cs208 Homework 1

Anthony Rentsch

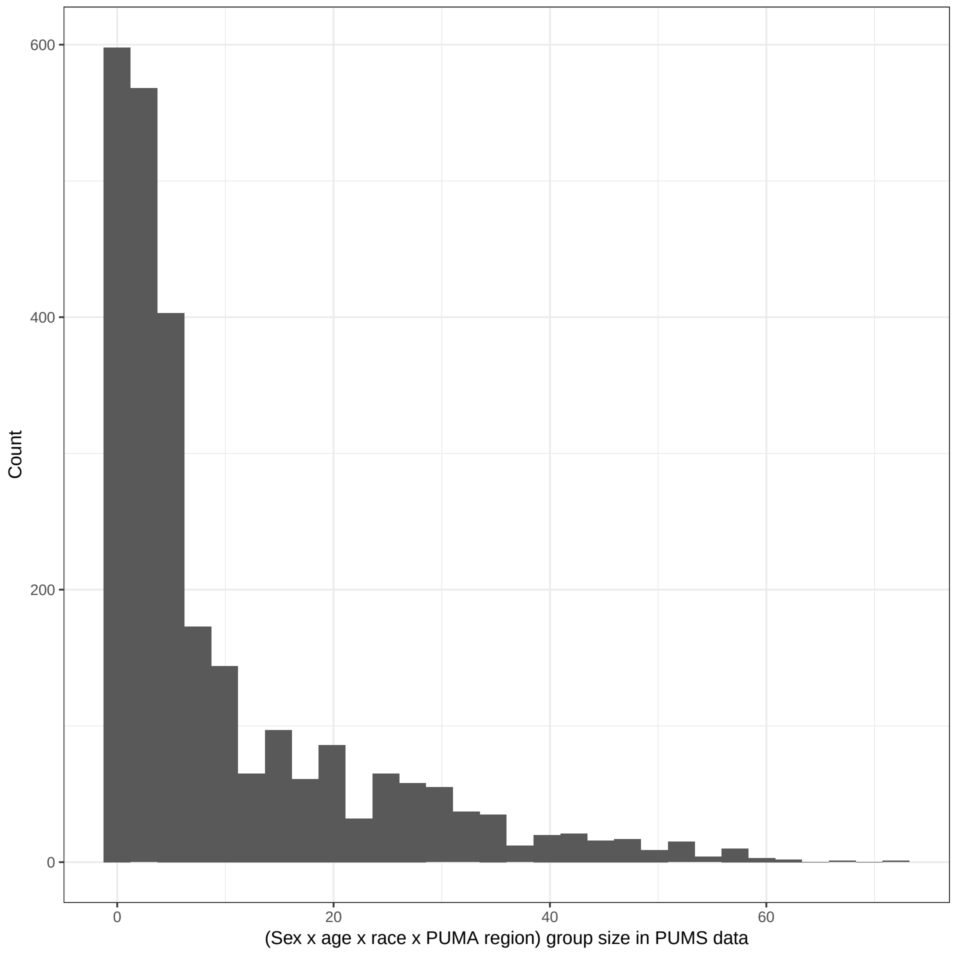
2/26/2019

For this assignment, I collaborated with Bhaven Patel and Lipika Ramaswamy. Find the code I wrote for this assignment in my Github repository.

Question 1

Similar to Latanya Sweeney's original attack, I would use the Georgia state voter file to attempt to re-identify individuals in the 2010 Census PUMS dataset for Georgia. As a result of the Voting Rights Act of 1965, several states - including Georgia - began asking people to self-report their race when filling out voter registration forms. Thus the statewide Georgia voter file, which costs just $250, could be used to conduct an attack on the PUMS dataset with race, gender, age, and PUMA region (derived from address) as available quasi-identifiers.

When we have these four attributes at our disposal, we are able to uniquely identify 2.3% of the individuals in the PUMS data. Note that this is much lower than the number of people that Sweeney was able to re-identify since she had access to each individual's exact birthdate, not just their age. Furthermore, roughly 7% of individuals in the sample share their combination of sex, race, age, and PUMA region with less than 10 other people; if we had slightly more auxiliary data we might have a reasonable chance to identify these people, too.

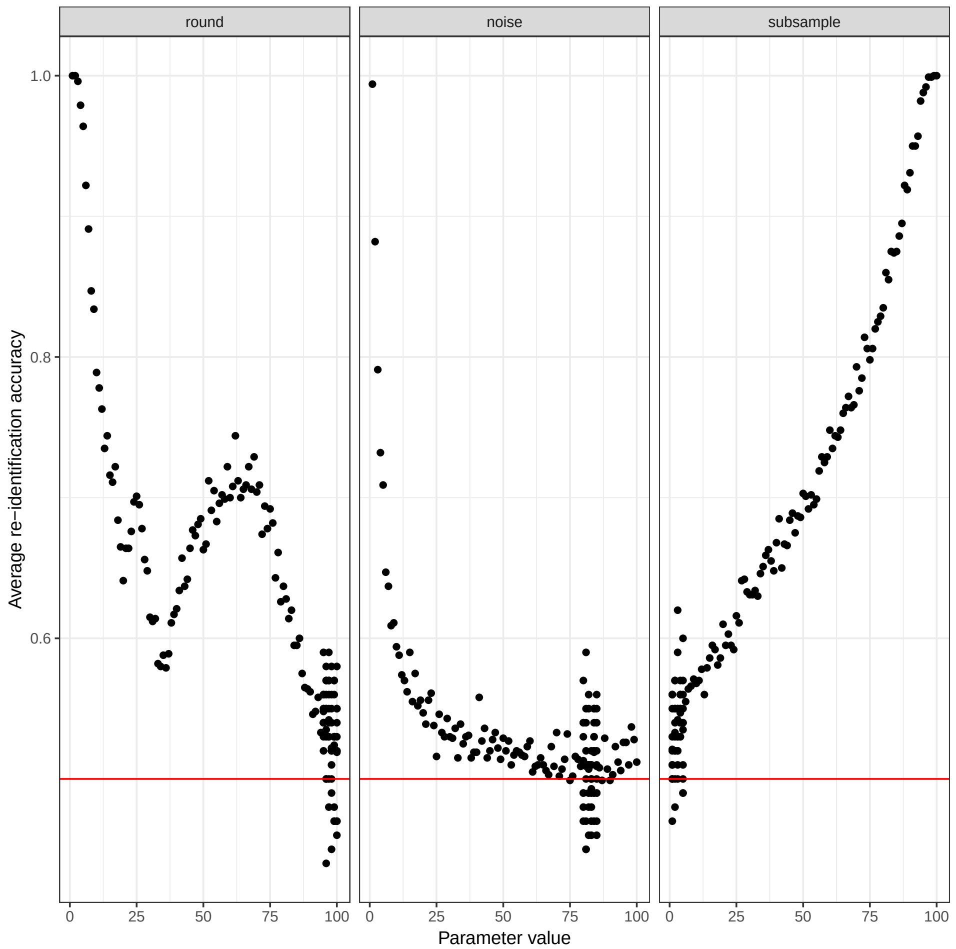


Question 2

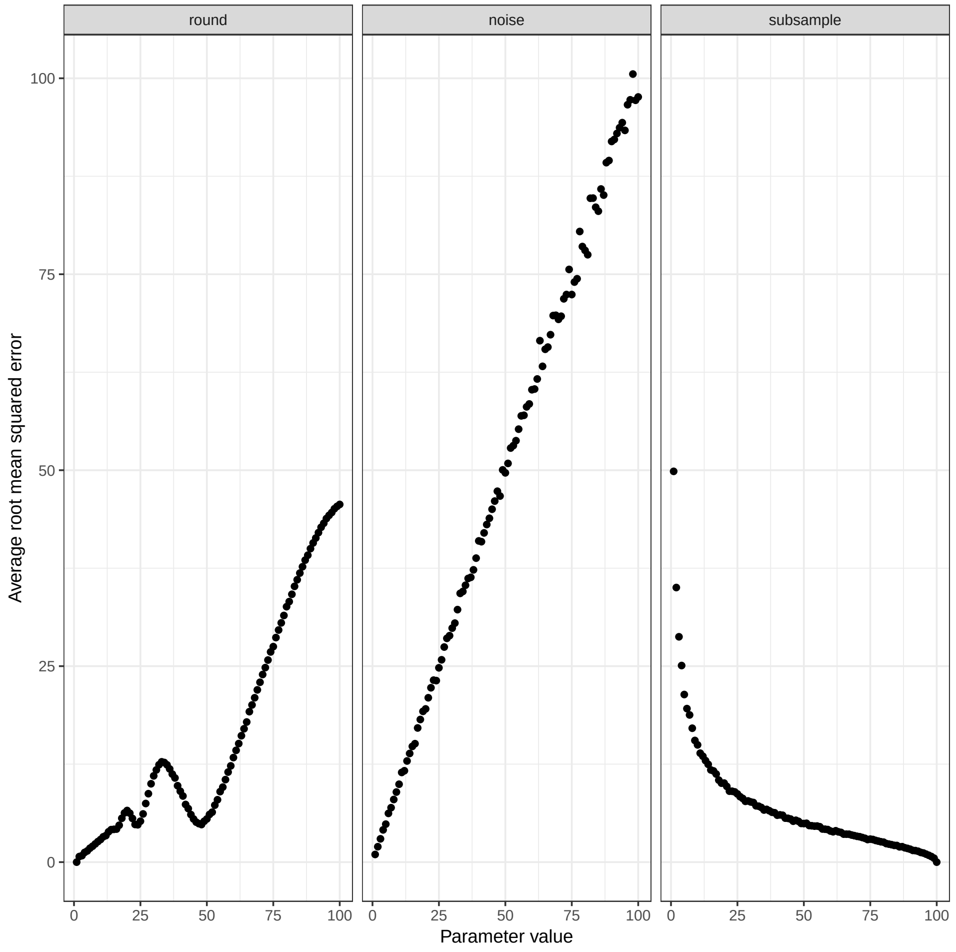
The first plot below shows the relationship between the defense parameter (R for rounding, ϵ for noise, and T for subsampling) and the average re-identification accuracy for 10 experimental regression attacks run at each parameter value. In general, we observe that re-identification accuracy goes up when privacy is worse, which is the expected behavior. As we increase the standard deviation of the added Gaussian noise the accuracy goes down and as we increase the number of people we subsample the re-identification accuracy goes up, as expected. For rounding there is a seemingly out-of-place spike in the re-identification accuracy rate around 50. Upon further inspection, this spike makes sense because 96% of the 100-person sample are U.S. citizens, so when our query randomly samples roughly 50 people the expected value of the query is approximately 50.

Also note that, since the attack would fail at 50% accuracy, we can set defense parameters at $R = 100$, $T = 5$, and $\epsilon = 100$. I've added additional data points around these values, i.e., the results from all of the individual experiments within 5 parameter values of this point.

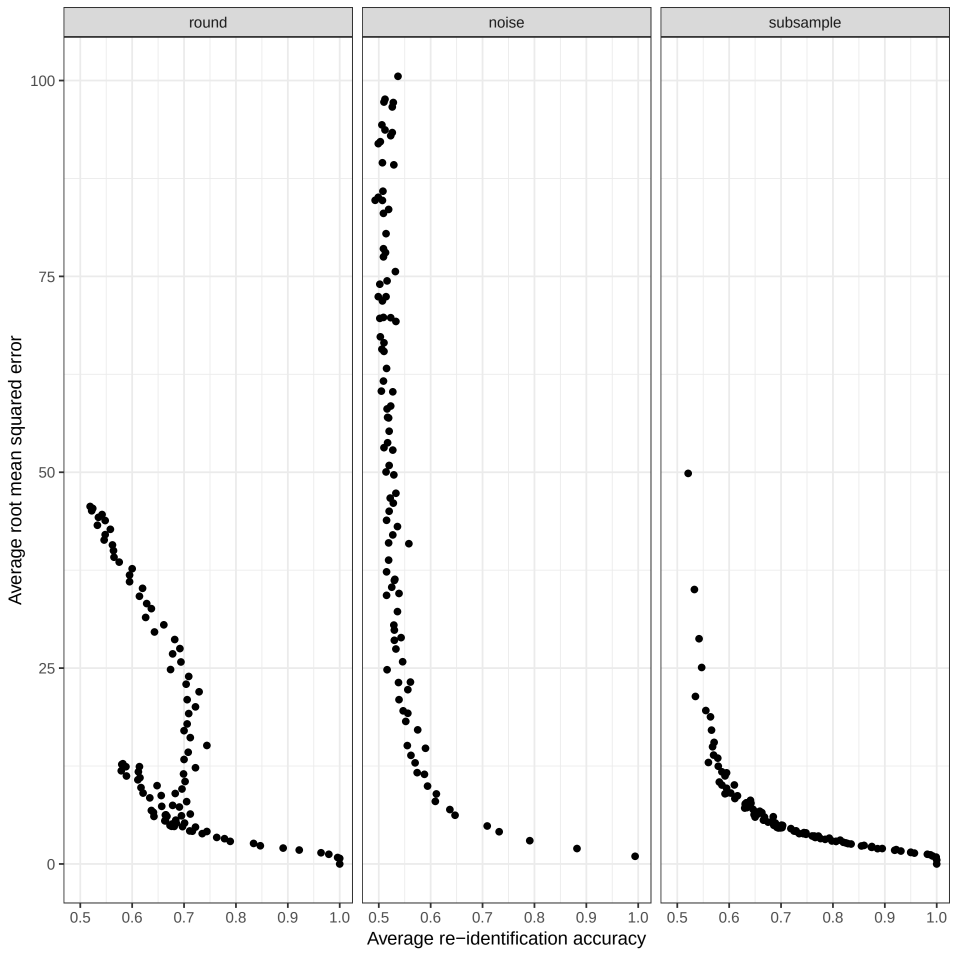
For the noise injection defense, it appears that the attack fails for a slightly lower value of ϵ but I choose 100 as the optimal value for Question 3 to test the effect of injecting the most noise possible under the problem constraints. For the subsampling defense, while I find $T = 1$ to be approximately the optimal parameter to thwart the regression attack, I choose $T = 5$ to use moving forward so that I can examine what happens when we subsample at least a few people and not just one person.



The next plot shows the relationship between the defense parameter and the utility of the query, as measured by the root mean squared error between the actual answer to the query and query mechanism result. We see that larger values of R for the rounding defense lead to larger RMSE values (except for the previously explained bump at 50), larger values of ϵ for the noise defense lead to larger RMSE values, and larger values of T for the subsampling defense lead to lower RMSE values.



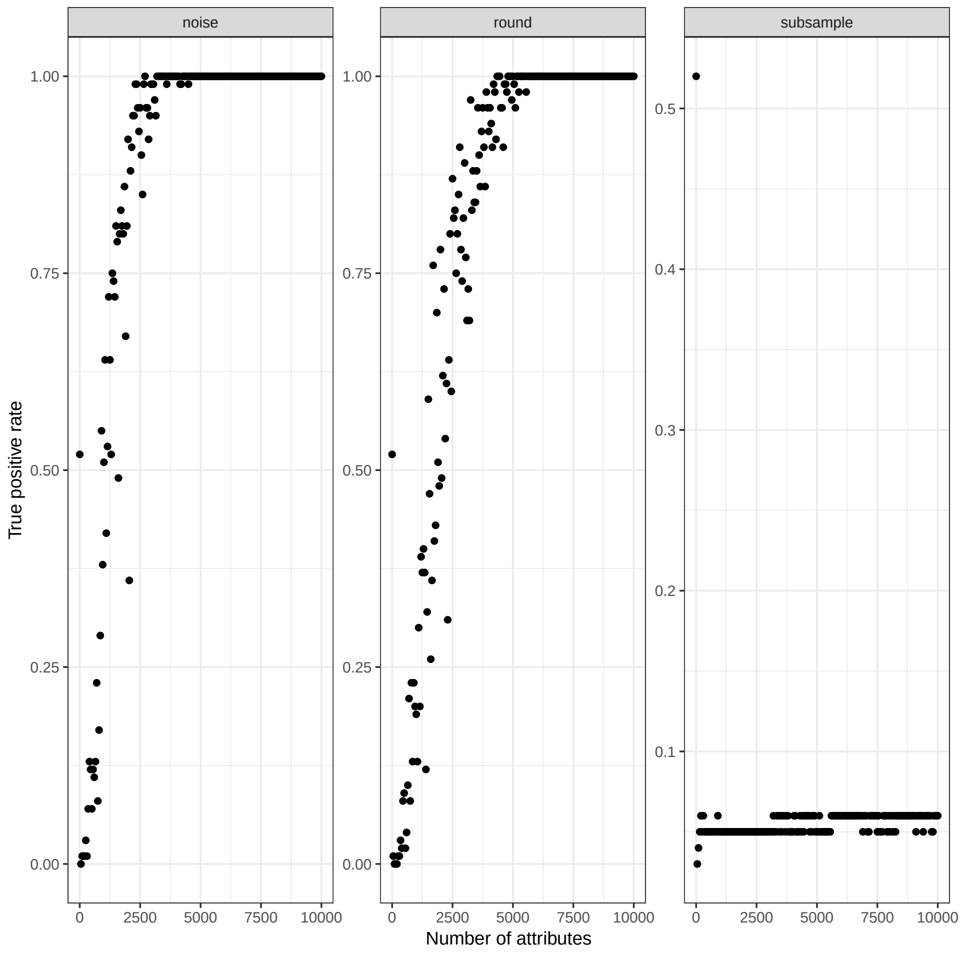
Comparing the privacy-utility tradeoff directly, we see that there is an approximately exponential decay in RMSE as the re-identification accuracy increases. In other words, as we shed privacy we see huge improvements in accuracy, but the size of those improvements tends to decrease as we approach near perfect re-identification accuracy. Again, the curve for rounding appears a bit strange, but this is a reflection of the idiosyncratic nature of the query's behavior around 50, i.e., rounding to the nearest multiple of 50 a sum with an expected value of 50 leads to low RMSE even when re-identification accuracy is relatively low.



Question 3

We would expect membership attacks to be successful even when reconstruction attacks are unsuccessful. In the plots below I depict the relationship between the number of attributes an adversary can query from a dataset and the true positive rate of a membership attack (conducted using the Dwork test statistic and rescaling population and sample means to live in $[-1, 1]$). Additionally, note that I only allow the adversary to query the derived attributes of the dataset, which are formed by the hashing process used in Question 2. The defense parameters I use are: $R = 100$, $\epsilon = 100$, and $T = 5$.

For both the noise injection and rounding defenses, the membership attack reaches near perfect true positive rates after the release of only about 2,200 and 4,200 attributes, respectively. This means that these defenses cannot hold off membership attacks when up to n^2 attributes are released, as we saw in lecture. For subsampling, however, the membership attack's performance plateaus at roughly T/n and does not change as we increase the number of attributes released. Based on this analysis, subsampling is the best defense against a membership attack and, referring back to the first plot in Question 2, subsampling with a small value of T (for example, $T < 10$) still provides strong utility.



Question 4

Bhaven Patel 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.

Appendix

I put the code for all of my analyses here. You can also find it on Github.

Question 1

```
grouped_pums_full <- pums_full %>% group_by(puma, sex, asian, black, latino, age) %>%
  summarise(n = n())

q1_plot <- ggplot(grouped_pums_full) + geom_histogram(aes(n)) +
  labs(x="(Sex x age x race x PUMA region) group size in PUMS data", y = "") +
  theme_bw()

sum(grouped_pums_full$n == 1)/sum(grouped_pums_full$n)
sum(grouped_pums_full$n < 10)/sum(grouped_pums_full$n)
```

Question 2

```
# define table that adversary has access to
adversary_db <- pums_100 %>% select(sex,age,educ,income,latino,black,asian,married,divorced,
                                    children,disability,militaryservice,employed,englishability)
adversary_data_matrix <- data.matrix(adversary_db)

# query function
# INPUTS:
# random_vector (array of ints of size=length(data))
# data (dataframe)
# defense ("round","noise","subsample")
# defense_parameter (int)
# OUTPUT:
# sum of query mechanism and of actual query (for comparison)
query <- function(random_vector, data, defense="round", defense_parameter=NULL){

  # random subsets
  # use matrix multiplication for more efficient computation
  # this returns indicies for subset
  # Note: I do hashing inside query function rather than passing it in for ease of computation
  data_matrix <- data %>% select(-state,-puma,-uscitizen,-fips) %>% data.matrix()
  subset_inds <- ((data_matrix %*% random_vector) %% P) %% 2
  data$subset_ind <- subset_inds[,] # save this for subsample defense
  subset <- data[subset_inds==1,]
  sum_actual <- sum(subset$uscitizen)

  if (defense == "round") {
    sum_defended <- plyr::round_any(sum(subset$uscitizen), defense_parameter)
  }
  else if (defense == "noise"){
    sum_defended <- sum(subset$uscitizen) + rnorm(n=1, mean=0, sd=defense_parameter)
  }
  else if (defense == "subsample") {
    subsample <- data %>% sample_n(defense_parameter) %>% filter(subset_ind==1)
    sum_defended <- sum(subsample$uscitizen) * (nrow(data)/defense_parameter)
  }
  return(list(sum_defended=sum_defended, sum_actual=sum_actual))
}

# set parameters
```

```r
n <- nrow(adversary_db)
num_trials <- 2*n
num_experiments <- 10
P <- 131 # large prime for randomly hashing vectors
attack_performance <- data.frame()

# run attack

for(defense_type in c("round","noise","subsample")) {

  for(param_value in 1:100) {

    for(experiment in 1:num_experiments){

      # initialize results matrix and actual results vector
      results <- matrix(NA, nrow=num_trials, ncol=n+1)
      actual_results <- c()

      for(trial in 1:num_trials){
        # random vector
        r_vec <- sample(x=(0:P-1), size=length(adversary_db))
        # do hashing to generate inds used in sum from adversary_db
        used_inds <- ((adversary_data_matrix %*% r_vec) %% P) %% 2

        # perform query
        res <- query(random_vector=r_vec, data=pums_100, defense=defense_type,
                     defense_parameter=param_value)
        # append sum and inds to matrix of results
        results[trial,] <- c(res$sum_defended, used_inds)
        # save actual results in a vector
        actual_results <- c(actual_results, res$sum_actual)
      }

      # convert matrix into df
      results_df <- data.frame(results)
      # run regression
      regression <- lm(X1 ~ . - 1, data=results_df)
      estimates <- regression$coef

      # save relevant results
      acc <- (sum((estimates > 0.5) == pums_100$uscitizen) / 100)
      rmse <- sqrt(mean((results_df$X1 - actual_results)^2))

      # append row to results df
      new_row <- data.frame(defense_type,experiment,param_value,acc,rmse)
      attack_performance <- rbind(attack_performance, new_row)
    }
  }
}

# make column names informative
names(attack_performance) <- c("defense","experiment","param_value","accuracy","rmse")

# create df with average values for each parameter setting for each defense type
```

```
attack_avg_perf <- attack_performance %>% group_by(defense, param_value) %>%
  summarise(mean_acc = mean(accuracy), mean_rmse = mean(rmse))

# create df with extra data points around transition
extra_points <- attack_performance[(attack_performance$defense=="round" &
                                   attack_performance$param_value%in%95:100) |
                                   (attack_performance$defense=="noise" &
                                   attack_performance$param_value%in%80:85) |
                                   (attack_performance$defense=="subsample" &
                                   attack_performance$param_value%in%1:5),
                                   c("defense","param_value","accuracy","rmse")]
names(extra_points) <- c("defense","param_value","mean_acc","mean_rmse")
attack_avg_perf_w_extra_points <- dplyr::bind_rows(attack_avg_perf, extra_points)

# plot results

q2_plot_rmse_acc <- ggplot(attack_avg_perf, aes(x=mean_acc, y=mean_rmse)) +
  geom_point() + facet_wrap(~defense) +
  labs(x="Average re-identification accuracy", y="Average root mean squared error") +
  theme_bw()
pdf("q2_plot_rmse_acc.pdf", width=8, height=8)
q2_plot_rmse_acc
dev.off()

q2_plot_acc <- ggplot(attack_avg_perf_w_extra_points, aes(x=param_value, y=mean_acc)) +
  geom_point() +
  geom_hline(yintercept = 0.5, col="red", lty=1) +
  facet_wrap(~defense) +
  labs(x="Parameter value", y="Average re-identification accuracy") +
  theme_bw()
pdf("q2_plot_acc.pdf", width=8, height=8)
q2_plot_acc
dev.off()

q2_plot_rmse <- ggplot(attack_avg_perf, aes(x=param_value, y=mean_rmse)) +
  geom_point() + facet_wrap(~defense) +
  labs(x="Parameter value", y="Average root mean squared error") +
  theme_bw()
pdf("q2_plot_rmse.pdf", width=8, height=8)
q2_plot_rmse
dev.off()
```

Question 3

```
# set parameters
k.attributes <- seq(0, 10000, by=50)
null.sims <- 1000
optimal_defense_params <- list(round=100, noise=100, subsample=5)
alpha <- 0.001
P <- 197

# generate matrix with each column being a vector of random values
rand_matrix <- replicate(10000, sample(x=(0:P-1), size=14)) %>%  data.matrix()
```

```
# get random predicates as new attributes for ...
# ... PUMS 100
pums_100_matrix <- pums_100 %>% select(-state,-puma,-uscitizen,-fips) %>% data.matrix()
pums_100_new <- ((pums_100_matrix %*% rand_matrix) %% P) %% 2

# ... PUMS full
pums_full_matrix <- pums_full %>% select(-state,-puma,-uscitizen,-fips) %>% data.matrix()
pums_full_new <- ((pums_full_matrix %*% rand_matrix) %% P) %% 2

# since I'm using derived attributes, population probs are all 0.5 and rescaled means are 0
population_probs <- rep(0.5, 10000)
population_means <- 2*(population_probs-0.5)

### functions ##
# 1. query mechanism w/ three defenses for purposes of membership attack
membershipQuery <- function(data, defense="round", defense_parameter) {
  # get column sums from data
  col_sums <- colSums(data)

  # perturb actual sum
  if(defense=="round") {
    col_sums_defended <- plyr::round_any(col_sums, defense_parameter)
    col_means_defended <- col_sums_defended/100
  }
  else if(defense=="noise") {
    col_sums_defended <- col_sums + rnorm(1, mean=0, sd=defense_parameter)
    col_means_defended <- col_sums_defended/100
  }
  else if(defense=="subsample") {
    inds <- sample(1:nrow(data), size=defense_parameter)
    subsample <- data[inds,]
    col_means_defended <- colMeans(subsample)
  }

  # convert means to between -1 and 1
  col_means_defended <- (col_means_defended-0.5)*2

  return(col_means_defended)
}
# 1. a. get sample means for each defense type
sample_means_round <- membershipQuery(pums_100_new, defense="round",
                                      defense_parameter=optimal_defense_params["round"][[1]])
sample_means_noise <- membershipQuery(pums_100_new, defense="noise",
                                      defense_parameter=optimal_defense_params["noise"][[1]])
sample_means_subsample <- membershipQuery(pums_100_new, defense="subsample",
                                          defense_parameter=optimal_defense_params["subsample"][[1]])
# 2. create data from 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
    history[i,] <- x
  }
  return(history)
}
# 3. test stat
test.Dwork <- function(alice, sample.mean, population.mean){
  test.statistic <- sum(alice * sample.mean) - sum(population.mean * sample.mean)
  return(test.statistic)
}
# 4. null distribution and critical value
nullDistribution <- function(null.sims=1000, alpha=0.05, fun, population.prob, sample.mean){
  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)
    hold[i] <- eval(fun(alice=nullAlice, sample.mean=sample.mean, population.mean=population.mean))
  }
  nullDistribution <- sort(hold, decreasing=TRUE)
  criticalValue <- nullDistribution[round(alpha*null.sims)]
  return(list(nullDist=nullDistribution, criticalVal=criticalValue))
}



###
# run attack
###

membership_history <- matrix(NA, nrow=(3*length(k.attributes)), ncol=3)
ind <- 1

for(k in k.attributes){
  for(defense in c("round","noise","subsample")) {
    # checkpoint
    print(paste0(defense, " ", k))

    # access appropriate noisy sample means
    if(defense=="round") { sample_means <- sample_means_round }
    else if(defense=="noise") { sample_means <- sample_means_noise }
    else if(defense=="subsample") { sample_means <- sample_means_subsample }

    # get critical value
    nullDist <- nullDistribution(alpha=alpha,
                                 fun=test.Dwork,
                                 population.prob=population_probs[1:k],
                                 sample.mean=sample_means[1:k])

    true_pos <- 0
    for(i in 1:nrow(pums_100_new)){
      # use new row as Alice
      alice <- pums_100_new[i,1:k]
      alice[alice==0] <- -1
      # conduct tests
      test.alice <- test.Dwork(alice=alice,
                               sample.mean=sample_means[1:k],
                               population.mean=population_means[1:k])

      # increment counter if we ID a true positive
      if(test.alice >= nullDist$criticalVal) {
        true_pos = true_pos + 1
      }
    }
    # store values
    membership_history[ind,]<- c(k, defense, true_pos/100)

    ind = ind + 1
  }
}

# save results
membership_history_df <- as.data.frame(membership_history)
names(membership_history_df) <- c("num_attributes","defense","tpr")
membership_history_df$num_attributes <- as.numeric(as.character(membership_history_df$num_attributes))
membership_history_df$tpr <- as.numeric(as.character(membership_history_df$tpr))
write.csv(membership_history_df, file="membership_history.csv")
# membership_history_df <- read.csv("membership_history.csv")

# plot results
q3_plot <- ggplot(membership_history_df, aes(num_attributes, tpr)) +
  geom_point() +
  facet_wrap(~defense, scales = "free") +
  labs(x="Number of attributes", y="True positive rate") + theme_bw()
pdf("q3_plot.pdf", width=8, height=8)
q3_plot
dev.off()
```