HW4a

CS208

Lipika Ramaswamy

https://github.com/lipikaramaswamy/cs208_lr/tree/master/homework/HW4a (https://github.com/lipikaramaswamy/cs208_lr/tree/master/homework/HW4a)

Collaborators: Bhaven Patel, Karina Huang and Anthony Rentsch

R Setup

In [8]:
```r
## Setup
# rm(list=ls())
library(ggplot2)
options(repr.matrix.max.cols=50, repr.matrix.max.rows=100)

# Random draw from Laplace distribution
#
# mu numeric, center of the distribution
# b numeric, spread
# size integer, number of draws
#
# return Random draws from Laplace distribution
# example:
#
# rlap(size=1000)

rlap = function(mu=0, b=1, size=1) {
    p <- runif(size) - 0.5
    draws <- mu - b * sgn(p) * log(1 - 2 * abs(p))
    return(draws)
}

# Sign function
#
# Function to determine what the sign of the passed values should be.
#
# x numeric, value or vector or values
# return The sign of passed values
# example:
#
# sgn(rnorm(10))

sgn <- function(x) {
    return(ifelse(x < 0, -1, 1))
}
```

Problem 1: Learning Conjunctions in the SQ Model

(a)

Describe and implement centralized and local DP versions of the above SQ algorithm, dividing the privacy budget equally among each of the d estimates $\hat{p}_j$. Keep the threshold t as a free parameter that you can choose.

Centralized DP version of SQ algorithm

The centralized DP version of the SQ algorithm involves the following:

- Use the dataset D to obtain a true value for p_j where $j = 1, \ldots, d$.
  - The conjunction of $x[j] = 0$ and $y = 1$ is calculated from the data and stored in a conjunction matrix.
  - Using this conjunction matrix, the true values of p_j are calculated for attribute, j.
  - The true set S is determined from these true probabilities, p_j, which will be 0 for all $j \in S$.
- Add Laplace noise with scale $\frac{1/n}{\epsilon/d} = \frac{d}{n\epsilon}$ to each p_j and yield $\hat{p}_j$.
  - The global sensitivity of the proportion of n rows where the conjunction of the jth bit is 0 and the outcome is 1, is 1/n.
  - There will be d releases, one for each attribute, so we divide the epsilon equally.
- Given a threshold t, output the set of attributes, $\hat{S}$, where the estimate, $\hat{p}_j$, is less than the threshold.

In [9]:
```
# SQ Algorithm implemented using centralized differential privacy
#
# Parameters:
# x is the dataframe with only predictor columns
# y is the response/predicted variable column of the dataframe
# epsilon is the privacy budget
# t is the threshold for choosing the
#
# Returns:
# column indices for an estimate of the set that predicts y
# column names for an estimate of the set that predicts y
# column indices for the true set that perfectly predicts y
# column names for the true set that perfectly predicts y


dpSQAlgCentral <- function(x,
                           y,
                           epsilon,
                           t){

    # get dimensions of data
    nrows = dim(x)[1]
    d = dim(x)[2]

    # assign epsilon per column for central release
    epsilon.split = epsilon/d

    # make an empty matrix to store conjunction
    conjunction.matrix <- matrix(0, nrow = nrows, ncol = d)

    # get conjunction matrix
    for(i in 1:nrows){

        # check if the y attribute is 1
        if(y[i] == 1){

            # if attribute is 0, then store TRUE
            conjunction.matrix[i,] <- (x[i,] == 0)
        }
    }

    # get the mean of columns to get true vals
    true.p.vec <- colMeans(conjunction.matrix)

    # get set S, where true probabilities are below threshold
    S <- which(true.p.vec == 0)
    S.colnames <- colnames(x)[S]

    # draw random laplace noise of size d = number of attributes
    noise <- rlap(mu=0, b = d/(nrows * epsilon), size=d)

    # add noise to means
    noisy.p.vec <- true.p.vec + noise

    # get set S hat, where noisy probabilities are below threshold
    S.hat <- which(noisy.p.vec < t)
```

```
    S.hat.colnames <- colnames(x)[S.hat]

    return(list(dp.column.indices = S.hat, dp.column.names = S.hat.colna
mes,
                true.column.indices = S,   true.column.names = S.colname
s))

    }
```

Local DP version of SQ algorithm

The localized DP version of the SQ algorithm involves the following:

- Use the dataset D to obtain a true value for p_j where $j = 1, \ldots, d$.
  - The conjunction of $x[j] = 0$ and $y = 1$ is calculated from the data and stored in a conjunction matrix.
  - Using this conjunction matrix, the true values of p_j are calculated for attribute, j.
  - The true set S is determined from these true probabilities, p_j, which will be 0 for all $j \in S$.
- Randomized response is performed on each element, c_i of the conjunction matrix per the formula: $$ Q(c_i) = \left{

  ```
  \begin{array}{ll}
      c_i & w.p. \frac{e^{\epsilon/d}}{e^{\epsilon/d}+1} \\
      1-c_i & w.p. \frac{1}{e^{\epsilon/d}+1}
  \end{array}
  ```

  \right. $$
  - Note that the epsilon used for each randomized response is the total privacy budget divided by the number of attributes, as we get a mean for each attribute (described in detail later).
- The conjunction matrix is columnwise summed to obtain $\hat{p}_j$ and then a correction is applied to scale it appropriately to ensure that the expectation of the estimated sum is equal to the true sum, as shown below:

$$
\begin{aligned}
E[\hat{p_j}] &= E\Big[\frac{1}{n}\sum_{i=1}^{n}\hat{x}_i[j]\Big] \\
&= \frac{1}{n}\sum_{i=1}^{n}E[\hat{x}_i[j]] \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(x_i[j]\cdot\frac{e^\epsilon}{1+e^\epsilon} + (1-x_i[j])\cdot\frac{1}{1+e^\epsilon}\right) \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(x_i[j]\cdot\frac{e^\epsilon-1}{e^\epsilon+1} + \frac{1}{1+e^\epsilon}\right) \\
&= \frac{1}{1+e^\epsilon} + \frac{1}{n}\sum_{i=1}^{n}\left(x_i[j]\cdot\frac{e^\epsilon-1}{e^\epsilon+1}\right)
\end{aligned}
$$

We want this expectation to be $p_j = \frac{1}{n}\sum_{i=1}^{n} x_i[j]$.

Setting the two sides equal and manipulating the terms, we find that if we wanted to apply the correction to the column sums obtained after the randomized response is completed, we would use the following multiplicative and additive factors to the sum:

$$
\begin{aligned}
\text{multiplicativeFactor} &= \frac{e^\epsilon+1}{e^\epsilon-1} \\
\text{additiveFactor} &= -\frac{n}{e^\epsilon-1}
\end{aligned}
$$

.

- After applying the correction to the sums, we divide by the length of the data (n) to get the required probability estimates, $\hat{p_j}$.
- Given a threshold t, output the set of attributes, $\hat{S}$ where the estimate, $\hat{p}_j$, is less than the threshold.

In [10]:

```
# local release function for a vector
localReleaseVector <- function(x.vec, values=c(-1,1), epsilon){
    draw.vec <- runif(n=length(x.vec), min=0, max=1)
    cutoff <- 1/(1+exp(epsilon))
    to.return <- x.vec
    for(i in 1:length(draw.vec)){
        if(draw.vec[i]<cutoff){
            to.return[i] <- values[!values%in%x.vec[i]]
        }
    }
    return(to.return)
}
```

In [11]:
```
# SQ Algorithm implemented using local differential privacy
#
# Parameters:
# x is the dataframe with only predictor columns
# y is the response/predicted variable column of the dataframe
# epsilon is the privacy budget
# t is the threshold for choosing the
#
# Returns:
# column indices for an estimate of the set that predicts y
# column names for an estimate of the set that predicts y
# column indices for the true set that perfectly predicts y
# column names for the true set that perfectly predicts y


dpSQAlgLocal <- function(x, y,
                         epsilon,
                         t){

    # get dimensions of data
    nrows = dim(x)[1]
    d = dim(x)[2]

    # assign epsilon per local release
    epsilon.split = epsilon/d

    # make an empty matrix to store conjunction
    conjunction.matrix <- matrix(0, nrow = nrows, ncol = d)

    # get conjunction matrix
    for(i in 1:nrows){

        # now check if the y attribute is 1
        if(y[i] == 1){

            # if attribute is 0, then store TRUE
            conjunction.matrix[i,] <- (x[i,] == 0)
        }
    }

    ### TRUE VALUES FOR p_j
    true.means <- colMeans(conjunction.matrix)

    # get set S, where true probabilities are below threshold
    S <- which(true.means == 0)
    S.colnames <- colnames(x)[S]


    ### now do randomized response on each value in the conjunction matr
ix

    # make a matrix to store LDP values
    conjunction.dp = matrix(0, nrow = nrow(conjunction.matrix), ncol = n
col(conjunction.matrix))

    # loop through rows in the conjunction matrix
```

```
    for(row in 1:nrow(conjunction.dp)){

        # do a local DP release on each row of the conjunction matrix us
ing local release function for vectors
        conjunction.dp[row,] = localReleaseVector(x = conjunction.matrix
[row,],
                                                  values = c(0,1),
                                                  epsilon = epsilon.spli
t)

        }

    # get means of RR bits for each attribute in conjunction matrix
    column.sums <- colSums(conjunction.dp)

    # apply correction to the sums
    mul <- (exp(epsilon.split) + 1)/(exp(epsilon.split) - 1)
    add <- (- nrows )/ (exp(epsilon.split) - 1)
    corrected.col.sums <- column.sums * mul + add

    # get probabilities
    dp.col.probs <- corrected.col.sums/nrows

    # get set S hat, where noisy probabilities are below threshold
    S.hat <- which(dp.col.probs < t)
    S.hat.colnames <- colnames(x)[S.hat]

    return(list(dp.column.indices = S.hat, dp.column.names = S.hat.colna
mes,
                true.column.indices = S,   true.column.names = S.colname
s))

}
```

In [ ]:

```
### TEST WITH DUMMY DATA

# load test data
mydata <- read.csv('../../data/hw4testdata.csv')

# Centralized DP DEMO
res = dpSQAlgCentral(x = mydata[,1:10], y = mydata[,11], epsilon = 1, t
= 1e-2)
print('Centralized DP method demo results:')
print(res)

# Localized DP DEMO
res2 = dpSQAlgLocal(x = mydata[,1:10], y = mydata[,11], epsilon = 1, t =
1e-2)
print('Localized DP method demo results:')
print(res2)
```

(b)

For a threshold, t, dataset of size n, privacy loss parameter ϵ and set of features, S,

$$\begin{aligned} Pr[\hat{S} \not\supset S] &= Pr[\exists j \in Ss.t. j \notin \hat{S}] \\ &\leq \sum_{j \in S} Pr[j \notin \hat{S}] && \text{(by union bound)} \\ &= \sum_{j \in S} Pr[p_j > t] \\ &= \sum_{j \in S} Pr[\sum_{i=1}^{n} \hat{x_i}[j] > nt] \end{aligned}$$

In the **centralized model**,

$$\begin{aligned} Pr[\hat{S} \not\supset S] &\leq \sum_{j \in S} Pr[p_j > t] \\ &= \sum_{j \in S} Pr[p_j + Lap\big(\frac{1/n}{\epsilon/d}\big) > t] \\ &= \sum_{j \in S} Pr[Lap\big(\frac{1/n}{\epsilon/d}\big) > t] && \text{(as } p_j = 0 \ \forall j \in S\text{)} \\ &= |S| \cdot \int_t^{\infty} \frac{exp\big(\frac{-|y|}{d/n\epsilon}\big)}{2d/n\epsilon} \\ &= |S| \frac{n\epsilon}{2d} \cdot \int_t^{\infty} exp\big(\frac{-yn\epsilon}{d}\big) \\ &= |S| \frac{n\epsilon}{2d} \cdot \frac{d}{n\epsilon} \cdot \Big[- exp\big(\frac{-yn\epsilon}{d}\big)\Big]_t^{\infty} \\ &= |S| \frac{1}{2} \cdot exp\big(\frac{-tn\epsilon}{d}\big) \end{aligned}$$

Substituting the above into $Pr[\hat{S} \not\supset S] \leq 0.1$ and $|S| \leq d$,

$$\begin{aligned} Pr[\hat{S} \not\supset S] &\leq 0.1 \\ |S| \frac{1}{2} \cdot exp\big(\frac{-tn\epsilon}{d}\big) &\leq 0.1 \\ exp\big(\frac{-tn\epsilon}{d}\big) &\leq \frac{0.2}{|S|} \\ \frac{-tn\epsilon}{d} &\geq log\left(\frac{0.2}{|S|}\right) && \text{(for log of a value between 0 and 1)} \\ -t &\geq \frac{d}{n\epsilon} log\left(\frac{0.2}{|S|}\right) \\ t &\leq \frac{-d}{n\epsilon} log\left(\frac{0.2}{|S|}\right) \\ t &\leq \frac{-d}{n\epsilon} log\left(\frac{0.2}{d}\right) && \text{(as } |S| \leq d\text{)} \end{aligned}$$

Thus, we have an upper bound for t.

In the **local model**,

$$Pr[\hat{S} \not\supset S] \leq \sum_{j \in S} Pr[p_j > t]$$

$$= \sum_{j \in S} Pr\Big[\sum_{i=1}^{n} \hat{x_i}[j] > nt \Big]$$

As $p_j = 0 \; \forall j \in S$, we can treat each $\hat{x_i}$ as drawn from a Bernoulli distribution with a probability of success (i.e. 1) being the probability of flipping in the randomized response mechanism. Thus, $\hat{x_i}[j] \sim Bernoulli\big(\frac{1}{1+e^\epsilon}\big)$. The sum of all these random variables with follow a Binomial distribution, i.e. $\sum_{i=1}^{n} \hat{x_i}[j] \sim Bin(n, \frac{1}{1+e^\epsilon})$. We can use a normal approximation (Z) to the binomial, with mean, $\mu = \frac{n}{1+e^\epsilon}$ and variance, $\sigma^2 = \frac{n \cdot e^\epsilon}{(1+e^\epsilon)^2}$.

Thus, we have:

$$Pr[\hat{S} \not\supset S] \leq \sum_{j \in S} Pr\Big[\sum_{i=1}^{n} \hat{x_i}[j] > nt \Big]$$

$$= \sum_{j \in S} Pr\left[\frac{\big(\sum_{i=1}^{n} \hat{x_i}[j] \big) - \frac{n}{1+e^\epsilon}}{\frac{n \cdot e^\epsilon}{(1+e^\epsilon)^2}} > \frac{nt - \frac{n}{1+e^\epsilon}}{\frac{n \cdot e^\epsilon}{(1+e^\epsilon)^2}} \right]$$

$$= |S| \cdot Pr\left[\frac{\big(\sum_{i=1}^{n} \hat{x_i}[j] \big) - \frac{n}{1+e^\epsilon}}{\frac{n \cdot e^\epsilon}{(1+e^\epsilon)^2}} > \frac{nt - \frac{n}{1+e^\epsilon}}{\frac{n \cdot e^\epsilon}{(1+e^\epsilon)^2}} \right]$$

$$\leq d \cdot Pr\left[\frac{\big(\sum_{i=1}^{n} \hat{x_i}[j] \big) - \frac{n}{1+e^\epsilon}}{\sqrt{\frac{n \cdot e^\epsilon}{(1+e^\epsilon)^2}}} > \frac{nt - \frac{n}{1+e^\epsilon}}{\sqrt{\frac{n \cdot e^\epsilon}{(1+e^\epsilon)^2}}} \right]$$

$$= d \cdot Pr\left[Z > \frac{nt - \frac{n}{1+e^\epsilon}}{\sqrt{\frac{n \cdot e^\epsilon}{(1+e^\epsilon)^2}}} \right]$$

$$= d \cdot \Phi\left(\frac{nt - \frac{n}{1+e^\epsilon}}{\sqrt{\frac{n \cdot e^\epsilon}{(1+e^\epsilon)^2}}} \right)$$

Substituting the above into $Pr[\hat{S} \not\supset S] \leq 0.1$,

$$Pr[\hat{S} \not\supset S] \leq 0.1$$

$$d \cdot \Phi\left(\frac{nt - \frac{n}{1+e^\epsilon}}{\sqrt{\frac{n \cdot e^\epsilon}{(1+e^\epsilon)^2}}}\right) \leq 0.1$$

$$\Phi\left(\sqrt{\frac{n}{e^\epsilon}} \cdot ((1+e^\epsilon)t - 1)\right) \leq \frac{0.1}{d}$$

$$t \leq \frac{1}{(1+e^\epsilon)}\left(\sqrt{\frac{e^\epsilon}{n}} \cdot \Phi^{-1}\left(\frac{0.1}{d}\right) + 1\right) \qquad \text{(As inverse CDF is monotonicall}$$

Again, we have an upper bound for t.

(c)

In this problem it makes sense to consider all the "not" columns, i.e. not sex, not married, etc. that represent the other label in the demographic characteristics. The dataset has been transformed to include these columns. It perhaps makes sense to change the algorithm itself to take into account all "not" columns for any dataset that the algorithm might be used on, but for the sake of implementing this in time, I chose this approach to ensure reasonable answers for the CA PUMS data.

NOTE:

1. In part (b), analytical upper bounds depending on ϵ, d and n, were calculated for t. Given that these can be thought of "worst case" thresholds, the implementations below use a threshold of $0.1 \times t$.
2. False positive and false negative rates have been calculated on the samples in each bootstrap as opposed to the remaining dataset (test set). This was done to save time, even though it's likely that performance of these metrics will be different for different on the test set.

In [24]:
```
# read in PUMS data
PUMS <- read.csv('../../data/CaPUMS5Full.csv')

# given some exploratory work I did on the dataset that reveals that the
male sex (coded as 0)
# is a predictor of 'targetted'. I will be including all 'not' variables
in the dataset

for(col in colnames(PUMS)[1:10]){
    PUMS[,paste('not_',col)] <- 1 - PUMS[,col]
}

colnames(PUMS)
```

'sex' 'married' 'black' 'asian' 'collegedegree' 'employed' 'militaryservice' 'uscitizen' 'disability' 'englishability' 'blackfemale' 'targetted' 'not_ sex' 'not_ married' 'not_ black' 'not_ asian' 'not_ collegedegree' 'not_ employed' 'not_ militaryservice' 'not_ uscitizen' 'not_ disability' 'not_ englishability'

In [29]:
```
PUMS[1:10,]
```

sex	married	black	asian	collegedegree	employed	militaryservice	uscitizen	disability	english
1	1	0	0	0	1	0	1	0	
1	1	0	0	0	0	0	1	0	
0	1	0	0	0	0	1	1	0	
0	0	0	0	0	0	0	1	1	
0	1	0	0	0	0	0	1	0	
1	1	0	0	0	0	0	1	0	
0	1	0	0	0	1	0	1	0	
1	1	0	0	0	0	0	1	0	
0	1	0	0	0	1	0	1	0	
1	1	0	0	0	1	0	1	0	

In [31]:
```
## Function to analytically calculate a value for the threshold
## for the centralized model based on calculations above
analyticaltCentralized <- function(d, n, epsilon){
    return((- d * log (0.2/d))/(n * epsilon))
}

## Function to analytically calculate a value for the threshold
## for the local model based on calculations above
analyticaltLocal <- function(d, n, epsilon){
    exp.e = exp(epsilon)
    return((1/(1 + exp.e)) * ( ((sqrt(exp.e)/sqrt(n)) * qnorm(0.1/d)) +
1))
}
```

In [34]:
```
## CENTRALIZED DP MECHANISM FOR SQ ALGORITHM
res.central = dpSQAlgCentral(x = PUMS[,c(1:10, 13:22)], y = PUMS[, 12],
 epsilon = 1,
                             t = (0.1) * analyticaltCentralized(d = 10,n
= nrow(PUMS), epsilon = 1))
print('Results of Centralized DP algorithm')
print(res.central)

## LOCAL DP MECHANISM FOR SQ ALGORITHM
reslocal = dpSQAlgLocal(x = PUMS[,c(1:10, 13:22)], y = PUMS[, 12], epsil
on = 1,
                        t = (0.1) * analyticaltLocal(d = 10,n = nrow(PUM
S), epsilon = 1))
print('Results of Localized DP algorithm')
print(reslocal)
```

```
[1] "Results of Centralized DP algorithm"
$dp.column.indices
[1]  6  8 10

$dp.column.names
[1] "employed"       "uscitizen"      "englishability"

$true.column.indices
[1]  6  8 10 11

$true.column.names
[1] "employed"       "uscitizen"      "englishability" "not_ sex"

[1] "Results of Localized DP algorithm"
$dp.column.indices
[1]  6  8 10 11 14 19

$dp.column.names
[1] "employed"        "uscitizen"       "englishability"  "not_ sex"
[5] "not_ asian"      "not_ disability"

$true.column.indices
[1]  6  8 10 11

$true.column.names
[1] "employed"       "uscitizen"      "englishability" "not_ sex"
```

Neither the centralized not the local mechanism suceed in reconstructing the correct variables in this data. In the centralized version of the SQ algorithm, the column `not_sex` (male) is not being released as a predictor, even though it is a true predictor. In the localized version, some additional columns are being included in the released predictors (`not_asian` , `not_disability`) in addition to the true columns that are predictors of the `targeted` variable.

There is randomness with respect to the noise added for both models. This is simply the outcome of one of the runs of these algorithms. It could well be different for the next run.

In [36]:

```
# utility function to get predictions from the algorithm
getPreds <- function(model.results, df){
    # subset df to only the columns selected by the algorithm
    if(length(model.results$dp.column.names) > 1){
        dp.df = df[,model.results$dp.column.names]
        # calculate row sums in this selection
        y.pred.rowsums = rowSums(dp.df)
        y.pred = (y.pred.rowsums == length(model.results$dp.column.names
))
        return(as.integer(as.logical(y.pred)))
    }
    if(length(model.results$dp.column.names) == 1){
        dp.col = df[,model.results$dp.column.names]
        y.pred = (dp.col == 1)
        return(as.integer(as.logical(y.pred)))
    }
}
```

In [69]:
```
list.of.n = c(100, 500, 1000, 2500, 5000, 7500,  10000, 25000, 50000, 75
000, 100000, 250000, 500000)

n.boot = 10

# make empty matrices to store results
FPR.C = FPR.L = FNR.C = FNR.L = matrix(0.0, nrow = n.boot, ncol = length
(list.of.n))

counter = 1
# for each value of n i.e. size of sample to try
for (size in list.of.n){

    writeLines(paste("Sample size: ", size))

    # for each iteration aka bootstrap
    for(boot in 1:n.boot){

        # sample data
        sample.index = sample.int(n = nrow(PUMS), size = size, replace =
TRUE)
        PUMS.sample = PUMS[sample.index,]
        sample.y.true = PUMS.sample[,12]

        #### CENTRALIZED
        optimal.t.centralized = analyticaltCentralized( d = 10,
                                            n = size,
                                            epsilon = 1)

        central.results = dpSQAlgCentral(x = PUMS.sample[,c(1:10, 13:22
)], y = sample.y.true,
                                         epsilon = 1,
                                         t = 0.1 * optimal.t.centralized
)

        while(length(central.results$dp.column.indices) == 0){
            central.results = dpSQAlgCentral(x = PUMS.sample[,c(1:10, 13
:22)], y = sample.y.true,
                                         epsilon = 1,
                                         t = 0.1 * optimal.t.centralized
)
        }

        # calculate predictions based on results of model
        ### NOTE: using the sample instead of the remaining dataset for
 some computational savings
        y.pred = getPreds(central.results, PUMS.sample)


        # FALSE NEGATIVE RATE
        ## 1 - true positive rate

        # get indices with true 1s - (labelled positives)
        true.positive.indices = which(sample.y.true == 1)
        # get true positive rate
        tpr = (sum(y.pred[true.positive.indices] == sample.y.true[true.p
```

```
ositive.indices])
              )/length(true.positive.indices)
        # store FNR
        FNR.C[boot, counter] = 1 - tpr

        # FALSE POSITIVE RATE
        ## 1 - TrueNegativeRate

        # get indices with true 0s - (labelled negatives)
        true.negative.indices = which(sample.y.true == 0)
        # get true negative rate
        tnr = (sum(y.pred[true.negative.indices] == sample.y.true[true.n
egative.indices])
              )/(length(true.negative.indices))
        # store FPR
        FPR.C[boot, counter] = 1 - tnr


        ####################
        #### LOCALIZED

        optimal.t.local = analyticaltLocal( d = 10,
                                            n = size,
                                            epsilon = 1)

        local.results = dpSQAlgLocal(x = PUMS.sample[,c(1:10, 13:22)], y
= sample.y.true,
                                     epsilon = 1,
                                     t = 0.1*optimal.t.local)
        while(length(local.results$dp.column.indices) == 0){
            local.results = dpSQAlgLocal(x = PUMS.sample[,c(1:10, 13:22
)], y = sample.y.true,
                                         epsilon = 1,
                                         t = 0.1*optimal.t.local)
        }

        # calculate predictions based on results of model
        y.pred = getPreds(local.results, PUMS)

        # FALSE NEGATIVE RATE
        ## 1 - true positive rate

        # get indices with true 1s - (true positives)
        true.positive.indices = which(sample.y.true == 1)
        # get true positive rate
        tpr = sum(y.pred[true.positive.indices] == sample.y.true[true.po
sitive.indices]
                  )/length(true.positive.indices)
        # store FNR
        FNR.L[boot, counter] = 1 - tpr

        # FALSE POSITIVE RATE
        ## 1 - TrueNegativeRate

        # get indices with true 0s - (true negatives)
        true.negative.indices = which(sample.y.true == 0)
        # get true negative rate
```

```
        tnr = sum(y.pred[true.negative.indices] == sample.y.true[true.ne
gative.indices]
                  )/(length(true.negative.indices))
        # store FPR
        FPR.L[boot, counter] = 1 - tnr
    }

    counter = counter + 1
}

# calculate averages for FPR and FNR
avg.FPR.C = colMeans(FPR.C)
avg.FNR.C = colMeans(FNR.C)
avg.FPR.L = colMeans(FPR.L)
avg.FNR.L = colMeans(FNR.L)
```

In [70]:

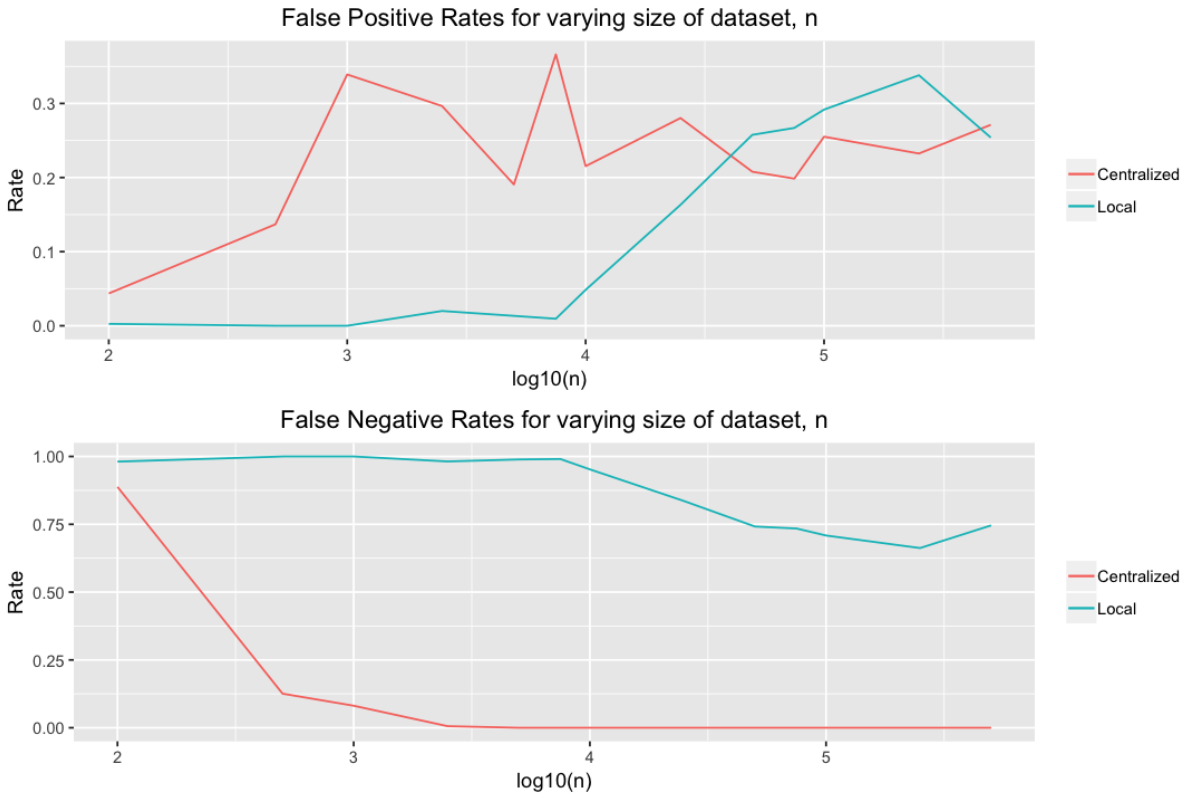
```
# make a df for plotting
rates.for.plot = data.frame(list.of.n, avg.FPR.C, avg.FNR.C, avg.FPR.L,
 avg.FNR.L)

# plot FPR
p1 <- ggplot(rates.for.plot) +
    geom_line(aes(x=log10(rates.for.plot$list.of.n), y=rates.for.plot$av
g.FPR.C,
                  color = "Centralized")) +
    geom_line(aes(x=log10(rates.for.plot$list.of.n), y=rates.for.plot$av
g.FPR.L,
                  color = "Local")) +
    labs(x = "log10(n)", y = 'Rate', title =
         'False Positive Rates for varying size of dataset, n') +
    theme(plot.title = element_text(hjust = 0.5), legend.title = element
_blank())



# plot FNR
p2 <- ggplot(rates.for.plot) +
    geom_line(aes(x=log10(rates.for.plot$list.of.n), y=rates.for.plot$av
g.FNR.C,
                  color = "Centralized")) +
    geom_line(aes(x=log10(rates.for.plot$list.of.n), y=rates.for.plot$av
g.FNR.L,
                  color = "Local")) +
    labs(x = "log10(n)", y = 'Rate', title =
         'False Negative Rates for varying size of dataset, n') +
    theme(plot.title = element_text(hjust = 0.5), legend.title = element
_blank())
```

In [71]:
```
options(repr.plot.width=9, repr.plot.height=6)
require(gridExtra)
grid.arrange(p1, p2, nrow=2)
```



Confusion matrix:

		Predicted class	
		P	N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

The plots above show trend of false positive rates and false negative rates for the classifier built on the output of the SQ algorithm using both centralized and local DP mechanisms.

False Positive Rates:

- Centralized model: For the centralized DP implementation of the SQ algorithm, the false positive rate seems to fluctuate 0.1 to 0.3 range as n increases up to 10^4 and then stays around 0.25 for larger values of n.
- Local model: The local DP implementation of the SQ algorithm yields low false positive rates for low values of n, but starts increasing and fluctuating for higher values between 10^4 and 10^5.

We had set the false positive rate of the SQ algorithm to be at most 0.1 using the threshold, and this has somehow translated into the classifier having high FPR for larger values of n. The number of true negatives falls as n increases.

False Negative Rates:

- Centralized model: The false positive rate starts out around 0.75 for low sample sizes, and plummets to close to 0 for n just less than 10^3. The false negative rate of the output of the SQ algorithm is at most the sum of probabilities for all elements in the output. So if were to think of this false negative rate for the classification as capturing what's going on in the SQ algorithm, it might be that the threshold is low enough that we're only ever selecting predictors with a true p_j of 0, resulting in a low false negative rate.
- Local model: For the centralized DP implementation of the SQ algorithm, the false negative rate is very close to 1.0 for n up to 10^4, and then it starts to drop off. This means that the threshold that we have set is low enough that none of the predictors are being released, and the classifier is only predicting everything as 0, i.e. our true positive rate ends up being 0.