HW 1: Reidentification, Reconstruction and Membership Attacks

Bhaven Patel

2/26/2019

I collaborated Anthony Rentsch and Lipika Ramaswamy on this homework.

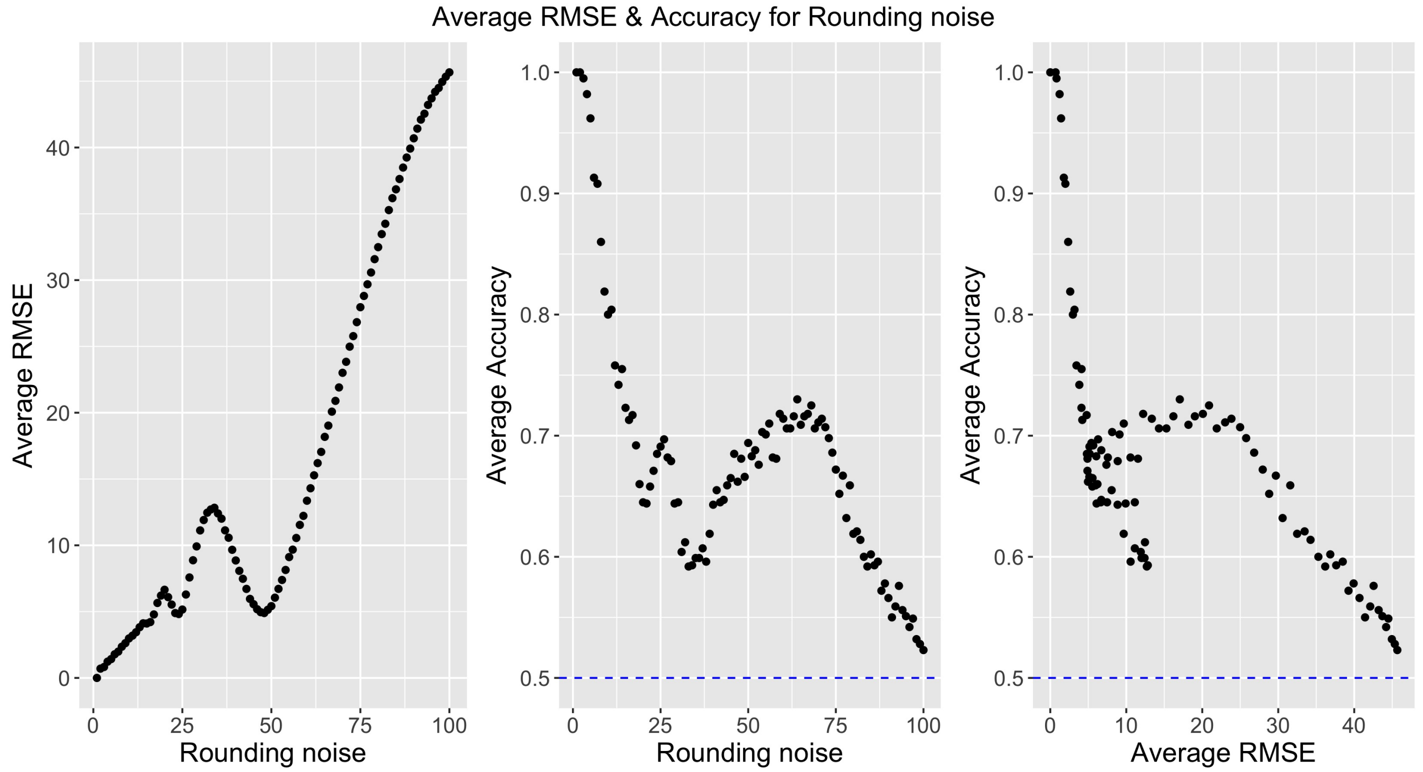
1. Reidentification Attack

To perform a reidentification attack on this PUMS dataset from the 2000 Census, I could obtain the Georgia voter registration file, which contains an individual's residential address, year of birth, gender, and race (Georgia's voter registration list (http://sos.ga.gov/index.php/elections/order_voter_registration_lists_and_files)). The individual's residential address could be used to identify which PUMA code he or she belongs to. A voter's "year of birth" would be used to calculate his/her age, while race would be used to match against the "latino", "black", and "asian" features we have in the PUMA dataset. Lastly, a voter's gender would be used to match against the "sex" feature in the PUMA dataset.
Assuming the individuals from the voter registration file are also in the 5% PUMA sample, I was able to uniquely identify 598 individuals in the PUMA dataset based on the available features from the voter registration file. This translated to reidentifying 2.3% of the individuals in the dataset.
Below is my code for performing the reidentification attack. It can also be accessed on Github (https://github.com/bhavenp/cs208/blob/master/homework/HW1/HW1_1.R).

```
library(plyr);
library(dplyr);

#read in the PUMS
data <- read.csv("../../data/FultonPUMS5full.csv");

#groupby PUMA region, age, sex, and race in data
data_gb <- data %>% group_by(puma, sex, age, latino, black, asian) %>% summarise(n=
n());
#get rows of groupby where we only have unique individuals
unique_indivs <- data_gb[data_gb$n == 1, ];
print(nrow(unique_indivs))
#calculate the percent of people that I could uniquely
perc_unq <- nrow(unique_indivs) / nrow(data);
print(perc_unq)
```

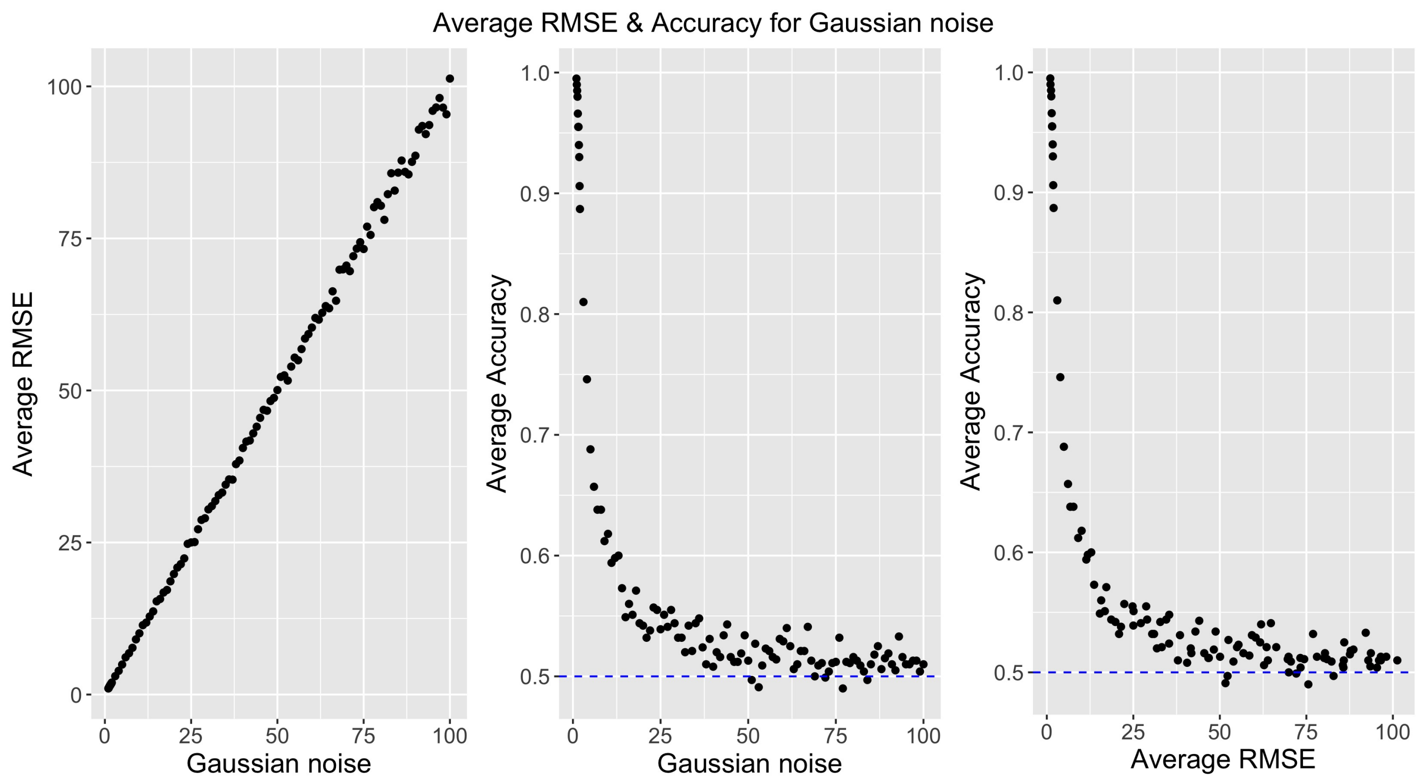
2. Reconstruction Attack

Below are graphs illustrating how adding “Rounding” noise to the answers to the queries in a reconstruction attack affects the RMSE of the answer and the accuracy of the reconstruction. The dashed blue line is drawn at 0.5, which is when the reconstruction attack fails because the attacker should be able to achieve an accuracy of 0.5 by simply flipping a coin to generate the sensitive bit for each individual.



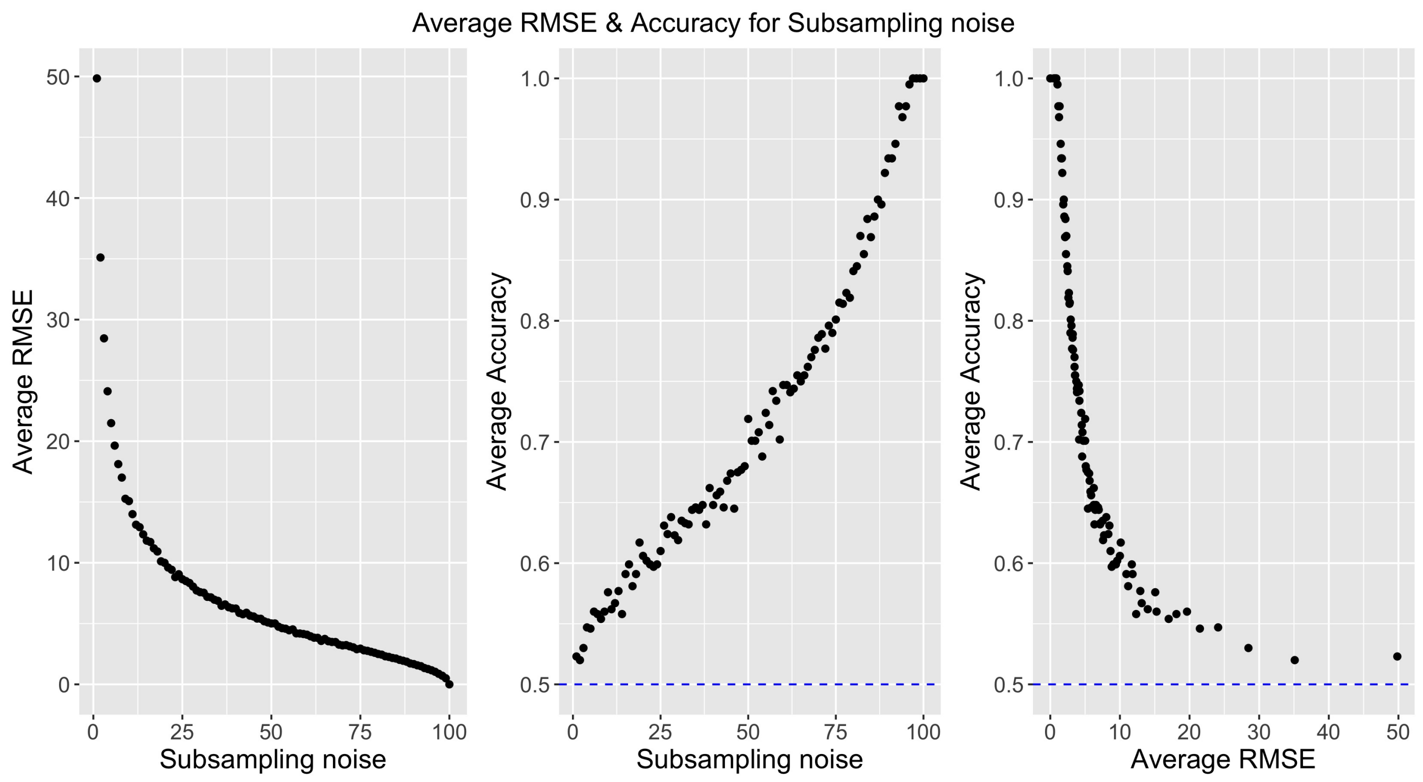
We see that as the “Rounding” noise increases, for the most part the RMSE of the answer also increases and the accuracy of the reconstruction decreases. There is an interesting increase in accuracy from ~30 to ~60 in the “Rounding” noise parameter because for most of the random subsets, the true sum of the sensitive bit (US Citizen) was around 50. Thus, the noisy answers are close to their true sum, as seen by the dip in RMSE from noise parameters ~30 to ~60 in the first graph. This occurrence causes the “Accuracy vs. RMSE” graph to have a unique shape where there is a cluster of points with an RMSE of ~5-12 and an accuracy between 60% adn 70%. Using a noise parameter of 100 for “Rounding” limits the attacker to achieving just baseline accuracy on the reconstruction, but it also provides very little utility.

Below are graphs illustrating how adding “Gaussian” noise to the answers to the queries in a reconstruction attack affects the RMSE of the answer and the accuracy of the reconstruction. The dashed blue line is drawn at 0.5 to indicate baseline accuracy for the attacker.



As expected, as Gaussian noise increases, the RMSE between the true value and the noisy answer increases and accuracy of reconstruction decreases; the relationship between RMSE and accuracy could be described as exponential decay. The accuracy of reconstruction drops very quickly even with adding noise from a Gaussian distribution with small standard deviations of 1.5-3. Noise parameters of 11 standard deviations and greater for “Gaussian” noise limit the attacker to achieving no more than 60% accuracy on the reconstruction attack, however the utility decreases as more Gaussian noise is added.

Below are graphs illustrating how adding noise using “Subsampling” to the generate noisy answers to the queries in a reconstruction attack affects the RMSE of the answer and the accuracy of the reconstruction. The dashed blue line is drawn at 0.5 to indicate the baseline accuracy for reconstruction.



As expected, as the “Subsampling” size increases, the RMSE of the answers decreases and the accuracy increases since the noisy count better approximates the true count with larger subsample sizes. The accuracy increases almost linearly, which was interesting to see since I expected an exponential increase. Like with the “Gaussian” noise, the relationship between RMSE and accuracy can be described by a exponential decay. Using a noisy parameter of 50 for “Subsampling” allows the attacker to achieve ~70% reconstruction accuracy and only an RMSE of ~5, providing some privacy and utility.

The code for my reconstruction attack can be accessed on Github (https://github.com/bhavenp/cs208/blob/master/homework/HW1/HW1_2.R) and is displayed below.

```
library(plyr); #import this library for doing rounding for noise
library(ggplot2); #import library for plotting
library(grid);

#### Parameters ####
prime <- 113; # prime number for hashing creating random vectors to pass to query
n <- 100;        # Dataset size
k.trials <- 2*n;  # Number of queries
num_exps <- 10; #number of experiments
noise_input <- "Subsampling"; # What type of noise will be used as defense. Can be
"Rounding", "Gaussian", or "Subsampling"

noise_vec <- c(1:100); #noise parameters for Rounding and Subsampling
# noise_vec <- c(seq(1, 1.9, 0.1), 2:100); #noise parameters for Gaussian


#### Import Data ####
pums_100 <- read.csv(file="../../data/FultonPUMS5sample100.csv"); #read in data fro
m data folder
#trim dataset to only the PUB categories plus "uscitizen"
pums_100_trimmed <- pums_100[, c("sex", "age", "educ","income", "latino", "black",
"asian",  "married", "divorced", "children", "employed", "militaryservice", "disabi
lity", "englishability", "uscitizen")];

#### Query function ####
query <- function(random_vec, data, prime, noise_type, noise_param){
  data_mat <- data.matrix(data[, 1:14]); # convert all rows and PUB categories into
matrix
  #multiply matrix by random vector and
  person_in_sum <- (data_mat %*% random_vec) %% prime %% 2;
  us_citz_sum <- sum(person_in_sum * data[, 15]);

  if(noise_type == "Rounding"){
    noisy_sum <- round_any(us_citz_sum, noise_param); #this function will round the
sum according to a multiple of noise_param
  }
  else if(noise_type == "Gaussian"){
    noisy_sum <- us_citz_sum + rnorm(n=1, mean=0, sd=noise_param); #add noise sampl
ed by from N(0, noise_param)
  }
  else{
    subsamp_ind <- sample(x=1:length(person_in_sum), size=noise_param, replace=FALS
E); #get a random sample of the indices w/o replacement
    subsamp_sum <- sum(person_in_sum[subsamp_ind] * data[subsamp_ind, 15]); #get th
e US citizen count for the subsample
    noisy_sum <- subsamp_sum * (n / noise_param); #sum to report will be subsamp_su
m x (scaling factor)
  }

  # return the actual sum, the noisy sum and indices. I am returning the indices he
re so that I don't have to recalculate which individuals were included in the sum.
```

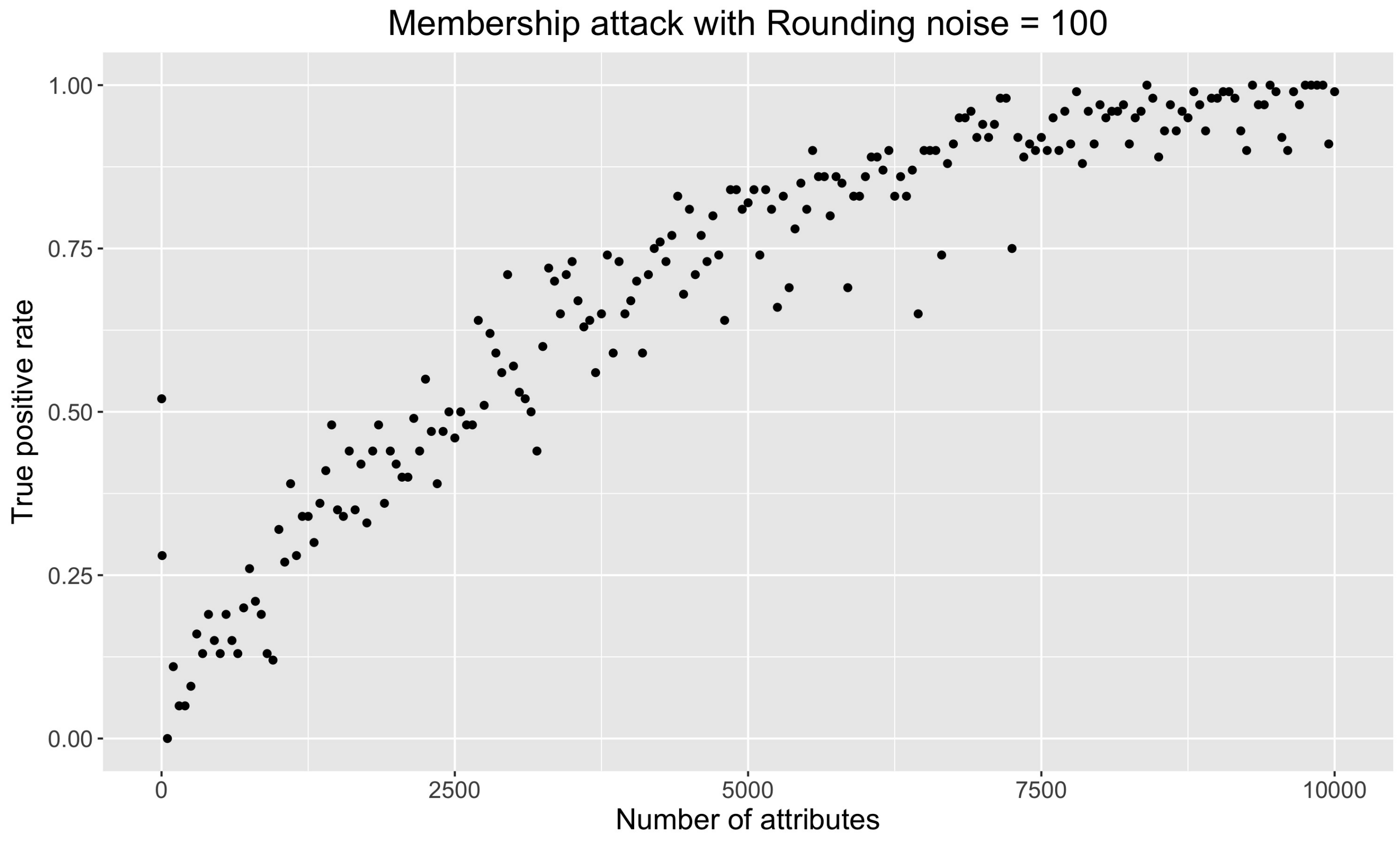
```
  return(list(us_citz_sum=us_citz_sum, noisy_sum=noisy_sum, indices=person_in_su
m));
}

#### Run experiment function ####
#### Give a prime number used for querying, data frame of PUB values, and how noise
should be added to query
run_experiment <- function(prime, data_input, noise_type, noise_to_add){
  #### Here we run our query repeatedly and record results
  history <- matrix(NA, nrow=k.trials, ncol=n+2);  # a matrix to store results in

  for(i in 1:k.trials){
    rand_vec <- sample(0:prime-1, size = ncol(data_input)-1);
    res <- query(random_vec=rand_vec, data=data_input, prime=prime, noise_type=nois
e_type, noise_param=noise_to_add);
    history[i,] <- c(res$us_citz_sum, res$noisy_sum, res$indices);  # save into our
history matrix
  }

  #### Convert matrix into data frame
  xnames <- paste("x", 1:n, sep="");
  varnames<- c("y", xnames);
  # convert noisy sum and indices in matrix into data frame
  releaseData <- as.data.frame(history[, 2:ncol(history) ]);
  names(releaseData) <- varnames; #add column names to data frame

  #### Run a linear regression
  formula <- paste(xnames, collapse=" + ");    # construct the formula, y ~ x1 ...
xn -1
  formula <- paste("y ~ ", formula, "-1");
  formula <- as.formula(formula);
  # print(formula);

  output <- lm(formula, data=releaseData);                    # run the regression
  estimates <- output$coef;                                   # save the estimates
  estimate_conv <- (estimates>0.5); # convert estimates to binary values

  sensitiveData <- data_input[, "uscitizen"];
  exp_acc <- sum(estimate_conv == sensitiveData) / n; #calculate the fraction of US
CITIZEN correctly reconstructed for this experiment
  #calculate RMSE between the exact value of our query and the noisy answer we pass
ed back
  exp_rmse <- ( sum((history[, 2] - history[, 1]) ** 2) / nrow(history)) ** 0.5;

  #return the
  return(list(exp_rmse=exp_rmse, exp_acc=exp_acc))
}


print(Sys.time())
#### Run through all noise parameters, each with 10 experiments ####
final_results <- matrix(NA, nrow=length(noise_vec), ncol=3);  # a matrix to store r
```

```
esults in

for(i in 1:length(noise_vec)){
  noise_to_add <- noise_vec[i];
  agg_rmse <- c(); #empty vector to hold the RMSE values of individual experiments
  agg_acc <- c(); #empty vector to hold the accuracy values of individual experimen
ts
  #need to go through num_exps for each noise parameter
  for(e in 1:num_exps){
    exp_res = run_experiment(prime = prime, data_input=pums_100_trimmed, noise_type
=noise_input, noise_to_add=noise_to_add);
    agg_rmse <- c(agg_rmse, exp_res$exp_rmse); #add RMSE to vector
    agg_acc <- c(agg_acc, exp_res$exp_acc); #add acc to vector
  }
  #put average of RMSEs and Accs from the experiments into the matrix
  final_results[i, ] <- c(noise_to_add, mean(agg_rmse), mean(agg_acc));
}
print(Sys.time())

final_results <- as.data.frame(final_results);
colnames(final_results) <- c("Param_vals", "RMSE", "Acc")
#### Plot results ####
f_size = 15;
fifty = 0.5;
# Plot average RMSE of reconstruction against noise input
p_rmse <- ggplot(data = final_results, aes(x=final_results$Param_vals, y=final_resu
lts$RMSE)) + geom_point();
p_rmse <- p_rmse + labs(x=paste(noise_input, "noise"), y = "Average RMSE") + theme
(plot.title = element_text(hjust=0.5), text = element_text(size=f_size));
# Plot average accuracy of reconstruction against noise input
p_acc <- ggplot(data = final_results, aes(x=final_results$Param_vals, y=final_resul
ts$Acc)) + geom_point();
p_acc <- p_acc + geom_hline(yintercept = fifty, linetype="dashed", color = "blue");
p_acc <- p_acc + labs(x=paste(noise_input, "noise"), y = "Average Accuracy") + them
e(plot.title = element_text(hjust=0.5), text = element_text(size=f_size));
# Plot average RMSE vs average accuracy of reconstruction
p_rmse_acc <- ggplot(data = final_results, aes(x=final_results$RMSE, y=final_result
s$Acc)) + geom_point();
p_rmse_acc <- p_rmse_acc + geom_hline(yintercept = fifty, linetype="dashed", color
= "blue");
p_rmse_acc <- p_rmse_acc + labs(x="Average RMSE", y = "Average Accuracy") + theme(p
lot.title = element_text(hjust=0.5), text = element_text(size=f_size));

#create grid for plotting
gs <- grid.arrange(p_rmse, p_acc, p_rmse_acc, nrow=1, ncol=3, top=textGrob(paste("A
verage RMSE & Accuracy for",noise_input,"noise"), gp=gpar(fontsize=15)) );

#### Export the graph

ggsave(filename = paste("./figs/regAttack", noise_input, "noise.jpg", sep = "_"), p
lot=gs, width = 11, height = 6);
```

```
#### Save data so I don't have to run the simualtion again if I want to re-plot the
data
write.csv(final_results, paste("./regAttack", noise_input, "noise.csv", sep =
"_"));
```

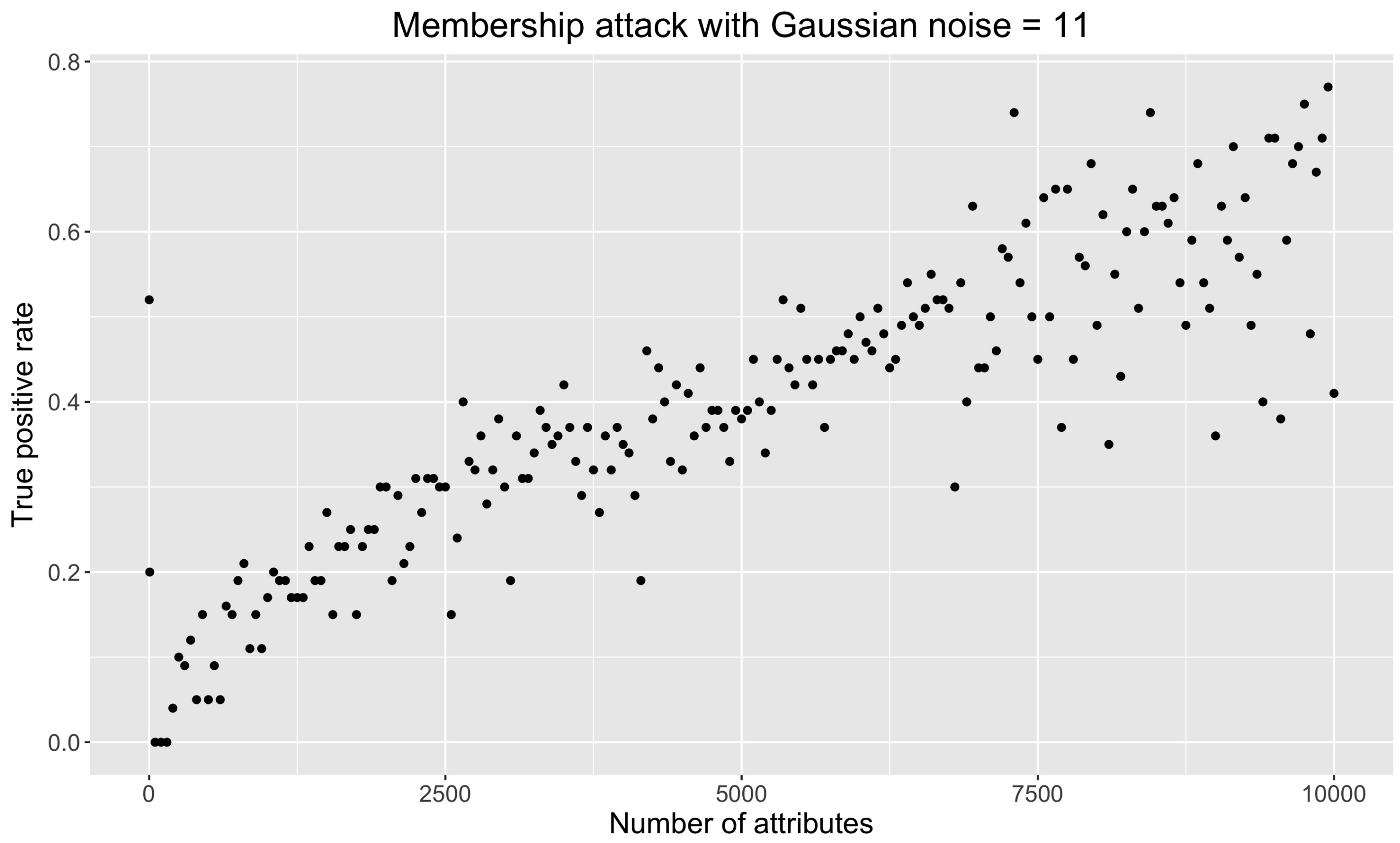
3. Membership Attack

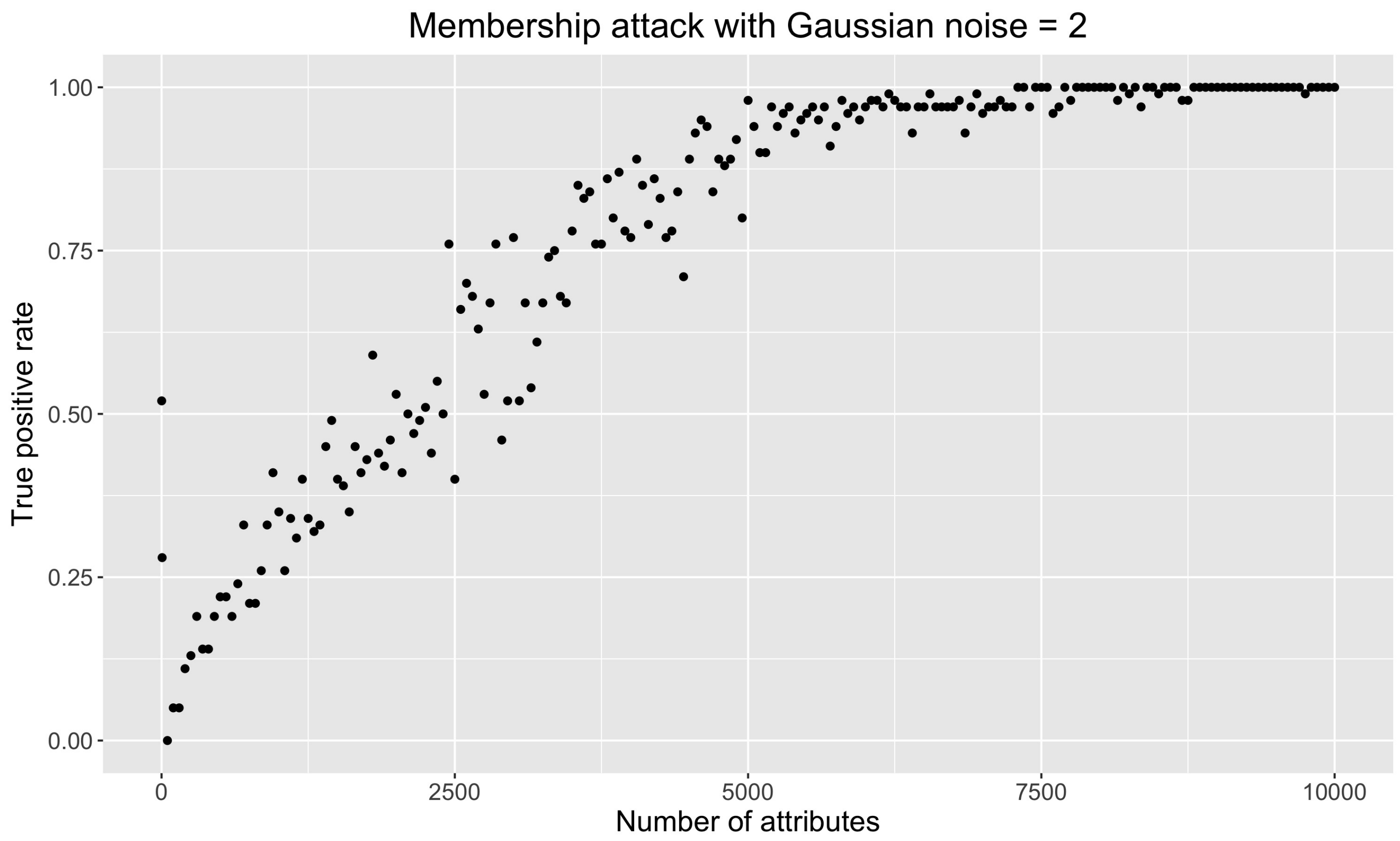
The graph below shows the results of my “Rounding” defense using a noise parameter of 100 against the membership attack as the number of attributes available to the attacker increases (I continue to increase the number of attributes by 50). I round the sum of each column in the sample dataset to a multiple of 100 before calculating the mean because 100 was the value at which the reconstruction attack failed. My false-positive rate false-positive rate (α) was set to 0.001.



It takes almost about 2500 attributes before the attacker can correctly identify about 50% of the individuals in the sample, and the attacker approaches 100% identification as the number of attributes approaches 10,000, since our error per query is $\sqrt{n}$ where n is the number of attributes/queries. From this, we can conclude that rounding is not a good defense against membership attacks.

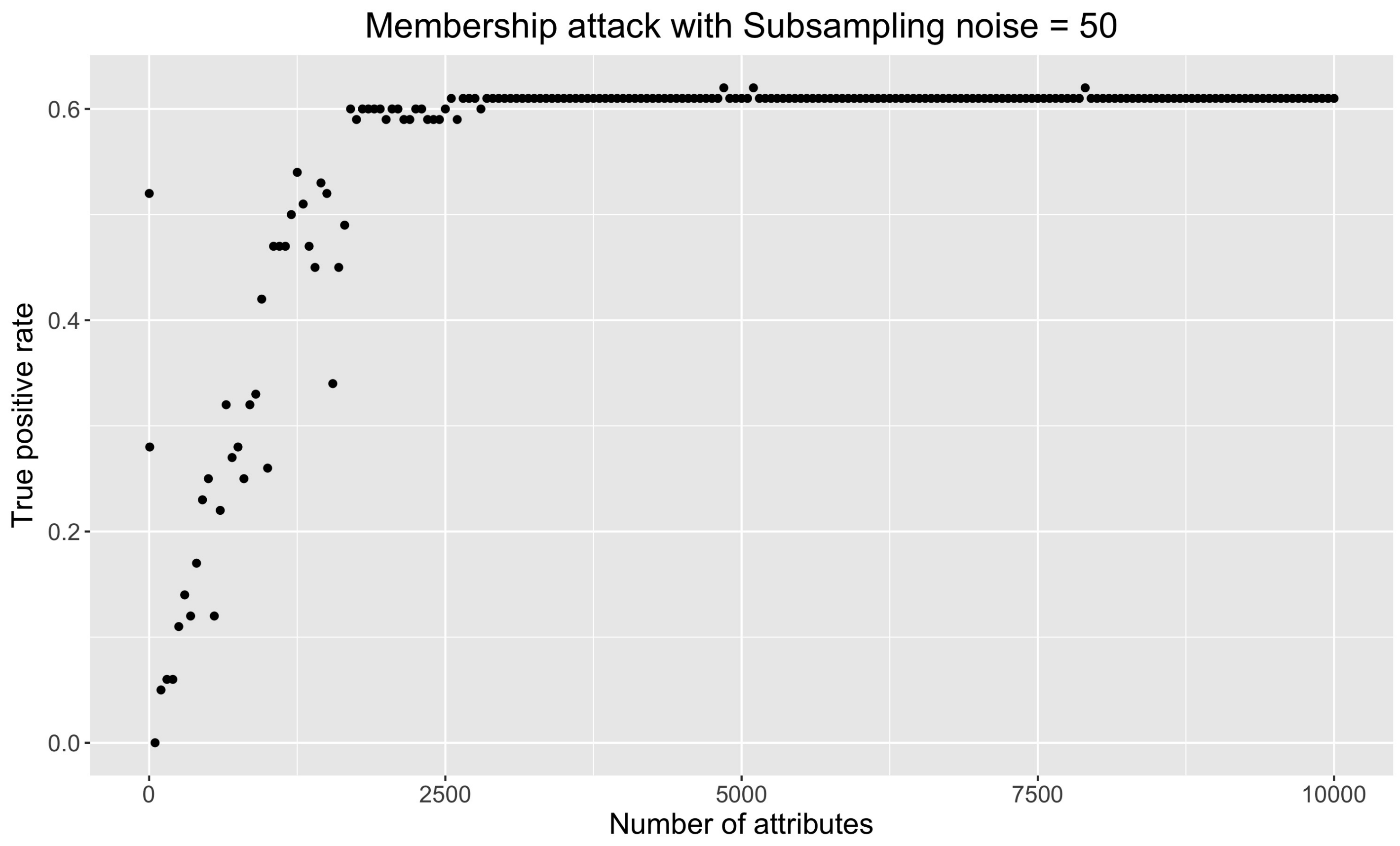
Below are graphs showing the results of my “Gaussian” defense using a noise parameter of 11 (standard deviations) against the membership attack as the number of attributes available to the attacker increases. I add a different amount of noise (sampled from a Gaussian distribution with mean 0 and standard deviation 11) to the sum of each column in the sample dataset before calculating the mean. With a Gaussian noise of 11, the reconstruction attack achieve ~60% accuracy and allows the membership attack to demonstrate how accuracy increases with more attributes. I also show that with Gaussian noise of 2, the membership attack can achieve 100% accuracy. My α was set to 0.001.

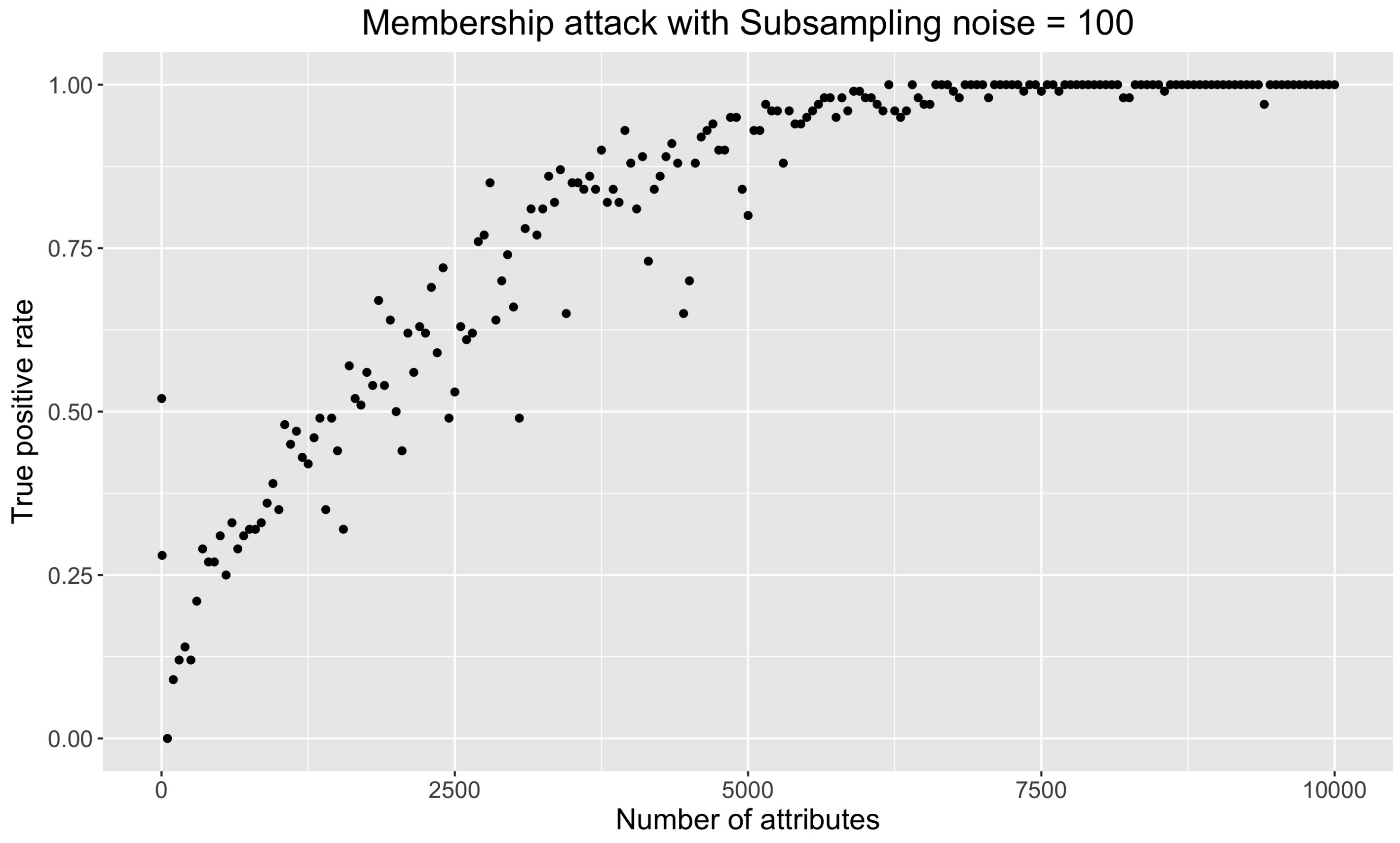




The membership attack does not achieve 100% accuracy with Gaussian noise of 11 standard deviations, but the accuracy does get to a range of 60-80% at large numbers of attributes. When I decrease the noise to only 2, the attacker is able to achieve 100% accuracy at ~7500 attributes. Using Gaussian noise in a range between 2-11 would probably provide a pretty good privacy versus utility tradeoff.

Below are graphs showing the results of my “Subsampling” defense using a noise parameter of 50 and 100 against the membership attack as the number of attributes available to the attacker increases. To generate means for the columns in the sample, I subsample 50 individuals from the sample and then report the means of the columns for just those 50 individuals. I used 50 and 100 as my noise parameters for "Subsampling" because it also allowed me to see how accuracy of the membership attack ranged with the number of attributes; lower noise parameters give decreasing true positive rates. My false-positive rate (α) was set to 0.001.





For noise parameter of 50, the membership attack accuracy increases very quickly for lower number of attributes and then maxes out at 60% after ~2000 attributes are used. It is somewhat successful since the true-positive rate surpasses 50%. It is logical that the subsampling defense with parameter 50 only allows ~60% accuracy since only 50 individuals are used to calculate the sample means.
Only once I increase the subsampling parameter to 100 do I get 100% membership attack accuracy because now the attacker has the true sample means. This graph (noise parameter of 100) follows a similar trend as the membership attack with “Rounding” noise of 100 because the true positive rate continues to increase as more attributes are used. Subsampling looks to be a good defense against membership attacks, however its provides low utility at small subsampling sizes.

The code for my membership attack can be accessed on Github (https://github.com/bhavenp/cs208/blob/master/homework/HW1/HW1_3.R) and and is displayed below.

```
library(plyr); #import this library for doing rounding for noise
library(ggplot2); #import library for plotting
library(gridExtra);

#### Parameters ####
prime <- 113; # prime number for hashing creating random vectors to pass to query
n <- 100;        # Dataset size
num_attr <- n*n; #calculate the number of attributes needed
noise_type <- "Subsampling"; # What type of noise will be used as defense. Can be
"Rounding", "Gaussian", or "Subsampling"
# noise_val <- 100; #noise to introduce for Rounding
# noise_val <- 11; #noise to introduce for Gaussian
noise_val <- 50; #noise to introduce for Subsampling


#### Import Data ####
pums_full <- read.csv(file = "../../data/FultonPUMS5full.csv"); #read in the full d
ata
#trim the population to PUB categories
pums_full_trimmed <- pums_full[, c("sex", "age", "educ", "latino", "black", "asia
n",  "married", "divorced", "children", "employed", "militaryservice", "disabilit
y", "englishability")];
pums_100 <- read.csv(file="../../data/FultonPUMS5sample100.csv"); #read in sample d
ata from data folder
#trim sample to only PUB cols
pums_100_trimmed <- pums_100[, c("sex", "age", "educ", "latino", "black", "asian",
"married", "divorced", "children", "employed", "militaryservice", "disability", "en
glishability")];
#---------------------------------------------------------------------#

#### Do some cleaning of the data ####
#dummify age by comparing each individual's age to the mean
pums_full_trimmed$age <- ifelse(pums_full_trimmed$age > mean(pums_full_trimmed$ag
e), 1, 0);
pums_100_trimmed$age <- ifelse(pums_100_trimmed$age > mean(pums_full_trimmed$age),
1, 0);

#dummify education column in the population and sample
for(e in unique(pums_full_trimmed$educ)){
  pums_full_trimmed[paste("educ", e, sep = "_")] <- ifelse(pums_full_trimmed$educ =
= e, 1, 0);
  pums_100_trimmed[paste("educ", e, sep = "_")] <- ifelse(pums_100_trimmed$educ ==
e, 1, 0);
}
#remove the education column in the population and sample
pums_full_trimmed <- subset(pums_full_trimmed, select = -c(educ));
pums_100_trimmed <- subset(pums_100_trimmed, select = -c(educ)); #remove the educat
ion column

#generate random predicates
preds_to_add <- as.matrix( replicate(num_attr-ncol(pums_full_trimmed), sample(0:pri
me-1, size=ncol(pums_full_trimmed)), simplify=TRUE) );

#generate binary values for each row in sample using predicates
preds_to_add_pop <- (as.matrix(pums_full_trimmed) %*% preds_to_add) %% prime %% 2;
#generate binary values for each row in sample using predicates
preds_to_add_samp <- (as.matrix(pums_100_trimmed) %*% preds_to_add) %% prime %% 2;

#create final matrices, which represent our population and our sample of 100 with 1
0,000 attributes
pums_full_final <- cbind(as.matrix(pums_full_trimmed), preds_to_add_pop);
pums_100_final <- cbind(as.matrix(pums_100_trimmed), preds_to_add_samp);

## Generate underlying population attributes
pop_prob <- colMeans(pums_full_final);
#----------------------------------------------------------------------#

#### Bulid Null Distribution ####
## A utility function to create data from the population
rmvbernoulli <- function(n=1, prob){
  history <- matrix(NA, nrow=n, ncol=length(prob))
  for(i in 1:n){
    x<- rbinom(n=length(prob), size=1, prob=prob); #
    x[x==0] <- -1      # Placeholder for transformation
    history[i,] <- x
  }
  return(history)
}

#### function to generate noisy sample means from data
gen_sample_probs <- function(data, noise_type, noise_param){
  if(noise_type == "Rounding"){
    col_sums = colSums(data);
    noisy_means <- round_any(col_sums, noise_param) / nrow(data); #this function wi
ll round the sums according to a multiple of noise_param and divide by the number o
f data points
    return( 2*(noisy_means - 0.5) );
  }
  else if(noise_type == "Gaussian"){
    noisy_sums <- colSums(data) + rnorm(n=ncol(data), mean=0, sd=noise_param); #add
noise sampled by from N(0, noise_param)
    noisy_means <- noisy_sums / nrow(data);
    return( 2*(noisy_means - 0.5) );
  }
  else{
    subsamp_ind <- sample(x=1:nrow(data), size=noise_param, replace=FALSE); #get a
random sample of the indices w/o replacement
    noisy_means <- colMeans(data[subsamp_ind, ]);  #get column means from subsample
    return( 2*(noisy_means - 0.5) );
  }
}
```

```
## A null distribution and critical value generator. Taken from membershipAttack.r
nullDistribution <- function(null.sims=1000, alpha=0.05, test_stat, population.pro
b, sample_means){
  population.mean <- 2*(population.prob-0.5)
  hold <- rep(NA,null.sims);

  for(i in 1:null.sims){
    nullAlice <- rmvbernoulli(n=1, prob=population.prob); #get an Alice that is jus
t from the population
    hold[i] <- eval(test_stat(alice=nullAlice, sample.mean=sample_means, populatio
n.mean=population.mean));
  }
  nullDistribution <- sort(hold, decreasing=TRUE);
  criticalValue <- nullDistribution[round(alpha*null.sims)];
  return(list(nullDist=nullDistribution, criticalVal=criticalValue));
}

#function that defines the Dwork test statistic. Taken from membershipAttack.r
test.Dwork <- function(alice, sample.mean, population.mean){
  test.statistic <- sum(alice * sample.mean) - sum(population.mean * sample.mean);
  return(test.statistic)
}
#---------------------------------------------------------------------#

#### Perform Attack ####
range_d = c(1, 5, seq(50, 10000, by=50)); #range of number of attributes to go thro
ugh

history <- matrix(NA, nrow=length(range_d), ncol=3);
myalpha <- 1 / (10*n); #myalpha is 0.001
population.mean <- 2*(pop_prob - 0.5); #scale population means/probs to -1 and 1
#get noisy sample means for all 10,000 attributes
sample_means <- gen_sample_probs(pums_100_final, noise_type=noise_type, noise_param
=noise_val);

print(Sys.time())
#loop through the number of attributes
row_counter = 1;
for(d in range_d ){ #loop through the different number of attributes
  print(d);
  sample_means_d <- sample_means[1:d]; #cut sample means to 1:d attributes
  #generate a new test statistic for a new null distribution based on 'd' attribute
s
  output <- nullDistribution(alpha = myalpha, test_stat = test.Dwork,
                             population.prob = pop_prob[1:d], sample_means = sample
_means_d);
  # test_stats_for_samp <- c();
  true_pos = 0;
  for(a in 1:nrow(pums_100_final)){ #loop through the 100 people in the sample
    alice <- pums_100_final[a, 1:d]; #choose Alice from sample. Cut to 1:d columns
    alice[alice==0] <- -1; #change 0's for alice to -1's
    # Conduct test for test statistic
    test.alice.Dwork <- test.Dwork(alice=alice, sample.mean=sample_means_d,
                                   population.mean=population.mean[1:d]);
    # test_stats_for_samp <- c(test_stats_for_samp, test.alice.Dwork); #for debuggi
ng
    if( test.alice.Dwork >= output$criticalVal){ #check if test statistic is greate
r than critical value
      true_pos = true_pos + 1;
    }
  }
  tp_rate = true_pos / nrow(pums_100_final); #calculate the true positive rate
  history[row_counter, ] <- c(d, tp_rate, output$criticalVal);
  row_counter = row_counter + 1;
}
print(Sys.time())



#### Plot results ####
f_size = 15;
history <- as.data.frame(history);
colnames(history) <- c("num_attr", "tpr", "crit_val");
# Plot average RMSE of reconstruction against noise input
p <- ggplot(data = history, aes(x=history$num_attr, y=history$tpr)) + geom_point();
p <- p + labs(title = paste("Membership attack with", noise_type, "noise =", noise_
val), x="Number of attributes", y = "True positive rate") + theme(plot.title = elem
ent_text(hjust=0.5), text = element_text(size=f_size));

#### Export the graph
ggsave(filename = paste("./figs/memAttack", noise_type, "noise.jpg", sep = "_"), pl
ot=p, width = 10, height = 6);
#### Save data so I don't have to run the simualtion again if I want to re-plot the
data
write.csv(history, paste("./memAttack", noise_type, "noise.csv", sep = "_"));
```

4. Final Project Ideas

Anthony Rentsch and I will be working together for the final project. We are interested in implementing an attack on a real-life dataset that contains some potentially sensitive information, especially datasets from the social and life sciences (i.e., medical records, political surveys). After implementing an attack, we would formulate a strategy for a differentially private algorithm/mechanism that could be used to preserve individual privacy while also enabling researchers to gain utility for different use cases, including descriptive queries and inferential/predictive modeling. Specifically, we would be interested in seeing how a differentially private algorithm affects the performance of machine learning models. We would also be interested in examining how a differentially private algorithm could be implemented in a publicly available application that provides useful diagnostic information; for instance, we could use the City of Boston's public 311 service requests or Field Interrogation and Observation data (similar to stop-and-frisk data) to provide updates on the current safety/well-being of different neighborhoods, while limiting any privacy loss.