Synthetic Customer Support Data Generation

Ali Bahari
School of Electrical and Computer Engineering
University of Tehran
Tehran, Iran
ali.bahari@ut.ac.ir

***Abstract*—Synthetic Data Generation (SDG) is a growing field of research which focuses on generating new data that closely resembles the real data. Time-series data generation has recently attracted a lot of attention and different models have been proposed to tackle this novel task. In this report, I will first introduce multiple time-series data generation models. Then, I will present the customer support dataset which will be used for time-series data generation. The methods that were utilized for data generation will be explained and finally, the produced results will be examined.**

Keywords—Synthetic Data Generation, Time-series Data Generation

I. Introduction

With the progress in AI development, there will be an ever-growing need for data. Creating large datasets is a challenging, time-consuming and expensive task. So, Synthetic Data Generation (SDG) has been proposed as a suitable alternative. Synthetic data is closely similar to the real data and can be used to augment the existing dataset without disrupting any patterns. Different types of data makes SDG a difficult task. Recently, methods utilizing Generative Adversarial Network (GAN) and Variational Autoencoders (VAE) have achieved remarkable results for synthetic image generation. With the success of GAN and VAE, these models are now being used for generating other types of data such as time-series data. Different methods have been developed by the researchers for time-series data generation using GAN such as: DoppelGANger [1] and TimeGAN [2]. The focus in this report is on DoppelGANger. The given project involves customer support data generation. A dataset of customer support records is given and the goal is to generate and analyze new synthetic data. DoppelGANger is used in the implementation of this project for data generation. In this report, I will start by introducing DoppelGANger. Then, I will examine the dataset and explain the methods which convert the dataset to the desired format for the model. Finally, I will explain the implementation in detail and show the results.

II. Time-series Data Generation - DoppelGANger

At first, I started by reading the paper introducing Banksformer [3] which gave me a great overview of the existing methods for time-series data generation. After going through the resources [1, 2, 3, 4] and their codes/documentation I decided to use DoppelGANger since the paper for it was published a year after TimeGAN and the code for it was well documented. The code for DoppelGANger is available on GitHub. The gan folder holds the implementation and will be used for the generation task. The master branch of the repository uses TensorFlow 1 so it is outdated. I first tried TensorFlow's tf_upgrade_v2 script to convert the code to TensorFlow 2 however, there were conversion issues with some lines of code. The repository has a TF2 branch which implements the code in TensorFlow 2 but it is not a full rewrite. As a result, there were many warnings for deprecation. However, there was no problem in running the code and training the model. I also found a TensorFlow 2 version of the code on GitLab [5] but there was no difference in code execution so I decided to go with the DoppelGANger's GitHub TF2 version. There is also a PyTorch implementation of DoppelGANger [8] available but there are some differences between the PyTorch version and the GitHub TF2 version. Now, the dataset's structure must change to be suitable for DoppelGANger's input. In the next section, these changes will be explained.

III. Customer Support Dataset

The given dataset consists of 306419 records of customer support reports. A user which belongs to a team will handle a task which was created at a point in time. The user will start at a certain time and finish the task at a different time. The customer will rate the service and mention whether his/her issue was resolved or not. Each of these properties is presented by a column in the dataset. First, the dates in the dataset need to be in datetime format. For view_date and action_date in each row, I replaced the values with their time difference with creation_date in seconds. This way, by having creation_date, view_date and action_date can easily be calculated. Also, we will have a number instead of datetime to feed the model. For the categorical values, the one-hot encoded version is needed so, the values are replaced by their one-hot encoded version. Customer_problem_resolved has only 2 unique values so its one-hot encoding is presented in only one column consisting of 0 (False) and 1 (True) values. However, for DoppelGANger's input, each unique categorical value needs to be presented by a separate column no matter how many unique values the initial column has. So, there must be two columns for one-hot encoding of customer_problem_resolved. Next, we need to convert the dataset into sequences and specify each record's features and attributes. Each user behaves in a different way. At each point in time, a user resolves a customer issue which can be expressed by a set of features. It makes sense to consider that each user has a sequence of responses and that we can choose each user's records as a single time-series sequence. So, the total number of sequences will be equal to the number of total users which is 50. After identifying the sequences we need to set the features and attributes. Features are variables that are dependent on time and are observed at each time point. On the other hand, attributes are fixed variables that do not vary over time. We can have no attribute but we must at least have 1 feature. For each user, we can consider view_date, action_date, task_type, customer_satisfaction and customer_problem_resolved as features since the values in these column change over time. Creation_date for each user will be sorted and transformed to the difference between two consecutive datetimes which is our time step. The only column left is user_team. Since each user only belongs to one team and his/her team does not change at all, user_team is an attribute for each user.

IV. DoppelGANger Input

Now that sequences, features and attributes are chosen, we have to prepare them as the input for DoppelGANger model.

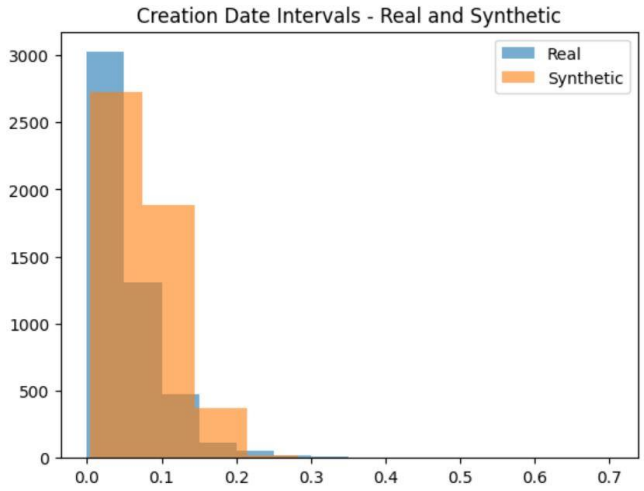
DoppelGANger requires 5 arrays as input: data_feature_output, data_attribute_output, data_feature, data_attribute and data_gen_flag. Data_feature_output and data_attribute_output indicate the features' and attributes' details such as dimension, type and normalization respectively. There are 6 features, 3 of them are of type continious, have a dimension of 1 and are normalized to zero and one. The other 3 are of type discrete since they are categorical values and their dimension is equal to their one-hot encoding length which is the number of unique values in the intial column. Also there is no need for normalization for the categorical values. All is_gen_flag values are set to False since this value will be set by the model itself in training. There is only one attibute and it is handled the same way as the features.Next, for creating data_feature array we first need to set Lmax which is the length of the largest sequence. The user with the most number of responses has 8526 responses which makes the length of the longest sequence equal to 8526. I set Lmax to 8600. All the users' sequences will fit in the array and the remaining spots will be filled by zeros (zero padding). So, the dimensions of data_feature will be (50, 8600, 13). 50 for the number of users, 8600 as Lmax and 13 for the total number of features including both categorical (one-hot encoded) and numerical variables. For each user, I first handled creation_date as mentioned before and then normalized creation_date, view_date and action_date. For normalization minmax is used which scales the values between 0 and 1, the same as the normalization in data_feature_output. The final array will be data_feature. Data_attribute is fairly the same. There are 4 teams and each user belongs to a team so the data_attribute's dimesions will be (50, 4). 50 for the number of users and 4 for the number of teams. Data_gen_flag indicates the activation of each feature in a time-series sequence. The dimension will be (50, 8600). 50 for the number of users and 8600 for Lmax. All of the values for each user in data_gen_flag will be equal to 1 since 1 means that the feature in the sequence is activated at the current time step. At the start of training, the value must be 1 so to activate all features for each sequence and use them all in the training. Now, every input is ready so we can build the model itself.

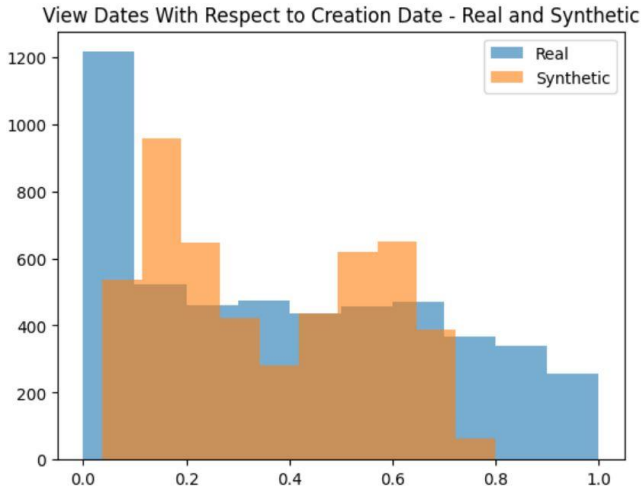
V. DoppelGANger Model

The model has feature and attribute generators and two discriminators. The hyperparameters set for the generators and discriminators can be seen in the Jupyter Notebook. The hyperparameters are mostly the general values mentioned in the DoppelGANger paper [1] or the Banksformer parameters report [3]. The number of epochs, batch size, the learning rate for the generators and the discriminators, number of layers in the RNN, number of units in each layer of the RNN, number of layers in the attribute generator, number of units in each layer of the attribute generator, number of layers in the discriminators and number of units in each layer of the discriminators were tested with numerous values and had noticable effects on training. After building and training the DoppelGANger model using the inputs and components mentioned so far, we can generate new data from noise using the trained model. The model will output new values for features and attributes which can easily be converted to the format in the dataset.

VI. New Generated Data

For comparing the new generated data with the real data from the tickets dataset, the new generated data needs processing to have the right format. The generated data will have the same shape as data_feature array so we need to extract each sample's sequence and put all the sequences together. Also, there will be records that have all zeros which are useless for comparison. These records are the result of padding the sequences with zeros. We want to compare the values of each column in the two datasets. For comparison, we first need to know that whether the synthetic data has captured the distribution of the real data. By plotting the values for each numerical column, we can somewhat understand this. Moreover, we can do the same with categorical columns by counting and plotting the number of unique categorical values. If the plots in both cases are fairly similar, we can say that the model has done a decent job. Also, we can use Wasserstein distance for numerical values and Jensen-Shannon distance for categorical values. These distances compute the similarity between two probability distributions and can show how well the model generated synthetic data. There is one other way of comparison and that is checking the similar patterns in the two datasets. For example, in the real dataset, there can be fewer requests at midnight but more requests at 9:00 A.M. The generated records have to keep this kind of pattern too. This comparison can also be visualized. The hyperparameters mentioned in section V played a major role in getting different results. I have also provided some photos of results with different combination of values for hyperparameters in the files. Larger batch sizes need less epochs with a higher learning rate. The DoppelGANger's paper mentioned that the value of 400 for epochs can be enough. But [4] has reached better results with higher number of epochs. I chose 400 epochs which took near 200 minutes to train. I believe higher epochs can produce better results too since going from 100 to 200, 300 and 400 epochs resulted in better outputs from the model. But, since the training process on CPU took so long, I decided not to go over 400. The issue with DoppelGANger's code was that suprisingly, running the code on GPU did not make the training process faster. Even after using the library mentioned on the GitHub repository of DoppelGANger, GPUTaskScheduler [6], I did not get any outcome. In [7] the authors also encountered the same issue with GPUTaskScheduler and only executed the code on CPU. The PyTorch version of DoppelGANger runs only on CPU too. Number of layers and units in the model was also important. Simple models with lower number of layers and units did not perform well. The generator needs to be a bit more complex than the discriminator. This way the generator can challenge the discriminator, resulting in a tougher competition between the two and a better performance at the end. Also, for attribute generator, since the number of training samples is not that high, we can have much lower number of layers and units. In the figures below (Figure 1, 2, 3), also available in the provided Jupyter Notebook, you can observe the visual comparison between the real and synthetic data distribution. The synthetic data distribution is fairly similar to the real data distribution.





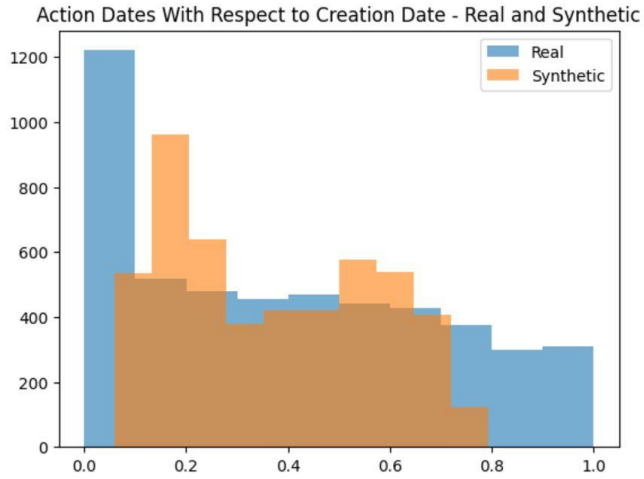
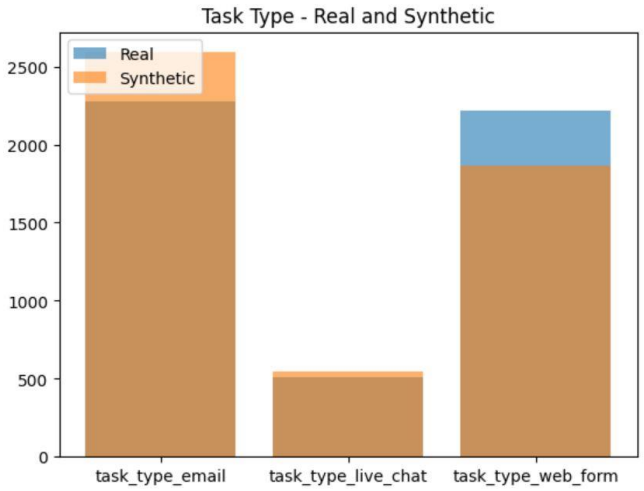
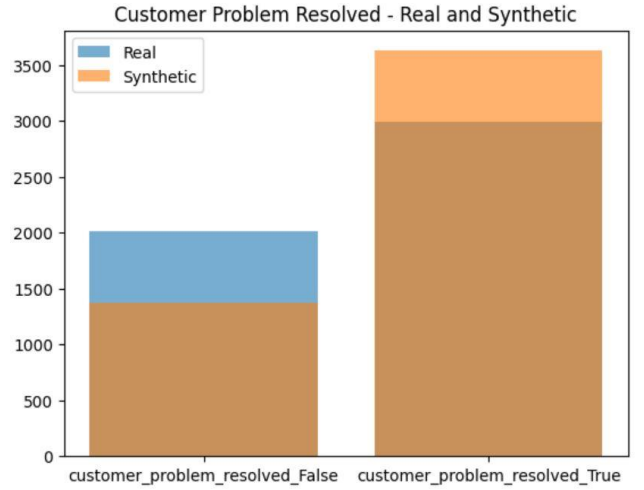


Fig. 1. Synthetic vs. real numerical data (features) - creation_date (top), view_date (middle) and action_date (bottom)





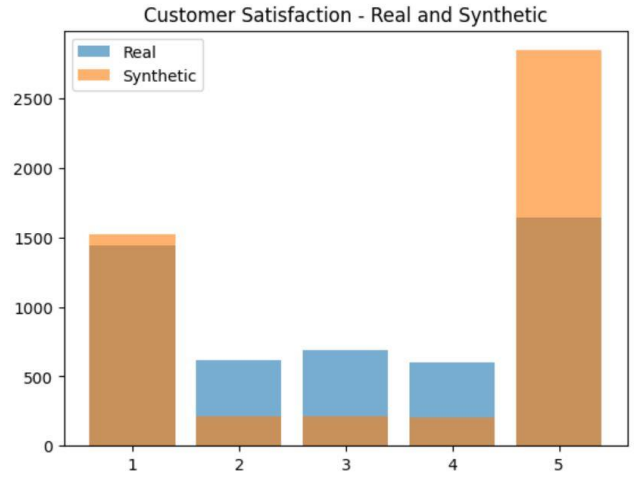


Fig. 2. Synthetic vs. real categorical data (features) – task_type (top), customer_problem_resolved (middle) and customer_satisfaction (bottom)

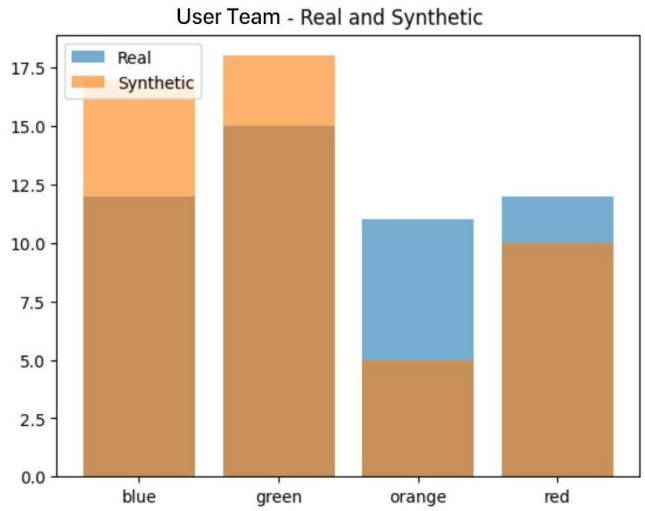


Fig. 3. Synthetic vs. real categorical data (attributes) – user_team

For comparing the distributions in figures above, 5000 records were sampled randomly from both real and synthetic datasets and the values of both were plotted on histograms. Wasserstein distance for creation_date, view_date and action_date is equal to 0.033, 0.082 and 0.084 respectively. Jensen-Shannon distance for task_type, customer_problem_resolved, customer_satisfaction and user_team is equal to (0.605, 0.786, 0.635), (0.684 , 0.488), (0.701, 0.799, 0.803 , 0.807, 0.625) and (0.741, 0.695, 0.776, 0.794) respectively.

VII. Conclusion

Applications of GAN can be seen in different areas of AI. Time-series data generation is no exception and models like TimeGAN and DoppelGANger have been developed to tackle this task. The results of DoppelGANger for time-series generation on customer support dataset were shown in this report. The model achieves decent results. There is however room for improvement. Hyperparameter tuning can help produce better results. Since training the model on CPU instead of GPU takes a long time, hyperparameter tuning can get tedious and take days. A rewrite of the DoppelGANger code on TensorFlow 2 can help the problem. Being able to run the model on GPU is a necessity.

References

1. Lin, Z., Jain, A., Wang, C., Fanti, G. and Sekar, V., 2020, October. Using gans for sharing networked time series data: Challenges, initial promise, and open questions. In Proceedings of the ACM Internet Measurement Conference (pp. 464-483).
2. Yoon, J., Jarrett, D. and van der Schaar, M. (2019) 'Time-series Generative Adversarial Networks', in Wallach, H. et al. (eds) Advances in Neural Information Processing Systems. Curran Associates, Inc.
3. Nickerson, K. et al. (2023) 'Banksformer: A Deep Generative Model For Synthetic Transaction Sequences', in Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2022, Grenoble, France, September 19–23, 2022, Proceedings, Part VI. Berlin, Heidelberg: Springer-Verlag, pp. 121–136. doi: 10.1007/978-3-031-26422-1_8.
4. https://pub.towardsai.net/generating-synthetic-sequential-data-using-gansa1d67a7752ac
5. https://git.tu-berlin.de/navid_ashrafi/doppleganger_tensorflow-2/
6. https://github.com/fjxmlzn/GPUTaskScheduler
7. Schaap, A. et al. (2021) 'Time series Synthesis using GANs - A take on DoppelGANger'
8. https://github.com/gretelai/doppelganger-torch