Applied Data Science HW2

Xiaoyao Yang
Columbia University

March 5, 2014

1 Sergeis problem

1.1 Categorical Variables with few level

First, we read data set using read.table function. And then we use For loop as well as model.matrix function to transfer Categorical Variables with limited level to Dummy Matrix. Noted that we only use for loop here one time and use model.matrix for other variables.

```
dat = read.table("columbia_data_set.csv", sep = ",", header = T)
n <- dim(dat)[1]
hour <- dat$hour
I_hour <- matrix(0, nrow = n, ncol = 24)
for (i in 1:n) {
    I_hour[i, hour[i]] <- 1
}
# hour<-as.factor(dat$hour) model.matrix(~hour-1)->a

### state
state <- as.factor(dat$state)
dummy_state <- model.matrix(~state - 1)
### browser.id
browser.id <- as.factor(dat$browser.id)
dummy_browser.id <- model.matrix(~browser.id - 1)
### ad.size
ad.size <- as.factor(dat$ad.size)
dummy_ad.size <- model.matrix(~ad.size - 1)
### ad.size
ad.size <- as.factor(dat$ad.size)
dummy_ad.size <- model.matrix(~ad.size - 1)
```

And we can take a look at first several rows of our Dummy Set

```
head(dummy_ad.size)

##   ad.size160x600 ad.size300x250 ad.size728x90
## 1              0              1             0
## 2              1              0             0
## 3              0              1             0
## 4              1              0             0
## 5              0              0             1
## 6              0              1             0
```

If we want to combined these Dummy Matrix. We can use cbind() function to do it.

1.2 Site.id

```
table(dat$site.id)

##
##     1     2     3     4     5     6     8     9    10    11    12    13
##   185    99    44    12    21    14   133    48     1     6     3    34
##    14    15    16    17    18    19    20    22    23    24    26    27
##     9     1   201     1     1   175    35    83     1     3    10   175
##    30    31    32    33    34    35    36    37    38    39    40    41
##    16    37  1826    19   358    81    58  1109    14     4     9     9
##    42    43    44    47    48    49    50    51    52    53    54    55
##    55    67   368    69    55   403    29    14    25    57   158   477
##    56    57    58    59    61    62    63    64    66    67    68    69
##    55     1    11     1    32 45851    35    15     2     5     7   199
##    70    71    72    73    74    75    76    77    78    79    80    81
##    26   151   118     6    18     4    22   715   129     1    98    11
##    82    83    84    85    86    87    88    89    90    91    92    93
##     8    30    37     6   384    58     8   147    17     5    38     3
##    95    99   100   101   102   103   104   105   107   108   109   110
##    10    10     6   156  4121    30     2    23    68     2    10     1
##   111   112   113   115   116   119   120   122   124   126   127   128
##    32    10    60     5    31    71  5780    79     1  1394    92     2
##   131   132   133   134   135   136   137   138   139   140   142   143
##    40   100    89     3    12    14    31   126    64    39   104   121
##   144   145   146   147   148   149   150   151   152   153   154   155
##    28     1    46    37    23    16     7    32    12    50    36    55
##   156   157   158   159   161   162   163   164   165   166   167   168
##    14   123     1    18     3    26   189     3    14     1    18     6
##   169   170   172   173   174   175   176   177   178   179   180   181
##     1     3    51     5    38    30     2    72    14    44     4   171
##   182   183   184   185   186   187   188   189   190   191   192   193
##    16    30     3    97   293    33   462 11365    23   868    11  1328
##   194   195   198   199   200   202   203   204   205   206   207   209
##    32    40   492   103   369    31    52    81   184    81    20    10
##   210   211   212   213   214   215   217   218   219   220   221   222
##    34   121    69    14    15  1598     4    23    76   879   303   804
##   223   224   225   229   231   232   233   234   236   237   238   242
##     1    82  2151     2    17   399     1    23    29    26     2     1
##   244   246   247   248   249   250   252   253   254   255   256   257
##    41     4    43  1430     9   216     3    12     4    16     1   306
##   259   260   261   262   264   266   267   268   269   270   271   272
##   135     2    60     1     9   325   291    11    44   109    21     4
##   273   274   275   276   277   278   279   280   281   282   283   284
##    48     4     5    30    12   366    57    90   307    64    24    24
##   285   286   287   288   289   290   291   295   296   297   298   299
##   184   632    89  3376     3     1     2     1    24     1    34     3
##   301   302   303   306   307   308   310
##     1     3     4     3     2     9     3
```

From the table above we can see that there are too many different site.id. Thus, we pick up most frequent id and put other id in one column.

As a result, there is 256 different level for site.id variable. After simulate the cumulative density function, we finally collect 136 variables from 256 and put the rest variable into "Other" column.

```
### site.id
site <- dat$site.id
site.t <- table(site)
temp <- sort(site.t, decreasing = TRUE)
site.cdf <- cumsum(temp)/sum(temp)
site.names <- names(temp)[site.cdf < 0.99]
site.num <- as.numeric(site.names)
# order number
site.names <- site.names[order(site.num)]
n.site <- length(site.names)
I_site <- matrix(0, nrow = n, ncol = n.site + 1)
colnames(I_site) <- c(site.names, "Others")
for (i in 1:n) {
    l.temp <- which(site[i] == as.numeric(site.names))
    if (length(l.temp) == 0)
        I_site[i, n.site + 1] <- 1 else I_site[i, l.temp] <- 1
}
```

We can take a look at the first rows of our Dummy Matrix for Site.id

```
head(I_site)

##      1 2 3 8 9 13 16 19 20 22 27 31 32 34 35 36 37 42 43 44 47 48 49 50 52
## [1,] 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## [2,] 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## [3,] 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## [4,] 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## [5,] 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## [6,] 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
##      53 54 55 56 61 62 63 69 70 71 72 77 78 80 83 84 86 87 89 92 101 102
## [1,]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0   0   0
## [2,]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0   0   0
## [3,]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0   0   1
## [4,]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0   0   0
## [5,]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0   0   0
## [6,]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0   0   0
##      103 107 111 113 116 119 120 122 126 127 131 132 133 137 138 139 140
## [1,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [2,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [3,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [4,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [5,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [6,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##      142 143 144 146 147 151 153 154 155 157 162 163 172 174 175 177 179
## [1,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [2,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [3,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [4,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [5,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [6,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##      181 183 185 186 187 188 189 191 193 194 195 198 199 200 202 203 204
## [1,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [2,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [3,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

```
## [4,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [5,]   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0
## [6,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0
##      205 206 210 211 212 215 219 220 221 222 224 225 232 236 237 244 247
## [1,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [2,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [3,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [4,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [5,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [6,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##      248 250 257 259 261 266 267 269 270 273 276 278 279 280 281 282 285
## [1,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [2,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [3,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [4,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [5,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## [6,]   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##      286 287 288 298 Others
## [1,]   0   0   1   0      0
## [2,]   0   0   1   0      0
## [3,]   0   0   0   0      0
## [4,]   0   0   0   0      1
## [5,]   0   0   0   0      0
## [6,]   0   0   0   0      0
```

2 Textbook Page 39

2.1 Regression

First, we generate 6 sets of data follow by Normal Distribution (they are independent to each other). Then, we use 3 of them as x1 and 3 of them as x2. In this chapter, we use linear regression to classify our simulated data.

```
set.seed(10)
x1 <- c(rnorm(100, 5, 2.5), rnorm(100, 5, 2), rnorm(100, 10, 2))
x2 <- c(rnorm(100, 5, 2), rnorm(100, 10, 2), rnorm(100, 7, 2.5))
y <- c(rep(0, 100), rep(1, 100), rep(2, 100))

plot(c(0, 17), c(0, 20), type = "n", xlab = "X1", ylab = "X2")
points(x1[1:100], x2[1:100], col = 1)
points(x1[101:200], x2[101:200], col = 2)
points(x1[201:300], x2[201:300], col = "cornflowerblue")
dat2 <- data.frame(y = y, x1 = x1, x2 = x2)

reg <- lm(y ~ x1 + x2)
betas <- reg$coefficients
x <- c(1:15000)/1000
y <- (1 - 1/3 - betas[1] - betas[2] * x)/betas[3]
lines(x, y, lwd = 2, col = "green")
y1 <- y <- (1 + 1/3 - betas[1] - betas[2] * x)/betas[3]
lines(x, y1, lwd = 2, col = "green")
```

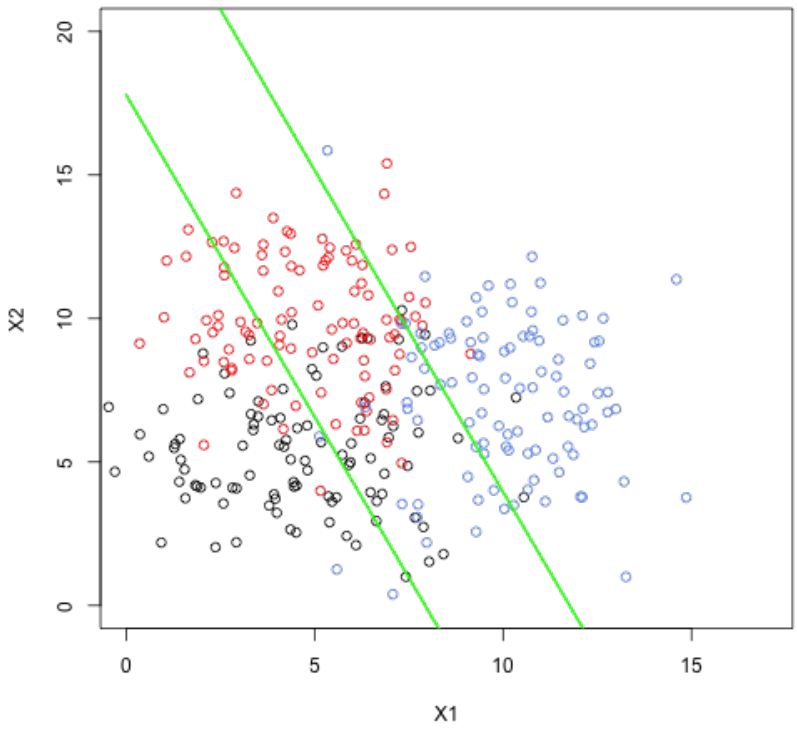

Figure 1: Three class simulated data seperated by linear regression

Figure 1 shows our data point as well as the linear regression classifier. Here we try to seperate three classes with Linear regression. We simply set each classes value as 0, 1 and 2. And if response value less than 2/3, we believe it belongs to class 0; if it is larger than 2/3 but less than 4/3, that point belongs to class 1; if the response value is larger than 4/3, we put it into class 2.

There are several drawbacks for this methods. One of them is that we assumed all data should fall in to our plane and thus all possible points have response from 0 to 2, which is obviously not true. We might use Cross Validation to find the best seperate point (here is 2/3 and 4/3). Still, even though we can find different ways to improve linear regression method, it not regular to use linear regression to solve classification problem.

2.2 KNN

```
grid <- function() {
    xx1 <- seq(0, 15, length = 1000)
    xx2 <- seq(0, 20, length = 1000)
    dat3 <- matrix(0, ncol = 2, nrow = 1000 * 1000)
    for (i in 1:1000) {
        for (j in 1:1000) {
            dat3[j + i * 1000 - 1000, 1] <- xx1[i]
            dat3[j + i * 1000 - 1000, 2] <- xx2[j]
        }
    }
    mygrid <- data.frame(dat3)
}

NN <- function() {
    Neighbors <<- rep(0, nrow(mygrid))
    for (i in c(1:nrow(mygrid))) {
        distances <- (mygrid$X1[i] - x1)^2 + (mygrid$X2[i] - x2)^2
        sort.distances <- sort.int(distances, index.return = TRUE)
        # k=15
        if ((sum(sort.distances$ix[1:15] > 200)/15) > 0.5)
            Neighbors[i] <- 3 else if ((sum(sort.distances$ix[1:15] > 100)/15) > 0.5)
            Neighbors[i] <- 2 else Neighbors[i] <- 1
    }
    Neighbors <<- Neighbors
}

plotKNN <- function() {
    plot(c(0, 15), c(0, 20), type = "n", xlab = "X1", ylab = "X2")
    points(mygrid$X1[Neighbors == 1], mygrid$X2[Neighbors == 1], col = "skyblue")
    points(mygrid$X1[Neighbors == 2], mygrid$X2[Neighbors == 2], col = "pink")
    # points(mygrid£X1[Neighbors==3],mygrid£X2[Neighbors==3],col='green')
    points(x1[101:200], x2[101:200], col = "RED", pch = 20)
    points(x1[1:100], x2[1:100], col = "BLUE", pch = 20)
    points(x1[201:300], x2[201:300], col = "black", pch = 20)
}
```

The KNN classifier is shown in Figure 2. Instead of using knn funciton in package “class”, we generate grid - a 1000000*2 Matrix - trying to cover every point in our plane. And by determining which class every point belongs to, we can obtain the KNN classifier boundry of our plane. If we want to draw that classifier line specifically, we can find the point whose at least two value of distance to class that calculated before are identical. Those point with identical distance consist of our KNN classifier.

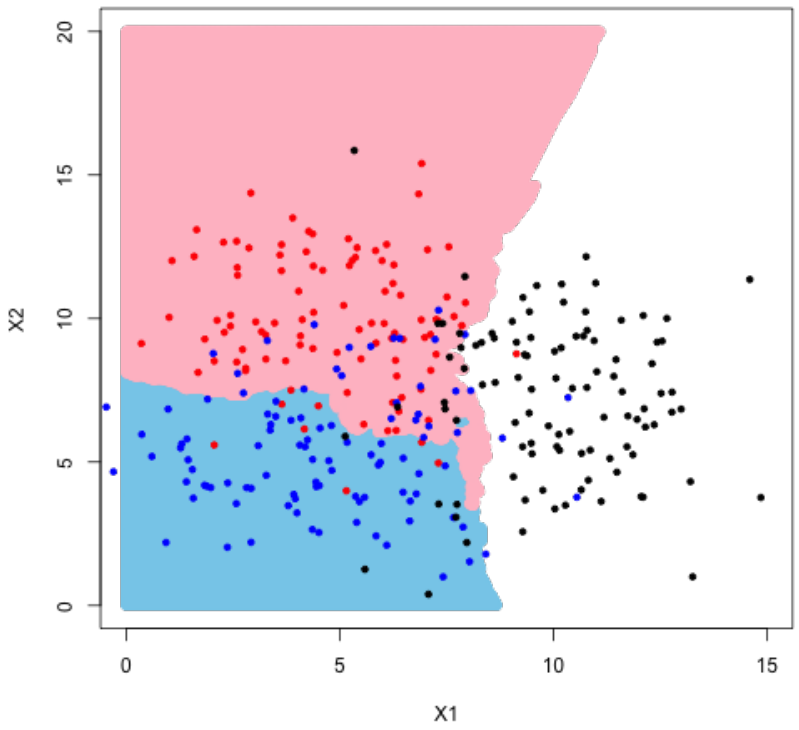

Figure 2: KNN Classifier with 3 classes

2.3 Bayes Optimal Classifier

```
f1 <- function(x1, x2, mu1, sigma1, mu2, sigma2) {
    const1 <- 1/(2 * pi * sigma1 * sigma2)
    f = exp((-(x1 - mu1)^2/sigma1^2 - (x2 - mu2)^2/sigma2^2)/2) * const1
    return(f)
}
BC <- function() {
    # apply(a,1,which.max)
    classifier <- matrix(0, nrow = nrow(mygrid), ncol = 3)
    classifier[, 1] <- f1(mygrid$X1, mygrid$X2, mu1 = 5, sigma1 = 2.5, mu2 = 5,
        sigma2 = 2)
    classifier[, 2] <- f1(mygrid$X1, mygrid$X2, mu1 = 5, sigma1 = 2, mu2 = 10,
        sigma2 = 2)
    classifier[, 3] <- f1(mygrid$X1, mygrid$X2, mu1 = 10, sigma1 = 2, mu2 = 7,
        sigma2 = 2.5)
    classifier1 <- apply(classifier, 1, which.max)

}
plotBC <- function() {
    plot(c(0, 15), c(0, 20), type = "n", xlab = "X1", ylab = "X2")
    points(mygrid$X1[classifier1 == 1], mygrid$X2[classifier1 == 1], col = "skyblue")
    points(mygrid$X1[classifier1 == 2], mygrid$X2[classifier1 == 2], col = "pink")
    points(mygrid$X1[classifier1 == 3], mygrid$X2[classifier1 == 3], col = "grey")
    points(x1[101:200], x2[101:200], col = "RED", pch = 20)
    points(x1[1:100], x2[1:100], col = "BLUE", pch = 20)
    points(x1[201:300], x2[201:300], col = "black", pch = 20)
}
```

The Figure 3 one Page 9 shows the Optimal Bayes Classifier. Here we use the same strategy as KNN. Since we have the exact density function of all three data set, we can calculate probability of every points form "mygrid" for each category. And then we can simply compare three probabilities for each point and determine which class it belongs to (belongs to class that has highest probability).

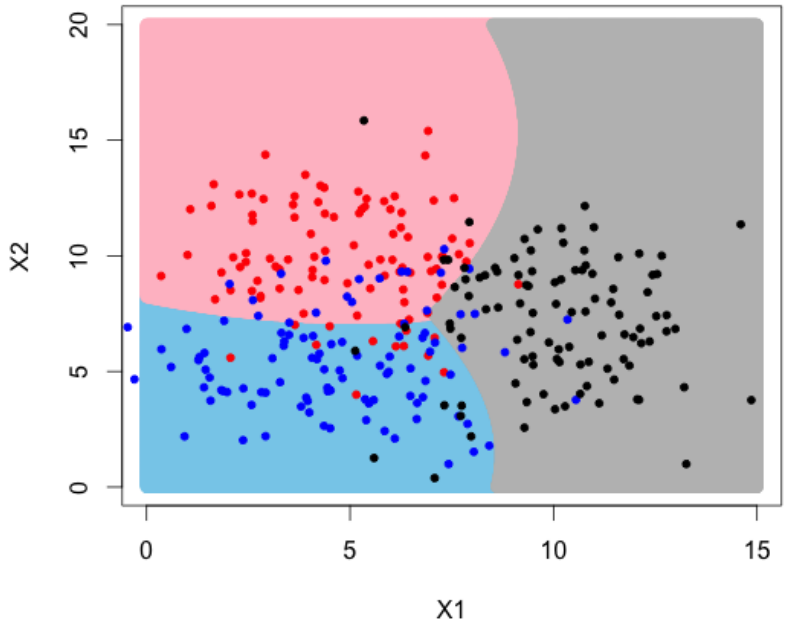

Figure 3: Bayes Optimal Classifier for 3 classes