A Summary of Techniques Applied on Tabular Data Using Transformers Variants

BY UNDERML

Centro de Investigación en Computación. IPN

Abstract

Many public datasets are stored in tabular data, and it solution efficency using novelous methods still been variable on the proposed method. With the expansion of attention usage in deep learning, some ideas have emerged in order to apply this technique on tabular data. In this article we summarize some techniques that have been applied in order to process tabular data using transformers, also, we propose some new features that could be considered in the transformers architecture for data aggregation, and analysis of these methods is provided. Finally, we generate a public Python package that include all features discusssed here.

1 Introduction

Tabular data has been one of the most extended ways to store data, and is very used in real applications such as credit card owners behavior (https://www.kaggle.com/competitions/amex-default-prediction/), technology accuracy improvement (https://www.kaggle.com/competitions/smartphone-decimeter-2022) and sports predictions (https://www.kaggle.com/competitions/nfl-big-data-bowl-2022).

The tabular data have some common characteristics (https://papl.cs.brown.edu/2016/intro-tabular-data.html): (a) They consists of rows and columns; Each example is a row and each of their characteristics is a column; (b) Each row has the same columns as the other rows, in the same order; (c) A given column has the same type, but different columns can have different types. Any model processing tabular data should have some features that allow us to treat this kind of information, it means, (a) Each row should be processed independently; (b) Each column represents a characteristic, then, features can't be permuted while training or once trained; (c) Each feature feeded in the model has anown meaning andhas it data type.

Given the popularity growth of deep learning techniques into the industry, achieve a high accuracy becames a must when searching to solve a real problem, that is why many efforts has been applied in order to reach this goal. The most novelous efforts, given the popularity of attention mechanisms performance, drove many researches to apply this technique into tabular data, by modyfing regular transformers [12][9], using attention as a feature selector[2] among others. Despite this, other tree-based ensemble techniques[7][10] have shown a competitive prediction accuracy, and offers short times for training phase in comparisson with deep learning methods, what makes it a highly favourable solution whea dealing with tabular data. Any way, tree-based methods have several limitations: (a) They are not suitable for continuous training for streaming data; (b) They cannot deal with sequeential data in a end-to-end pipeline; (c) some methods to handle missing and noisy data are not suitable for them[8].

The most elemental way to process tabular data using deep learning is through a Multi Layer Perceptron (MLP). By encoding each categorical feature and concatenating them with numerical features we build a vector, of fixed length, that can be feed into the MLP and get the expected output. This solution is really easy and have a great power[13] but it has a big drawback; This solution is a complete blackbox model, then, it doesn't have an interpretability. In comparisson with it, one of the biggest attention-based models, the transformers[20] provides an interpretability using the attention matrices values, or even, in tabular transformers[12] we can see how the in-layers generated embeddings have some interesting properties that drive to some kind of interpretability.

In this paper we provide a summary and a empirical comparison of two main techniques used to process tabular data using tranformers variations, an analysisof those models and their properties, we propose some other methods for embedding aggregation and a Python programming package thatcontains the architectures shown here.

2 Models for Tabular Data

2.1 Multi-layer Perceptron

The Multi-layer Perceptron (MLP) formalization is:

$$\text{MLP}(x) = \text{Linear}(\text{MLPBlock}(\ldots(\text{MLPBlock}(x))))$$
$$\text{MLPBlock} = \text{ReLU}(\text{Linear}(x))$$

Despite these are basic models, in 2021, Kadra [13] empirically assess the impact of regularization cocktails in 40 tabular datasets demonstrating that well-regularized MLPs significantly outperform specialized neural network architectures, and they even outperform strong traditional ML methods, such as XGBoost[7].

2.2 Tabular Transformer

Introduced in 2020 by Huang [12], this architecture was built upon self-attention based Transformers[20]. They transform the categorical features into contextual embeddings and apply a regularization method on numerical features. The main idea is to use the contextual embeddings to achive a high prediction accuracy. Their results shown an improvement of at least 1% on Area Under Curve (AUC) metric compared with deep learning methods and matches with tree-based ensembled models. Furthermore, their contextual embeddings are robust against missing and noisy data,and provide some interpretability.

2.3 FT-Transformer

This architecture was introduced by Gorishniy in 2021[9], the main difference with Tabular Transformer is that they build embedding for numerical and categorical variables. An additional [CLS] embedding is stacked to features embeddings an then they are then processed by the Transformer module. The final representation of the [CLS] token is used for prediction. They results shown an outperformance over other deep learning solutions in multiple tasks but they solution isn't universal since the Gradient Boosting Decision Trees[7] still being superior.

2.4 Other models

- TabNet[2]: Uses sequential attention to choose the features that are more important in a decission, allowing interpretability.
- Node[17]: A deep learning architecture specialized in tabular data by using ensembles of oblivious decision trees, but benefits from both end-to-end gradient-based optimization and the power of multi-layer hierarchical representation learning.
- GBDT[7]: A sparsity-aware algorithm for sparse data and weighted quantile sketch for approximate tree learning.

3 Standarization of Transformers Architectures for Tabular Data

We took the two previous architectures as the baseline; The tabular tranformer proposed in 2020 by Huang [12] and the architecture proposed by Gorishniy in 2021[9]. The main difference is that the Huang architecture process the numerical data out of multi-head attention mechanism, while the Gorishniy architecture generates an embedding for numerical data and process it inside the multi-head attention(MHA) mecahnism.

As a generalized data flow of mentioned architectures we can consider the next steps:

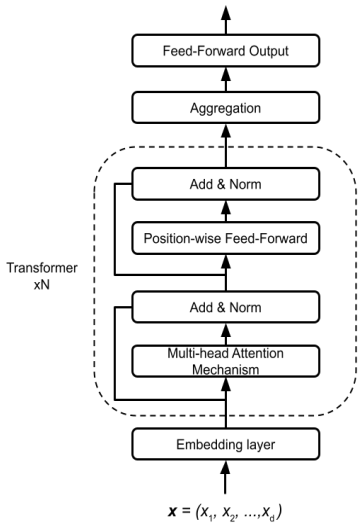

Figure 1. The proposed architecture for tabular data.

The main components assembled in this architecture and their utilities are:

Embedding layer: The embedding layer transforms the input vector $f = [f_1, \ldots, f_d]$ into embedding representations $X \in \mathbb{R}^{d \times l}$. The numerical embedding is performed depending on the desired behavior. If we want to use multi-head attention mechanism to process numerical data the applied transformation for feature i is:

$$x_i^{(\text{num})} = b_i + f_i^{(\text{num})} \hat{W}_i \in \mathbb{R}^l$$

Where b_i is a bias, $\hat{W}_i$ are a normalized weights[18] and $f_i^{(\text{num})}$ is the i-th numerical feature. In the other hand, if the decision is not to use the multi-head attention mechanism, the numerical transformation is:

$$X^{(\text{num})} = \text{LayerNorm}(f^{(\text{num})}) \in \mathbb{R}^l$$

Where LayerNorm is the method introduced by Ba[3], and $f^{(\text{num})} \in \mathbb{R}^{l^{(\text{num})}}$ is the vector formed by the concatenationof all numerical features. Finally, for categorical variables, the applied transformation is:

$$x_i^{(\text{cat})} = b_i + f_i^{(\text{cat})} W_i \in \mathbb{R}^l$$

Where b_i is the bias, $f_i^{(\text{cat})}$ is the one hot encoding representation of i-th categorical feature, and W_i are the weights. Once the selected transformations are applied, if we choose to passthrough the numerical variables these are concatenated to the output of the aggregation module, otherwise, these are stacked to categorical features to build the encoder input.

Multi-head Attention Mechanism: This modules applies the multi-head attention process described by Vaswani[20] to produce a sequence of new features $\phi_1, \ldots, \phi_d$ as follows: Let X be a matrix where the rows are the embedding vectors $\mathbf{x}_1, \ldots, \mathbf{x}_d$ of the input f, and Q, K, and V matrices of parameters of dimension $l \times s$ where s is the size of the head. The attention mechanism builds an *attention* matrix for each input x:

$$A = \text{softmax}\left(\frac{(XQ)\,(XK)^T}{\sqrt{d}} \right)$$

The matrix A is a matrix of dimension $d \times d$ where each row $\boldsymbol{a}_i$ is a probability distribution used to define the vector

$$\boldsymbol{\theta}_i = a_{i,1} V\boldsymbol{x}_1 + a_{i,2} V\boldsymbol{x}_2 + \cdots + a_{i,d} V\boldsymbol{x}_d$$

which represents the transformation $(\boldsymbol{x}_1,...,\boldsymbol{x}_n) \rightarrow \boldsymbol{\theta}_i$ of the i-th embedding vector. Hence, $\boldsymbol{\theta}_i$ is a linear combination of simple linear transformations of the original features where the weights of the linear combination are given by the attention vector $\boldsymbol{a}_i$. We argue that the attention mechanism can be seen as a *soft* feature selection algorithm. The output of this process is a sequence of vectors $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_d$ each of dimension h. Finally, this process can be repeated multiple times and the outputs are concatenated to form the final output $\hat{\boldsymbol{h}}_1, \ldots, \hat{\boldsymbol{h}}_d$ of the multi-head attention mechanism where $\hat{\boldsymbol{h}}_i = [\boldsymbol{\theta}_{i,1} ... \boldsymbol{\theta}_{i,k}]$ is the concatenation of the outputs of each head.

Add & Norm: The next step add its inputs (the previous-step and the residual one) and applies a layer normalization[3]. The applied transformation generates $\boldsymbol{h}_i = \text{LayerNorm}(\hat{\boldsymbol{h}}_i + \boldsymbol{x}_i)$.

Position-wise Feed-Forward Network: The next step in the transformers architecture is to apply the same feed forward network $\hat{\phi}$ to all the vectors $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_d$ produced by the first add & norm layer. The result is a sequence of vectors $\hat{\phi}(\boldsymbol{h}_1), \ldots, \hat{\phi}(\boldsymbol{h}_d)$.

Add & Norm: As the previous add & norm layer, the transformation performed by this stape is $\phi(\boldsymbol{h}_i) = \text{LayerNorm}(\boldsymbol{h}_i + \hat{\phi}(\boldsymbol{h}_1))$.

Agreggation: Once all features passed through the transformer encoding layer a vectors aggregation $\boldsymbol{\varphi} = \textbf{Agg}(\{\phi(\boldsymbol{h}_1), \ldots, \phi(\boldsymbol{h}_d)\})$ is applied. The proposed aggregation methods are:

1. **Concatenation**. The first method is the one used in the Tabular transformer architecture[12] and it concatenates all embedding obtained from the encoder.

2. **Classification token transformer (CLS)**. This aggregation method adds a [CLS] token used to perform the classification, such that, before input X into the encoder a [CLS] embedding must be stacked too as performed by Gorishniy[9].

3. **Recurrent Output Network**. Another considered aggregation method, in order to reduce the trainable parameters, is by using a recurrent neural network, where the vectors $\phi(\boldsymbol{h}_1), \ldots, \phi(\boldsymbol{h}_d)$ are feed one at each time step.

4. **Maximum Pooling**. In this aggregation method, the maximum pooling strategy[11] is applied on vectors $\phi(\boldsymbol{h}_1), \ldots, \phi(\boldsymbol{h}_d)$.

5. **Mean Pooling**. In this aggregation method, the mean pooling strategy[11] is applied on vectors $\phi(\boldsymbol{h}_1), \ldots, \phi(\boldsymbol{h}_d)$.

6. **Sum**. Generatea classification vector by sum all vectors $\phi(\boldsymbol{h}_1), \ldots, \phi(\boldsymbol{h}_d)$.

Feed-Forward Output Network: The last step consist of feed $\boldsymbol{\varphi}$ to a feed forward network that will produce the final output of the supervised problem. In case that we have choosen to passthrough the numerical parameters, theinput to this module must be the concatenation of $\boldsymbol{\varphi}$ and $E^{(\text{num})}$.

4 Experiments

In this section, we compare the methods used in transformers for tabular data, emphasizing on the aggregation methods and how this selection could affect in the task efficency performance.

4.1 Datasets

We select 38 open datasets from OpenML[6], then we generate some features based on their metadata:

- features_instances= n_features/n_instances
- percent_numerical=n_numerical/n_features
- percent_missing_values=n_missing_values/(n_features * n_instances)

Once done, we applied KMeans[16] algorithm using $n = 5$, in order to get 5 representative datasets, enough different among them but similar with the others, and then, we select those datasets that were closer to the cluster centroids. Figure 2 shows the spatial datasets distribution. The summary of selected datasets and their properties are shown in table 1.

Name	**OpenML ID**	**features_instances**	**percent_numerical**	**percent_missing_values**
anneal ✣	2	0.043	0.158	0.650
kr-vs-kp	3	0.012	0.000	0.000
credit-g	31	0.021	0.350	0.000
vehicle	54	0.022	1.000	0.000
kc1	1067	0.010	1.000	0.000
adult ✣	1590	0.000	0.429	0.009
walking	1509	0.000	1.000	0.000
phoneme	1489	0.001	1.000	0.000
skin-seg	1502	0.000	1.000	0.000
ldpa	1483	0.000	0.714	0.000
nomao ✣	1486	0.003	0.754	0.000
blood	1464	0.007	1.000	0.000
bank	1461	0.000	0.438	0.000
connect	40668	0.001	0.000	0.000
shuttle	40685	0.000	1.000	0.000
higgs	23512	0.000	1.000	0.000
australian	40981	0.022	0.429	0.000
car	40975	0.004	0.000	0.000
segment	40984	0.009	1.000	0.000
jungle	41027	0.000	1.000	0.000
numerai	23517	0.000	1.000	0.000
helena	41169	0.000	1.000	0.000
jannis	41168	0.001	1.000	0.000
volkert	41166	0.003	1.000	0.000
miniboone	41150	0.000	1.000	0.000
apsfailure	41138	0.002	1.000	0.083
jasmine ✣	41143	0.049	0.056	0.000
sylvine ✣	41146	0.004	1.000	0.000
dionis	41167	0.000	1.000	0.000
aloi	42396	0.001	1.000	0.000
ccfraud	42397	0.000	1.000	0.000
clickpred	1219	0.000	1.000	0.000
Covertype	293	0.000	1.000	0.000
albert	41147	0.000	0.333	0.082
blastchar	42178	0.003	0.158	0.000
online-shoopers	42993	0.001	0.824	0.000
qsar	1494	0.040	1.000	0.000
spambase	44	0.013	1.000	0.000

Table 1. OpenML[6] selected datasets. Those datasets with mark (✣) are the closest to their respective cluster centroids.

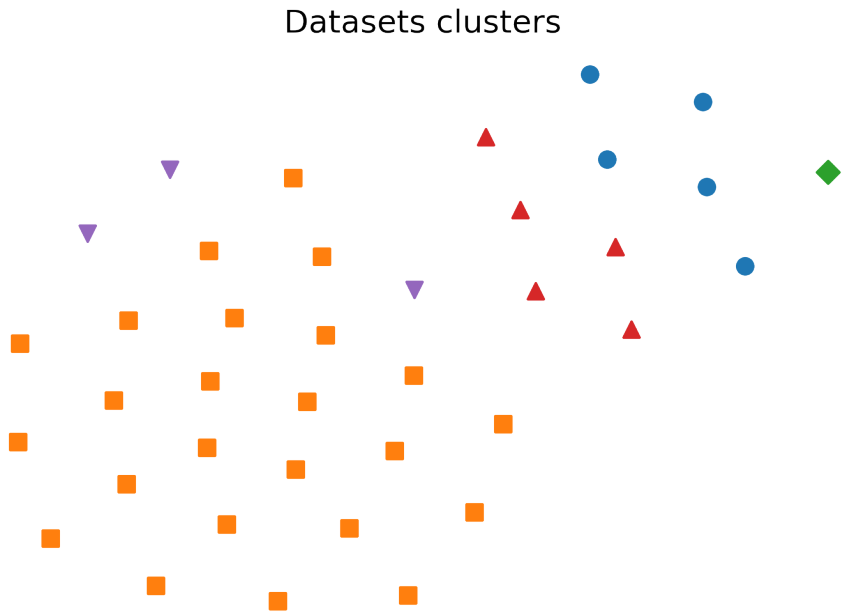

Figure 2. Datasets groups generated by appling KMeans[16] and t-SNE[19] dimensionality reduction.

4.2 Implementation Details

4.2.1 Preprocessing Pipeline

The preprocessing pipeline was shared accross all datasets: The first step was the data imputation using a KNN-based imputer with $n = 10$, after that, the numerical features were normalized with mean 0 and standard deviation 1, and the categorical features were encoded using an ordinal encoding. The categorical ordinal encoding will be automatically transformed into One hot encoding inside the embedding creation.

4.2.2 Hyperparameters Optimization

For each one of 5 selected datasets, a hyperparameter search was performed, this pipeline was shared among all datasets without disctintions, to this, we used Tune[15] with Optuna[1]. The optimization algorithm was Tree-structured Parzen Estimator (TPE)[4] and Asynchronous HyperBand Scheduler (ASHA)[14] for early stopping during the hyperparameters search. The hyperparameters optimization metric was balanced accuracy[5], the optimization parameters metric was Cross-entropy loss, with a processing batch size of 128, and Stochastic Gradient Descent, ASHA's early stopping of 15, 100 maximum epochs, starting seed set to 11, and 30 samples for performing Bayesian optimization (TPE).

The hyperparameter search space was the same for each aggregator, and it is shown in table 2

Parameter	Search space
Number of layers	[1, 5]
Optimizer lr	loguniform(1e-5, 1e-1)
Number of heads	{1, 2, 4, 8, 16, 32}
MLP Hidden size	{32, 64, 128, 256, 512, 1024}
Dropout	uniform(0, 0.5)
Embedding size	{32, 64, 128, 256, 512, 1024}
Numerical passthorugh	{False, True}
Params. Opt. Algorithm	{SGD}
Params. Opt. Metric	{Cross-entropy loss}
Params. Opt. Early stopping	{15}
Params. Opt. E. S. Metric	{balanced_accuracy}

Table 2. Hyperparameters shared search space.

Additionally, for RNN aggregator, other parameters were considered, those are in table 3

Parameter	Search space
Cell	{LSTM, GRU}
Hidden size	{32, 64, 128, 256, 512, 1024}
Number of layers	[1, 3]
Dropout	uniform(0, 0.5)

Table 3. Additional considered hyprparameters for RNN aggregator.

4.2.3 Final Model Fitting

Once the hyperparameter optimization was done, the selected model was that one that performs better on the hyperparameter optimization metric, and was fitted without limitations, it means, the hyperparameters restrictions stablished in order to improve the searching time were deleted and an exhaustive serch was performed.

Parameter	Value
Max epochs	500
LR Scheduler	ReductiononPlateu
LR Scheduler patience	5

Table 4. Hyperparameters shared search space.

4.3 Results

5 Analysis

The generalized architecture shown before complay these properties: Despite the transformer architecture was thought for sequential data processing, it also works without variating the input size during the time, and because the embedding is not shared among features, the input cannot be permuted.

Differences with a MLP

Can the progress with transformers be validated?

- Importance of Embeddings

Plots

- Seaborn

- # of params of models vs accuracy vs speed (circles)

6 Conclusions

Bibliography

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: a next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[2] Sercan O. Arik and Tomas Pfister. Tabnet: attentive interpretable tabular learning. 2019.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. 2016.

[4] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24, page 0. Curran Associates, Inc., 2011.

[5] Kay Henning Brodersen, Cheng Soon Ong, Klaas Enno Stephan, and Joachim M. Buhmann. The balanced accuracy and its posterior distribution. In *2010 20th International Conference on Pattern Recognition*, pages 3121–3124. 2010.

[6] Giuseppe Casalicchio, Jakob Bossek, Michel Lang, Dominik Kirchhoff, Pascal Kerschke, Benjamin Hofner, Heidi Seibold, Joaquin Vanschoren, and Bernd Bischl. OpenML: an R package to connect to the machine learning platform OpenML. *Computational Statistics*, pages 1–15, 2017.

[7] Tianqi Chen and Carlos Guestrin. Xgboost: a scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794. New York, NY, USA, 2016. Association for Computing Machinery.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: pre-training of deep bidirectional transformers for language understanding. 2018.

[9] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. 2021.

[10] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[11] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. 2018.

[12] Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. Tabtransformer: tabular data modeling using contextual embeddings. 2020.

[13] Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Well-tuned simple nets excel on tabular datasets. 2021.

[14] Lisha Li, Kevin Jamieson, Afshin Rostamizadeh, Katya Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. Massively parallel hyperparameter tuning. 2018.

[15] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: a research platform for distributed model selection and training. *ArXiv preprint arXiv:1807.05118*, 2018.

[16] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[17] Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. 2019.

[18] Tim Salimans and Diederik P. Kingma. Weight normalization: a simple reparameterization to accelerate training of deep neural networks. 2016.

[19] Laurens van der Maaten and Geoffrey E. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.