Instituto Superior Técnico, Feedzai

MECD

Tabular Data Normalization for Deep Learning

Author:
Rita Leite
92646
February 2023

1 Introduction

Deep Learning has proven to be a powerful tool, with outstanding performance in various realms, from visual recognition to language processing tasks. However, its success has not yet been widely extended to tabular data.

Tabular data is the most common form of data, present in a variety of real world applications, such as click through rate for adds prediction, credit risk prediction and anomaly detection.

Feedzai is a company that specializes in using AI to fight financial fraud. Among the solutions they develop, one is a software to detect instances of fraud in online bank account opening. This task is a complex one, as it requires maintaining a high sensitivity in order to prevent instances of fraud, while guaranteeing genuine consumers are not flagged, which could alienate costumers and be detrimental to Feedzai's clients.

Recently, new Deep Learning architectures have been proposed to deal specifically with tabular data, and have proven successful in various fields (see [11] and [6]). However, research on this area suffers from one major weakness: the lack of publicly available large scale real world data. To address this obstacle, Feedzai has made publicly available a suite of tabular data artificially generated on a real world bank account opening fraud datasets (see [8]).

Due to the heterogeneity and disparity in distribution of its features, tabular data presents a unique challenge to Deep Learning methods, which are highly sensitive to the scale and ordering of the input. To bridge this gap and ensure a stable learning process, normalization techniques are key. The goal of this project is therefore to make a realistic and fair comparison of different Normalization techniques, and assess their impact on Deep Learning models, in particular in the Fraud Detection Field.

The rest of the paper is organized as follows: Section 2 presents a review of the most recent literature on Deep Learning for tabular data. Section 3 delves into the primary challenges that arise when working with tabular data in the context of Deep Learning. In section 4, the methods tested in this paper are presented in depth and in section 5 the performed experiment is presented and the results analyzed in depth. Finally, section 6 contains the conclusion and future work.

2 Related Work

Strategies to deal with tabular data and make it more amenable to Deep Learning methods are quite diverse, but can be separated according to the approach they take. In [2], the authors propose dividing these techniques into 3 main categories: **Transformation Methods**, **Specialized Architectures** and **Regularization Methods**.

Transformation Methods encompass all transformations and encoding methods done to both numerical and categorical features, during the preprocessing stages of the model training. These include encodings for categorical features (such as label encoding or one-hot encoding) or more complex methods, such as VIME [13]. Similarly, [3] proposes training a Gradient Boosted Decision Tree on the target value, to then use the node conditions of the fitted tree to derive a new dataset of binary values, which can then be fed to any neural network.

Specialized Architectures consist of novel neural network architectures developed specifically to deal with tabular data. Examples of these include the TabTransformer [7], which employs transformer layers to gain more information on the categorical features, the TabNet [1] and the Deep Cross Network [12], both of which are particularly concerned with sparse features.

The final category are **Regularization Methods**, which concerns all regularization techniques used in Deep Learning, from Weight Decay, to having Dropout Layers and Batch Normalization. In [9], the authors defend that simply using an optimal combination of these techniques can lead to simple Multi-Layer Perceptrons excelling on their given tasks.

For the purpose of this study, the focus will be

primarily on the Transformation Methods, although a Specialized Architecture (the TabTranformer), was also tested.

3 The Challenges of Tabular Data

Unlike homogeneous data, such as images, text and audio, which neural networks are specifically geared to handle, tabular data has certain characteristics that make it more complex and harder to manipulate. Additionally, tabular data can be costly to obtain, as it often requires feature engineering or some other kind of manual labor. This means fewer observations available and often large class imbalance.

Firstly, data quality is a major concern when dealing with tabular datasets. Missing values and outliers are especially ubiquitous in tabular data, and must be handled with care.

On the other hand, tabular data is heterogeneous by nature, with a mix of categorical and numerical features, and no existing restrictions on how each of these features are distributed or relate to one another.

Typically, categorical data is handled in one of two ways. The first is simply performing label encoding, meaning, attributing to each category an integer that will be henceforth associated with that class. This can be problematic, as it encodes an order or numerical meaning to data that might not have one. The second approach, one-hot encoding, mitigates this issue, by creating a new column for each different category. However, it often leads to an increased dimensionality of the data. This is especially problematic in instances of categories with high cardinality, which can be quite common in tabular data. Consider for example, the case of click through rate prediction or recommendation systems, where demographic information can play an important role in the decision making process. Variables such as a user's country of residence and occupation can have a considerable amount of possible categories. This in turn leads to increasingly sparse data.

4 Methods

4.1 Treatment for numerical features

When dealing with numerical features, the primary issue to be dealt with is the varying scales along the different variables. Having features with similar distributions is beneficial in order to make the training process faster and to prevent the optimization process becoming stuck in a local minima. In order to address this issue, various transformation procedures can be used.

4.1.1 Transformation Methods

Changing the scale and location of the features is one option available to guarantee the data is more homogeneous and shares in the same distribution. For the purpose of this project, 3 normalization techniques were chosen.

Z-score Normalization

In Z-score normalization, all the features are normalized in order to guarantee their overall mean is 0 and standard deviation 1, and thus individually approach the distribution $\mathcal{N}(0, 1)$.

Let $\mathbf{z}_i$ be the normalized version of the feature vector $\mathbf{x}_i$, s_i be the sample standard deviation for the same feature. Z-score normalization is defined as follows:

$$\mathbf{z}_{zscore,i} = \frac{\mathbf{x}_i - \bar{\mathbf{x}}_i}{s_i} \tag{1}$$

Min-Max Normalization

Min-Max normalization uses the minimum and maximum observation of a column to guarantee its normalized version ranges between 0 and 1.

$$\mathbf{z}_{minmax,i} = \frac{\mathbf{x}_i - min(\mathbf{x}_i)}{max(\mathbf{x}_i) - min(\mathbf{x}_i)} \tag{2}$$

Robust Normalization

This normalization technique is similar to Z-score normalization, but instead uses robust estimators of scale and location, therefore attenuating the effect of outliers. Let $median(\mathbf{x}_i)$ be

the sample median for the $i-th$ feature, and $s_{IQR,i}$ be the sample standard deviation, taking only into consideration only the values inside the Interquartile Range (IQR).

$$\mathbf{z}_{robust,i} = \frac{\mathbf{x}_i - median(\mathbf{x}_i)}{s_{IQR,i}} \tag{3}$$

4.1.2 Numerical Embeddings

The goal of numerical embeddings is to find an n-dimensional representation of a numerical feature. With this in mind, we use an embedding for numerical features as proposed in [5]. In this article, the authors propose the Piecewise Linear Encoder (PLE).

Algorithm 1: Piecewise Linear Encoder

Data: Numerical feature vector $\mathbf{x}_i$, Embedding dimension n

Step 1: Split value range of $\mathbf{x}_i$ into a set of n disjoint intervals $B_1^i, ..., B_n^i$, where $B_t^i = [b_{t-1}^i, b_t^i)$

Step 2: Set vectorial representation as $PLE(x_{ij}) = [e_1, ..., e_n]$, where

$$e_t = \begin{cases} 0, \ x < b_{t-1} \wedge t > 1 \\ 1, \ x > b_t \wedge t < n \\ \dfrac{x - b_{t-1}}{b_t - b_{t-1}}, \text{ otherwise} \end{cases}$$

The PLE works by, for a given numerical feature, splitting its value range into a set of disjoint intervals, $B_1, ..., B_T$, where T is the pre-defined embedding dimension. Once this is done, the embedding components corresponding to bins with right boundaries lower than the observed value are set to 1. The component corresponding to the bin the observed value belongs to is replaced by a standardized version of the value, and the rest are set to 0. Algorithm 1 shows the procedure of PLE. When it comes to obtaining the range for the intervals B_t^i (or bins), the approach chosen is to use Target Aware Bins, which can be computed by training a decision tree on the column $\mathbf{x}_i$ to predict the target value, and using the obtained regions for its leaves as the bins. The number of nodes of the decision tree must be set as n.

4.2 Techniques for Categorical Features

4.2.1 Encoding Techniques

To address the previously mentioned challenges of dealing with categorical features, a good solution is to map them into continuous representations. With this goal in mind, three different encoding techniques were used: Target Encoding, Catboost Encoding[1] and Count Encoding.

Target Encoding

Target encoding consists of replacing each feature with the posterior probability of the target for each given category. In this case, it is assumed that the prior probability of the target over all of the training data $P(Y = 1)$ is the same as $P(Y = 0)$. Let X_{train} be the training set, n its number of samples, and $\mathbf{x}_i$ be the $i-th$ categorical feature with j one of its possible categories. Thus, the following mapping is performed:

$$\begin{aligned} \phi_{ij} &= P(Y = 1 | \mathbf{x}_i = j) \\ &= \frac{|x \in X_{train} : x_{ki} = j \wedge y_k = 1\}|}{|X_{train}|}, k = 1 : n \end{aligned} \tag{4}$$

CatBoost Encoding

Catboost Encoding works on the same principle as Target Encoding. However, unlike the previous encoder, it does not consider all of the existing observations of a category to calculate the posterior probability. Instead, it only considers the observations seen up to the current one, with an additional permutation being made to the data before beginning the encoding process. This method was proposed in [4], and can be used to avoid leakage of information on the target values.

Count Encoding

Count Encoding does not require any knowledge

[1] both CatBoost and Target Encoding are supervised methods which measure the effect of each category might have in the target statistics

on the target value and, thus, can be considered an unsupervised method. In it, each category is simply replaced with the count of observations in the training that belonging to said category.

$$\phi_{ij} = \frac{|\{x_k \in X_{train} : x_{ki} = j\}|}{|X_{train}|} \tag{5}$$

4.2.2 Embeddings

The goal of embedding layers in Deep Learning architectures is to learn an n-dimensional vectorial representation of categorical data. They have been applied with great success in various areas, as is the case of Natural Language Processing [10]. Essentially, an embedding layer consists of a look-up table of weights.
Consider the feature categorical vector $\mathbf{x}_i$, containing k categories, and consider the embedding matrix $\mathbf{W}^i$ with dimension $(n \times k)$. Thus, the n-dimensional embedding for the $j-th$ category is given by:

$$\psi_j^i = \mathbf{W}_j^i \tag{6}$$

For the purpose of this study, an embedding will be learned for each categorical feature, and its weights fitted along with the rest of the model.
For setting the dimension of the embeddings, the following rule is used:
Let m_j be the embedding dimension for the $j-th$ categorical features, n_{cat} the number of categorical features in the dataset and α an adjustable hyperparameter that scales the embedding dimensions.

$$m_j = \alpha[\sqrt{\#\mathbf{x}_j}],\ j = 1 : n_{cat} \tag{7}$$

4.3 TabTransformer

The TabTransformer is a tabular data modeling architecture proposed in [7]. It makes use of the transformer architecture to provide and exploit contextual embeddings of categorical features, taking as inspiration the success self-attention based transformers have had in NLP. It has three main blocks: the first is an embedding layer for categorical features. Afterwards, the embedded columns are fed to a stack of transformers, and the resulting output is then combined with the normalized numerical columns and passed through a Multi-Layer Perceptron which outputs the result. A scheme of the architecture can be seen in 1.

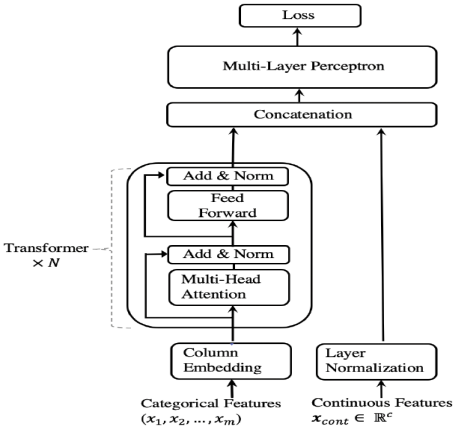

Figure 1: Architecture of the TabTransformer [7]

5 Experiment

5.1 Experiment Set up

In order to test the different techniques to deal with tabular data, three publicly available datasets were used:

- **BAF**[2]: the Bank Account Fraud dataset was provided by Feedzai (see [8]). Each entry contains a synthetically generated bank account opening application, along with a binary variable with information on whether the operation is fraudulent. The dataset contains 31 features, 5 of which are categorical. However, none of the categorical features have a large number of categories available, with the highest cardinality on the training set being 9. The prevalance is of 0.01025 and 0.01474 in the train and test set respectively.

[2] https://www.kaggle.com/datasets/sgpjesus/bank-account-fraud-dataset-neurips-2022
[3] https://www.kaggle.com/competitions/avazu-ctr-prediction/data

- **CTR**[3]: the Click Through Rate dataset is to for a click prediction task. It contains 21 categorical features, some of which with over 2000 different categories on the training set. The prevalence in the train and test set is of 0.1650 and 0.1762 respectively.
- **Fraud**[4]: the fraud dataset contains transactions made by credit cards in september 2013. It contains 29 numerical features, which are the result of a PCA transformation (done to keep confidential information from the data). The prevalence in the train and test set is of 0.001830 and 0.00077 respectively.

Dataset	# Train	# Val	# Test
Fraud	227844	28480	28480
BAF	794989	108168	96843
CTR	600000	200000	200000

Table 1: Dataset Dimensions

Both the **BAF** and **Fraud** datasets are highly unbalanced, another common issue in the realm of Fraud Detection. To mitigate this, the Area Under the ROC curve will be used as a metric to evaluate the different methods, along with the Recall at a 5% False Positive Rate .
The Multilayer Perceptron is used as the base for the Deep Learning methods that will be tested. To test the effect of all the different methods presented in Section 4, all combinations of categorical and numerical treatment techniques are considered.
The LightGBM method was chosen as the baseline against which the Deep Learning models will be tested. Because of the nature of this algorithm, normalization techniques are not expected to greatly impact its performance.
To find the optimal hyperparameters, a Random Search is performed on the relevant hyperparameters for all methods, with 20 different models being trained for each of them, and the best ones being selected according to the AUROC on the validation set. The hyperparameter spaces used can be seen in Appendix A.

5.2 Results

In Table 3 are presented the results for the AUC and Recall at a FPR of 5% scores for the test set of the best performing model on the validation set.

Normalization for Numerical Features

The results show that Zscore and MinMax normalization consistently outperform Robust Normalization and not using any normalization in both the **BAF** and **CTR** datasets. When looking to the **Fraud** dataset we find that Robust Normalization manages to reach a better AUC score. It is important to note that its features are the result of performing Principal Component Analysis on an original dataset, which is not ideal when trying to compare the effects of the different normalization methods.

Numerical Embeddings

Numerical embeddings consistently outperform other methods for treating numerical features, both in terms of recall and AUC. Looking to figure 2, it is clear that using numerical embeddings in general leads to a significant improvements in the Recall when compared to other normalization methods. This result is particularly interesting, as it is not standard practice to use embeddings for numerical features.

On Encodings for Categorical Features

The usage of encodings for categorical features has proven to be a computationally inexpensive and effective way to deal with this kind of features. As we are converting these features to numerical ones, it is therefore suitable to normalize them, obtaining in the process a more homogeneous data distribution. Specifically, CatBoost and Target Encoding outperform all other methods of Categorical Features. Indeed, when paired with numerical normalization, the resulting MLP's are capable of closely matching the LightGBM algorithm on Recall(5%) and AUC.

[4] `https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud`

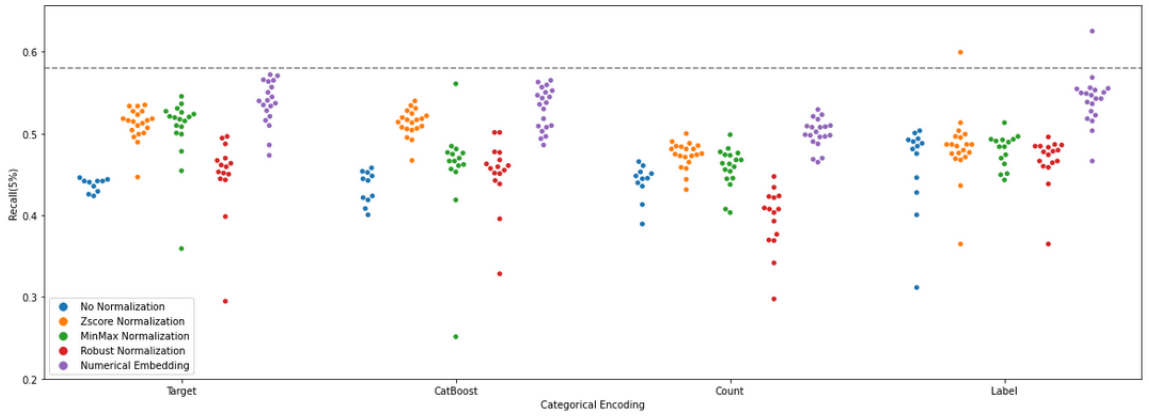

Figure 2: Swarmplot showing the Recall on a 5% FPR per type of categorical encoding. Results shown are for the test set of the **BAF** dataset. Dashed line represents recall for best LightGBM model on test set

On the effectiveness of Categorical Embeddings

The usage of categorical embeddings in the **BAF** proved surprisingly ineffective in comparison to using any other kind of encoding, as can be seen in Table 3. It is known that, for this dataset, the feature with the most categories has a cardinality of 9, which is relatively low. In order to understand how informative the categorical features are, a set of classification models are trained on the LightGBM algorithm, with only continuous features. The results for the AUC score for the LightGBM, with and without categorical features are 0.89902 and 0.8737 respectively (shown in Table 3). The decrease in this metric is small enough to consider the possibility that the categorical features are not informative enough, which could justify the weak performance of the categorical embeddings. Indeed, in the presence of uninformative features, the increase in dimensions caused by the embedding layer could result in inducing additional noise to the training process.

The same, however, is not true for the **CTR** dataset, where using categorical embeddings leads to results comparable with those of the other categorical encodings. In contrast to **BAF**, the **CTR** data has only categorical features, some of which with over 2000 available categories. This leads to an important conclusion: no solution fits all, and when dealing with tabular data it is important to be mindful of the characteristics of the data at hand.

TabTransformer

The TabTransformer yielded good results in the CTR dataset, ranking second for AUC in the test set, behind the LightGBM. However, its results for the BAF dataset were weaker, with this model barely beating the results given by a simple MLP with Label Encoding and no normalization. This is an indication that the TabTransformer is not effective when dealing with low dimensional and uninformative categorical features, most likely due to its complexity.

Model	# params.	$\bar{t}_{epoch}$
TabTransformer	18.4K	82.757s
MLP w. Embedding	873	24.915s
MLP	304	12.011s

Table 2: Number of trainable parameters and average training time for one epoch, for batch size of 1024. Results obtained on the **BAF** dataset

Indeed, the complexity of the TabTranformer is a factor that must be taken into account, as the

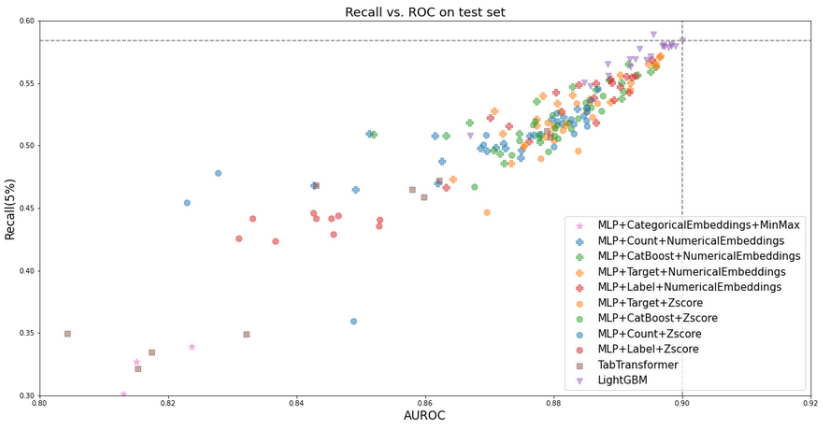

Figure 3: Recall at a 5% FPR vs AUROC (Zoomed). Each dot is a model trained, with a different set of hyperparameters. The scores are for the test set of **BAF** dataset. The dashed lines represent the scores for the best LightGBM model on the validation.

transformer layers naturally result in an increase in hyperparameters that must be trained. Table 2 shows the average training times for a simple MLP, a MLP with an embedding layers and a TabTransformer. Clearly, the TabTransformer, although effective in dealing the categorical features with high cardinality, requires a considerably higher amount of resources to train and store.

LightGBM

The results for the LightGBM without any preprocessing of the data still manage to beat Deep Learning techniques in most cases, and is the most consistent in its performance. Additionally, the scatterplot seen in Figure 3 clearly shows that the LightGBM is the most consistent in its performance, immediately followed by the MLPs trained with numerical embeddings. Indeed, looking to Table 3, it can be seen that some MLP models closely resemble those obtained by the LightGBM.
Despite this, it is important to note that the preprocessing methods that give the MLP its edge (the Numerical Embeddings and Target and CatBoost encodings) are supervised. It is therefore worth considering the possibility that the MLP may have an unfair advantage over the LightGBM. To study this possibility, the LightGBM model was trained over already preprocessed data treated with the above mentioned methods. The results are shown together with the rest in Table 3.
It is found that the usage of any preprocessing method in the LightGBM leads to a slight decrease in performance, showing that the LightGBM does not benefit from performing any kind of treatment to the data.

6 Conclusion

Through the course of this study, it has been possible to conclude that Deep Learning techniques are capable of matching Gradient Boosting Tree Ensembles through performing the right transformations to tabular data. In the course of this study, it has been found that no solution fits all, meaning that careful consideration must be given to the data at hand.

Lastly, it has been found that using embeddings for numerical features (as proposed in [5]) can greatly improve the performance of a simple Multi-Layer Perceptron. For the future, a deeper analysis of specialized architectures for tabular data should be performed. Additionally, and taking into consideration the success of using supervised transformation methods, testing encoding methods that do not require independence between different features, and instead map the whole set of features to a new representation could present interesting results.

References

[1] Sercan Ömer Arik and Tomas Pfister. "TabNet: Attentive Interpretable Tabular Learning". In: *CoRR* abs/1908.07442 (2019). arXiv: 1908.07442. URL: http://arxiv.org/abs/1908.07442.

[2] Vadim Borisov et al. "Deep Neural Networks and Tabular Data: A Survey". In: *CoRR* abs/2110.01889 (2021). arXiv: 2110.01889. URL: https://arxiv.org/abs/2110.01889.

[3] Vadim Borisov et al. "DeepTLF: robust deep neural networks for heterogeneous tabular data". In: *International Journal of Data Science and Analytics* (2022). ISSN: 23644168. DOI: 10.1007/s41060-022-00350-z.

[4] Anna Veronika Dorogush et al. "Fighting biases with dynamic boosting". In: *CoRR* abs/1706.09516 (2017). arXiv: 1706.09516. URL: http://arxiv.org/abs/1706.09516.

[5] Yury Gorishniy, Ivan Rubachev and Artem Babenko. *On Embeddings for Numerical Features in Tabular Deep Learning*. 2022. DOI: 10.48550/ARXIV.2203.05556. URL: https://arxiv.org/abs/2203.05556.

[6] Yury Gorishniy et al. "Revisiting Deep Learning Models for Tabular Data". In: *CoRR* abs/2106.11959 (2021). arXiv: 2106.11959. URL: https://arxiv.org/abs/2106.11959.

[7] Xin Huang et al. "TabTransformer: Tabular Data Modeling Using Contextual Embeddings". In: *CoRR* abs/2012.06678 (2020). arXiv: 2012.06678. URL: https://arxiv.org/abs/2012.06678.

[8] Sérgio Jesus et al. "Turning the Tables: Biased, Imbalanced, Dynamic Tabular Datasets for ML Evaluation". In: *Advances in Neural Information Processing Systems* (2022).

[9] Arlind Kadra et al. "Regularization is all you Need: Simple Neural Nets can Excel on Tabular Data". In: *CoRR* abs/2106.11189 (2021). arXiv: 2106.11189. URL: https://arxiv.org/abs/2106.11189.

[10] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[11] Weiping Song et al. "AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks". In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. CIKM '19. Beijing, China: Association for Computing Machinery, 2019, pp. 1161–1170. ISBN: 9781450369763. DOI: 10.1145/3357384.3357925. URL: https://doi.org/10.1145/3357384.3357925.

[12] Ruoxi Wang et al. "Deep & Cross Network for Ad Click Predictions". In: *CoRR* abs/1708.05123 (2017). arXiv: 1708.05123. URL: http://arxiv.org/abs/1708.05123.

[13] Jinsung Yoon et al. "VIME: Extending the Success of Self- and Semi-supervised Learning to Tabular Domain". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 11033–11043. URL: https://proceedings.neurips.cc/paper/2020/file/7d97667a3e056acab9aaf653807b4a03-Paper.pdf.

A Hyperparameter Grid Space

A.1 Multi-Layer Perceptron

- *activation function*: $ReLU$, $LeakyReLU$, $Tanh$
- *learningrate* $\in [0.00001, 0.01]$
- *dropout rate* $\in [0, 0.1]$
- *weight decay* $\in [0.00001, 0.001]$
- *batch size* : 2^i, with $i \in 6, 7, 8, 9, 10$
- *batch normalization* : True, False
- $\alpha \in [1.5, 3]$
- *max nr.* of *layers*: 5
- *layer dimension* : 2^i, *with* $i \in \{1, ..., k\}$ *and* 2^k the largest square number inferior or equal to the *input size*

A.2 Numerical Embeddings

- the *embedding dimensions* were set to 10 for the **BAF** dataset and to 4 in the **CTR** dataset

A.3 TabTransformer

- *embedding dimension*: $\{8, 16, 32, 64\}$
- *number of transformer blocks*: $\{2, 4, 8\}$
- *attention dropout* $\in [0, 0.1]$
- MLP *dropout* $\in [0, 1]$
- *embedding dropout* $\in [0, 1]$
- $feedforward$ *dropout* $\in [0, 0.1]$
- $transformer$ *activation* : $ReGLU, GEGLU$
- MLP *activation* : $ReLU, LeakyReLU, Tanh$
- *learning rate* $\in [0.00001, 0.01]$
- *weight decay* $\in [0.00001, 0.001]$
- *batch size* : $\{2^8, 2^9, 2^{10}, 2^{11}, 2^{12}\}$
- *batch normalization* : True, False
- *batch normalization for continuous input*: True, False
- MLP *layers*: $2 * l, 4 * l$, where l in the dimension of the input of the MLP

A.4 LightGBM

- *number of iterations* $\in \{200, 1200\}$
- *learning rate* $\in [0.001, 1.00]$
- *number leaves* $\in \{5, .., 500\}$
- *boosting type*: *goss*
- *min data in leaf* $\in \{5, ..., 200\}$
- *max bins* $\in \{5, ..., 20\}$

			BAF		CTR		Fraud	
Model	Treatment for Cat. Feat.	Treatment for Cont. Feat	AUC	Recall(5%)	AUC	Recall(5%)	AUC	Recall(5%)
MLP	Label Encoding	No Normalization	0.8752	0.4923	0.69143	0.21	0.96623	0.81818
		Zscore	0.87659	0.51331	—	—	0.95767	0.86364
		MinMax	0.87848	0.51331	—	—	0.979	0.86364
		Robust	0.856	0.4958	—	—	**0.98397**	0.81818
		Numerical Embedding	0.8927	0.55602	—	—	0.97812	**0.909091**
	Target Encoding	No Normalization	0.85281	0.43557	0.70921	0.18252	—	—
		Zscore	0.88508	0.52731	0.71263	0.12846	—	—
		MinMax	0.88571	0.53641	0.73869	0.20562	—	—
		Robust	0.86455	0.4944	0.69828	0.1615	—	—
		Numerical Embedding	0.896530	0.57073	—	—	—	—
	CatBoost Encoding	No Normalization	0.85169	0.45238	0.73721	0.20482	—	—
		Zscore	0.88746	0.52801	0.74125	0.20589	—	—
		MinMax	0.8653	0.47619	0.73423	0.21176	—	—
		Robust	0.86812	0.5014	0.72815	0.19691	—	—
		Numerical Embedding	0.89168	0.56513	—	—	—	—
	Count Encoding	No Normalization	0.86287	0.46569	0.63965	0.11217	—	—
		Zscore	0.87303	0.5	0.61715	0.12338	—	—
		MinMax	0.86868	0.4986	0.58975	0.08844	—	—
		Robust	0.84801	0.42297	0.60746	0.09048	—	—
		Numerical Embedding	0.88373	0.52941	—	—	—	—
	Categorical Embedding	No Normalization	0.84697	0.39776	0.73988	0.21022	—	—
		Zscore	0.8501	0.45588	—	—	—	—
		MinMax	0.84306	0.41106	—	—	—	—
		Robust	0.80373	0.30252	—	—	—	—
		Numerical Embedding	0.86556	0.47549	—	—	—	—
LGBM	Label Encoding	No Normalization	0.89902	0.57983	0.74151	**0.213**	0.96906	0.81818
		Numerical Embedding	**0.8994**	**0.58403**	—	—	0.84656	0.72727
	Target Encoding	No Normalization	0.89839	0.57563	0.73231	0.20192	—	—
		Numerical Embedding	0.89815	0.58053	—	—	—	—
	CatBoost Encoding	No Normalization	0.89793	0.57283	**0.75068**	0.2126	—	—
		Numerical Embedding	0.89852	0.57913	—	—	—	—
	Count Encoding	No Normalization	0.89706	0.57913	0.7112	0.19341	—	—
		Numerical Embedding	0.89668	0.57353	—	—	—	—
	Only Numerical Features		0.8737	0.4972				
TabTransformer			0.87952	0.507	0.74296	0.20425	—	—

Table 3: ROCAUC for test set and Recall with FPR at 5%. Best results are shown in bold, and best non-LightGBM results underscored