Supporting Database Constraints in Synthetic Data Generation based on Generative Adversarial Networks

Wanxin Li

Cheriton School of Computer Science, University of Waterloo
Waterloo, Ontario, Canada

Problem and Motivation

With unprecedented development in machine learning algorithms, it is crucial to have available large amount of data to verify the correctness and efficiency of these algorithms. Due to privacy concerns, we may not always have enough real data to use; it is important to develop techniques to generate synthetic data which are similar to the real world data. In our research, we focus on data synthesization for relational databases where the database constraints [1] of the original data must be imposed to the generated data.

To the best of our knowledge, no study has been conducted on supporting database constraints in synthetic data generation. Theoretically, we can filter out records that violate database constraints after generation. However, this can cause a large portion of the generated records being thrown way. In this paper, we aim to significantly reduce the likelihood of generating invalid records. We offer solutions by designing extensions to Tabular Generative Adversarial Network (TGAN) [5] algorithm for supporting database constraints. Specifically, we propose functions to encode database constraints as additional penalties into the TGAN loss function. We designed and implemented a prototype for our approach, and compare the performance of different extensions by a set of experiments.

Contributions

- Database constraints are represented by non-differentiable boolean functions. Neural networks require loss functions to be differentiable. To encode constraints as additional penalties to loss functions, we design differentiable approximation functions which transform the satisfiability of constraints into numerical values.
- We propose scaling and combining methods in loss functions to account for the differences among the original penalties related to the similarity between the generated data and the original data, and the additional penalties related to database constraints.

Extended TGAN Algorithm - Tables

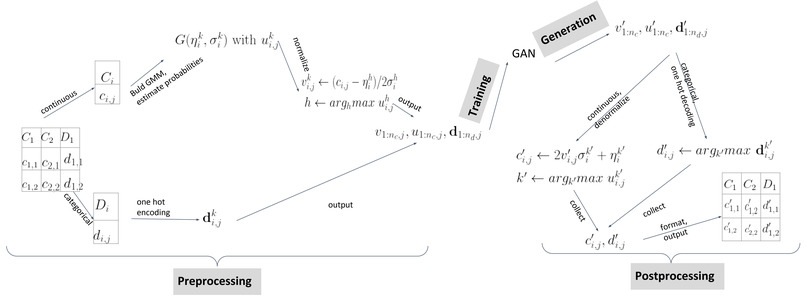
Table 1: $\mathscr{C}_t(X_t)$ for categorical variables of the form of $T.A - c = (\neq)0$ or $T.A - T.B = (\neq)0$ for the table $T(A, B)$, and c is the c^{th} category of $T.A$. $|A|$ is the total number of categories of $T.A$. $\mathbf{d}'_A$ is a vector of the estimated probabilities for A's categories. $\mathbf{d}'^{c}_A$ is the estimated probability for the c^{th} category of $T.A$. abs takes the absolute value. sum takes the sum of vector elements.

constraint form \ op	$=$	$\neq$
$T.A - c = 0 / T.A - c \neq 0$	$1 - \mathbf{d}'^{c}_A$	$\mathbf{d}'^{c}_A$
$T.A - T.B = 0 / T.A - T.B \neq 0$	$sum(abs(\mathbf{d}'_A - \mathbf{d}'_B))/\lvert A\rvert$	

Table 2: additional penalty ap_t for constraint $\mathscr{C}_t(X_t)$. $\beta < 0$ and $\gamma > 0$. In our experiments, $\beta = -0.1, \gamma = 1$.

penalty type \ constraint type	$\mathscr{C}_t(X_t) = 0$	$\mathscr{C}_t(X_t) \neq 0$		$\mathscr{C}_t(X_t) \geq 0$
penalty_cat (categorical)	$\mathscr{C}_t(X_t)$	$T.A - T.B = 0$: $-\mathscr{C}_t(X_t)$	$T.A - c = 0$: $\mathscr{C}_t(X_t)$	NA
penalty_cont (continuous)	$-e^{\beta\mathscr{C}_t(X_t)^2} + 1$	$e^{\beta\mathscr{C}_t(X_t)^2}$		$\frac{1}{1+e^{\beta-\gamma\mathscr{C}_t(X_t)}}$

Original TGAN Algorithm



We divide original TGAN algorithm into four phases: preprocessing, training, generation and postprocessing.
In preprocessing, columns are divided into continuous and categorical columns. For each continuous columns, a GMM model is built, and a probability $u_{i,j}^k$ for each record $c_{i,j}$ coming from the kth Gaussian model is estimated. $v_{i,j}$ is the normalized value of $c_{i,j}$. Categorical records are represented by one-hot encoding. $\mathbf{d}_{i,j}^k$ is the binary indicator if $d_{i,j}$ comes from the kth category. In training, the generator takes $v_{1:n_c,j}, u_{1:n_c,j}, \mathbf{d}_{1:n_d,j}$ and iteratively learns how to produce more qualified $v'_{1:n_c}, u'_{1:n_c}, \mathbf{d}'_{1:n_d,j}$. Finally, a well-trained generative adversarial network (GAN) is returned by the training process. In generation, the trained GAN produces a final set of $v'_{1:n_c}, u'_{1:n_c}, \mathbf{d}'_{1:n_d,j}$. In postprocessing, the algorithm denormalizes and decodes the output from generation, and combines $c'_{i,j}, d'_{i,j}$ into input format.

Extended TGAN Algorithm

Optimal changes are numbered using (Ⅰ), (Ⅱ), (Ⅲ). Table 1 and 2 are given in Extended TGAN Algorithm - Tables section.

- (Ⅰ) In preprocessing, $v_{i,j} \leftarrow \sum_{k=1}^{m} u_{i,j}^k (c_{i,j} - \eta_i^k)/2\sigma_i^k$.
- In training, for a given set of constraints $\mathscr{C}$, we construct an additional penalty term ap used in the loss function by the following steps:
  - Split $\mathscr{C}$ into atomic constraints $\{\mathscr{C}_1, ..., \mathscr{C}_T\}$. For example, for a table T(A, B, C), $\mathscr{C} = (T.A \geq T.B \text{ or } T.D \neq 3)$ will be split into $\{\mathscr{C}_1 = (T.A \geq T.B), \mathscr{C}_2 = (T.D \neq 3)\}$.
  - For $\mathscr{C}_t$ in $\{\mathscr{C}_1, ..., \mathscr{C}_T\}$
    - Each $\mathscr{C}_t$ is represented by an equation with 0 on the right hand side. For example, $T.A \geq T.B$ will be transformed to $T.A - T.B \geq 0$: $\mathscr{C}_t(X_t) \geq 0$ and $(X_t = T.A - T.B, op = \geq)$.
    - If X_t involves continuous columns, the formulae for $\mathscr{C}_t(X_t)$ is given in Table 1.
    - (Ⅱ) If X_t Involves continuous columns, the formula for $\mathscr{C}_t(X_t)$ is $\mathscr{C}_t(X_t) = X_t(c'_{k:l,j})$ where $c_{k:l}$ denote all referenced columns in X_t and $c'_{i,j} = \sum_{k=1}^{m} u'_{i,j}(2v'_{i,j}\sigma_i^k + \eta_i^k)$.
    - The formulae for the t^{th} additional penalty ap_t are given in Table 2.
    - The formula for the final ap is obtained by combining ap_t given the original constraint $\mathscr{C}$: for example, $\mathscr{C} = (\mathscr{C}_1 \ OR \ \mathscr{C}_2) \ AND \ \mathscr{C}_3$ then $ap = softmax(softmin(ap_1, ap_2), ap_3)$
  - We experiment with two new generator loss functions:
    - $gloss_1 = original\_loss + ap$ (1)
    - $gloss_2 = original\_loss * (1 + ap)$ (2)
  - (Ⅲ) In postprocessing, $c'_{i,j} = \sum_{k=1}^{m} u'_{i,j}(2v'_{i,j}\sigma_i^k + \eta_i^k)$.

Experimental Results

35 experiments have been performed with 5 settings (vertical) and 7 constraints (horizontal). The numbers of records violating database constraints have been reduced by 22% to 100% with E4. The numbers of invalid records can be summarized by the following table:

Table 3: $\mathscr{C}1$ is for categorical columns. $\mathscr{C}3$, $\mathscr{C}4$, $\mathscr{C}5$, $\mathscr{C}8$ are for continuous columns. $\mathscr{C}6$ and $\mathscr{C}7$ are for a mix of continuous and categorical columns.

description \ constraint id	$\mathscr{C}1$	$\mathscr{C}3$	$\mathscr{C}4$	$\mathscr{C}5$	$\mathscr{C}6$	$\mathscr{C}7$	$\mathscr{C}8$
$E1$: TGAN	947	1980	27	249	463	1241	5089
$E2$: TGAN with (I), (III)	928	346	836	316	809	2831	9713
$E3$: TGAN with (I), (II), (III); use $gloss_1$	166	451	0	0	10	1363	0
$E4$: TGAN with (I), (II), (III); use $gloss_2$	179	231	1	0	30	968	0
$E5$: TGAN with (II); use $gloss_2$	201	281	0	5	1	2302	5035

Conclusion

We conclude that experiments with E4 setting are the most promising one for future improvement. The success of E4 can be justified by the following:

- To overcome the non-differentiability of argmax in original TGAN, we used a consistent weighted sum of Gaussian models in preprocessing, training and postprocessing.
- Formulae in Table 2 are differentiable approximations to quantify the satisfiability of the constraints.
- softmax is a differentiable approximation for AND, so is softmin for OR.
- The range for $original\_loss$ is $[0, +\infty]$. The range for ap is [-1,1]. (2) works better than (1) as multiplication in (2) uses ap to penalize $original\_loss$ but addition in (1) can be problematic if ap and $original\_loss$ are not in the same scale.

Future Directions

There are many open problems in this area related to designing a general approach for supporting any database constraints in the generation algorithms based on GAN. In the future work, we plan to explore new types of constraints such as functional dependencies, foreign key constraints, etc. The runtime of the GAN algorithms is also a real issue: the current implementation based on TGAN, for some experiments, take hours to finish. Lastly, it will be interesting to apply our extended TGAN algorithm into industrial pipelines.

References

[1] W. Fan. Dependencies revisited for improving data quality. In Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems PODS, pages 159–170, 2008.
[2] N. Park, M. Mohammadi, K. Gorde, S. Jajodia, H. Park, and Y. Kim. Data synthesis based on generative adversarial networks. Proceedings of the VLDB Endowment, 11(10):1071–1083, 2018.
[3] N.Patki,R.Wedge,and K.Veeramachaneni.The synthetic data vault.In 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA), pages 399–410, 2016.
[4] J.Xu,Z.Zhang,T.Friedman,Y.Liang, and G.V.d.Broeck. A semantic loss function for deep learning with symbolic knowledge. pages 5498–5507, 2018.
[5] L. Xu and K. Veeramachaneni. Synthesizing tabular data using generative adversarial networks. arXiv preprint arXiv:1811.11264, 2018.

Acknowledgements

This work has been done under the supervision of Dr. Anisoara Nica during my co-op internships at SAP Labs, Waterloo, Ontario, Canada.