MSiA 400 Lab Assignment 3

Shenglang: szg1224

Problem 1

Problem 1a

We just plug in the formual and get the result of the MME to be $\boldsymbol{\theta} = (\theta_1, \cdots, \theta_6)$?

$$\theta = \frac{n_i}{\sum_{i=1}^{N} n_i}$$

Problem 1b

We simply plug in the formula and the result is as follows

Log-likelihood

$$\sum_{i=1}^{N}(n_i + \alpha_i - 1)(\ln(\theta_i)) = (18 * ln(\theta_1)) + (11 * ln(\theta_2)) + (9 * ln(\theta_3)) + (25 * ln(\theta_4)) + (18 * ln(\theta_5)) + (19 * ln(\theta_6))$$

MAP

$$\theta = \frac{n_i + \alpha_i - 1}{\sum_{i=1}^{N} n_i + \alpha_i - 1}$$

### For alpha = 1 We just plug in the alpha=1, then we can get

$$\theta = \frac{n_i}{\sum_{i=1}^{N} n_i}$$

Problem 2

Problem 2a

First roll fair die: $0.5 * \frac{1}{6} = \frac{1}{12}$

Posterior First Roll

$\pi_1 = Fair$

$$P(\pi_0 = Fair|\pi_1 = Fair, x_0 = 4) = \frac{P(\pi_1 = Fair, x_0 = 4|x_0 = Fair)P(\pi_0 = Fair)}{P(\pi_1 = Fair, x_0 = 4)}$$

$$= \frac{P(\pi_1 = Fair, x_0 = 4|\pi_0 = Fair)P(\pi_0 = Fair)}{P(\pi_1 = Fair, x_0 = 4)P(\pi_0 = Fair) + P(\pi_1 = Fair, x_0 = 4|\pi_0 = Weight)P(\pi_0 = Weight)}$$

$$= \frac{P(\pi_1 = Fair|\pi_0 = Fair)P(x_0 = 4|\pi_0 = Fair)P(\pi_0 = Fair)}{P(\pi_1 = Fair|\pi_0 = Fair)P(x_0 = 4|\pi_0 = Fair)P(\pi_0 = Fair) + P(\pi_1 = Fair|\pi_0 = Weight)P(x_0 = 4|\pi_0 = Weight)P(\pi_0 = Weight)}$$

$$= \frac{0.75 * \frac{1}{6} * \frac{1}{2}}{0.75 * \frac{1}{6} * \frac{1}{2} + 0.25 * \frac{4}{3} * \frac{1}{2}}$$

$$= 0.619$$

$\pi_1 = Weight$

$$P(\pi_0 = Fair|\pi_1 = Weight, x_0 = 4) = \frac{P(\pi_1 = Weight, x_0 = 4|x_0 = Fair)P(\pi_0 = Fair)}{P(\pi_1 = Weight, x_0 = 4)}$$

$$= \frac{P(\pi_1 = Weight|\pi_0 = Fair)P(x_0 = 4|\pi_0 = Fair)P(\pi_0 = Fair)}{P(\pi_1 = Weight|\pi_0 = Fair)P(x_0 = 4|\pi_0 = Fair)P(\pi_0 = Fair) + P(\pi_1 = Weight|\pi_0 = Weight)P(x_0 = 4|\pi_0 = Weight)P(\pi_0 = Weight)}$$

$$= \frac{0.25 * \frac{1}{6} * \frac{1}{2}}{0.25 * \frac{1}{6} * \frac{1}{2} + 0.75 * \frac{4}{3} * \frac{1}{2}}$$

$$= 0.153$$

Posterity Final State

$\pi_1 = Fair$

$$P(\pi_{19} = Fair|\pi_{18} = Fair, x_{19} = 1) = \frac{P(x_{19} = 1|\pi_{19} = Fair)P(\pi_{18} = Fair|\pi_{19} = Fair)}{P(\pi_{18} = Fair, x_{19} = 1)}$$

$$= \frac{P(x_{19} = 1|\pi_{19} = Fair)P(\pi_{18} = Fair|\pi_{19} = Fair)}{P(\pi_{18} = Fair, x_{19} = 1|\pi_{19} = Fair) + P(\pi_{18} = Fair, x_{19} = 1|\pi_{19} = Weight)}$$

$$= \frac{\frac{1}{6} * 0.75}{0.75 * \frac{1}{6} + 0.25 * \frac{2}{13}}$$

$$= 0.765$$

$\pi_1 = Weight$

$$P(\pi_{19} = Fair|\pi_{18} = Weight, x_{19} = 1) = \frac{P(x_{19} = 1|\pi_{19} = Fair)P(\pi_{18} = Weight|\pi_{19} = Fair)}{P(\pi_{18} = Weight, x_{19} = 1)}$$

$$= \frac{P(x_{19} = 1|\pi_{19} = Fair)P(\pi_{18} = Weight|\pi_{19} = Fair)}{P(\pi_{18} = Weight, x_{19} = 1|\pi_{19} = Fair) + P(\pi_{18} = Weight, x_{19} = 1|\pi_{19} = Weight)}$$

$$= \frac{\frac{1}{6} * 0.25}{0.25 * \frac{1}{6} + 0.75 * \frac{2}{13}}$$

$$= 0.265$$

In Between States

$$P(\pi_x = A|\pi_{x-1} = B, x_x = y, \pi_{x+1} = C) = \frac{P(x_x = y|\pi_x = A)P(\pi_{x-1} = B|\pi_x = A)P(\pi_{x+1} = C|\pi_x = A)}{P(\pi_{x-1} = B, x_x = y, \pi_{x-1} = C)}$$

$$P(\pi_x = A|\pi_{x-1} = B, x_x = y, \pi_{x+1} = C) = \frac{P(x_x = y|\pi_x = A)P(\pi_{x-1} = B|\pi_x = A)P(\pi_{x+1} = C|\pi_x = A)}{P(\pi_{x-1} = B, x_x = y, \pi_{x+1} = C|\pi_x = A) + P(\pi_{x-1} = B, x_x = y, \pi_{x+1} = C|\pi_x = \sim A)}$$

Problem 2b

Here I will use the similar code from the lab 1.

```
firstlast = function(p0,p1,x0){
  if(p1=='W'){
    if(p0=='F'){
      num = 0.25*(1/6)
      denom = 0.25*(1/6)+0.75*(wprob[x0])
      return(num/denom)
    }
    else{
      num = 0.75*(wprob[x0])
      denom = 0.75*(wprob[x0])+0.25*(1/6)
      return(1-(num/denom))
    }

  }
  else{
    if(p0=='F'){
      num = 0.75*(1/6)
      denom = 0.75*(1/6)+0.25*(wprob[x0])
      return(num/denom)
    }
    else{
      num = 0.25*(wprob[x0])
      denom = 0.75*(wprob[x0])+0.25*(1/6)
      return(1-(num/denom))
    }
  }

}
```

```
between = function(px,pb,x,pa){

  if(pb=='W'){
    if(pa=='F'){
      if(px=='F'){
        num = (1/6)*0.25*0.75
        denom = 0.25*(1/6)*0.75+0.75*(wprob[x])*0.25
        return(num/denom)
      }
      else{
        num = (wprob[x])*0.75*0.25
        denom = 0.75*(wprob[x])*0.25+0.25*(1/6)*0.75
        return(1-(num/denom))
      }

    }
    else{
      if(px=='F'){
        num = (1/6)*0.25*0.25
        denom = 0.25*(1/6)*0.25+0.75*(wprob[x])*0.75
        return(num/denom)
      }
      else{
        num = (wprob[x])*0.75*0.75
        denom = 0.75*(wprob[x])*0.75+0.25*(1/6)*0.25
        return(1-(num/denom))

      }
    }
  }
  else{
    if(pa=='F'){
      if(px=='F'){
        num = (1/6)*0.75*0.75
        denom = 0.75*(1/6)*0.75+0.25*(wprob[x])*0.25
        return(num/denom)
      }
      else{
        num = (wprob[x])*0.25*0.25
        denom = 0.25*(wprob[x])*0.25+0.75*(1/6)*0.75
        return(1-(num/denom))

      }
    }
    else{
      if(px=='F'){
        num = (1/6)*0.75*0.25
        denom = 0.75*(1/6)*0.25+0.25*(wprob[x])*0.75
        return(num/denom)
      }
      else{
        num = (wprob[x])*0.25*0.75
        denom = 0.25*(wprob[x])*0.75+0.75*(1/6)*0.25
        return(1-(num/denom))
      }
    }
  }

}
```

```
#set.seed(123)
counts = c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
rolls = c(4, 4, 5, 2, 2, 4, 6, 6, 1, 4, 1, 1, 3, 5, 5, 2, 5, 4, 2, 1)
states = c('F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F', 'F')
wprob = c(2/13, 2/13, 1/13, 4/13, 2/13, 2/13)
for(i in 1:15000){

  for(j in 1:20){

    if(j==1){
      p = firstlast(states[1],states[2],rolls[1])
    }
    else if(j==20){
      p = firstlast(states[20],states[19],rolls[20])
    }
    else{
      p = between(states[j],states[j-1],rolls[j],states[j+1])
    }
    r = runif(1)
    #cat(p,' ',j,' ')
    if(p>r){
      states[j]='F'
      counts[j] = counts[j] + 1
    }
    else{
      states[j]='W'
    }
  }

}
```

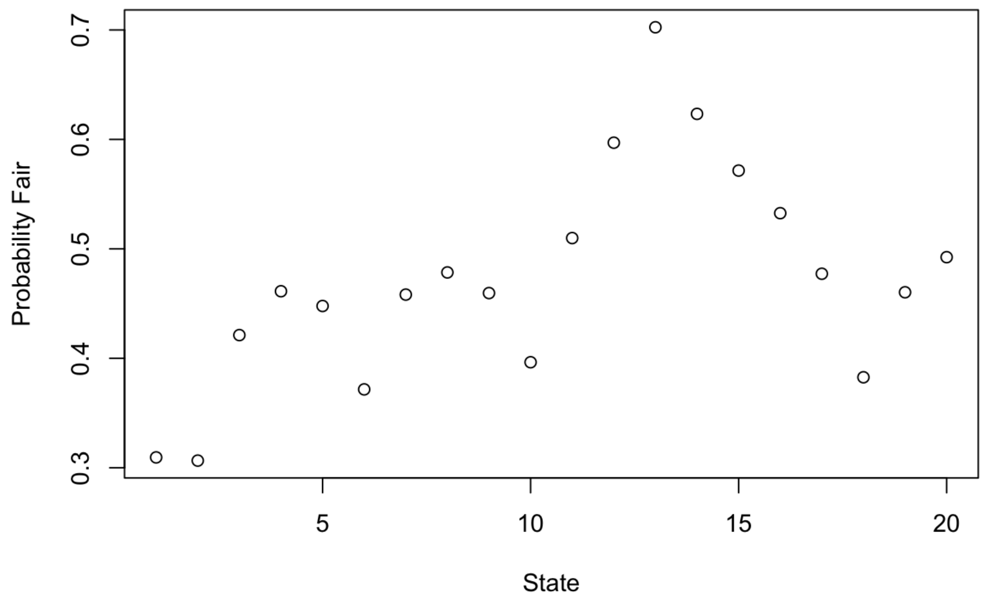
```
counts/15000
```

```
##  [1] 0.3094667 0.3065333 0.4212000 0.4612667 0.4478000 0.3716000 0.4582000
##  [8] 0.4784667 0.4596000 0.3964000 0.5098667 0.5970000 0.7024667 0.6233333
## [15] 0.5715333 0.5326000 0.4772667 0.3826667 0.4603333 0.4924000
```

```
seq(1,20)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
plot(seq(1,20),counts/15000,xlab = 'State',ylab = 'Probability Fair')
```



Problem 3

I will first use the following code to generate the training and testing set.

```
admits <- read.csv("gradAdmit.csv")
set.seed(12123)
n = nrow(admits) # number of samples
# hold out 20% for testing
sample = sample.int(n = n, size = floor(.2*n), replace = F)
train = admits[-sample,]; test = admits[sample,]
```

Problem 3a

By the following code, we can get

```
#Train set
table(train$admit)/nrow(train)
```

```
##
##        0        1
## 0.690625 0.309375
```

31.5 percent of students were admitted

```
#Test set
table(test$admit)/nrow(test)
```

```
##
##    0    1
## 0.65 0.35
```

32.5 percent of students were admitted

Problem 3b

Using the following code, we can fit a SVM model

```
library(e1071)
library(caret)
```

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

```
## Loading required package: lattice
```

```
smodel <- svm(factor(admit)~.,kernel='polynomial',cost=1,gamma=1,coef0=0,degree=5, data=train)
testpredict = predict(smodel,newdata=test)
finaltest = confusionMatrix(factor(test$admit), testpredict)
finaltest$byClass
```

```
##          Sensitivity          Specificity       Pos Pred Value
##            0.6760563            0.5555556            0.9230769
##       Neg Pred Value            Precision               Recall
##            0.1785714            0.9230769            0.6760563
##                   F1           Prevalence       Detection Rate
##            0.7804878            0.8875000            0.6000000
## Detection Prevalence    Balanced Accuracy
##            0.6500000            0.6158059
```

Problem 3c

By using the algorithm

```
library(performanceEstimation)
```

The percentage would be ~116 = (0.684375 - 0.315625) / 0.315625 percent of minority over-sampling.

```
train$admit = as.factor(train$admit)
s <- smote(admit~.,data=train,perc.over = 116, perc.under = 0)
#train <- rbind(train,s)
table(s$admit)
```

```
##
##     0     1
##     0 11583
```

```
maj <- train[which(train$admit==0),]
nTrain <- rbind(s,maj)
table(nTrain$admit)
```

```
##
##     0     1
##   221 11583
```

Problem 3d

```
smodel <- svm(admit~.,kernel='polynomial',cost=1,gamma=1,coef0=0,degree=5, data=nTrain)
testpredict = predict(smodel,newdata=test)
finaltest = confusionMatrix(factor(test$admit), testpredict)
finaltest$byClass
```

```
##          Sensitivity          Specificity       Pos Pred Value
##                   NA                 0.35                   NA
##       Neg Pred Value            Precision               Recall
##                   NA                 0.00                   NA
##                   F1           Prevalence       Detection Rate
##                   NA                 0.00                 0.00
## Detection Prevalence    Balanced Accuracy
##                 0.65                   NA
```

It’s clear to see that the precision and specificity decreased, while the recall increased for the model on the combined dataset.

Problem 4

Problem 4a

The probability can be easily caculated by the following.

```
pexp(10*pi,lower.tail = FALSE)
```

```
## [1] 0.00000000000002271101
```

Problem 4b

We can calculate it easily by this

```
integrand = function(x) {exp(-1*x)*sin(x)}
integrate(integrand, lower = 10*pi, upper = Inf)
```

```
## 0.00000000000001135156 with absolute error < 0.0000000000000071
```

Problem 4c

I think choosing a shifted exponential distribution is a good choice. This distribution allows for probabilities to fit the criteria above versus a different distribution that only changed the rates that will not result in much of a bias.

Problem 4d

We can calculate with the following code

```
set.seed(12345)
tot = 0
for (i in 1:1000000){
  x_star = rexp(1, rate = 1) + 10*pi
  if (x_star>=10*pi){
    g = sin(x_star)
  }else{
    g = 0
  }
  r = pexp(x_star, lower.tail = FALSE)/pexp(x_star-10*pi, lower.tail = FALSE, rate = 1)
  prod = r*g
  tot = tot + prod


}
tot/1000000
```

```
## [1] 0.00000000000001135953
```