Synthetic data generation for classification problems

Alejandro Salamanca Ruiz

Abstract: Several techniques to generate synthetic data already exist. Nevertheless, these techniques are used exclusively in image generation scenarios. This work proposes a new approach to generating synthetic data that extrapolates ideas from state-of-the-art models to create a model that can generate synthetic data from categorical and numerical variables. This new technique is compared with other classical approaches, and the results are presented in the final section.

1. Introduction

One of the pain points for Data Science or AI projects is when data is scarce. With all the hype around Big Data and how today we live in a world where data is everywhere, it seems awkward to have problems with low amounts of data. Still, the reality is that collecting data in many businesses applications is not trivial due to security reasons, regulations, ethical issues, or just because, for some companies, gathering large amounts of data is very expensive.

There is where synthetic data comes into play. Today many companies claim to produce synthetic data that is better than real data, but the way they do it remains unclear, and even more their results. A report from Gartner predicts that by 2030 most of the data used in AI will be artificially generated. There are many state-of-the-art approaches to generating unstructured data like images and text. However, this work will tackle the synthetic data generation problem for structured data. Furthermore, this work will only concern categorical, numerical, and ordinal variables in a classification scenario.

Many statistical tools can address the problem of generating data for univariate distributions. Nevertheless, when the data we deal with is multivariate, the problem increases its complexity, and more so if we are not assuming normality in the features' distributions.

For problems like image generation, we find very sophisticated models like VAEs and GANs. The problem with these models is that they are built for generating unstructured data and require many training data. So how can we generate synthetic data from a reduced training sample?

This work proposes a method that extrapolates the architecture of a GAN to generate structured data. The data to be used comes from a dataset originally from CDC (Centre for disease and control prevention). It consists of annual telephone surveys to gather data on the health status of U.S. residents. This dataset includes data from the 2020 survey. It consists of 401,958 rows and 279 columns. For the purpose of this work, a cleaned and pruned version of the dataset done by Kamil Pytlak [1] is used.

Using the same classifier, different experiments are conducted to compare the classifier's performance using the original data, data generated with some SMOTE techniques, data generated with the ADASYN technique, and the newly proposed method.

The experimental results are shown in the experimental approach and experiments section, while conclusions and further considerations are given in the conclusions section.

2. Literature review

There are many methods for overcoming the small dataset problem, particularly the imbalanced data problems. The simplest ones rely on repeating data points. This is called oversampling. This method generates synthetic data points by randomly repeating data points for some class or following some probabilistic approach. The issue with these approaches is that they repeat the original data, so they are not generating new synthetic data. The Synthetic Minority Over-sampling Technique (SMOTE) proposed by Chawla et al.[2] solves this problem. It proposes a method that samples a data point A from a specific class, finds its k-nearest neighbors, then randomly selects one of the neighbors B, and from the line that connects A and B in the feature space, draws a point. Many variations of this technique address some issues like the overgeneralization problem that arises from the fact that most of the new data points will be generated in the areas where the majority of the original data is located. The Adaptive Synthetic Sampling Approach (ADASYN) [3] considers the distribution of the data to generate the new synthetic data. Another extension to SMOTE is Borderline-SMOTE [4]. It consists in selecting data points of the minority class that are misclassified to oversample in those regions because they are more influential in the classification problem.

More complex techniques for generating data use deep learning architectures to balance data. Wan et al. Propose a method for high dimensional data generation [5]. The intuition behind their approach is that VAEs can learn complex data distributions. Some samples are drawn from the latent variable space, and finally, a decoder generates new data based on the conditional distribution of the data given the latent variables.

Finally, the Generative Adversarial Networks come into play. This framework proposes a combination of two models: a discriminator and a generator. The discriminative model determines whether a sample is from the model distribution or the

data distribution, while the generator tries to produce fake data that resembles real data. As we can see in the work by Goodfellow et al. [6], these methods are used in scenarios with high-dimensional data as image generation.

3. GAN for synthetic structured data

This work proposes a new approach for generating synthetic data in scenarios where we just have categorical, numerical, and boolean features. The two neural networks are proposed in order to build the complete GAN model.

The generator is a NN composed of several fully connected layers. It takes as input a vector from the latent space (given by parameter) and outputs a vector with the dimension of the original data. The layers have Relu activation functions except the last one, which has a sigmoid function to make the output go from 0 to 1.

The discriminator is a NN composed of several fully connected layers. It takes as input a vector with the same dimension as the original data and outputs a number between 0 and 1. This last layer represents the probability of the input vector being a fake o real sample. The three first layers have a Relu activation function, while the last one has a sigmoid activation.

Then, the final model is the ensemble of these two models. First, the generator creates fake samples that are passed to the discriminator with the same number of real samples. The discriminator learns to differentiate based on the batch of real and fake samples. Then, the generator generates a batch of fake samples again and passes them to the discriminator, which predicts which ones are fake and real. The difference this second time is that the loss will propagate through the two networks, but the updates will be just for the generator parameters. This is how the generator learns to build better samples. Finally, the weights in the generator model are updated based on the performance of the discriminator model. When the discriminator is good at detecting fake samples, the generator updates more. Contrarily, the generator model updates less when the discriminator model is not doing a good job. This GAN is built so that the generator model is only concerned with the discriminator's performance on fake examples.

4. Experimental Approach

As stated before, the data to be used comes from the 2020 survey performed by the CDC. The classification problem for which data will be generated is predicting whether a person has heart disease or not. The data has three types of variables that need to be considered: ordinal, categorical, and numerical variables. A KNN algorithm is trained to perform the classification task. The number of neighbors that the KNN will use through all the experiments is 5.

First, a preprocessing function is built. This preprocessing function takes the original data set, recognizes the types of variables, performs one-hot encoding for categorical and boolean variables, scales the numerical values, and finally stacks everything in one matrix.

Then, the generate_data_classical_models function is built. This function generates synthetic samples with classical methods like SMOTE, ADASYN, Borderline-SMOTE, and SVMSMOTE. The function takes the original data set, the original labels, the target label, and a percentage of new samples that will be created. The method uses the technique given by the parameter to create new samples from the ones that have the target label. The following equation gives the number of new samples:

$$\text{Num\_of\_Samples} = \text{Percentage x num\_Samples\_target label}$$

This function allows further experimentation.

A GAN class is developed to contain the two NNs. This class has all the methods to create both models, generate new synthetic data, sample real data, and train.

The experiments split the original data into a training set and a test set. The training set accounts for 85% of the original data, while the test data accounts for 15%. First, the training data is used for creating a purely synthetic data set. This synthetic data set is then used for training the KNN model, and finally, the model is assessed with the original test set.

This experimental procedure has many parameters to be tunned:

- The size of the original data set
- The hyperparameters of the classical over sampling approaches
- The number of neighbors of the KNN model
- The generative model architecture
- The discriminative model architecture
- The number of synthetic samples generated by each model
- The number of epochs the GAN is trained
- The latent space used for the generative model in the GAN

The parameter space is intractable. Some parameters need to be fixed.

<u>Tunable parameters in the experiments</u>

- The size of the original data set
- The generative model architecture
- The discriminative model architecture
- The latent space of the generative model

<u>Static parameters in the experiments</u>

- The classical models and the GANs will always create 0.8 new samples with respect with the original training samples with label 1, and 0.5 new samples with respect to the original training samples with label 0.
- The hyperparameters of the classical over sampling approaches
- The number of neighbors of the KNN model
- The number of epochs the GAN is trained

These combinations of training specifications allow analyzing how each technique performs concerning a baseline and what is the reach for the amount of data they can generate without compromising the classifier performance.

The metric used to compare performance is Accuracy.

5. Experiments

Frst configuration:

Number of original samples	**Generative model**	**Discriminative model**	**Latent Space**

500	model.add(Dense(self.latent_dim, input_dim=self.latent_dim))model.add(LeakyReLU(alpha=0.2))model.add(Dense(50, activation='relu'))model.add(Dense(52, activation='sigmoid'))	model.add(Dense(26, activation='relu'))
		model.add(Dense(13, activation='relu'))
		model.add(Dense(6, activation='relu'))
		model.add(Dense(1, activation='sigmoid'))
		
		opt = Adam(learning_rate=0.0002, beta_1=0.5)
			30 and 50

Results:

- SMOTE: 0.9466
- SVMSMOTE: 0.9066
- GAN with LS of 30: 0.6133
- GAN with LS of 50: 0.6266

Second configuration:

Number of original samples	**Generative model**	**Discriminative model**	**Latent Space**

1000	model.add(Dense(self.latent_dim, input_dim=self.latent_dim))model.add(LeakyReLU(alpha=0.2))model.add(Dense(50, activation='relu'))model.add(Dense(52, activation='sigmoid'))	model.add(Dense(26, activation='relu'))
		model.add(Dense(13, activation='relu'))
		model.add(Dense(6, activation='relu'))
		model.add(Dense(1, activation='sigmoid'))
		
		opt = Adam(learning_rate=0.0002, beta_1=0.5)
			30 and 50

Results:

- SMOTE: 0.98
- SVMSMOTE: 0.9133
- GAN with LS of 30: 0.1333
- GAN with LS of 50: 0.8266

Third configuration:

Number of original samples	**Generative model**	**Discriminative model**	**Latent Space**

10000	model.add(Dense(self.latent_dim, input_dim=self.latent_dim))model.add(LeakyReLU(alpha=0.2))model.add(Dense(50, activation='relu'))model.add(Dense(52, activation='sigmoid'))	model.add(Dense(26, activation='relu'))
		model.add(Dense(13, activation='relu'))
		model.add(Dense(6, activation='relu'))
		model.add(Dense(1, activation='sigmoid'))
		
		opt = Adam(learning_rate=0.0002, beta_1=0.5)
			30 and 50

Results:

- SMOTE: 0.992
- SVMSMOTE: 0.9106
- GAN with LS of 30: 0.924
- GAN with LS of 50: 0.813

Fourth configuration:

Number of original samples	**Generative model**	**Discriminative model**	**Latent Space**

500	model.add(Dense(self.latent_dim, input_dim=self.latent_dim))
	model.add(LeakyReLU(alpha=0.2))
	model.add(Dense(20, activation='relu'))
	model.add(Dense(30, activation='relu'))
	model.add(Dense(52, activation='sigmoid'))
		model.add(Dense(26, activation='relu'))
		model.add(Dense(13, activation='relu'))
		model.add(Dense(6, activation='relu'))
		model.add(Dense(1, activation='sigmoid'))
		
		opt = Adam(learning_rate=0.0002, beta_1=0.5)
			30 and 50

Results:

- SMOTE: 0.96
- SVMSMOTE: 0.88
- GAN with LS of 30: 0.8133
- GAN with LS of 50: 0.7466

Fifth configuration:

Number of original samples	**Generative model**	**Discriminative model**	**Latent Space**

1000	model.add(Dense(self.latent_dim, input_dim=self.latent_dim))
	model.add(LeakyReLU(alpha=0.2))
	model.add(Dense(20, activation='relu'))
	model.add(Dense(30, activation='relu'))
	model.add(Dense(52, activation='sigmoid'))
		model.add(Dense(26, activation='relu'))
		model.add(Dense(13, activation='relu'))
		model.add(Dense(6, activation='relu'))
		model.add(Dense(1, activation='sigmoid'))
		
		opt = Adam(learning_rate=0.0002, beta_1=0.5)
			30 and 50

Results:

- SMOTE: 0.9333
- SVMSMOTE: 0.9266
- GAN with LS of 30: 0.2133

- GAN with LS of 50: 0.1933

Sixth configuration:

Number of original samples	Generative model	Discriminative model	Latent Space
10000	model.add(Dense(self.latent_dim, input_dim=self.latent_dim)) model.add(LeakyReLU(alpha=0.2)) model.add(Dense(20, activation='relu')) model.add(Dense(30, activation='relu')) model.add(Dense(52, activation='sigmoid'))	model.add(Dense(26, activation='relu')) model.add(Dense(13, activation='relu')) model.add(Dense(6, activation='relu')) model.add(Dense(1, activation='sigmoid')) opt = Adam(learning_rate=0.0002, beta_1=0.5)	30 and 50

Results:

- SMOTE: 0.99
- SVMSMOTE: 0.9213
- GAN with LS of 30: 0.874
- GAN with LS of 50: 0.8933

Seventh configuration:

Number of original samples	Generative model	Discriminative model	Latent Space
500	model.add(Dense(self.latent_dim, input_dim=self.latent_dim)) model.add(LeakyReLU(alpha=0.2)) model.add(Dense(20, activation='relu')) model.add(Dense(30, activation='relu')) model.add(Dense(52, activation='sigmoid'))	model.add(Dense(16, activation='relu')) model.add(Dense(8, activation='relu')) model.add(Dense(1, activation='sigmoid')) opt = Adam(learning_rate=0.0002, beta_1=0.5)	30 and 50

Results:

- SMOTE: 0.9333
- SVMSMOTE: 0.8666
- GAN with LS of 30: 0.32
- GAN with LS of 50: 0.1466

Eight configuration:

Number of original samples	Generative model	Discriminative model	Latent Space
1000	model.add(Dense(self.latent_dim, input_dim=self.latent_dim)) model.add(LeakyReLU(alpha=0.2)) model.add(Dense(20, activation='relu')) model.add(Dense(30, activation='relu')) model.add(Dense(52, activation='sigmoid'))	model.add(Dense(16, activation='relu')) model.add(Dense(8, activation='relu')) model.add(Dense(1, activation='sigmoid')) opt = Adam(learning_rate=0.0002, beta_1=0.5)	30 and 50

Results:

- SMOTE: 0.9733
- SVMSMOTE: 0.94
- GAN with LS of 30: 0.56
- GAN with LS of 50: 0.1466

Nineth configuration:

Number of original samples	Generative model	Discriminative model	Latent Space
10000	model.add(Dense(self.latent_dim, input_dim=self.latent_dim)) model.add(LeakyReLU(alpha=0.2)) model.add(Dense(20, activation='relu')) model.add(Dense(30, activation='relu')) model.add(Dense(52, activation='sigmoid'))	model.add(Dense(16, activation='relu')) model.add(Dense(8, activation='relu')) model.add(Dense(1, activation='sigmoid')) opt = Adam(learning_rate=0.0002, beta_1=0.5)	30 and 50

Results:

- SMOTE: 0.9933
- SVMSMOTE: 0.904
- GAN with LS of 30: 0.6726
- GAN with LS of 50: 0.9006

These configurations were performed to experiment how

- Different data set sizes influence synthetic data generation
- Incrementing the layers of the generative model influences quality of synthetic data
- Decreasing the layers of the discriminative model influences quality of synthetic data
- Latent space dimension influences synthetic data generation

Implementing the experiments, failures with ADASYN and Borderline SMOTE techniques were noticed. Due to the high dimensionality of the preprocessed data, these techniques could not generate new samples. ADASYN for example, could not generate samples because more synthetic data is generated for minority class samples that are harder to learn than those minority samples that are easier to learn.

The results bring an excellent behavior of SMOTE technique with all data set sizes followed by the SVMSMOTE. The worst performance of the SMOTE technique results in an accuracy of 0.93, while the best performance is 0.992.

On the other side, the new method was very unstable. No pattern could be extrapolated from the experiments. Some experiments with few samples had good and bad performances simultaneously (0.81 and 0.14 for two 500 samples configurations). The same unpredictable behavior happens when tunning the generator, the discriminator, and the latent space. The best performance of the new method is in the third configuration (0.924), but it does not reach the performance of the SMOTE technique in that same configuration.

6. Conclusions

The newly proposed method did not prove effective against the classical techniques. On the other hand, the SMOTE technique worked well even in a high-dimensional space. Nevertheless, the new approach reached promising results as high accuracies between 0.8 and 0.9 were reached. These are promising results because a noticeable accomplishment is reaching an accuracy of 0.9 in a test set (with real data) that the model has never seen with training data that is completely synthetic. Also, many other experiments could improve the results of this new approach by trying other architectures, other numbers of epochs, learning rates, and other data sets.

7. References

[1] Kamil Pytlak. (February 2022). [Personal Key Indicators of Heart Disease], https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease

[2] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “Smote: synthetic minority over-sampling technique,” Journal of artificial intelligence research, Vol. 16, 2002, pp. 321–357.

[3] Haibo He, Yang Bai, E. A. Garcia and Shutao Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), 2008, pp. 1322-1328, doi: 10.1109/IJCNN.2008.4633969.

[4] Han, H., Wang, WY., Mao, BH. (2005). Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In: Huang, DS., Zhang, XP., Huang, GB. (eds) Advances in Intelligent Computing. ICIC 2005. Lecture Notes in Computer Science, vol 3644. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11538059_91

[5] Z. Wan, Y. Zhang and H. He, "Variational autoencoder based synthetic data generation for imbalanced learning," 2017 IEEE Symposium Series on Computational Intelligence (SSCI), 2017, pp. 1-7, doi: 10.1109/SSCI.2017.8285168.

[6] Goodfellow Ian J., Pouget-Abadie Jean, Mirza Mehdi, Xu Bing, Warde-Farley David, Ozairt Sherjil, Courville Aaron, Bengio Yoshua "*Generative Adversarial Nets",* *https://doi.org/10.48550/arxiv.1406.2661*