Untitled

Isaac Kleisle-Murphy

February 17, 2020

1.)

i.)

According to the paper, synthesized variable fields included `Payroll`, `Employment`, `Multiunit`, `Firstyear`, `Lastyear`, while `SIC` was left as is. Further, `County` (and by extension, `State`) was simply not released, while `Year` and `ID` were created and released. The paper also notes the existence of additional, confidential variables that were not used to generate this synthetic dataset, but may be used in future sets.

ii/iii.)

In order to simulate the multiple variables in play, Kinney et al. built a nested series of models – i.e. they imputed "outwards" from the categorical targets to the continuous targets. Importantly, the output of one synthesis could be an input for the next, thus giving the "nested" structure described above. More specifically, this structure involved

1.) Model category $Firstyear|County, SIC \sim Multinom$ with (flat) dirichlet priors, i.e. the multinomial-dirichlet conjugacy.

2.) Model category $Lastyear|Firstyear, County, SIC \sim Multinom$ with (flat) dirichlet priors, i.e. the multinomial-dirichlet conjugacy. Note that the output of 1 is an input here.

3.) Model category $Multiunit|Lastyear, Firstyear, County, SIC \sim Multinom$ with (flat) dirichlet priors, i.e. the multinomial-dirichlet conjugacy. Here, the outputs of 1 and 2 are inputs.

4.) Model continuous $Employment|Multiunit, Lastyear, Firstyear, County, SIC$ via a hybrid AR1/MLR model, with an added KDE smoothing transformation at the end. Note that the outputs of 1-3 were, in addition to the AR1's lag component, covariates in this linear regression.

5.) Model continuous $Payroll|Employment, Multiunit, Lastyear, Firstyear, County, SIC$ via a hybrid AR1/MLR model, with an added KDE smoothing transformation at the end. Note that the outputs of 1-4 were, in addition to the AR1's lag component, covariates in this linear regression.

As shown above, this model was built "outwards" from the categorical fits above. In this way, they were able to synthesize multiple variables at once. Notably, this puts principal importance on models 1-2, as the remainder of the models rely/are conditional on their accuracy.

iv.)

As noted on page 365, they generated a single synthetic entry for over 21 million records. So it appears that $m = 1$.

v.)

Annual comparisons of each synthetic marginal values/summary stats with the true marginal values/summary stats was one way the authors measured utility: for instance, they remarked that between 1976 and 2000, synthetic `Employment` and true `Employment` had an average 1.3% discrepancy (think MAE, converted to percentage) over those years, while synthetic `Payroll` and true `Payroll` had an average 8% discrepancy over those years. Further, the authors note that within non-synthesized categories of location and industry, employments, establishments, and payrolls generally aligned with the true summary statistics in these categories. This finding appears to have been made via visual inspection/comparison of group-wise scatterplots for these variables (similar to that in question 2).

Further, the authors also corroborated the accuracy of their synthesis by comparing transformations of the synthetic data with transformations of the true data. For instance, job creation/destruction metrics can be backed out of year-to-year employment figures, so the authors calculated these measures for the true data and then calculated the same measures for their own data. To their credit, these transformations aligned closely, although the synthetic data consistently showed a lower job creation rate than the true data. In a similar manner, the authors also plugged both the synthesic data and the true data into a pre-fit linear economic growth regression, before comparing the performances of the two. Again, the synthetic data exhibited similar trends to that of the true data, when plugged into the model, further illustrating the robustness of the synthetic data.

Though their analysis of utility/accuracy was fairly exhaustive, other utility measures could have included:

- correlations between the various transformations of the two datasets (i.e. job flow metrics, the growth regression), as well as correlations for the plots that were subject to visual inspection.
- MAE estimates not just for marginal rates (i.e. one variable in a vacuum), but perhaps MAEs/percentage discrepancies when faceted over one or two additional covariates. This reporting would, of course, be subject to it not disclosing individual entries.
- A brief case study in how small perturbations in the early components of the nested model (such as `Firstyear`, `Lastyear`), that is those outputs most foundational to the overall model, would have affected accuracy. This could have spoken to the structural "stability" of the model.

vi.)

Due to a paucity of matching algorithms/appropriate software, the authors could not have a computer comb through the synthetic data and pick out individuals. As such, the authors analyzed disclosure risks in the following ways. First, they analyzed the probability that true birth/death years (`Firstyear`, `Lastyear`, respectively) matched those of the synthetic – in most cases, this probability was low, as the synthetic value was close but not exactly equal. This alone, they argue, "confounded" reidentification substantially.

Second, the authors also point to the KDE transform of the continuous variables as an additional buffer against reidentification – that is, a single vector of covariates cannot be "tracked" through the data, as the smoother obfuscates this vector's potential uniqueness.

Third, the authors also note that while the overall shape of the synthetic data matches that of the true data, correlations between the value of each entry are minimal. This also speaks to limited disclosure risks.

Fourth, as discussed on the authors found that outliers in the true data rarely align with the outliers in the synthetic data: they found that in fewer than 5% of cases, the maximum synthetic employment value corresponded to the maximum true employment value.

Fifth, the authors applied more complex algorithms, which compared year-to-year transition probabilities, to see if year-to-year behavior in the synthetic dataset aligned with year-to-year behavior in the true dataset. It did not – in fact, even if the intruder knew pretty much everything about the true and synthetic data, they might still have trouble.

This disclosure risk analysis seemed pretty robust to me, and with little knowledge of the field, I don't have too many additional disclosure risk tests to add. However, I think there is one non-mathematical check that would be worthwhile: take the synthetic data to the accounting departments/C-Suites of the 10-20 most "vulnerable" companies (provided they can be contacted), and see if they can identify themselves in the synthetic data. If most cannot, then identities are likely secure.

2.)

i.)

As the authors briefly describe, analysis-specific measures have their pros and cons. Among the pros: they lend themselves well to particularized and detailed analyses, allowing statisticians to "get in the weeds" of whether a masked data set really resembles the true data with regard to some property. However, while this is true for those particularized an detailed analyses, it may not be helpful – Woo notes it may even be harmful – for other analyses. So if there's one deeply-nuanced aspect of a masked data set that requires analysis, analysis-specific measures may be one's best bet.

ii.)

Conversely, global utility measures are better for broader analyses; generally, they tend to focus on comparison summary statistics/parameter estimates across the entire synthetic and true data sets. As Woo characterizes, they are "broad yet blunt," reporting on the general distributional trends/similarities across data sets. Thus, for the more 10,000 foot level analysis of a masked data set's utility – i.e. whether it generally obfuscates identities, reflects true distributions, etc. The downside here is that this broader analysis may overlook nuances in the data, such as poorly preserved relationships between subcategories of the data.

iii.)

Fundamental similarities between the proposed measures include the following: - all rely on some pseudo-classification function (logistic regression, clustering, or a true/false less than) that attempts to take a row of the merged data and position it (whether in euclidian space, classification probability, or on a number line) in relation to both masked and true observations - Under these metrics, these pseudo-classification functions should have a difficult time discerning the differences between unmasked and masked entries in the merged dataset, and should accordingly position them in an integrated manner (i.e. it can't tell the difference between masked and unmasked). In other words equal probability of classification and/or placement is a good thing. - All are fairly simple in their computation.

iv.)

Before applying any of Woo's utility measures, I recreate my (admittedly poor) synthetic dataset, as proffered in Lab 2. This code has been directly cut/pasted from my Lab 2 submission.

```r
suppressMessages(require(dplyr))
suppressMessages(require(ggplot2))
suppressMessages(require(runjags))
suppressMessages(require(coda))
suppressMessages(require(tidyr))
suppressMessages(require(fastDummies))
options(warn = -1)
setwd("~/Documents/Swat_2020/Data_Privacy/Data-Confidentiality/datasets")
```

```r
CEsample = read.csv("CEsample.csv")%>%
  mutate(logInc = log(TotalIncomeLastYear),
         logExp = log(TotalExpLastQ))


the_data = list("logInc" = CEsample$logInc,
                "logExp" = CEsample$logExp,
                "r" = CEsample$Race,
                "R" = max(CEsample$Race),
                "u" = CEsample$UrbanRural,
                "U" = max(CEsample$UrbanRural),
                "N" = nrow(CEsample),
                "mu_b0" = 0,
                "mu_b1" = 0,
                "prec_b0" = 1,
                "prec_b1" = 1
                )


the_formula = "

model{

#model
for (i in 1:N){
  logInc[i] ~ dnorm(B_0[r[i], u[i]] + B_1[r[i], u[i]]*logExp[i], inv_sigma_sq[r[i]])
}

#priors
for (rc in 1:R){
  for (ur in 1:U){
    B_0[rc, ur] ~ dnorm(b_0, tau_sq_0)
    B_1[rc, ur] ~ dnorm(b_1, tau_sq_1)
  }
  inv_sigma_sq[rc] ~ dgamma(1,1)
  sigma[rc] <- sqrt(pow(inv_sigma_sq[rc], -1))
}

#hyperpriors
b_0 ~ dnorm(mu_b0,prec_b0)
b_1 ~ dnorm(mu_b1,prec_b1)
tau_sq_0 ~ dgamma(1,1)
tau_sq_1 ~ dgamma(1,1)
}

"




initsfunction <- function(chain){
  .RNG.seed <- c(1,2)[chain]
  .RNG.name <- c("base::Super-Duper",
                 "base::Wichmann-Hill")[chain]
  return(list(.RNG.seed=.RNG.seed,
              .RNG.name=.RNG.name))
}


posterior_jags  <- run.jags(the_formula,
                            n.chains = 2,
                            data = the_data,
                            monitor = c("B_0", "B_1", "sigma"),
                            adapt = 1000,
                            burnin = 5000,
                            sample = 2500,
                            thin = 100,
                            inits = initsfunction)
```

```
## Calling the simulation...
## Welcome to JAGS 4.3.0 on Tue Feb 18 13:52:09 2020
## JAGS is free software and comes with ABSOLUTELY NO WARRANTY
## Loading module: basemod: ok
## Loading module: bugs: ok
## . . Reading data file data.txt
## . Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 994
##    Unobserved stochastic nodes: 34
##    Total graph size: 6019
## . Reading parameter file inits1.txt
## . Reading parameter file inits2.txt
## . Initializing model
## . Adaptation skipped: model is not in adaptive mode.
## . Updating 5000
## -------------------------------------------------| 5000
## ************************************************** 100%
## . . . . Updating 250000
## -------------------------------------------------| 250000
## ************************************************** 100%
## . . . . . Updating 0
## . Deleting model
## .
## Note: the model did not require adaptation
## Simulation complete.  Reading coda files...
## Coda files loaded successfully
## Calculating summary statistics...
## Calculating the Gelman-Rubin statistic for 30 variables....
## Finished running the simulation
```

```
#extract, store in dictionary for quicker retrieval
posterior_df = data.frame(as.mcmc(posterior_jags))
posterior_list = lapply(colnames(posterior_df), function(x) posterior_df%>%pull(x))%>%
  `names<-`(colnames(posterior_df))
```

```r
synth_helper <- function(posterior_list, df, ii = NULL){

  #randomly select posterior draw to use

  if (is.null(ii)){
    ii = sample(1:nrow(df), 1)
  }


  beta_suffixes = paste0(".", df$Race, ".", df$UrbanRural, ".")
  sigma_suffixes = paste0(".", df$Race, ".")

  b0 = sapply(beta_suffixes, function(x) posterior_list[[paste0("B_0", x)]][ii])%>%as.vector()
  b1 = sapply(beta_suffixes, function(x) posterior_list[[paste0("B_1", x)]][ii])%>%as.vector()


  mu_post = b0+b1*df%>%pull(logExp)
  sig_post = sapply(sigma_suffixes, function(x) posterior_list[[paste0("sigma", x)]][ii])%>%as.vector()

  result = rnorm(nrow(df), mu_post, sig_post)

  return(result)
}


synth_logInc <- function(posterior_list, df, m=1, ...){

  set.seed(4*m-1)
  sample_idx = sample(1:length(posterior_list[[1]]), m, replace = F)

  lapply(sample_idx,
         function(y)
           synth_helper(posterior_list, df, ...)
         )%>%
    return()
}



synth_single = synth_logInc(posterior_list, CEsample, m = 1)

synth_single_df = data.frame(logInc_true = CEsample$logInc,
                             logInc_synth = unlist(synth_single),
                             logExp = CEsample$logExp,
                             Race = as.character(CEsample$Race),
                             UrbanRural = as.character(CEsample$UrbanRural))
```

Now, we're ready to measure global utility. First, I code up a function to measure propensity score:

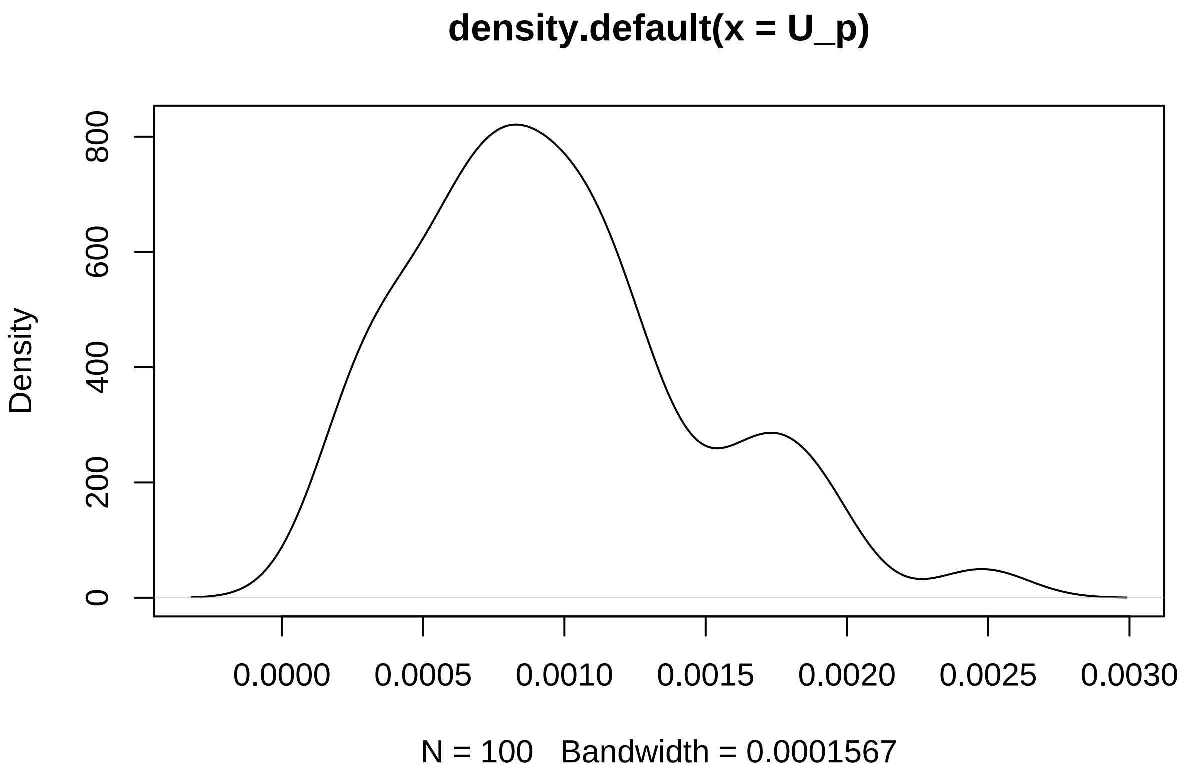
```r
propensity_util_score <- function(synth_df, true_df, order = 1, c = .5){

  propensity_df = bind_rows(synth_df, true_df)
  propensity_df$logInc = scale(propensity_df$logInc)
  propensity_df$logExp = scale(propensity_df$logExp)
  frm = as.formula(paste0("t ~ .", ifelse(order>1, paste0("^", order), "")))
  clf_logistic = glm(frm, data = propensity_df, family = "binomial")

  p_hats = predict(clf_logistic, newdata = propensity_df, type = "response")%>%
    as.vector()%>%
    suppressMessages()

  return(mean((p_hats - c)^2))

}
```

We use this function to calculate U_p for our synthetic draw from the last lab. For now, we'll stick to first order interaction (lest we grossly overfit, thus wrongly suppressing propensity score) We have (note that one of the race-urban interactions is missing, thus R barks at us):

By this utility measure, the synthetic draw performed decently, as $U__p = $0.0013119. However, this was only one draw – let's see the distribution of U_p across multiple draws, say $m = 100$:

```
U_p; plot(density(U_p))
```

```
##   [1] 0.0005892393 0.0008622214 0.0005656067 0.0009244874 0.0002589116
##   [6] 0.0011765733 0.0016698717 0.0003370791 0.0007232041 0.0010450372
##  [11] 0.0009201082 0.0016024442 0.0015158788 0.0003742548 0.0008390168
##  [16] 0.0007069938 0.0018267172 0.0012925013 0.0006550338 0.0007656920
##  [21] 0.0015842589 0.0012919191 0.0008813562 0.0011833120 0.0013486908
##  [26] 0.0011538800 0.0002212777 0.0010219842 0.0016608600 0.0012795836
##  [31] 0.0007827266 0.0019653092 0.0017331082 0.0005516135 0.0011780048
##  [36] 0.0010278256 0.0017405242 0.0018867536 0.0010062391 0.0008416203
##  [41] 0.0007898800 0.0024397456 0.0019181614 0.0006253454 0.0006059449
##  [46] 0.0009087830 0.0003039932 0.0008998767 0.0003973688 0.0001499587
##  [51] 0.0010968814 0.0005062605 0.0019140404 0.0008144119 0.0011917129
##  [56] 0.0025196947 0.0012750101 0.0006350987 0.0010234250 0.0003823654
##  [61] 0.0006011239 0.0006961600 0.0011021515 0.0005404458 0.0006517225
##  [66] 0.0002531765 0.0018825982 0.0013804739 0.0001866535 0.0008776915
##  [71] 0.0003581277 0.0010670838 0.0010845655 0.0008713125 0.0009245002
##  [76] 0.0002573063 0.0007968398 0.0013799000 0.0006779865 0.0010675141
##  [81] 0.0006353432 0.0010039420 0.0008936339 0.0006786529 0.0011130817
##  [86] 0.0017361299 0.0008002874 0.0004121349 0.0003947695 0.0016920854
##  [91] 0.0005599381 0.0003979205 0.0007602015 0.0003158842 0.0005137384
##  [96] 0.0011903601 0.0002007482 0.0011580299 0.0010830212 0.0007707665
```



Indeed, we see that in taking multiple synthetic draws, we still obtain decent U_p measures.

We can also repeat the above analysis, but with higher-order interaction in the logistic regression. However, this might lead to overfit, incorrectly suppressing our propensity measure when we "recycle" the stacked data.

I cannot find Woo's precise clustering algorithm, so for ease of fitting I will use K-means here. Just as Woo did, I will try a variety of cluster centers, ranging from 5-200. Finally, for this trial run of the method, each cluster will be weighted equally, though this is certainly an area for more in-depth exploration in future analyses.

```r
cluster_utility_score <- function(g, true_df, synth_df){

  df_dummied = bind_rows(true_df, synth_df)%>%
    fastDummies::dummy_cols(., c("UrbanRural", "Race"), remove_first_dummy = T)%>%
    mutate_if(is.numeric, scale)
  df_dummied$logExp = scale(df_dummied$logExp); df_dummied$logInc = scale(df_dummied$logInc)
  cl_obj = kmeans(df_dummied%>%dplyr::select(-t), centers = g, iter.max = 500)

  df_dummied$cl = cl_obj$cluster

  group_analyze = df_dummied%>%
    group_by(cl)%>%
    summarise(pct_synth = sum(t)/n(),
              n = n())

  return(mean((group_analyze$pct_synth - .5)^2))

}
```

In performing this cluster analysis for our single synthetic draw, and over clusters $G = \{5, 10, 20, 25, 100, 200\}$,

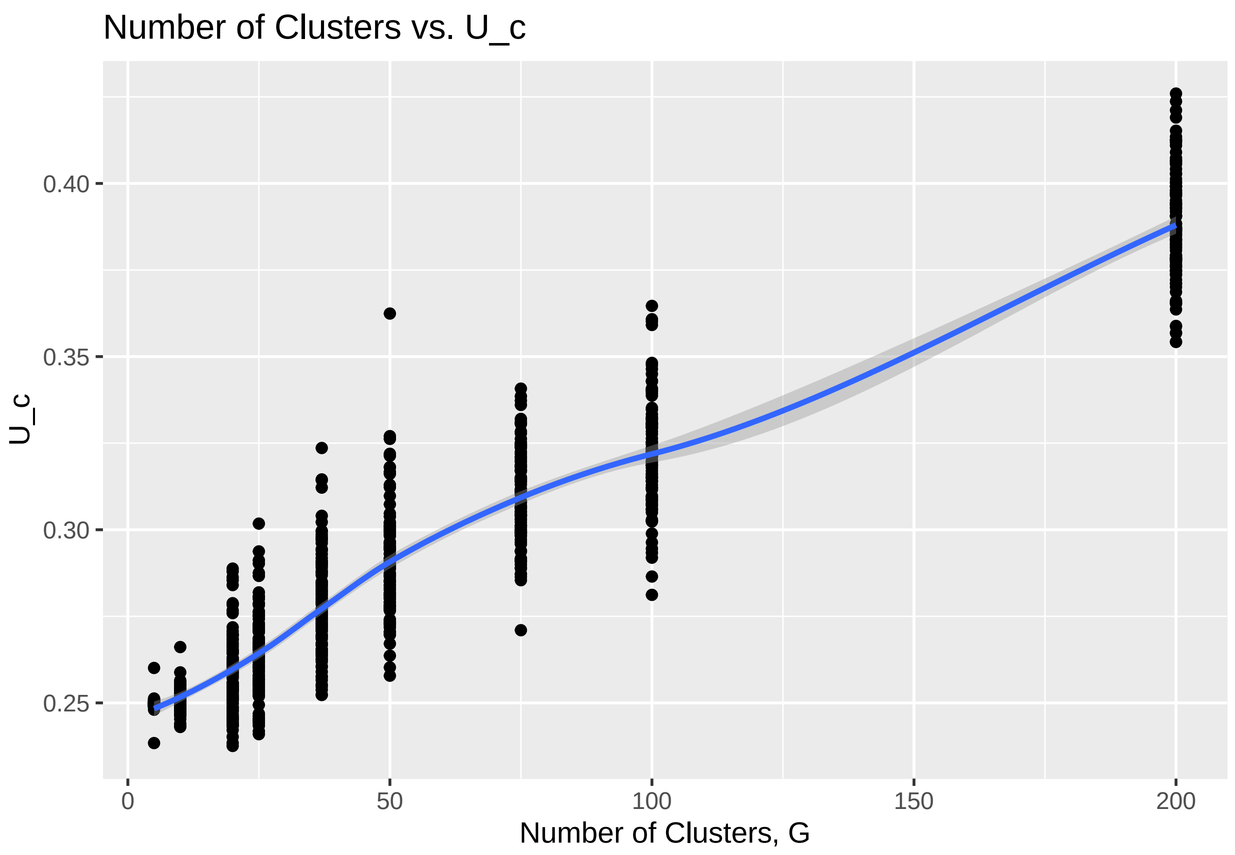
```r
g_seq = c(5, 10, 20, 25, 37, 50, 75, 100, 200)

true_df = synth_single_df[, c("logInc_true", "logExp", "Race", "UrbanRural")]%>%
  rename(logInc = logInc_true)%>%
  mutate(t = 0)

synth_df = synth_single_df[, c("logInc_synth", "logExp", "Race", "UrbanRural")]%>%
  rename(logInc = logInc_synth)%>%
  mutate(t = 1)

sapply(g_seq, cluster_utility_score, true_df, synth_df)
```

```
## [1] 0.2496279 0.2468425 0.2595674 0.2820568 0.2634312 0.2819601 0.3300353
## [8] 0.3053889 0.3918194
```

we achieve U_c's all less than .05 – again, a pretty good scores, though these increase noticably as the clusters become increasingly granular. Further, in repeating this for the $m = 100$ synthetic draws, this performance appears non-anomalous, as we have

```r
synth_100 = synth_logInc(posterior_list, CEsample, m = 100)
U_c_mat = matrix(rep(NA, 100*length(g_seq)), ncol = length(g_seq))

for (ss in 1:length(synth_100)){

  synth_single_df = data.frame(logInc_true = CEsample$logInc,
                               logInc_synth = synth_100[[ss]],
                               logExp = CEsample$logExp,
                               Race = as.character(CEsample$Race),
                               UrbanRural = as.character(CEsample$UrbanRural))

  true_df = synth_single_df[, c("logInc_true", "logExp", "Race", "UrbanRural")]%>%
  rename(logInc = logInc_true)%>%
  mutate(t = 0)

  synth_df = synth_single_df[, c("logInc_synth", "logExp", "Race", "UrbanRural")]%>%
    rename(logInc = logInc_synth)%>%
    mutate(t = 1)

  U_c_mat[ss, ] = sapply(g_seq, cluster_utility_score, true_df, synth_df)
}

U_c_df = U_c_mat%>%
  data.frame()%>%
  `colnames<-`( g_seq)%>%
  gather(G, U_c)%>%
  mutate(G = as.numeric(as.character(G)))

ggplot(U_c_df, aes(x = G, y = U_c))+
  geom_point()+
  stat_smooth() +
  labs(title = "Number of Clusters vs. U_c", x = "Number of Clusters, G", y = "U_c")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Indeed, even when taking multiple synthetic samples, and calculating the U_c score for each, the U_c scores are largely low, stable, and rising as G increases.

Finally, we apply the empirical CDF method to the synthetic `logInc` data, both for the single synthetic draw and the 100 draws.

```r
empirical_utility_score <- function(true_df, synth_df, var = "logInc"){

  Sx = sapply(c(true_df%>%pull(var), synth_df%>%pull(var)),
            function(x) sum(true_df%>%pull(var)<=x)/length(true_df%>%pull(var)))

  Sy = sapply(c(true_df%>%pull(var), synth_df%>%pull(var)),
            function(x) sum(synth_df%>%pull(var)<=x)/length(synth_df%>%pull(var)))

  Um = max(abs(Sx - Sy)); Us = mean((Sx - Sy)^2)

  return(c(Um, Us))
}
```

For the single draw, we get:

```r
true_df = synth_single_df[, c("logInc_true", "logExp", "Race", "UrbanRural")]%>%
  rename(logInc = logInc_true)

synth_df = synth_single_df[, c("logInc_synth", "logExp", "Race", "UrbanRural")]%>%
    rename(logInc = logInc_synth)

emp_cdf_score1 = empirical_utility_score(true_df, synth_df, var = "logInc")
```

For the synthetic sample, we get \$U_c = \$ 0.056338 and \$U_s = \$ 0.0010802 – both desirable scores. For the 100 draws, the results are similar:

```r
synth_100 = synth_logInc(posterior_list, CEsample, m = 100)
U_mat = matrix(rep(NA, 200), ncol = 2)

for (ss in 1:length(synth_100)){

  synth_single_df = data.frame(logInc_true = CEsample$logInc,
                               logInc_synth = synth_100[[ss]],
                               logExp = CEsample$logExp,
                               Race = as.character(CEsample$Race),
                               UrbanRural = as.character(CEsample$UrbanRural))

  true_df = synth_single_df[, c("logInc_true", "logExp", "Race", "UrbanRural")]%>%
    rename(logInc = logInc_true)

  synth_df = synth_single_df[, c("logInc_synth", "logExp", "Race", "UrbanRural")]%>%
    rename(logInc = logInc_synth)

  U_mat[ss, ] = empirical_utility_score(true_df, synth_df, var = "logInc")

}
```

In looking at its derivation for one variable, this utility measure bears considerable similarity to a two-sided K-S test. However, their difference is subtle: whereas a proper K-S test would calculate the ecdfs by taking the max (supremum) distance between the ecdfs, the U_c method takes the max/supremum distance between the ecdfs evaluated at each point in the merged data. In this way, the two are similar in that it's a distance between ecdfs – however, what differs is the vector being fed into the ecdf.

Nevertheless, this also suggests that for a synthetic and true data pairing, perhaps a regular old two-sample K-S test could be an additional global utility measure. Like with any such utility measure, it would undoubtedly have its shortcomings, but it would provide another "broad but blunt" description of distributional difference for a synthetic and true pairing.

v.)

Of the proposed global utility measures, I have the most reservation about the clustering method, simply because of the hyperparameter tuning involved. At a gut level, a global utility measure is something that shouldn't require tuning or semi-arbitrary decisions – that is, it ought to be independent, empirical, and non-malleable (to the greatest degree possible). However here, the hyperparameterization of G undercuts this empiricism – in my mind, a measure that depends in large part on the user's "input setting" may not be best for "broad but blunt," apples-to-apples comparisons. If we don't even know what the optimal G is, how can we let the clusters tell us what the best synthesis is.

To a lesser extent, the tuning required for the logistic regression also bothers me. By excessively increasing the order and interaction of the regression, one could grossly overfit the model. Then, once the fitting data was recycled into the model, in order to compute propensity score, this overfit would be disguised (i.e. not held accountable to a test dataset). However, this sort of tuning ambiguity can be better mitigated. For example, the logistic regression can be fit up to first order interaction, or, for sufficiently large datasets, the merged data can be split (stratified) into training and testing, allowing for more nuanced feature selection methods (such as lasso or ridge). In this way, the logistic regression/propensity score approach is more viable. Particularly on large data sets, I think it is a good measure of global utility.

Finally, I also like the empirical CDF methods, U_c and U_s. Yes, as Woo acnowledges, the synthetic and true ecdfs may often align closely, but the unambiguous (empirical – requires little human "instruction") nature of the measure is appealing. What's more, the empirical approach effectively amounts to a 2-sided K-S test, a well-respected test for differences in distributions. Given that the ultimate objective of global utility is to measure overall differences in distribution, this is a good thing.

3.)

Through its open-data portal, the City of Seattle (where I'm from) releases all sorts of data, from public safety data to social services data. One particularly vulnerable dataset on this portal is its client level "Aging and Disability Services" dataset, which includes racial, socioeconomic, and location-related information about people on elderly and/or disability services in Seattle. Let's take an exploratory look at the dataset (using 2016 data)

```r
setwd("~/Downloads")
ageDisData <- read.csv("Aging_and_Disability_Services_-_Client_Level_Data_2016.csv")

summary(ageDisData)
```

```
##    ActivityID         ClientID                              GeographicLocation
##  Min.   :4088027   Min.   :   138   South Urban                       :119159
##  1st Qu.:4172232   1st Qu.: 83780   East Urban                        : 38278
##  Median :4264712   Median :130661   Seattle Neighborhoods: SE Seattle: 32236
##  Mean   :4267973   Mean   :122778   Seattle Neighborhoods: Downtown  : 24839
##  3rd Qu.:4365910   3rd Qu.:168510                                     : 20937
##  Max.   :4448151   Max.   :204776   North Urban                       : 20740
##  NA's   :10        NA's   :10       (Other)                           : 72064
##      AgeRange      EthnicityCode    RaceCode       IncomeCode     LiveAlone
##  75 to 79:47690    :   449      Min.   :0.000   Min.   :0.000    :   497
##  70 to 74:45528   n: 26483      1st Qu.:2.000   1st Qu.:1.000   0:    10
##  80 to 84:44064   N:222136      Median :3.000   Median :1.000   n:     3
##  65 to 69:41838   u: 23654      Mean   :3.901   Mean   :1.071   N:192975
##  85 to 89:32884   U: 35674      3rd Qu.:6.000   3rd Qu.:1.000   U: 12829
##  60 to 64:28112   y:  3836      Max.   :8.000   Max.   :8.000   y:    15
##  (Other) :88137   Y: 16021      NA's   :10      NA's   :10      Y:121924
##  LimitedEnglish              Language      NutritionalRisk  SingleParent
##   :   513                       :142512          :185124    :    10
##  0:    10      English          : 73703    U      : 61722    : 96901
##  n:    15      Russian          : 17549    N      : 58853   N:186461
##  N:174702      Spanish          : 12827    Y      : 22241   U: 41949
##  U: 15483      Chinese-Cantonese: 12494    1      :   163   Y:  2932
##  Y:137530      Ukranian         :  7039    2      :    32
##                (Other)          : 62129    (Other):   118
##  HouseholdWithChildren Homeless   DisabilityStatus Unincorporated
##   :    10               :    10    :    10           :    10
##   : 61590               : 67986    :   703           : 67433
##  N:181890              N:240570   -:     5          N:184867
##  U: 54055              U: 13915   N:101478          U: 67693
##  Y: 30708              Y:  5772   U: 29652          Y:  8250
##                                   Y:196405
##
##  NumberofChildren   RelationshipToRecipientCode Kinship     Veteran
##  Min.   : 0.00000   Min.   :0.0000               :    10     :    10
##  1st Qu.: 0.00000   1st Qu.:0.0000              :303275     :   465
##  Median : 0.00000   Median :0.0000             0:    39    N:233394
##  Mean   : 0.02379   Mean   :0.2535             N: 21350   U: 76636
##  3rd Qu.: 0.00000   3rd Qu.:0.0000             U:    25   Y: 17748
##  Max.   :10.00000   Max.   :9.0000             Y:  3554
##  NA's   :20         NA's   :20
##  Eating     Toileting  Walking    GettingPlaces Transferring Dressing
##   :    10    :    10    :    10    :    10       :    10      :    10
##   :142642    :142733    :142501    :232806       :191103      :142671
##  N: 67053   N: 44853   N: 17266   N: 21758      N: 34940     N: 31492
##  U: 48275   U: 48297   U: 49608   U: 61202      U:   212     U: 48292
##  Y: 70273   Y: 92360   Y:118868   Y: 12477      Y:101988     Y:105788
##
##
##  Bathing    MedicalManagement Cooking       Shopping       Chores
##   :    10          :143101        :    10       :    10       :    10
##   :142642   Y      :103177        :142647       :142576       :142590
##  1:    10   U      : 48327    2016:    10   1034:     1   2616:     9
##  N: 26006   N      : 33628    N   : 22072   872 :     9   2693:     1
##  U: 48445          :    10    U   : 48502   N   :  7308   N   : 20855
##  Y:111140   1      :     1    Y   :115012   U   : 48445   U   : 48546
##             (Other):     9                  Y   :129904   Y   :116242
##  Driving     HeavyHousework   Phoning    MoneyManagement   DivisionID
##    :    10          :226110    :    10        :    10    Min.   :   1.00
##    :142598   U      : 60304    :142557        :193786    1st Qu.:   1.00
##  11:     1   N      : 26687   1:     9   Case :     9    Median :   1.00
##  19:     9   Y      : 15132   8:     1   Meals:     1    Mean   :   1.12
##  N : 23164          :    10   N: 51607   N    : 54298    3rd Qu.:   1.00
##  U : 49335   134    :     7   U: 49818   U    :  2406    Max.   :4036.00
##  Y :113136   (Other):     3   Y: 84251   Y    : 77743    NA's   :10
##   ServiceMonth     ServiceYear      AgencyID          SiteID
##  Min.   : 1.000   Min.   :2016   Min.   : 871.0   Min.   :2600
##  1st Qu.: 4.000   1st Qu.:2016   1st Qu.: 880.0   1st Qu.:2616
##  Median : 6.000   Median :2016   Median : 899.0   Median :2654
##  Mean   : 6.494   Mean   :2016   Mean   : 923.3   Mean   :2659
##  3rd Qu.: 9.000   3rd Qu.:2016   3rd Qu.: 944.0   3rd Qu.:2691
##  Max.   :12.000   Max.   :2016   Max.   :1210.0   Max.   :2799
##  NA's   :20       NA's   :20     NA's   :20       NA's   :20
##  ServiceAreaID    ServiceTypeID    UnitsProvided       UnitsProvidedType
##  Min.   :  8.00   Min.   : 12.00   Min.   :  0.000   Case        :98094
##  1st Qu.: 11.00   1st Qu.: 31.00   1st Qu.:  1.000   Meals       :93877
##  Median : 19.00   Median : 57.00   Median :  1.000   Activity    :58935
##  Mean   : 22.82   Mean   : 73.82   Mean   :  4.243   Contact     :47582
##  3rd Qu.: 19.00   3rd Qu.:134.00   3rd Qu.:  3.000   One-way Trip:10347
##  Max.   :138.00   Max.   :159.00   Max.   :987.000   Session     : 8129
##  NA's   :20       NA's   :20       NA's   :20        (Other)     :11289
##    ContractID
##  Min.   :3215
##  1st Qu.:3806
##  Median :3851
##  Mean   :3889
##  3rd Qu.:3946
##  Max.   :4060
##  NA's   :20
```

Clearly, this is an expansive dataset, and the disclosure risks are obvious. With knowlege of only a few highly-descriptive variables – for example veteran status, neighborhood, race, and number of children – an individual could plausibly be identified in the data. This could be harmful for a number of reasons: for one, it could embarass the individuals who are on city services. Additionally, by identifying them specifically within an already vulnerable population, it could make them a target for scammers or other nefarious actors.

Given the amount of information contained in this data (everything from the individual's ability to dress themself to their eduction), there are countless ways to demonstrate and/or quantify disclosure risks. For example, we could write a script (as we did in the first lab) that scans across all permutations of variables, and tags particularly identifiable/unique subsets of the population based on these combinations. Alternatively, we could scrutinise joint density plots in order to identify outliers, or look at class imbalances to identify vulnerable traits.

However, to quantify/demonstrate the disclosure risks in this data, I also want to propose my own , basic Euclidian distance, to quantify an individual's uniqueness within the dataset. Specifically, for each response vector, I will measure the distance of that response vector to all other response vectors, to identify "nearby" or similar responses. Those with few similar responses, it follows, are most vulnerable, while those with many are least.

More precisely, this Euclidian distance goes as follows: 1.) I take all variables of interest (detailed below), 2.) convert categorical variables to dummies (drop the first level), and 3.) compute the Euclidian distance of one entry to that of another. For one entry, compare the distances to all rows, and then examine summary statistics of all these distances to get a sense of the entry of interest's overall distance (and by extension, uniqueness) from the rest of the data.

Before computation, a bit of data wrangling.

```r
code_vars = c("GeographicLocation", "AgeRange", "RaceCode", "IncomeCode", "Language", "AgencyID")
ynu_vars = c("Veteran","LiveAlone", "MedicalManagement", "Driving")

ageDisData_wr = ageDisData%>%
  filter(grepl("Seattle Neighborhoods", GeographicLocation))%>%
  mutate_at(code_vars, function(x)
      ifelse(x == ""|x==" ", 0, x))%>%
  mutate_at(ynu_vars, function(x)
      ifelse(x ==" "|x=="", "U", toupper(x)))%>%
  dplyr::select(c("ClientID", code_vars, ynu_vars))

ageDisData_du = ageDisData_wr%>%
  dplyr::select(-ClientID)%>%
  fastDummies::dummy_cols(., c(code_vars, ynu_vars), remove_first_dummy = T, remove_selected_columns = 
  as.matrix()
```

As encoded above, the variables included in similarity measure are coded variables `code_vars = c("GeographicLocation", "AgeRange", "RaceCode", "IncomeCode", "Language", "AgencyID")` and pseudo-binary (includes an unknown case) `ynu_vars = c("Veteran","LiveAlone", "MedicalManagement", "Driving")`. Certainly, we could include a whole host of other variables in future analyses, but this is probably a good starting point.

Importantly, I also restricted the data to people just in "Seattle Neighborhoods", as defined by the dataset. This was to ensure a.) reasonable run times and b.) a focus on the immediate Seattle area.

```r
dist_helper <- function(i){
  sapply(1:nrow(ageDisData_du),
         function(x){sum((ageDisData_du[i, ] - ageDisData_du[x, ])^2, na.rm = T)^.5})->result
  return(summary(result))
}

suppressMessages(library(doParallel))
```

```r
cl <- makeCluster(4)
registerDoParallel(cl)
set.seed(2020)
#compute 100 distances, as an example
distances = foreach(i=1:100) %dopar% {dist_helper(i)}
stopCluster(cl)

distance_df = distances%>%
  do.call("rbind", .)%>%
  data.frame()%>%
  arrange(desc(X1st.Qu.))

distance_df%>%head(100)
```

```
##      Min. X1st.Qu.   Median     Mean X3rd.Qu.     Max.
## 1       0 3.000000 3.464102 3.258017 3.605551 3.872983
## 2       0 3.000000 3.464102 3.258017 3.605551 3.872983
## 3       0 3.000000 3.464102 3.258017 3.605551 3.872983
## 4       0 3.000000 3.316625 3.207168 3.464102 3.872983
## 5       0 3.000000 3.316625 3.226458 3.464102 4.000000
## 6       0 3.000000 3.316625 3.237020 3.464102 4.000000
## 7       0 3.000000 3.162278 3.136451 3.316625 3.872983
## 8       0 3.000000 3.464102 3.283194 3.605551 3.872983
## 9       0 3.000000 3.464102 3.289806 3.605551 4.123106
## 10      0 2.828427 3.162278 3.090789 3.464102 3.741657
## 11      0 2.828427 3.316625 3.176519 3.464102 3.872983
## 12      0 2.828427 3.162278 3.074228 3.316625 3.741657
## 13      0 2.828427 3.162278 3.059147 3.316625 3.872983
## 14      0 2.828427 3.316625 3.190389 3.464102 4.000000
## 15      0 2.828427 3.162278 3.013419 3.316625 3.872983
## 16      0 2.828427 3.162278 3.013419 3.316625 3.872983
## 17      0 2.828427 3.316625 3.114939 3.464102 3.872983
## 18      0 2.828427 3.162278 3.024820 3.316625 3.872983
## 19      0 2.828427 3.316625 3.174808 3.464102 3.872983
## 20      0 2.828427 3.316625 3.174808 3.464102 3.872983
## 21      0 2.828427 3.316625 3.117348 3.464102 3.872983
## 22      0 2.828427 3.162278 3.063831 3.316625 4.000000
## 23      0 2.828427 3.162278 3.113717 3.464102 3.741657
## 24      0 2.828427 3.316625 3.153491 3.464102 4.000000
## 25      0 2.828427 3.162278 3.095946 3.464102 3.741657
## 26      0 2.828427 3.000000 3.032171 3.316625 4.000000
## 27      0 2.828427 3.000000 3.032171 3.316625 4.000000
## 28      0 2.828427 3.000000 2.980581 3.162278 3.605551
## 29      0 2.828427 3.000000 2.987048 3.316625 3.741657
## 30      0 2.828427 3.316625 3.154703 3.464102 4.000000
## 31      0 2.828427 3.162278 3.066343 3.464102 3.872983
## 32      0 2.828427 3.316625 3.108867 3.464102 3.872983
## 33      0 2.828427 3.316625 3.116864 3.464102 3.741657
## 34      0 2.828427 3.162278 3.091505 3.464102 3.741657
## 35      0 2.828427 3.162278 3.119814 3.464102 3.872983
## 36      0 2.828427 3.316625 3.099290 3.464102 3.872983
## 37      0 2.828427 3.162278 3.112117 3.316625 3.872983
## 38      0 2.828427 3.162278 3.112117 3.316625 3.872983
## 39      0 2.828427 3.000000 2.920263 3.000000 3.741657
## 40      0 2.828427 3.000000 2.920263 3.000000 3.741657
## 41      0 2.828427 3.162278 3.094728 3.464102 3.741657
## 42      0 2.828427 3.316625 3.127204 3.464102 4.000000
## 43      0 2.828427 3.316625 3.155741 3.464102 4.000000
## 44      0 2.828427 3.162278 3.084547 3.464102 3.872983
## 45      0 2.828427 3.162278 3.092020 3.464102 3.872983
## 46      0 2.828427 3.316625 3.107347 3.464102 3.872983
## 47      0 2.828427 3.316625 3.136406 3.464102 3.741657
## 48      0 2.828427 3.162278 3.076373 3.464102 3.872983
## 49      0 2.828427 3.316625 3.116864 3.464102 3.741657
## 50      0 2.828427 3.316625 3.110846 3.464102 3.872983
## 51      0 2.828427 3.162278 3.048428 3.316625 3.872983
## 52      0 2.828427 3.316625 3.108867 3.464102 3.872983
## 53      0 2.828427 3.162278 3.110849 3.464102 4.000000
## 54      0 2.828427 3.162278 3.115838 3.464102 3.741657
## 55      0 2.645751 3.000000 2.861283 3.162278 3.741657
## 56      0 2.645751 3.000000 2.913553 3.316625 3.741657
## 57      0 2.645751 3.000000 2.937633 3.316625 3.605551
## 58      0 2.645751 3.000000 2.844612 3.162278 3.605551
## 59      0 2.645751 3.000000 2.844612 3.162278 3.605551
## 60      0 2.645751 3.162278 3.020235 3.316625 3.872983
## 61      0 2.645751 3.000000 2.925955 3.316625 3.605551
## 62      0 2.645751 3.000000 2.910746 3.162278 3.741657
## 63      0 2.645751 3.000000 2.910746 3.162278 3.741657
## 64      0 2.645751 3.162278 3.040401 3.464102 3.741657
## 65      0 2.645751 3.000000 2.916664 3.316625 3.741657
## 66      0 2.645751 3.162278 2.936981 3.316625 3.741657
## 67      0 2.645751 3.162278 2.936981 3.316625 3.741657
## 68      0 2.645751 3.000000 2.881108 3.162278 3.464102
## 69      0 2.645751 3.000000 2.950286 3.162278 3.872983
## 70      0 2.645751 3.000000 2.853374 3.162278 3.741657
## 71      0 2.645751 3.000000 2.831686 3.162278 3.741657
## 72      0 2.645751 3.000000 2.831686 3.162278 3.741657
## 73      0 2.645751 3.162278 2.963930 3.316625 3.741657
## 74      0 2.645751 3.162278 2.976855 3.316625 3.741657
## 75      0 2.645751 3.000000 2.916606 3.316625 3.605551
## 76      0 2.645751 3.000000 2.891265 3.162278 3.741657
## 77      0 2.645751 3.162278 2.966624 3.316625 3.872983
## 78      0 2.645751 3.000000 2.861777 3.162278 3.605551
## 79      0 2.645751 3.000000 2.861777 3.162278 3.605551
## 80      0 2.645751 3.000000 2.938446 3.316625 3.872983
## 81      0 2.645751 3.162278 2.953039 3.316625 3.741657
## 82      0 2.645751 3.000000 2.911875 3.316625 3.605551
## 83      0 2.645751 3.000000 2.925897 3.316625 3.605551
## 84      0 2.645751 3.000000 2.925897 3.316625 3.605551
## 85      0 2.645751 3.000000 2.858343 3.162278 3.741657
## 86      0 2.645751 3.162278 2.988372 3.316625 3.741657
## 87      0 2.645751 3.162278 2.980899 3.316625 3.741657
## 88      0 2.645751 3.162278 3.001654 3.316625 3.741657
## 89      0 2.645751 3.000000 2.921135 3.316625 3.741657
## 90      0 2.645751 3.162278 3.021961 3.316625 3.872983
## 91      0 2.645751 3.162278 2.982074 3.316625 3.872983
## 92      0 2.645751 3.000000 2.886657 3.162278 3.741657
## 93      0 2.645751 3.162278 3.019169 3.316625 3.741657
## 94      0 2.645751 3.162278 3.027228 3.316625 3.741657
## 95      0 2.645751 3.162278 2.975595 3.316625 3.605551
## 96      0 2.645751 3.162278 2.975595 3.316625 3.605551
## 97      0 2.449490 3.000000 2.787554 3.162278 3.464102
## 98      0 2.449490 3.000000 2.778876 3.162278 3.605551
## 99      0 2.449490 2.828427 2.770506 3.162278 3.464102
## 100     0 2.236068 2.645751 2.546855 2.828427 3.162278
```

Because of the size of the data and the time it takes to knit, I wasn't able to complete a full Euclidian distance run (only the first 100 distances). However, this is meant to be an example of the framework I use for thinking about similarity and identifiability. If a single entry is "nearby" (i.e. small to zero Euclidian distance) to many other entries, that entry is less unique. However, if the majority of distances are larger, this entry may be more "isolated" within the dataset, and the risk of individual disclosure could be greater.

For example, in the table above, I have sorted these 100 entries by their first quartile difference – that is, the distance for which 75% of all entries are further away. The higher this number, the more entries that are further away, and the more unique/identfiable the entry. Notably, all of the distances computed here have a `Min` of 0, indicating there is at least one identical entry. However, I'm sure that with the inclusion of additional variables, certain people would become immediately identifiable.