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

Chan Mun Fai

Abstract

This document presents a comprehensive and intuitive *comparative metrics framework*. It evaluates the performance of data synthesis methods through the dimensions of general utility, specific utility and disclosure risk. Crucially, I aim to provide a practical guide to the implementation of this framework, such that users can readily implement and modify it for their own purposes.

1 Introduction

In this document[1], 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. Modifications were made to ensure easy and systematic implementation. In the event where certain code were not publicly available, I have coded them into functions and uploaded them into a Github repository.[2]

Many of the functions utilised in the *comparative metrics framework* are built-in functions of the **synthpop** package in R. However, I will discuss in Section 6 on how to enable these functions to be generalisable to synthetic datasets generated from all methods. For an overview of **synthpop**, see Nowok et al. (2016).

I have implemented this framework onto 3 different datasets and found the metrics to be sound and insightful. Unfortunately, these datasets are highly sensitive and cannot be publicly referenced. Nonetheless, I will provide some provisional findings and recommendations on best practices for data synthesis without giving explicit mention to the datasets used.

[1]In my attempt to enable this document to be as accessible as possible, I will employ highly intuitive - and possibly imprecise - explanations of key concepts.

[2]https://github.com/MUNFAI15/DiffPriv

2 General Utility

General utility is a measure of the similarity in distributions between the synthetic and original data. A visual examination of histograms or frequency tables (on univariate, bivariate and multivariate distributions) will be useful as a preliminary check, but becomes too infeasible when comparing between a large number of synthetic datasets. This is available through the COMPARE() and MULTI.COMPARE() functions in **synthpop** However, a single summary statistic score is necessary for quick and easy comparison of performance.

2.1 Univariate Distributions

Intuitively, this measure indicates how similar *individual* columns are between the original and synthetic dataset.

Ratio of Counts (ROC)

The **Ratio of Counts (ROC)** score evaluates the overlap in frequency counts between the original and synthetic dataset. Taub et el.(2017) introduces the mathematical equations behind it and I have coded as part of the ROC functions. The equation is as follows:

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

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. If $y_o = y_s$, the ROC score is 1. 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. Additionally, columns with few items in a category may perform especially poorly in the ROC score even if the numerical difference between datasets is small. For instance, consider Column X with a category α. In the original dataset, there are 2 observations of category α. In the synthetic dataset, there is 1 observation. Though the numerical difference is small, that equates to a 50% difference which may lead to an adverse worsening of the ROC score. Hence, the function ROC_SCORE automatically skips over such columns.[3]

ROC_ALL evaluates the ROC score for individual columns without aggregating them. One can identify columns which do particularly poorly and pay

[3]We can adjust the threshold that determines which variables to skip over. Specifically, we set a numerical threshold for the minimum number of items in a category. If a category in a variable has fewer items than this threshold, we skip over the entire variable.

special attention to them. ROC_NUMERIC evaluates the ROC score for numerical variables. Continuous numerical values in the original and synthetic data may not match exactly. For instance, the synthesis process may arbitrarily synthesise 10,000 as 10,001 in the synthetic dataset but one may still consider them to be of the same value. Hence, ROC_NUMERIC rounds numeric values off to a user-specified value. This is equivalent to placing the numbers into bins. Intuitively, the ROC scores for numeric variables will increase as the bins get larger.

Propensity Scores

For a detailed overview on **propensity scores**, refer to Raab et al.(2017). Very imprecisely, we calculate the propensity score by combining both the synthetic and original dataset and appending an additional column indicating if an observation is synthetic. We then run a logistic regression to calculate the probability of identifying whether an observation is synthetic.

Propensity scores can be tabulated using UTILITY.TAB and UTILITY.GENERAL from the **synthpop** package, with the former being a more practical function. One can select the column(s) to calculate the propensity score on, and the *ratio to degree of freedom* score is the most useful measure. It has a lower bound of 0, and a higher score indicates a bigger lack-of-fit and thus 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 using SYNTHPOP::COMPARE() would be useful. We calculate propensity scores for all variables and average them.

Aggregate of Ratio of Counts (ROC) and Propensity Scores

The ROC score and the propensity score are eventually aggregated in the final tabulation of performance. Empirically, they have a close relationship - synthetic datasets that perform well on the ROC score tend to do well on the propensity score, and vice versa. However, there can be differences in their results which justifies the need to use both metrics in combination. While the ROC score evaluates for a simple numerical difference/similarity, the propensity score uses a more sophisticated algorithm that measures some form of statistical difference. Variables that are statistically significant from their corresponding columns in the original dataset are more heavily penalised with the propensity score.

There may also be a difference in the ROC and propensity score for numerical values. The SYNTHPOP::UTILITY.TAB function automatically categorises numbers into bins though the number of bins cannot be specified by the user[4].

[4]I define a user as someone responsible for synthesising the data. Note the distinction with data end user. See footnote 5.

Thus, there may be a difference in results when the size of bins used by the ROC and propensity scores are different.

2.2 Bivariate / Multivariate Distributions

For bivariate and multivariate distributions, we will only focus on propensity scores.

Propensity Scores

We need a systematic way of selecting bivariate relationships to test for - if there were n different columns, there would be $\binom{n}{2}$ number of bivariate relationships, which may be too time-consuming to test for. I propose using the **varrank** package (or any other package, in fact) which provides the feature importance and mutual information between variables. Variables with strong inter-relationships in the original dataset will then be tested for in the propensity scores metric. Unfortunately, for multivariate distributions, one has to rely on intuition and discretion in choosing the columns to test for.

3 Specific Utility

Specific utility evaluates if we can get similar estimates when we run the same statistical analysis or machine learning model on both the synthetic and original datasets. However, one must have some *ex ante* assumptions of the analyses that a data end user[5] will be interested to run. Granted, such *ex ante* assumptions are highly prone to errors. Hence, I will propose that one accords the metrics on specific utility a lower weighting in the tabulation of the final performance of a synthetic dataset.

Confidence Interval Overlap (CIO) for logistic and linear regressions

Here, I focus only on logistic and linear regressions, though a similar approach can be easily extended to other forms of statistical analysis and machine learning models as long as an equivalent confidence interval exists. I make use of the LM.SYNDS() and GLM.SYNDS() functions available in the **synthpop** package. These functions run a regression(linear and logistic respectively) on the original and synthetic dataset and evaluate how close the confidence intervals are.

It is imperative that one runs a regression that is actually valuable to a data end user. A possible mechanism for doing so is as follows :

1. Identify variables that a data end user may be interested in studying.

[5]I define a data end user as an individual who is interested in obtaining and analysing the data provided by the data provider. This data end user will first explore the synthetic dataset provided and evaluate if it is useful for his/her purpose before sending in a formal request to obtain the original dataset.

2. Find the best specification of such regressions on the original dataset by backwards elimination.

3. Apply these regressions to the synthetic datasets using the functions SYNTHPOP::LM.SYNDS() for linear regressions and SYNTHPOP::GLM.SYNDS() for logistic regressions.

4. Retrieve the Confidence Interval Overlaps (CIO).

5. To evaluate the final specific utility, weigh each CIO by its corresponding adjusted R^2 on the original dataset and take the weighted average of all the CIOs. A regression with a higher predictive power and presumably greater utility to the data end user will receive a higher weighting.

Unfortunately, the aforementioned functions are imperfect. They are prone to computational errors particularly when the dependent variable is numerical. Ultimately, this justifies why the metric on specific utility is accorded a lower weighting in the final evaluation of overall performance.

4 Disclosure Risk

Differential Correct Attribution Probabibility (DCAP)

Taub et al. (2018) proposed the **Differential Correct Attribution Probability (DCAP)** approach. 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, race and ethnic grouping. Using these key information, the intruder attempts to extract *target information* that are more private - for instance a person's income level.

We present the method for calculating DCAP as introduced by Taub et al. (2018) and coded the following equations into R functions. This can be found in the DCAP_FUNCTIONS script. 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.

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

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.

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

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 (Elliot, 2014). In most cases, we will use both approaches and take their average. However, CAP scores tend to be higher when non-matches are treated as undefined; in some cases, they may even be significantly higher than the CAP score for the original dataset. In such an instance, we will only consider the case where non-matches are treated as 0.

The baseline CAP for record j is the marginal probability of its target variables estimated from the original dataset. This is coded as CAP_BASELINE. The assumption here is that the intruder will know the univariate distribution of a target variable for the population, presumably through summary statistics that are publicly available. We can think of the baseline CAP score as an approximate lower bound.

$$CAP_{b,j} = Pr(T_{o,j}) = \frac{1}{n}\sum_{i=1}^{n}[T_{o,i} = T_{o,j}] \tag{6}$$

For numerical variables, Equations 2 to 6 are modified in the same way as the ROC scores - numerical values are categorised into bins for which the size of the bins can be user-specified.

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 Overall Performance

The overall performance of a synthetic dataset can be evaluated through 2 approaches, with the latter approach still being a work in progress.

The first approach is to rank each synthetic dataset for each of the metrics in the *comparative metrics framework*. We then assign a weighting to each metric and calculate the weighed average of these ranks. The weighted average of ranks is useful in finding all-rounders or datasets that do generally well in most metrics. However, it gives no information about the difference in scores between synthetic datasets. Only the ordinal ranking of datasets is reflected here.

The second approach attempts to account for raw differences in scores between synthetic datasets. The idea is to have some mechanism to aggregate all metrics and then to take a weighted average of this aggregate score. The ranking of this weighted aggregate score provides an ordinal comparison of performance while numerical differences in the scores provide a clearer picture into the magnitude of difference. However, a reasonable method of aggregation has not yet been established. First, the different metrics scores are on different scales and it is challenging to apply the same standardisation to all metrics. Second, there are some metrics where a higher score denotes a better performance and other metrics where the opposite holds true.

Nevertheless, underlying both approaches is the mechanism of taking a weighted average, whereby the weights of each metric can be adjusted to the preference of the user. The summation of all weights equals to 1. For instance, one can give a weighting of 0.5 to both utility and disclosure risk so that both factors are equally weighed against one another. Furthermore, as mentioned in Section 3, the metrics on specific utility may be too narrow and can thus be assigned with a lower weighting.

6 Generalisation of synthpop functions

Many of the functions in the *comparative metrics framework* are built in functions of the **synthpop** package. It will be useful to allow this *comparative metrics framework* to generalise to all synthetic datasets. Provisionally, it appears that many new techniques for generating synthetic data are computationally more efficient and have an explicit focus on preserving **differential privacy** which may help to reduce disclosure risk. On the other hand, **synthpop** still holds the upper hand in its flexibility and versatility especially when it comes to rules assertion.

Synthpop functions only read in objects of class *synds*, a distinct class for synthetic data generated by **synthpop**. A *synds* object is simply a large list, with the first item being the synthetic dataset itself, and the other items being parameters associated with the synthetic dataset in question. This includes but is not limited to the visit sequence, the predictor matrix and the methods used. We can artificially ‘transform’ any synthetic dataset (saved as a CSV file) into the class *synds* by appending it with additional aforementioned items in the list. It is challenging to manually code for these items and express them in a format that is readable by **synthpop**. Rather, a convenient method is to :

1. Synthesise the original dataset with **synthpop**. An object of class *synds* will be generated.
2. Duplicate the object and replace its embedded dataset with any desired dataset. Datasets are stored in SYNS in the *synds* object.
3. Use the SYNTHPOP::COMPARE() function to ensure that there are no errors in this process.
4. Some pre/post-processing may be required to ensure that column names and variables are of the same format as the original dataset.

These steps work for any synthetic dataset with the same dimension and column names as the original dataset. Although this is merely a ‘quick hack’ solution, it seems to work without issues as the **synthpop** functions we use only depend on the synthetic dataset to calculate their final output. Arbitrarily changing the other items in the object made no difference to the outputs.

7 Conclusion

I hope that this document can serve as a practical guide to the implementation of a comprehensive *comparative metrics framework* in evaluating the best data synthesis method. Furthermore, this framework is intuitive, systematic and highly generalisable to other synthesis methods. However, there can be further development in an aggregation mechanism for evaluating the final performance.

References

Gillian M. Raab, Beata Nowok, and Chris Dibben. Guidelines for producing useful synthetic data, 2017.

Beata Nowok, Gillian Raab, and Chris Dibben. synthpop: Bespoke creation of synthetic data in r. *Journal of Statistical Software, Articles*, 74(11):1–26, 2016. ISSN 1548-7660. doi: 10.18637/jss.v074.i11. URL `https://www.jstatsoft.org/v074/i11`.

David Voas and Paul Williamson. Evaluating goodness-of-fit measures for synthetic microdata. *Geographical and Environmental Modelling*, 5(2):177–200, 2001. doi: 10.1080/13615930120086078. URL `https://doi.org/10.1080/13615930120086078`.

Joshua Snoke, Gillian M. Raab, Beata Nowok, Chris Dibben, and Aleksandra Slavkovic. General and specific utility measures for synthetic data. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 181(3):663–688, 2018. doi: 10.1111/rssa.12358. URL `https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/rssa.12358`.

Markus Hittmeir, Andreas Ekelhart, and Rudolf Mayer. Utility and privacy assessments of synthetic data for regression tasks. *2019 IEEE International Conference on Big Data (Big Data)*, 2019. doi: 10.1109/bigdata47090.2019.9005476.

Jennifer Taub, Mark Elliot, M. Pampaka, and D. Smith. Differential correct attribution probability for synthetic data: An exploration. In *PSD*, 2018.

Jennifer Taub, Mark Elliot, and Joseph W. Sakshaug. The impact of synthetic data generation on data utility with application to the 1991 uk samples of anonymised records. *Trans. Data Priv.*, 13:1–23, 2020.

Bill Howe, Julia Stoyanovich, Haoyue Ping, Bernease Herman, and Matt Gee. Synthetic data for social good. *CoRR*, abs/1710.08874, 2017. URL `http://arxiv.org/abs/1710.08874`.

Gillian M Raab and Beata Nowok. Inference from fitted models in synthpop.

Jun Zhang, Graham Cormode, Cecilia M Procopiuc, Divesh Srivastava, and Xiaokui Xiao. Privbayes: Private data release via bayesian networks. *ACM Transactions on Database Systems (TODS)*, 42(4):1–41, 2017.

Gregory Caiola and Jerome P Reiter. Random forests for generating partially synthetic, categorical data. *Trans. Data Priv.*, 3(1):27–42, 2010.

Beata Nowok. Utility of synthetic microdata generated using tree-based methods. 2015.

Jennifer Taub, Mark Elliot, and Gillian Raab. Creating the best risk-utility profile : The synthetic data challenge. 2019.

Mark Elliot. Final report on the disclosure risk associated with the synthetic data produced by the sylls team. Oct 2014.

Gilles Kratzer and Reinhard Furrer. varrank: An r package for variable ranking based on mutual information with applications to systems epidemiology, Jan 2020. URL `https://www.math.uzh.ch/pages/varrank/articles/varrank.html`.

Gilles Kratzer and Reinhard Furrer. varrank: an r package for variable ranking based on mutual information with applications to observed systemic datasets, Apr 2018. URL `https://arxiv.org/abs/1804.07134`.

8 Appendix : Provisional Findings and Recommendations

Given the sensitivity of the datasets, I can only provide a few provisional findings and recommendations without demonstrating the evidence/justifications behind them. Note that these findings and recommendations may not be generalisable to all datasets.

8.1 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. **Synthpop** allows for the assertion of rules during the synthesis process which addresses this problem.

However, when we do not explicitly assert for rules during data synthesis, these relationships will not necessarily be broken in the synthetic data. It appears that preservation of relationships when rules are not asserted happens due to mere chance. When relationships are broken, it is imperative to correct for them by asserting rules in the synthesis process. However, when relationships are not broken, asserting rules does not lead to a significant difference (or improvement) in performance in all metrics.

Recommendations

1. Rules are the first thing to check for, and should be accorded the highest priority in the event that they are violated in the synthetic data.
2. Come up with a list of possible relationships and check that these relationships indeed hold true in the original dataset. If a relationship largely but not completely holds true, it may be sensible to still code it as a rule, because the synthesis will remove the outliers and reduce their disclosure risk.
3. Synthesise the dataset without any rules asserted. Then, check which of the above relationships are violated. Select the most severe violations and assert them as rules in the synthesis process.
    - We classify the severity of violations by the number of records that do not preserve these relationships in the synthetic data divided by the number of records that do in the original data (assuming number of observations are the same in both)
    - If the synthesis process takes too long, use a smaller subsample instead.
    - If relationships are not violated, rules need not be applied as they do not seem to make a difference.

4. Alternatively, one can assert for rules in the very first step. At best they help to preserve relationships; at worst, they will not harm the performance. However, it is impossible to know which rules are the most important *a priori* without empirically checking if they are violated in the synthesised data.

Noteworthy are the limitations surrounding the syntax in **synthpop** when asserting rules. Between any group of variables, there can only be 1 rule. Rules that code for numerical relationships between variables are disallowed. There is also a limit to the number of rules due to computational issues. Moreover, visit sequences are limited as the variables that act as the conditions for the rules have to be synthesised beforehand. Keeping these in mind, it is impossible to assert for all possible rules *a priori*. Nevertheless, despite **synthpop's** limitations, there are not many synthesis methods that can allow for the explicit assertion of rules.

8.2 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 in **synthpop**. Visit sequences matter to the performance of a synthetic dataset 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 which provides us with mutual information and feature importance between variables. One can then use the following heuristics, though we do not have to concern ourselves with following them exactly.

- Place variables with important inter-relationships near the front of the visit sequence.
- Place variables with many categories at the back though this may contradict with the previous recommendation.
- If one is using rules in the synthesis process, ensure that the conditions for designing visit sequences for rules hold true.(See subsection 9.1)

8.3 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(which may take a few hours). Restarting R during synthesis will cause all local variables to be lost.

2. Alternatively, one can code for $m = 0$ in **synthpop** where m is the number of datasets to be synthesised. This allows the code to run and alert the user of any errors without actually synthesising a dataset.

3. For variables with many categories, it may be useful to combine the 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 the scores for general and specific utility.

4. When working with $DateTime$ formats, computational time is significantly improved if these are changed to numeric variables. The POSIXCT package allows easy conversion of $DateTime$ to numeric and back.

5. When working with $DateTime$, one may also use the POSTIXT package to remove the days (and months) depending on the level of precision required. For instance, when working with Date of Birth, one is often less interested in the actual day of birth, but more interested in the year (and month). This improves computational time slightly(to a much lesser extent than Point 4) but can be a useful step.

6. 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.