Abridged: *Comparative Metrics Framework* in R to evaluate the performance of Synthetic Data

Mun Fai Chan

ABSTRACT

This abridged document[1] presents a comprehensive *comparative metrics framework* to evaluate the performance of synthetic data based on general utility, specific utility and disclosure risk. This is one of the few publicly available benchmarks to evaluate the quality of synthetic data. I will also provide some recommendations on best practices for generating synthetic data.

1 INTRODUCTION

I introduce a *comparative metrics framework* heavily inspired by Taub et. al (2019). Rather than reinventing the wheel, my focus here is to provide a practical guide to its implementation. In the event where certain code were not previously publicly available, I uploaded them into an R package **cmf** [2].

The functions used here are either from the **cmf** package or the synthpop package, which is a package to synthesise data. However, these functions can be generalised to any synthetic data generated with any method.

2 GENERAL UTILITY

General utility is a measure of the similarity in distributions between synthetic and original data. A visual examination of histograms or frequency tables will be useful as a preliminary check, but becomes too infeasible when comparing between a large number of synthetic datasets. Instead, a single summary statistic score is necessary for easy comparison of performance.

2.1 Ratio of Counts(ROC)

This evaluates the overlap in frequency counts between the original and synthetic dataset. I coded the mathematical equations introduced by Taub et el. (2020) in the **cmf** package.

$$\frac{min(y_o, y_s)}{max(y_o, y_s)} \tag{1}$$

y_o denotes the frequency count (by proportion) for a particular variable of the original dataset. y_s denotes the same for a synthetic dataset. Intuitively, this is analogous to the intersect of the 2 datasets divided by their union. The ROC score is bounded by 1 and 0. The lower the ROC score, the more distant the 2 datasets are in distribution. The function ROC_score evaluates the ROC score for all individual columns and averages them in the final output.

ROC_numeric evaluates the ROC score for numerical variables, because continuous numerical values in the original and synthetic data may not match exactly. For instance, 10,000 may be arbitrarily synthesised as 10,001 in the synthetic dataset but one may choose to treat them as the same value. Hence, ROC_numeric rounds numeric values off to a user-specified value.

2.2 Propensity Scores

Intuitively, propensity score is the probability of a row of data being synthetic given its entries. One can select the column(s) to calculate the propensity score on. A higher ratio to degree of freedom score obtained from synthpop::utility.tab indicates a worse general utility. Ratios bigger than 3 or 4 suggests that a variable is problematic and may warrant further attention - visual examination of these columns is useful.

3 SPECIFIC UTILITY

Specific utility evaluates if we get similar estimates when running the same statistical analysis on both synthetic and original datasets.

3.1 Confidence Interval Overlap (CIO)

Here, I focus on logistic and linear regressions, though we can easily extend this to other machine learning inferences. I make use of the synthpop::lm.synds() and synthpop::glm.synds() functions to run regressions(linear and logistic respectively) on the original and synthetic dataset and evaluate how close confidence intervals are.

4 DISCLOSURE RISK

4.1 Differential Correct Attribution Probability (DCAP)

I coded functions for DCAP (Taub et al, 2018) in the R package **cmf**. DCAP works on the assumption that an intruder has certain *key information* about individuals. Such key information are highly observable or obtainable - for instance a person's gender. Using these key information, the intruder attempts to extract *target information* that are more private - for instance a person's income level. We define d_o as the original data and K_o and T_o as vectors for the key and target variables.

$$d_o = \{K_o, T_o\} \tag{2}$$

Similarly, d_s is the synthetic data.

$$d_s = \{K_s, T_s\} \tag{3}$$

The Correct Attribution Probability (CAP) for the record j is the probability of its target variables given its key variables. This is coded as the function CAP_original . We can think of the CAP score for the original dataset to be an approximate upper bound - it would not make much sense for any other dataset to reveal more information than the original dataset.

$$CAP_{o,j} = Pr(T_{o,j}|K_{o,j}) = \frac{\sum_{i=1}^{n} [T_{o,i} = T_{o,j}, K_{o,i} = K_{o,j}]}{\sum_{i=1}^{n} [K_{o,i} = K_{o,j}]} \tag{4}$$

[1] Original document available at https://github.com/MUNFAI15/DiffPriv

[2] Available for download in R using devtools::installgithub("MUNFAI15/cmf")

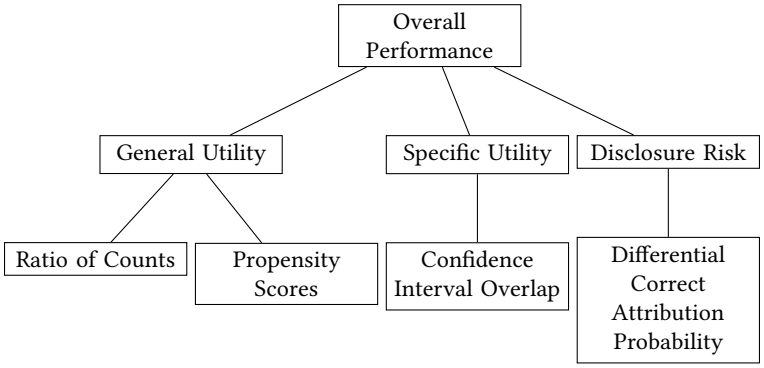

Figure 1: Overview of the Comparative Metrics Framework

where the square brackets are Iverson brackets and n is the number of records.

The CAP for the record j based on a corresponding synthetic dataset d_s is the same probability but derived from d_s. This has been coded as CAP_SYNTHETIC.

$$CAP_{s,j} = Pr(T_{o,j}|K_{o,j})_s = \frac{\sum_{i=1}^{n}[T_{s,i} = T_{o,j}, K_{s,i} = K_{o,j}]}{\sum_{i=1}^{n}[K_{s,i} = K_{o,j}]} \quad (5)$$

For any record in the original dataset for which there is no corresponding record in the synthetic dataset with the same key variables, the denominator in Equation 5 will be 0 and the CAP is therefore undefined. We can record non-matches as zero or treat them as undefined; the record is skipped over and does not count towards n.

To calculate the overall CAP for a synthetic dataset, we divide its CAP score by its corresponding upper bound - the CAP score for the original dataset. This process is repeated for all combinations of key and target variables.

5 PROVISIONAL FINDINGS AND RECOMMENDATIONS

I then apply the framework to analyse 3 different datasets. However, due to their sensitivity, I cannot reference them publicly and can only provide provisional recommendations for best practices.

5.1 Rules should be the foremost consideration in data synthesis.

When relationships between variables are violated in the synthetic data, they lead to patently spurious results. For instance, if all individuals below 16 should be single, we have to check that this relationship holds in the synthetic datataset. SYNTHPOP allows for the assertion of rules during the synthesis process which addresses this problem.

5.2 Visit sequences do not appear to have a significant and consistent impact on performance.

A visit sequence is the order in which variables are synthesised. Visit sequences matter to the performance of a synthetic data only insofar as multiple visit sequences generate random variations which increase the likelihood of finding a 'better' dataset. There appears to be no significant and consistent difference in performance for visit sequences generated by different mechanisms.

A quick method for generating different visit sequences is to use the VARRANK package in R to plot feature importance for each dependent variable. This is significantly faster than using Random Forests to find feature importance. A heuristic is to place the most important variables at the front of the visit sequence.

5.3 Speeding Up the Synthesis Process

(1) When experimenting with a new code, it is desirable to synthesise a small sub-sample of the dataset if the synthesis time is long. Use the SYNTHPOP::COMPARE() function to do a preliminary examination to ensure that the code is working. Errors may only appear after the synthesis has proceeded for some time. Restarting R during synthesis will cause all local variables to be lost.
(2) For variables with many categories, one can combine less frequent categories as an 'Others' category. This reduces computational time during synthesis and is useful for removing outliers who face a higher disclosure risk. On the other hand, one may risk worsening general and specific utility.
(3) When working with *DateTime* formats, computational time is significantly improved if these are changed to numeric variables. One may also remove the days (and months) depending on the level of precision required. This improves computational time slightly.
(4) When working with factors, ensure that factor levels are coded as their original names(for instance as strings) rather than as integers. This seemingly trivial difference may cause hours of difference in synthesis time.