Data Augmentation Techniques for Tabular Data

Whitepaper by Abinaya Mahendiran, Manager – Data Science, NEXT Labs | Vedanth Subramaniam, Intern, NEXT Labs

Contents

1.
Introduction

Data is the core of any Machine Learning (ML) or Deep Learning (DL) algorithm. Data must be provided in a format that an algorithm can understand. The main function of the ML/DL algorithm is to find patterns or information available in the data which is otherwise hidden from a human. The algorithm will fail to learn the underlying information if the data is not sufficient or if it is not in a suitable format.

2.
Why is Data so Important?

The golden rule of Machine Learning is that an algorithm is only as good as the data it's fed. 95% of advertisers, per the Interactive Advertising Bureau's Data Center of Excellence, have terabytes of user data, location information and user interest they can use to target prospects. But to effectively use ML, advertisers and marketers must have the right data for the problems they are trying to solve when building predictive models. Machine Learning also requires the data to be properly formatted, cleaned and organized to enable data scientists to develop predictive models. Cleaning, maintaining and governing data is just one part of developing ML models.

Any Machine Learning project involves two key steps revolving around data:

Data pre-processing

Data pre-processing is a process of preparing the raw data and converting it into a suitable format for an ML model to consume. It is the first and crucial step in the ML process. In any ML project, the data doesn't come readily formatted. It is mandatory to clean and prepare the data in the right format. Most ML practitioners face the challenge of processing data accurately and efficiently. It is believed that around 80% of a data scientist's time is spent on pre-processing the data and making it usable.

Feature engineering

For an ML model to be effective in predicting results, the algorithms must receive structured and relevant data. Feature engineering is the process of using domain knowledge to extract features (characteristics, properties, attributes) from raw data. A feature is a property shared by independent units on which analysis or prediction has to be done.

After completing the above two vital steps, model training is performed. Model training is an iterative procedure where the hyperparameters of the model are fine-tuned to achieve better and optimal results for the problem at hand. But tuning the hyperparameters or tinkering with the ML model might not always yield good or considerable results. After a certain number of iterations, if the model does not converge, then either the quality or the quantity of the data is to be questioned, sometimes even both need to be checked.

3. Data Augmentation

Data augmentation is a technique to artificially create new data from existing data and significantly increase the diversity of data available for training the models. This is done by applying domain-specific techniques to the subset of training data. Since the performance of a model is highly dependent on the quality and quantity of the dataset, using synthetically generated data might help improve the model performance to some extent.

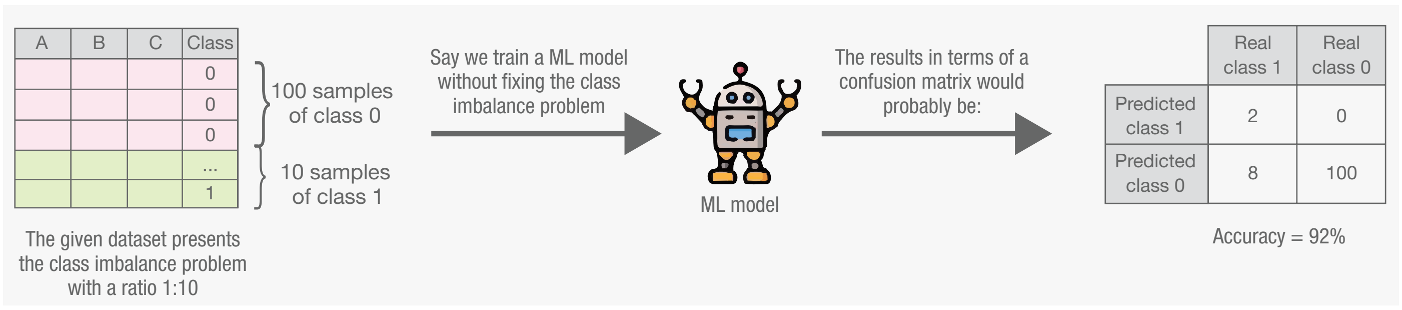
In an ML project, the dataset is divided into three subsets:

- Training set: a subset of data used to train the model
- Validation set: a subset of data used in the unbiased evaluation of a model which will in turn help in tuning model hyperparameters
- Test set: a subset of data used to test the generalization of the model

When the model fails to converge or if the desired result is not obtained, data augmentation can be done on the training set and validation set. Augmenting the test set would compromise the model. When dealing with image data, augmentation techniques such as cropping, padding and horizontal flipping are commonly used to train large neural networks and have proven to be effective in improving the model's accuracy and generalizing ability. However, when it comes to dealing with tabular data, there has been a lack of focus on improving the data quality through augmentation. In this paper, we explore, how to get started when dealing with class imbalance problems in a tabular dataset.

4. What Happens if You Train on an Imbalanced Dataset?

One of the underlying things to consider when dealing with imbalanced data in a classification problem is which metric to use. Accuracy is commonly used as the de facto metric. However, for the class imbalance problem, it's not right to use accuracy as a metric. This problem is better known as the accuracy paradox.


A
B
C
Class
0
0
0
...
1
100 samples of class 0
10 samples of class 1
The given dataset presents the class imbalance problem with a ratio 1:10
Say we train a ML model without fixing the class imbalance problem
ML model
The results in terms of a confusion matrix would probably be:
Accuracy = 92%

	Real class 1	Real class 0
Predicted class 1	2	0
Predicted class 0	8	100

Figure 1: Accuracy Paradox

When using accuracy as the metric to evaluate a Machine Learning model that was trained with a dataset with the class imbalance problem, the results can be misleading. As we can see in Figure 1, the accuracy is 92%, which would make us assume that the model is good enough. However, if we observe, we can see that the model learned to classify everything towards class 0, which generates the effect of having a good enough accuracy. In these cases, in addition to applying some method to fix the class imbalance problem, it is suggested to introduce other evaluation metrics such as precision, recall and F1-Score.

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The recall is the ratio of correctly predicted positive observations to all the observations in the actual class. Finally, to generalize the precision and recall metrics,

	Real class 1	Real class 0
Predicted class 1	2	0
Predicted class 0	8	100

True Positives = 2

False Positives = 0

True Negatives = 100

False Negatives = 8

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total \# samples}}$$

$$\text{Accuracy} = \frac{2 + 100}{110}$$

$$\boxed{\text{Accuracy} = 0.92}$$

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Precision} = \frac{2}{2 + 0}$$

$$\boxed{\text{Precision} = 1}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{Recall} = \frac{2}{2 + 8}$$

$$\boxed{\text{Recall} = 0.2}$$

$$\text{F1-Score} = \frac{2 \text{ x } (\text{Recall x Precision})}{\text{Recall} + \text{Precision}}$$

$$\text{F1-Score} = \frac{2 \text{ x } (0.2 \text{ x } 1)}{0.2 + 1} = \frac{0.4}{1.2}$$

$$\boxed{\text{F1-Score} = 0.33}$$

Figure 2: Metrics

we implement the F1-Score metric, which is understood as “the harmonic mean” between the precision and recall in other words, it provides a ratio between both metrics. Therefore, judging the model’s performance based on the accuracy metric alone in an imbalanced dataset is erroneous and before proceeding with model training, the dataset needs to be augmented.

5.

Sampling

Sampling is a method that allows us to get information about the population based on the statistics from a subset of the population (sample) without having to investigate every individual.

Data sampling is a collection of techniques that transform the training dataset to balance the class distribution. Once balanced, standard Machine Learning algorithms can be trained directly on the transformed dataset without any modification. This is one way to address the imbalance in the training data. There are many different types of data sampling methods that can be used, and there is no single best method to use on all classification problems. Like choosing the right model, careful experimentation is required to discover which sampling method works best for the project.

Random Oversampling

Random oversampling is the simplest oversampling technique to balance the imbalanced dataset. It balances the data by replicating the minority class samples. This does not cause any loss of information and results in an equal distribution of the majority and minority classes. Oversampling might be prone to overfitting, but is an extremely fast technique that is useful when dealing with large to extremely large tabular data.

Random Undersampling

Random undersampling method randomly removes samples from the majority class, with or without replacement. This is one of the earliest techniques used to alleviate the imbalance in the dataset; however, it may increase the variance of the classifier and is very likely to discard useful or important samples. Like oversampling, this is an extremely fast technique and is useful when handling large data.

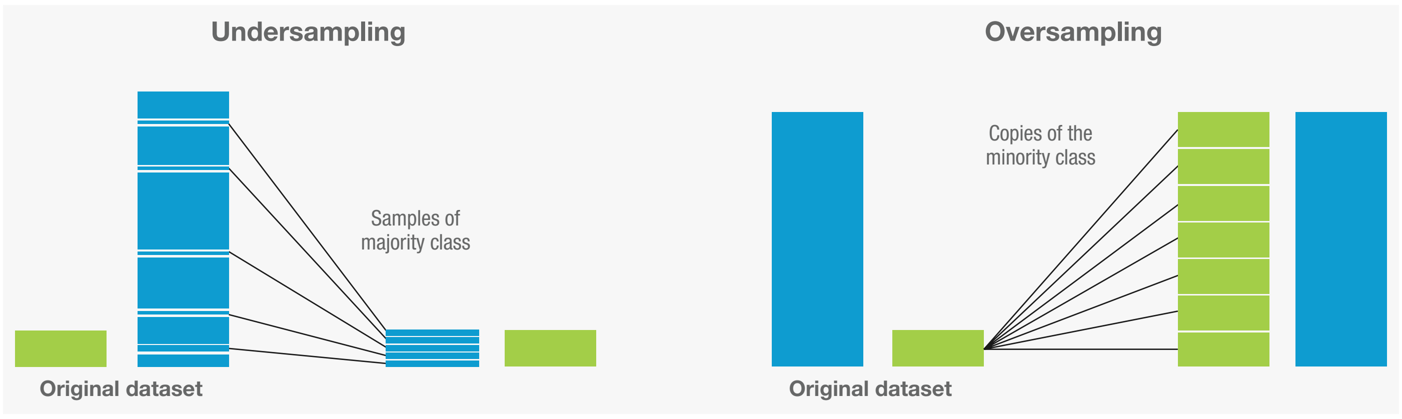


Figure 3: (Left) Undersampling and (Right) Oversampling

SMOTE

Random oversampling is prone to overfitting as the minority class samples are replicated. Overfitting can be avoided by SMOTE. SMOTE stands for Synthetic Minority Oversampling Technique. It creates new synthetic samples to balance the dataset.

SMOTE works by utilizing a k-Nearest Neighbor algorithm to create synthetic data. Samples are created through the following steps:

- Identify the feature vector and its nearest neighbor
- Compute the distance between the two sample points
- Multiply the distance with a random number between 0 and 1
- Identify a new point on the line segment at the computed distance
- Repeat the process for identified feature vectors

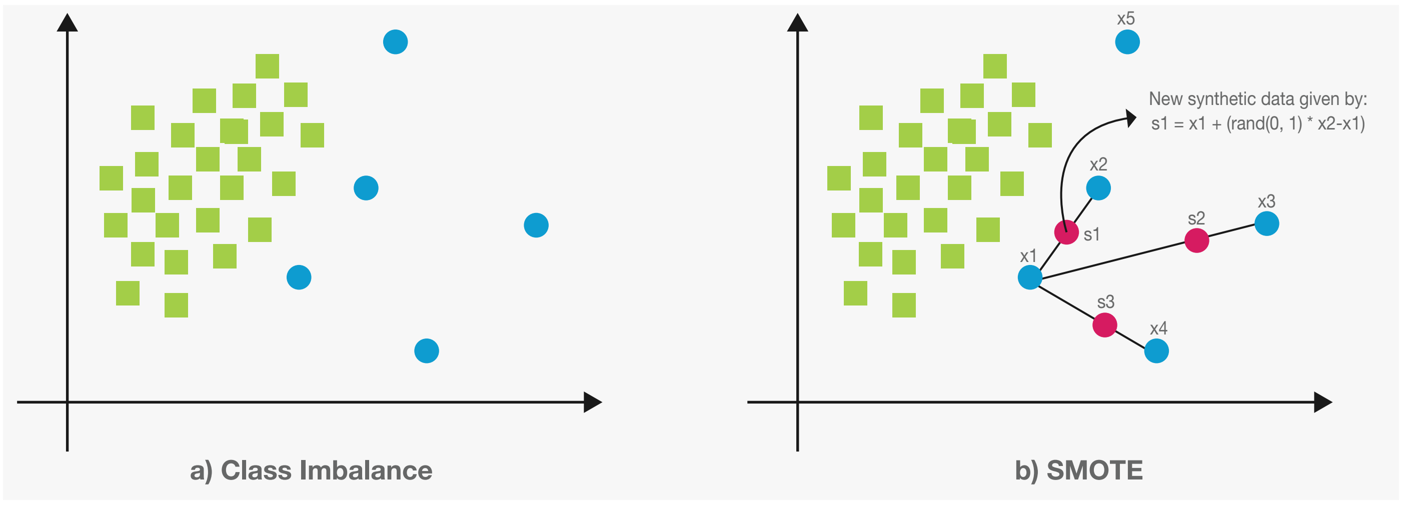


Figure 4: SMOTE

SMOTE-NC

SMOTE oversampling technique only works for the dataset with all continuous features. For a dataset with categorical features, Smote-NC (Nominal and Continuous) is used for sampling.

SMOTE can also be used for data with categorical features by one-hot encoding, but it may result in increased dimensionality. Therefore, SMOTE-NC is used when there are categorical features. SMOTE-NC can be used by denoting the features that are categorical, and SMOTE would resample the categorical data instead of creating synthetic data.

6.

Autoencoders-based Augmentation

Autoencoder is an unsupervised artificial neural network that learns how to efficiently compress and encode data and performs reconstruction of the data that is as close to the original input as possible. Autoencoder, by design, reduces data dimensions by learning how to ignore the noise in the data.

A Variational Autoencoder (VAE), with a slight variation in the architecture, provides a probabilistic method for describing an observation in latent space. Thus, rather than building an encoder that outputs a single value to describe each latent state attribute, VAE provides a probability distribution for each latent attribute. By constructing the encoder model to output a range of possible values (a statistical distribution), sampling can be randomly done from the distribution so that it can be fed into the decoder model.

The latent space is continuous in nature. VAE performs sampling by outputting a 2-dimensional vector (mean and variance) from a random variable. The vector is then used to get a sampled encoding which is passed to the decoder. As encodings are generated from a distribution with the same mean and variance as those of the inputs, the decoder learns from all nearby points referred to as the same latent space.

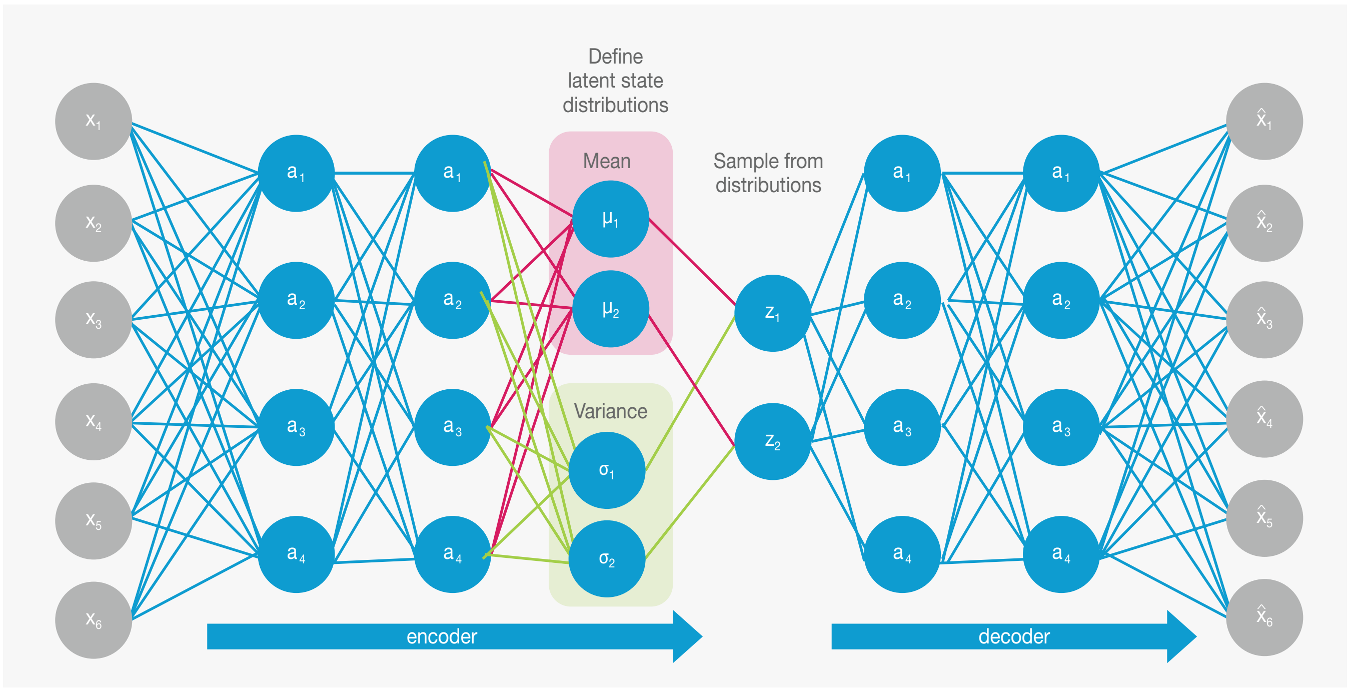

Define latent state distributions
Mean
Variance
Sample from distributions
x_1 x_2 x_3 x_4 x_5 x_6
a_1 a_2 a_3 a_4
μ_1 μ_2
σ_1 σ_2
z_1 z_2
$\hat{x}_1$ $\hat{x}_2$ $\hat{x}_3$ $\hat{x}_4$ $\hat{x}_5$ $\hat{x}_6$
encoder
decoder

Figure 5: Variational Autoencoders

Rather than directly outputting values for the latent state in a standard autoencoder, the encoder model of a VAE will output parameters describing a distribution for each dimension in the latent space. The assumption here is that the prior follows a normal distribution, so the two output vectors describe the mean and variance of the latent state distributions.

The decoder model will then generate a latent vector by sampling from these defined distributions and proceed to reconstruct the original input. Values that are close to one another in latent space should result in very similar reconstructions. However, this sampling process requires some extra attention. When training the model, the relationship of each parameter in the network is tuned to the final output loss using a technique known as backpropagation. However, this technique cannot be applied to a random sampling process. The reparameterization trick is leveraged where sample ε from a unit Gaussian is randomly sampled, and the sampled ε is randomly shifted by the latent distribution's mean μ and scaled by the latent distribution's variance σ. With this reparameterization, parameters of the distribution are optimized while still maintaining the ability to randomly sample from the distribution.

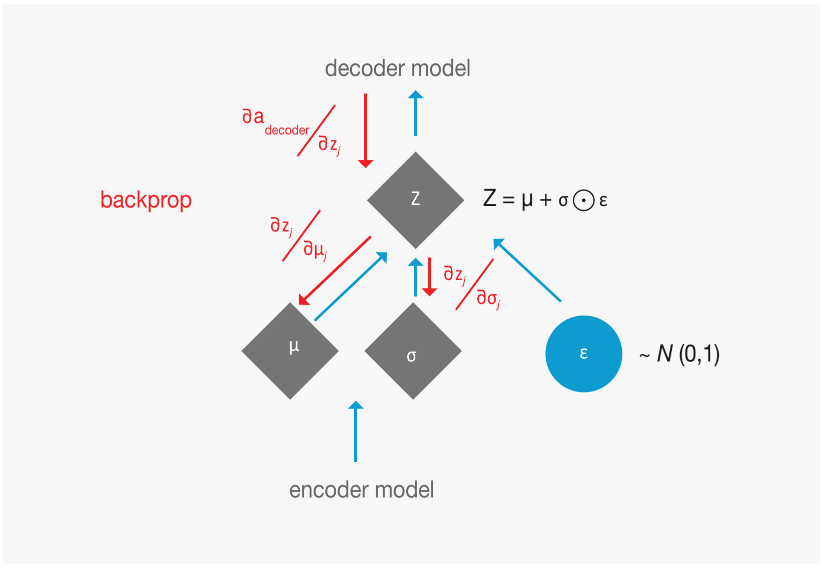

decoder model
$\partial a_{decoder} / \partial z_j$
backprop
Z
$Z = \mu + \sigma \odot \varepsilon$
$\partial z_j / \partial \mu_j$
$\partial z_j / \partial \sigma_j$
μ
σ
ε
~ N (0,1)
encoder model

Figure 6: Encoder and Decoder

The Variational Autoencoders (VAE) approach can be used to augment datasets for classification or regression tasks. For a classification task, the minority classes can be augmented into containing the same number of samples as the majority class, thus ensuring that the class imbalance is handled. In the case of a regression task, the desired number of samples can be generated.

7.
Conclusion

The data augmentation techniques described in the paper are applicable to tabular data in general. There are several other techniques that are available for textual, audio, video and image data. Most of the use cases that come up in the industry will require tabular data as the first step, but the availability might be scarce. In those scenarios, any of the above techniques can be used to increase the diversity and the quantity of tabular data.

Authors

Abinaya Mahendiran

Manager – Data Science, NEXT Labs

Abinaya Mahendiran holds a master's degree in Computer Science with a specialization in Machine Learning and Deep Learning from the International Institute of Information Technology Bangalore (IIIT-B). Her research areas include Natural Language Understanding/Processing, Machine Learning, Deep Learning and MLOps. She has extensive software engineering and data science experience. At NEXT Labs, she has been building and productionizing NLU/NLP solutions for various clients both on-premise and on the cloud. She loves contributing to open-source projects in her free time.

Vedanth Subramaniam

Intern, NEXT Labs

Vedanth is a Machine Learning intern at Mphasis NEXT Labs. His areas of interest include Computer Vision, Natural Language Processing and Explainable AI. He is currently pursuing his Bachelor's in Engineering at the College of Engineering Guindy, Anna University, and wants to specialize in the field of Deep Learning.