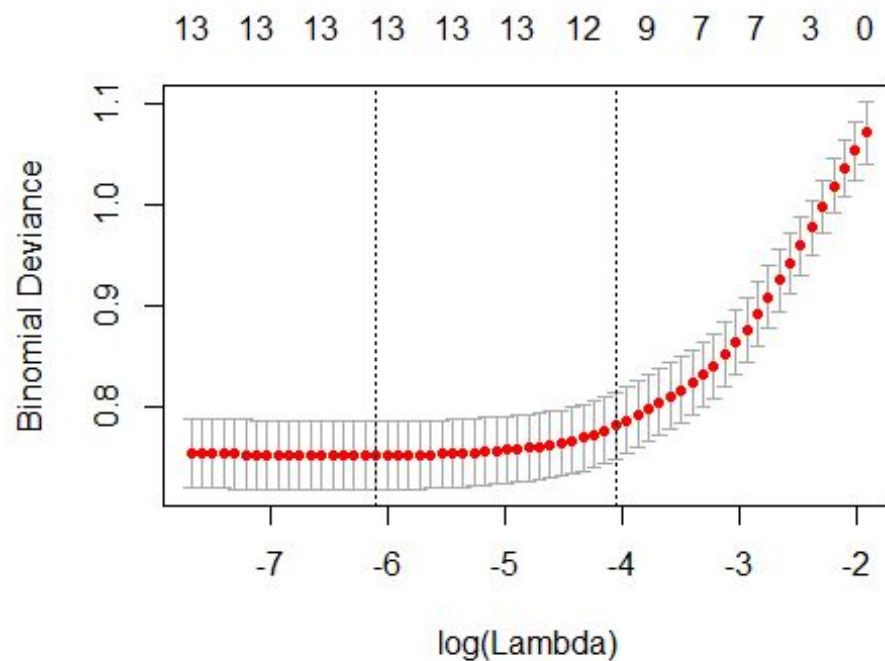
*# refactoring credit data frame as a matrix and mapping y (response variables) values from 0/1 to -1/1*

```
x <- as.matrix(credit_data[train, 3:15])
```

```
y <- 2*credit_data$Fail[train]-1
```

*#4)a. Use the cv.glmnet (with family=binomial) command to fit a regularized logistic regression to the same training data used in 3a (you may need a cast from data.frame to matrix and map y from 0/1 to -1/1). Plot the cross-validation curve. Explain the plot.*

```
credit.glmnet <- cv.glmnet(x, y, family = "binomial")  
plot(credit.glmnet)
```



*#Explanation: Error decreases between Lambda min and Lambda max and starts flattening out after about 10 predictors have non-zero coefficients.*

*#4)b. The object returned by `cv.glmnet()` contains the value of the best Lambda. Pass this value of Lambda to the `coef()` function to retrieve the corresponding coefficient vector. Print the coefficients. Compare to your answer in 3a.*

```
beta <- coef(credit.glmnet, lambda = credit.glmnet$lambda.1se)
print(beta)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
```

```
##              1
## (Intercept) -3.595794543
## Leverage    -0.321794081
## CumulProfit -1.720488002
## Liquid      -5.459086298
## OverDueDebt  1.292955269
## WorkCap     -0.496743660
## OperProfit   .
## ShortDebt    1.435705602
## GuarDebt    -0.186213667
## StateLag     0.004150093
## Fiscallag    .
## InFinan     .
```

```
## Links      2.538650202
## CapStruct  2.030230395
```

*#Explanation: Three predictors, OperProfit, FiscalLag and InFinan are equal to zero.*

*#4)c. (c) Use the predict function with the same value of lambda to predict on the test data. Show the confusion matrix. Compare the accuracy with 3c.*

```
x.test <- as.matrix(credit_data[-train, c(3:15)])
y.test <- credit_data$Fail[-train]

yHat <- as.numeric(predict(credit.glmnet, x.test, type = "class", lambda =
credit.glmnet$lambda.1se))
conf_mat <- table(y.test, yHat)

accuracy <- sum(diag(conf_mat))/sum(conf_mat)
accuracy

## [1] 0.8305085
```

## #Que 5)

a.

If  $x < 0.05$ , we will use observation in the interval  $[0, x+0.05]$  which represents a fraction of  $(100x + 5)\%$

If  $x > 0.95$ , the fraction of observations we will use is  $(105 - 100x)\%$

$$\int_{0.05}^{0.95} 10dx + \int_0^{0.05} (100x + 5)dx + \int_{0.95}^1 (105 - 100x)dx = 9 + 0.375 + 0.375 = 9.75.$$

So on average, the fraction of available observations we will use to make the prediction is 9.75%

b.

Assume  $X_1$  and  $X_2$  are independent, the fraction of available observations we will use to make the prediction is

$$9.75\% * 9.75\% = 0.951\%$$

c.

On average,  $9.75\%^{100} \approx 0$

d.

As  $p$  increases linearly, observations that are geometrically near decrease exponentially. When there are large number of dimensions, the percentage of observations that can be used to predict with KNN becomes very small. This means that for a set sample size, more features lead to fewer neighbors.

$$\lim_{p \rightarrow \infty} (9.75\%)^p = 0$$

e.

For  $p=1$ , side = 0.1

For  $p=2$ , side =  $0.1^{(1/2)} = 0.316$

For  $p=100$ , side =  $0.1^{(1/100)} = 0.977$

This is saying that when the number of features is high (i.e.  $p=100$ ), to use on average 10% of the training observations would mean that we would need to include almost the entire range of each individual feature.

## *#Que 6). Cross-validation.*

*#6)a.*

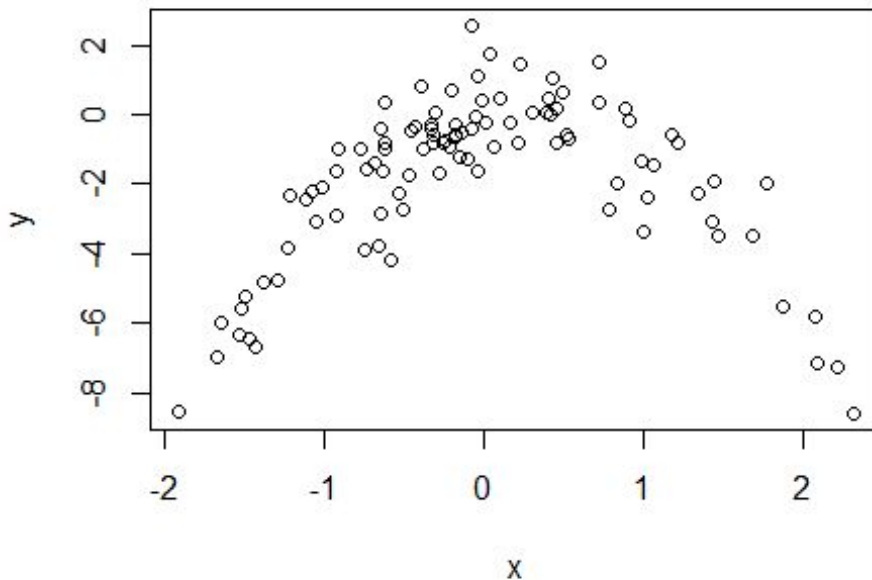
```
library(boot)
set.seed(1)
y <- rnorm(100)
x <- rnorm(100)
y <- x - 2*x^2 + rnorm(100)
```

$n = 100, p = 2.$

$$Y = X - 2X^2 + \epsilon$$

*#6)b.*

```
plot(x, y)
```



*#Relationship between X and Y is quadratic*

X from about -2 to 2. Y from about -8 to 2.

*#6)c.*

```
set.seed(1)
df <- data.frame(y, x, x2=x^2, x3=x^3, x4=x^4)
fit1 <- glm(y ~ x, data=df)
cv.err1 <- cv.glm(df, fit1)
cv.err1$delta

## [1] 5.890979 5.888812

fit2 <- glm(y ~ x + x2, data=df)
cv.err2 <- cv.glm(df, fit2)
cv.err2$delta

## [1] 1.086596 1.086326

fit3 <- glm(y ~ x + x2 + x3, data=df)
cv.err3 <- cv.glm(df, fit3)
cv.err3$delta

## [1] 1.102585 1.102227
```

```
fit4 <- glm(y ~ x + x2 + x3 + x4, data=df)
cv.err4 <- cv.glm(df, fit4)
cv.err4$delta
```

```
## [1] 1.114772 1.114334
```

*#6)d.*

```
set.seed(2)
df <- data.frame(y, x, x2=x^2, x3=x^3, x4=x^4)
fit1 <- glm(y ~ x, data=df)
cv.err1 <- cv.glm(df, fit1)
cv.err1$delta
```

```
## [1] 5.890979 5.888812
```

```
fit2 <- glm(y ~ x + x2, data=df)
cv.err2 <- cv.glm(df, fit2)
cv.err2$delta
```

```
## [1] 1.086596 1.086326
```

```
fit3 <- glm(y ~ x + x2 + x3, data=df)
cv.err3 <- cv.glm(df, fit3)
cv.err3$delta
```

```
## [1] 1.102585 1.102227
```

```
fit4 <- glm(y ~ x + x2 + x3 + x4, data=df)
cv.err4 <- cv.glm(df, fit4)
cv.err4$delta
```

```
## [1] 1.114772 1.114334
```

*#Results are exactly the same because LOOCV predicts every observation using the all of the rest (no randomness involved)*

*#6)e.*

*#The quadratic model using X and X^2 had the lowest error because the true model was generated using a quadratic formula.*

*#6)f.*

```
fit0 <- lm(y ~ poly(x,4))
summary(fit0)
```

```
##
```

```
## Call:
```

```
## lm(formula = y ~ poly(x, 4))
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.8914 -0.5244  0.0749  0.5932  2.7796
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.8277     0.1041  -17.549  <2e-16 ***
## poly(x, 4)1    2.3164     1.0415   2.224   0.0285 *
## poly(x, 4)2  -21.0586     1.0415 -20.220  <2e-16 ***
## poly(x, 4)3   -0.3048     1.0415  -0.293   0.7704
## poly(x, 4)4   -0.4926     1.0415  -0.473   0.6373
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.041 on 95 degrees of freedom
## Multiple R-squared:  0.8134, Adjusted R-squared:  0.8055
## F-statistic: 103.5 on 4 and 95 DF,  p-value: < 2.2e-16
```

*#Summary shows that only X and X^2 are statistically significant predictors. This agrees with the LOOCV results that indicate using only X and X^2 produces the best model.*