MSiA400_Lab3

Question 1

a

See PDF called question 1

b

See PDF called question 1

Question 2

a

```
x_i <- c(4, 4, 5, 2, 2, 6)
p_fair <- rep(1/6, 6)
p_unfair <- c(2/13, 2/13, 1/13, 4/13, 2/13, 2/13)
p_fair_all <- 0.5 * p_fair[4]
p_unfair_all <- 0.5 * p_unfair[4]
for (i in seq(2, 6)){
  curr_xi <- x_i[i]
  p_fair_all <- p_fair_all * 0.75 * p_fair[curr_xi]
  p_unfair_all <- p_unfair_all * 0.75 * p_unfair[curr_xi]
}
p_fair_all
```

```
## [1] 2.543132e-06
```

```
p_unfair_all
```

```
## [1] 6.292977e-06
```

All unfair die is more likely to get the given observation.

b

```
s <- c("f")
p_fair_out <- c(0.5 * 1/6)
p_unfair_out <- c(0.5 * p_unfair[4])


for (i in seq(2,6)){
    curr_xi <- x_i[i]
# p(fair | x_i = curr_xi)
    curr <- c("f", "f", "u", "u")
    curr_p_fair_prev_fair <- 0.75 * p_fair[curr_xi]

    curr_p_unfair_prev_fair <- 0.25 * p_unfair[curr_xi]

    curr_p_fair_prev_unfair <- 0.25 * p_fair[curr_xi]

    curr_p_unfair_prev_unfair <- 0.75 * p_unfair[curr_xi]

    p_fair_out <- append(p_fair_out, curr_p_fair_prev_fair)
    p_fair_out <- append(p_fair_out, curr_p_unfair_prev_fair)

    p_unfair_out <- append(p_unfair_out, curr_p_fair_prev_unfair)
    p_unfair_out <- append(p_unfair_out, curr_p_unfair_prev_unfair)


}
p_fair_out
```

```
##  [1] 0.08333333 0.12500000 0.07692308 0.12500000 0.03846154 0.12500000
##  [7] 0.03846154 0.12500000 0.03846154 0.12500000 0.03846154
```

```
p_unfair_out
```

```
##  [1] 0.15384615 0.04166667 0.23076923 0.04166667 0.11538462 0.04166667
##  [7] 0.11538462 0.04166667 0.11538462 0.04166667 0.11538462
```

Question 3

a

```
dat <- read_csv("gradAdmit.csv")
```

```
## Rows: 400 Columns: 4

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (4): admit, gre, gpa, rank

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
set.seed(123)
n <- nrow(dat)
train_i <- sample.int(n = n, size = floor(n * 0.8), replace = FALSE)
train <- dat[train_i, ]
test <- dat[-train_i, ]
```

```
table(train$admit)
```

```
##
##   0   1
## 216 104
```

```
table(test$admit)
```

```
##
##  0  1
## 57 23
```

In the train dataset, 32.5% were admitted. In the test dataset, 28.75% were admitted.

b

```
best_m <- svm(factor(admit) ~ ., data = train,
                            kernel = "polynomial",
                            degree = 4,
                            gamma = 0.01,
                            coef0 = 10,
                            cost = 10)

pred_train <- predict(best_m, newdata = train, type = 'response')
pred_test <- predict(best_m, newdata = test, type = 'response')

confusionMatrix(pred_train, factor(train$admit), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 209  84
##          1   7  20
##
##                Accuracy : 0.7156
##                  95% CI : (0.6628, 0.7644)
##     No Information Rate : 0.675
##     P-Value [Acc > NIR] : 0.06664
##
##                   Kappa : 0.1979
##
##  Mcnemar’s Test P-Value : 1.626e-15
```

```
##
##             Sensitivity : 0.19231
##             Specificity : 0.96759
##          Pos Pred Value : 0.74074
##          Neg Pred Value : 0.71331
##              Prevalence : 0.32500
##          Detection Rate : 0.06250
##    Detection Prevalence : 0.08438
##       Balanced Accuracy : 0.57995
##
##        'Positive' Class : 1
##
```

```
confusionMatrix(pred_test, factor(test$admit), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 55 19
##          1  2  4
##
##                Accuracy : 0.7375
##                  95% CI : (0.6271, 0.8296)
##     No Information Rate : 0.7125
##     P-Value [Acc > NIR] : 0.3613344
##
##                   Kappa : 0.1781
##
##  Mcnemar's Test P-Value : 0.0004803
##
##             Sensitivity : 0.1739
##             Specificity : 0.9649
##          Pos Pred Value : 0.6667
##          Neg Pred Value : 0.7432
##              Prevalence : 0.2875
##          Detection Rate : 0.0500
##    Detection Prevalence : 0.0750
##       Balanced Accuracy : 0.5694
##
##        'Positive' Class : 1
##
```

```
# precision = TP / (TP + FP)
p_test <- 4 / (4 + 2)

# recall = TP / (TP + FN)
r_test <- 4 / (4 + 19)

# specificity = TN / (TN + FP)
s_test <- 55 / (55 + 2)

p_test
```

```
## [1] 0.6666667
```

```
r_test
```

```
## [1] 0.173913
```

```
s_test
```

```
## [1] 0.9649123
```

c

There are 320 samples in the train set. There are 104 in admit, 216 in reject. Thus, we need 216 - 104 = 112 to obtain the most balanced dataset

```
perc <- 100 * (112-104) / 104
train$admit <- factor(train$admit)
train_df <- as.data.frame(train)
new_train <- SMOTE(form = admit~., train_df, perc.over = perc, perc.under = 0)
# table(new_train$admit)
train_full <- rbind(train_df, new_train)
table(train_full$admit)
```

```
##
##   0   1
## 216 216
```

d

```
best_m <- svm(factor(admit) ~ ., data = train_full,
                            kernel = "polynomial",
                            degree = 4,
                            gamma = 0.01,
                            coef0 = 10,
                            cost = 10)
pred_test <- predict(best_m, newdata = test, type = 'response')
confusionMatrix(pred_test, factor(test$admit), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 40 11
##          1 17 12
##
##                Accuracy : 0.65
##                  95% CI : (0.5352, 0.7533)
##     No Information Rate : 0.7125
##     P-Value [Acc > NIR] : 0.9109
```

```
##
##                   Kappa : 0.2074
##
##  Mcnemar’s Test P-Value : 0.3447
##
##             Sensitivity : 0.5217
##             Specificity : 0.7018
##          Pos Pred Value : 0.4138
##          Neg Pred Value : 0.7843
##              Prevalence : 0.2875
##          Detection Rate : 0.1500
##    Detection Prevalence : 0.3625
##       Balanced Accuracy : 0.6117
##
##        ’Positive’ Class : 1
##
```

```
# precision = TP / (TP + FP)
p_test_smote <- 12 / (12 + 19)

# recall = TP / (TP + FN)
r_test_smote <- 12 / (12 + 11)

# specificity = TN / (TN + FP)
s_test_smote <- 38 / (38 + 19)

p_test_smote
```

```
## [1] 0.3870968
```

```
r_test_smote
```

```
## [1] 0.5217391
```

```
s_test_smote
```

```
## [1] 0.6666667
```

The precision and specificity are decreased, but the recall is increased compared to the unbalanced dataset.

Problem 4

a

```
set.seed(123)
lambda <- 1
n <- ceiling((1 / lambda^2) / (10^(-6) * 0.01))
curr_i <- numeric(3)
real_i <- 1/(1+lambda^2)
```

```
curr_n <- n
x <- runif(curr_n, 0, 1)
y <- -log(x) / lambda
curr_i <- 1/curr_n * sum(sin(y) / lambda)
curr_i
```

```
## [1] 0.5000296
```

```
crit <- exp(-10 * pi)
sum(y > crit) / length(y)
```

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

b

```
set.seed(123)
lambda <- 1
n <- ceiling((1 / lambda^2) / (10^(-6) * 0.01))
curr_i <- numeric(3)
real_i <- 1/(1+lambda^2)

curr_n <- n
x <- runif(curr_n, 0, 1)
y <- -log(x) / lambda
est <- 1/curr_n * sum(sin(y) / lambda)
est
```

```
## [1] 0.5000296
```

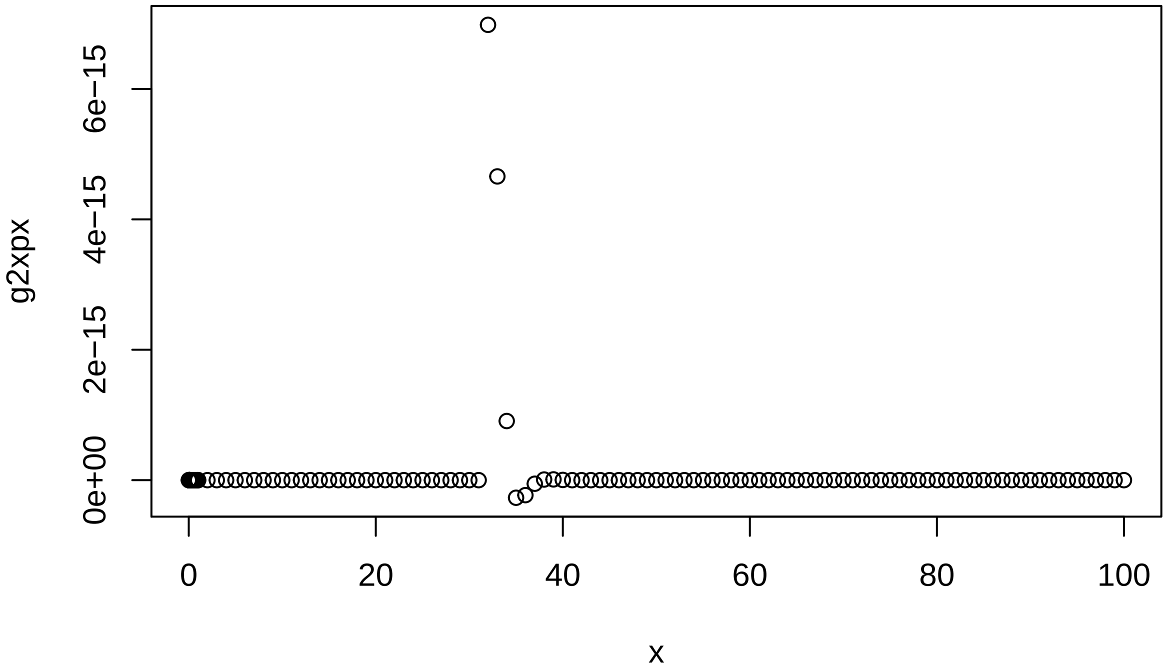
```
real <- 1 / (1 + 1^2)
real
```

```
## [1] 0.5
```

c

We can plot $g^2(x)p(x)$ to find the best $p^*(x)$

```
x <- c(seq(0.01, 0.1, 0.01), seq(0.1, 1, 0.1), seq(1, 100))
g2xpx <- exp(-x) * ifelse(x >= 10*pi, sin(x), 0)
plot(x, g2xpx)
```



Here, we can see that when $g^2(x)p(x)$ is small, it is equal to 0. Note the asymptotic behavior of the curve. Thus, we must ensure that when $x < 10\pi$, $p^*(x) < p(x)$. By same logic, when $x > 10\pi$, $p^*(x) > p(x)$. However, note that if $x < 10\pi$, $g(x) = 0$. Thus, we realistically only need to satisfy $x > 10\pi$, $p^*(x) > p(x)$. Thus, the simplest measure is to shift the exponential distribution $p(x)$ 10π to the right. Thus, $p^*(x) = e^{-x+10\pi}$

d

```
n <- 10^6
x <- runif(n, 0, 1)
y <- -log(x) + 10*pi
est <- 1/n * sum(sin(y) * (exp(-y) / exp(-y + 10*pi)))
est
```

```
## [1] 1.137324e-14
```