**TABCaps: A Capsule Neural Network for Tabular Data Classification with BoW Routing**

Jintai Chen¹, Kuanlun Liao¹, Yanwen Fang², Danny Z. Chen³, Jian Wu⁴

¹College of Computer Science and Technology, Zhejiang University
²Department of Statistics & Actuarial Science, The University of Hong Kong
³Department of Computer Science and Engineering, University of Notre Dame
⁴Department of Public Health, Zhejiang University School of Medicine

**Abstract**

Records in a table are represented by a collection of heterogeneous scalar features. Previous work often made predictions for records in a paradigm that processed each feature as an operating unit, which requires to well cope with the heterogeneity. In this paper, we propose to encapsulate all feature values of a record into vectorial features and process them collectively rather than have to deal with individual ones, which directly captures the representations at the data level and benefits robust performances. Specifically, we adopt the concept of capsules to organize features into vectorial features, and devise a novel capsule neural network called TABCaps to process the vectorial features for classification. In TABCaps, a record is encoded into several vectorial features by some optimizable multivariate Gaussian kernels in the primary capsule layer, where each vectorial feature represents a specific profile of the input record and is transformed into senior capsule layer under the guidance of a new straightforward routing algorithm. The design of routing algorithm is motivated by the Bag-of-Words (BoW) model, which performs capsule feature grouping straightforwardly and efficiently, in lieu of the computationally complex clustering of previous routing algorithms. Comprehensive experiments show that TABCaps achieves competitive and robust performances in tabular data classification tasks. Codes are available at [https://github.com/WhatAShot/TabCaps](https://github.com/WhatAShot/TabCaps).

1 Introduction

Tabular data are ubiquitous in real world applications, which records abundantly meaningful information such as medical examination results (Hassan et al., 2020) and company financial statements (Addo et al., 2018). Previous methods often processed a record by treating the scalar feature values as the operating units. For example, decision tree based methods (Breiman et al., 1984; Chen & Guestrin, 2016) used one tabular feature in each decision step, and neural networks (Gorishniy et al., 2021; Chen et al., 2022) elaborately executed feature-wise interactions to capture higher-level semantics. However, it is intractable to design effective feature-wise interaction approaches (Grinsztajn et al., 2022; Ng, 2004) due to the heterogeneity among features. In this paper, we propose a novel paradigm for supervised tabular learning, which encapsulates all feature values of records into vectorial features and directly conducts on the vectorial feature level. Such design utilizes the sufficient representation space of the vectorial feature format to probably learn the comprehensive data level semantics, and avoids executing complex interactions among heterogeneous features.

To this end, we borrow the concept of capsules (Sabour et al., 2017) to organize vectorial features, and propose a novel capsule neural network (CapsNet) called TABCaps for tabular data classification. In TABCaps, several optimizable multivariate Gaussian kernels encode all the features of each record into the primary capsules, in which features in a vector format represent the marginal likelihoods of the record in reference to the corresponding multivariate Gaussian distributions. We set the scale and location parameters of these Gaussian kernels learnable, thus allowing these kernels to model some plausible data patterns for the dataset and each primary capsule feature represents a specific profile.
of the input record that measures the likelihood to these data patterns. Unlike previous CapsNets that used one senior capsule to predict the belonging probability for one class, we allot multiple senior capsules for a class, motivated by ensemble learning. In previous CapsNets, primary capsule features were transformed into senior capsules by an affinity projection and a routing algorithm (Sabour et al., 2017; Hinton et al., 2018) that groups similar primary capsule features by clustering processes. In TABCAPS, a novel sparse projection method and a novel straightforward routing algorithm are proposed to perform the feature transformation from primary to senior capsules.

Our proposed routing algorithm is much more efficient than previous routing algorithms, since the primary capsules in previous CapsNets captured some unknown semantics from unstructured data (e.g., images) and had to apply feature clustering in an iterative process to attain higher-level semantics. In our TABCAPS, features in primary capsules are the likelihood w.r.t. the Gaussian distribution, which represents stable semantics and allows to perform a straightforward routing algorithm. Motivated by the bag-of-words (BoW) model (Salton & Lesk, 1965) which is efficient in similar information search (Pineda et al., 2011), we propose a straightforward routing (called BoW Routing) for TABCAPS. The proposed BoW Routing computes similarities between primary capsule features and the initialized senior capsule features by counting the co-occurrences of “words” (implemented to be learnable templates), and incorporates those primary capsule features whose similarities are beyond an adaptive threshold to update senior capsule features. Similar to previous CapsNets, we also present a tabular-data-suited decoder, which is tailored for tabular feature reconstruction tasks (e.g., missing value imputation).

Contributions. (i) For the first time, we propose to encapsulate all the features of a record in table into vectorial features as operating units, which avoids inefficient interactions between heterogeneous tabular features and directly learns the data-level semantics. (ii) We propose a CapsNet tailored for supervised tabular learning, in which we devise a new type of primary capsule with learnable multivariate Gaussian kernels and conduct a novel straightforward routing to synthesize senior capsule features at low costs. (iii) Experiments on real-world datasets validate that TABCAPS attain robust performances in tabular data classification.

2 RELATED WORK

Capsule Neural Networks. Capsule neural networks (Ribeiro et al., 2022) were first proposed to deal with image data, where a capsule represents an object whole or object part in a vector or matrix format. Firstly, primary capsules capture the basic object parts by using templates (Kosiorek et al., 2019), optical flow analysis (Sabour et al., 2021), or location-based operations (Hinton et al., 2018). Features in primary capsules are linearly projected to transpose the poses of object parts, so as to synthesize senior capsule features (representing object wholes) guided by a routing algorithm. Most routing algorithms performed iteration processes at high computational costs (Sabour et al., 2017; Hinton et al., 2018), since the primary capsules handle different semantics in dealing with different samples. To avoid iterations, some straightforward approaches were proposed (Choi et al., 2019; Ahmed & Torresani, 2019; Chen et al., 2021; Ribeiro et al., 2020). However, these straightforward
approaches were for image tasks and some of them got rid of the part-to-whole learning assumption. At present, CapsNets have been widely used for image processing (Rajasegaran et al., 2018; Hinton et al., 2021; Chen et al., 2021), point clouds (Zhao et al., 2019; 2020), recommendation in shopping (Li et al., 2019), and language tasks (Zhao et al., 2019a). However, to our best knowledge, no bespoke CapsNets were proposed for general supervised tabular learning tasks. This paper presents a novel CapsNet for tabular data, in which primary capsules learn some specific kinds of data patterns for the dataset rather than the data-adaptive “object parts”.

**Supervised Tabular Learning.** Various methods were proposed for tabular data processing, mainly including decision tree algorithms (Breiman et al., 1984; Quinlan, 1979; 1997; Good, 2016; Prokhorenkova et al., 2018; Ke et al., 2017) and neural network (NN) based methods (Arik & Pfister, 2020; Popov et al., 2019; Chen et al., 2022; Aburbul et al., 2021). Different from tree based methods that split tabular feature spaces by some information metrics (Quinlan, 1979; 1997; 2014; Breiman et al., 1984) and sought for possible decision paths, NN based methods often introduced attention and grouping mechanisms (Chen et al., 2022; Arik & Pfister, 2020; Guo et al., 2017; Wang et al., 2017; Gorishnyy et al., 2021; Yan et al., 2023) for tabular feature interactions to yield high-level features. However, tabular features in a table are heterogeneous and are relation-agnostic, and thus it is hard to conduct all the feature interactions in the right way.

### 3 Methodology

We propose to classify each record in a table by processing a vectorial feature containing all its feature values in order. See the pipeline of our TabCaps in Fig. 1(a), to better depict an input record, its raw features are first processed by a simple function (e.g., an Abstract Layer (Chen et al., 2022) or a linear projection layer) to yield more comprehensive features, which are then concatenated with its raw features and we obtain the basal vectorial feature of size $m$, $x \in \mathbb{R}^m$. Then, the vectorial feature is projected into primary capsules (in a vector format) by various learnable multivariate Gaussian kernels. Each vectorial feature (in a primary capsule) is transformed into feature votes for senior capsules by the proposed sparse projection (see Sec. 3.2.1), and senior capsule features are finally obtained by selecting and merging feature votes, guided by our BoW Routing (see Sec. 3.2.2).

#### 3.1 How to Encode Records into Primary Capsules?

Unlike the known CapsNets for images that utilized various primary capsules to represent different parts of objects, in TabCaps, different primary capsules represent various profiles for just one record. Formally, a record represented by a vectorial feature $x \in \mathbb{R}^m$ is transformed into a feature $u_i \in \mathbb{R}^m$ for the $i$-th primary capsule by a multivariate Gaussian kernel $k_i$, as:

$$u_i = k_i(x; \mu_i, \Sigma_i) = \frac{\exp\left(-\frac{1}{2}(x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i)\right)}{(2\pi)^{m/2} |\Sigma_i|},$$

where the location parameter vector $\mu_i \in \mathbb{R}^m$ and the scale parameters $\Sigma_i = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_m)$ of the multivariate Gaussian kernel $k_i$ are set to be learnable, so that TabCaps allows to use proper multivariate Gaussian kernels in different datasets. Since features in a table are typically regarded as independent, we set $\Sigma$ as a diagonal matrix. After Eq. (1), a layer normalization is performed on each $u_i$ ($i = 1, 2, \ldots, N_p$, $N_p$ is the count of the primary capsules) for normalization following Tsai et al. (2019). Specially, we also encapsulate the original feature values (denoted by $x$) to be an additional capsule feature.

This design can be interpreted that a primary capsule models an data pattern by determining the parameters $(\mu, \Sigma)$ of the multivariate Gaussian kernels, and various primary capsule features of an input record are represented as the likelihoods in reference to these data patterns. It is beneficial to find some dataset-specific data patterns, which allows to implicitly model the tabular feature relations.

#### 3.2 How to Synthesize Senior Capsule Features?

Motivated by ensemble learning, we use multiple senior capsules to learn the semantics for one target class and obtain the final prediction by voting, which also meets the need of irregular target pattern fitting (Grinsztajn et al., 2022). To attain the senior capsule features, primary capsule features are transformed by a sparse weight matrix into feature votes, and a newly proposed BoW Routing selects feature votes to synthesize senior capsule features. Note that the elements in vectorial features are linearly transformed (reversible), and the non-linear mapping (the routing) is performed by taking vectorial features as operating units.
3.2.1 Sparse Projection

The affinity projection in previous work (Sabour et al., 2017) uses a learnable weight $W_{ji} \in \mathbb{R}^{m \times n}$ to linearly map the $i$-th primary capsule feature $u_i \in \mathbb{R}^m$ to yield a feature vote $\hat{u}_{j|i} \in \mathbb{R}^n$ for the $j$-th senior capsule, by $\hat{u}_{j|i} = u_i W_{ji}$. We modify the weight matrix by adding a sparse operation $\text{entmax}$ (Peters et al., 2019), and the transformation is computed by:

$$\hat{u}_{j|i} = u_i W'_{ji} \cdot W_{ji} = \text{entmax}_\alpha(W_{ji}),$$

(2)

where $\alpha = 1.5$ as default. In this way, the weight matrix $W'_{ji}$ shall be sparse and only parts of elements of $u_{j|i}$ is used to synthesize the feature votes and the target-irrelevant elements are excluded. Different from sparse attention operations, the $\text{entmax}$ is applied to the weight matrix. Thus, the capsule feature component selection is consistent for different records.

3.2.2 BoW Routing

Given the synthesized feature votes $\hat{u}_{j|i}$ for the $j$-th senior capsule (where $i = 1, 2, \ldots, N_p$, $j = 1, 2, \ldots, N_s$, $N_p$ and $N_s$ are the amounts of primary capsules and senior capsules respectively), we use a routing algorithm to synthesize senior capsule features by selecting and merging $\hat{u}_{j|i}$. Note that the vanilla routing algorithm was a clustering-like approach (Sabour et al., 2017), because a primary capsule may capture any kinds of object parts from images and thus the “cluster centroids” are unknown before the routing step. In our model, each primary capsule learns concrete semantics that represent record profiles in reference to a specific multivariate Gaussian distribution, making it possible to predict the initialized senior capsule features (“cluster centroids”) directly using a straightforward function w.r.t. semantics (features) captured by primary capsules. Here we simply use the average of the feature votes $\hat{u}_{j|i}$ to initialize senior capsule features $v'_j$ by $v'_j = \sum_{i=1}^{N_p} \hat{u}_{j|i}/N_p$.

To select some feature votes for senior capsule feature synthesis, we measure the similarities between the initialized senior capsule feature $v'_j$ and the primary capsule feature vote $\hat{u}_{j|i}$. An intuitive solution is to compute the similarity directly. However, directly computing similarity might hinder the performance, which overlooks the special data patterns and importance differences of different vectorial feature elements. Motivated by the success of the bag-of-words (BoW) model in similarity measure for information retrieval, we introduce the BoW model to the routing algorithm. A classical BoW model first clusters the given features, and collects a word dictionary whose words are the centroids of feature clusters. Then, a feature can be quantized as the “occurrence” of various words, and feature similarities can be computed by counting the co-occurring words. In this way, the similarity is computed in the semantic space guided by the “words” in discrete steps. Our differentiable BoW Routing is depicted as follow.

Occurrence status computing. The dictionary words in the traditional BoW model are computed according to the extracted features. Motivated by Chen et al. (2021), we directly implement the word dictionary with a learnable parameter set of size $M$, $\{s_m \mid m = 1, 2, \ldots, M\}$, with each “word” $s_m \in \mathbb{R}^n$ (of the identical size as the feature votes $\hat{u}_{j|i}$). To compute the “occurrences” of the “words”, an occurrence status (a scalar) between $\hat{u}_{j|i}$ and the initialized senior capsule feature $v'_j$ for the “word” $s_m$ is defined by:

$$p_{j|i;m} = \text{sigmoid}(\hat{u}_{j|i}^T \cdot s'_m) \in (0, 1), \quad p_{j;m} = \text{sigmoid}(v'_j^T \cdot s'_m) \in (0, 1),$$

(3)

where $p_{j|i;m}$ and $p_{j;m}$ represent the degrees (i.e., the occurrence statuses) in which $\hat{u}_{j|i}^T$ and $v'_j^T$ reveal the semantics of $s_m$, we let $p_{j|i} = [p_{j|i;1}, p_{j|i;2}, \ldots, p_{j|i;M}]$ and $p_{j} = [p_{j;1}, p_{j;2}, \ldots, p_{j;M}]$ represent semantic codes of $\hat{u}_{j|i}$ and $v'_j$. In computing Eq. (3), the “word” $s_m$ is normalized into $s'_m$ by the $l_2$ norm to avoid the impact of the vector lengths. In traditional BoW models, an occurrence status is represented as binary (0 or 1), but by our routing algorithm, we generalize the discrete binary status to a continuous status form (in $(0, 1)$) using the “sigmoid” function.

Feature vote merging. To compute the similarity between the occurrence statuses $p_{j|i}$ and $p_j$, traditional BoW models may use the Jaccard distance. Since the occurrence statuses are continuous values in our routing algorithm, to which the Jaccard distance cannot be simply applied, we compute the pairwise distances $d_{j|i}$ between $p_{j|i}$ and $p_j$ using the squared Euclidean distance function $D$:

$$d_{j|i} = D(p_{j|i}, p_j) = \sum_{m=1}^{M} (p_{j|i;m} - p_{j;m})^2.$$  

(4)
Note that the squared Euclidean distance has been proved to be an effective approximation of the Jaccard distance (Reina et al., 2014). Based on the pairwise distances $d_{j|i}$ of feature votes and initialized senior capsule features, we can directly obtain the neighboring feature votes $\hat{u}_{j|i}$ to update the features in the $j$-th senior capsule by performing $k$-Nearest Neighbor ($k$-NN) search. However, we find that $k$-NN can lead to instability in training, since the distributions of feature votes are unknown and $k$-NN might select some feature votes with extreme values.

To gain training stability, we propose to use an adaptive threshold to determine a range around the initialized senior capsule feature $v_{i}'$ (see Fig. 1(b)). We set this threshold as a learnable positive value $\gamma^2 > 0$, and the selection of feature votes is determined by:

$$q_{j|i} = \varepsilon(\gamma^2 - d_{j|i}),$$

where $\varepsilon(\cdot)$ is the Heaviside function, and if and only if $q_{j|i} = 1$, it indicates that the feature vote from the $i$-th primary capsule, $\hat{u}_{j|i}$, is selected. Then, we compute the weights for the feature votes and obtain the final senior capsule feature $v_j$ by a weighted sum, as:

$$w_{j|i} = \text{softmax}(q_{j|i} \cdot e^{-d_{j|i}}), \quad v_j = \sum_{i}^{N_p} w_{j|i} \hat{u}_{j|i},$$

where the “softmax” operation guarantees all $w_{j|i} > 0$ (giving small weights to unselected votes), avoiding possible failure if no feature vote is selected to yield $v_j$. In this way, our proposed BoW Routing performs a straightforward process to obtain the senior capsule features.

**Remarks.** Distinct from previous straightforward routing algorithms (Chen et al., 2021; Ribeiro et al., 2020) for images, the efficiency of our BoW Routing partially results from the innovative definition of the primary capsules that learns concrete semantics, allowing to predict initialized senior capsule features directly, in lieu of learning unknown object parts and requiring clustering-like routing algorithms to group them. Our BoW Routing algorithm, with its strong ability to extract semantic patterns and flexible measurement of similarity, allows us to utilize a straightforward average method for initializing the senior capsule features.

### 3.3 Senior Capsule Voting for Classification

Similar to the known CapsNets, we let the vector lengths of the senior capsule features represent the existence probabilities of one semantics, and this work models each target class by multiple senior capsules motivated by ensemble learning. Suppose there are $K$ classes. We divide all the senior capsules into $K$ groups, such that the features in each group vote for one particular class. The classification outcomes are defined by:

$$c = \text{softmax}(l_1, \ldots, l_K), \quad \hat{y} = \text{argmax}(c),$$

where the vector $c$ represents the predicted probabilities for all the $K$ classes, and $\hat{y}$ denotes the predicted label. A value $l_k$ in Eq. (7) denotes the mean of the senior capsule feature vector lengths for the $k$-th class (i.e., in the group $G_k$), which is defined by:

$$l_k = \frac{\sum_{j \in G_k} ||v_j||_2}{|G_k|}.$$ 

Note that we drop 20% senior capsules in a group before voting in training. We use the margin loss in training our TABCAPS following Sabour et al. (2017), as:

$$\mathcal{L} = \sum_{k=1}^{K} T_k \max(0, t^+ - l_k)^2 + \lambda(1 - T_k) \max(0, l_k - t^-)^2,$$

where $T_k = 1$ if the $k$-th class is the target, and otherwise $T_k = 0$. We set $t^+ = 0.9, t^- = 0.1$, and $\lambda = 0.5$ following Sabour et al. (2017).

### 3.4 A Decoder for Tabular Feature Reconstruction

We build a decoder based on the Abstract Layers (Chen et al., 2022) for tabular feature reconstruction. We stack two Abstract Layers in sequence with a shortcut performing feature concatenation, as shown in Fig. 2. Finally, a fully connected layer is on top of the decoder to produce the reconstructed tabular features. Since there are multiple senior capsules voting for a target class, the decoder input is the concatenation of features in the group of senior capsules that votes for the correct class. We do not use the decoder as regularization like previous CapsNets (Sabour et al., 2017) and TABCAPS without the decoder can perform well on classification tasks (see Fig. 3). The decoder is used when we attempt to reconstruct raw features for some tasks (e.g., missing value imputation), which is trained under the specification of the mean square error (MSE) that is scaled down by $5 \times 10^{-6}$ and jointly works with the loss $\mathcal{L}$ (in Eq. (8)).
4 Experiments

4.1 Experimental Setups

Datasets. We conduct experiments on 8 real-world tabular datasets, including Heart Failure Prediction (Heart) (Chicco & Jurman, 2020), Click (Cup, 2012), Diabetes (Ukani, 2020), EEG Eye State (EEG) (Dua & Graff, 2015), Gas Concentrations (Gas) (Vergara et al., 2012), Hill-Valley (Hill) (Lee & Franz, 2008), Higgs-small (Higgs) (Baldi et al., 2014) and Epsilon (Chalenge, 2008). All of these datasets have few or no categorical features, eliminating the impacts of categorical feature embedding approaches in model comparison. In pre-processing, data are standardized by the z-score normalization. We follow the train-test split and the data pre-processing in (Popov et al., 2019) for the Click dataset. For the other datasets, we randomly divide 80% of the data into the training sets, and the rest 20% are for test. We further sample 15% training data for validation. Experiments were run for 5 times with regard to random seeds.

Implementation details and comparison baselines. Our method is implemented with PyTorch on Python 3.7, and is run on GeForce RTX 2080 Ti. In training, we set the batch size as 2,048. We use the QHAdam optimizer (Ma & Yarats, 2019) with weight decay rate $10^{-5}$, $\alpha = (0.7, 0.99)$, and $\beta = (0.95, 0.998)$. We compare TABCAPS with several baseline models, including TabNet (Arik & Pfister, 2020), NODE (Popov et al., 2019), FT-Transformer (Gorishniy et al., 2021), DANet-24 (Chen et al., 2022), Net-DNF (Abutbul et al., 2021), XGBoost (Chen & Guestrin, 2016), CatBoost (Prokhorenkova et al., 2018), the Vector CapsNet (Sabour et al., 2017), and fully connected neural networks (FCNNs) (Nair & Hinton, 2010) with lasso and mixup (Zhang et al., 2018) ($\alpha = 1.0$) respectively. The Vector CapsNet (Sabour et al., 2017) was designed for images; we replace its convolution layer by a fully connected layer for tabular data. Following (Popov et al., 2019), Arik & Pfister (2020), the performances of all the methods are hyperparameter-tuned on the validation sets (e.g., by grid search). The hyperparameter search spaces and search algorithms are given in Appendix A.1. We employ the “log-loss” metric to measure the model performances, which is considered as a more sensitive metric than accuracy, and is widely used (e.g., in Kaggle competitions). All the performances of the compared methods are obtained by running official codes.

4.2 Classification Performance Comparison

The classification performances are reported in Table 1. One can see that our TABCAPS obtains the best or second best performances on 7 out of 8 datasets, and is especially superior on some datasets by clear margins. In addition, we also report the computational complexity and efficiency on the Diabeties dataset, using the metrics of model size (# param.) and fps (the amount of records that a model processes per second). Since XGBoost and CatBoost are traditional machine learning models, we do not report their complexity. One can see that TABCAPS is much more efficient and greatly lighter (over $10 \times$) than the other state-of-the-art methods (e.g., FT-Transformer, Net-DNF, and DANet-24), and significantly outperforms the models with similar model complexity (e.g., regularized FCNNs and Vector CapsNet). Also, the results compared with Vector CapsNet shows the superiority of TABCAPS, obtaining better performances with lower complexity on tabular data.

4.3 Extreme Generalization Test

To further examine the generalization capability of TABCAPS, we attempt to test it with train-test split that the training data and test data are with very dissimilar feature values. We conduct t-SNE projections (with default hyper-parameters of the "sklearn" package) on all the records of datasets,
Vector CapsNet obtains (on log-loss values) during training, among several regularization strategies (e.g., FCNN with lasso, Extreme generalization performances. on training data keeps on decreasing or slightly fluctuating. One can see that the lasso and mixup log-loss values on the validation set after few training steps, and then their models gradually collapse and CatBoost), on the NODE and Net-DNF with ensemble learning strategies), and traditional boosting methods (XGboost and underlined “APS” factor. The reported log-loss values (the lower the better) are with a 100× factor. The model size (# param.) and inference speed (fps) are on the Diabetes dataset. The performances are reported as “mean±std”.

### Table 1: Classification Performances.

<table>
<thead>
<tr>
<th>Method</th>
<th>Click</th>
<th>Diabetes</th>
<th>EEG</th>
<th>Gas</th>
<th>Heart</th>
<th>Hill</th>
<th>Higgs</th>
<th>Epsilon</th>
<th># param.</th>
<th>fps</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGboost</td>
<td>66.07±0.00</td>
<td>65.86±0.09</td>
<td>70.65±0.01</td>
<td>31.50±0.04</td>
<td>35.65±0.01</td>
<td>69.65±0.09</td>
<td>54.55±0.04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Catboost</td>
<td>63.95±0.04</td>
<td>68.81±0.06</td>
<td>68.79±0.04</td>
<td>18.86±0.04</td>
<td>35.20±0.08</td>
<td>69.16±0.03</td>
<td>54.63±0.07</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TabNet</td>
<td>115.90±0.10</td>
<td>225.22±0.08</td>
<td>79.66±0.07</td>
<td>158.68±0.10</td>
<td>44.96±0.09</td>
<td>89.11±0.05</td>
<td>55.76±0.11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Net-DNF</td>
<td>67.62±0.02</td>
<td>58.79±0.05</td>
<td>68.26±0.04</td>
<td>15.12±0.03</td>
<td>55.37±0.07</td>
<td>48.30±0.04</td>
<td>55.73±0.06</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NODE</td>
<td>63.83±0.04</td>
<td>67.02±0.04</td>
<td>68.35±0.04</td>
<td>57.69±0.06</td>
<td>46.54±0.03</td>
<td>69.77±0.10</td>
<td>61.87±0.03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FT-Transformer</td>
<td>78.43±0.11</td>
<td>59.28±0.04</td>
<td>68.27±0.07</td>
<td>6.41±0.06</td>
<td>26.13±0.06</td>
<td>66.97±0.05</td>
<td>53.97±0.10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAnet-24</td>
<td>74.40±0.02</td>
<td>59.73±0.06</td>
<td>69.02±0.01</td>
<td>10.39±0.01</td>
<td>49.64±0.02</td>
<td>37.97±0.04</td>
<td>54.18±0.01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FCNN lasso</td>
<td>66.05±0.03</td>
<td>60.26±0.04</td>
<td>68.80±0.08</td>
<td>25.10±0.03</td>
<td>35.67±0.14</td>
<td>67.12±0.05</td>
<td>55.84±0.07</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vector CapsNet</td>
<td>106.12±0.03</td>
<td>67.08±0.04</td>
<td>67.17±0.04</td>
<td>61.31±0.02</td>
<td>76.85±0.07</td>
<td>75.85±0.02</td>
<td>106.58±0.06</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABCAPS (Ours)</td>
<td>64.72±0.05</td>
<td>66.09±0.02</td>
<td>67.84±0.01</td>
<td>163.19±0.04</td>
<td>60.84±0.04</td>
<td>64.74±0.09</td>
<td>62.79±0.02</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABCAPS (Ours)</td>
<td>63.35±0.04</td>
<td>58.40±0.02</td>
<td>67.47±0.01</td>
<td>8.75±0.02</td>
<td>34.50±0.03</td>
<td>17.88±0.04</td>
<td>54.70±0.07</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

and make the train-test split based on the t-SNE embedding (see the 2nd row in Table 2). We use the records inside the black rectangles for training (containing ~70% data) and the rest for test. Notably, the rectangles are biasedly placed and the records for training and test do not overlap in the t-SNE embedding, so that they are clearly in different distributions. The performances reported in Table 2 show that our TABCAPS obtains best or second best performances among deep learning approaches on most of datasets. Such results imply that our method can learn robust features and generalize well. Besides, our TABCAPS exhibits considerably better generalization performances than the Vector CapsNet with very clear margins (e.g., TABCAPS obtains 8.750 × 10^{-2} on Gas dataset, while the Vector CapsNet obtains 163.193 × 10^{-2}).

### 4.4 Performance Stability During Training

Many previous works required the early stop (Yao et al. 2007) to avoid over-fitting because they managed to exploit the potential of each tabular feature, and thus might be impacted by the uninformative features. However, our TABCAPS does not seem to require such training trick since it learns the data-level semantics directly. To show this, we compare the real-time validation performances (on log-loss values) during training, among several regularization strategies (e.g., FCNN with lasso, mixup (Zhang et al. 2018), commonly used neural networks (e.g., TabNet with sparse regularization, NODE and Net-DNF with ensemble learning strategies), and traditional boosting methods (XGboost and CatBoost), on the Click dataset. As shown in Fig. 3, most of these methods obtain their lowest log-loss values on the validation set after few training steps, and then their models gradually collapse as the training step increases. Notably, during the training process, all the corresponding loss values on training data keeps on decreasing or slightly fluctuating. One can see that the lasso and mixup...
Table 3: Ablation study results. For the missing value imputation (MVI) test, we report the performances of a XGBoost model trained on training sets with filled values. The log-loss values are 100×.

<table>
<thead>
<tr>
<th>Method</th>
<th>Routing &amp; projection</th>
<th>Classification fps</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) BoW Routing &amp; sparse projection</td>
<td>12.043±0.03</td>
<td>1666</td>
</tr>
<tr>
<td>(2) dynamic routing &amp; sparse projection</td>
<td>14.543±0.03</td>
<td>1342</td>
</tr>
<tr>
<td>(3) BoW Routing &amp; simple projection</td>
<td>13.106±0.03</td>
<td>1683</td>
</tr>
<tr>
<td>(4) BoW Routing fully connection</td>
<td>22.533±0.01</td>
<td></td>
</tr>
<tr>
<td>(5) Auto-Encoder with a classifier</td>
<td>26.338±0.07</td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td>Classification</td>
<td></td>
</tr>
<tr>
<td>(6)</td>
<td>2</td>
<td>12.043±0.03</td>
</tr>
<tr>
<td>(7)</td>
<td>3</td>
<td>6.254±0.04</td>
</tr>
<tr>
<td>(8)</td>
<td>4</td>
<td>11.138±0.03</td>
</tr>
</tbody>
</table>

Here we further inspect how our proposed BoW Routing determines the transformation paths for the vectorial features from primary capsules to senior capsules. We compute the averaged transformation weights \((w_{ij})\) in Eq. (6) over the whole test set and visualize them as in Fig. 4. One can see that a senior capsule feature is yielded by merging limited vectorial feature votes from primary capsules. Besides, we found the weights are sparse and the values are similar to different samples (with the standard deviation values varying from 0.001 to 0.003), which suggests that primary capsules model some semantically meaningful data patterns and our BoW Routing succeeds in finding out clear decision paths for classification.

4.5 EFFECTS OF BoW ROUTING

Here we further inspect how our proposed BoW Routing determines the transformation paths for the vectorial features from primary capsules to senior capsules. We compute the averaged transformation weights \((w_{ij})\) in Eq. (6) over the whole test set and visualize them as in Fig. 4. One can see that a senior capsule feature is yielded by merging limited vectorial feature votes from primary capsules. Besides, we found the weights are sparse and the values are similar to different samples (with the standard deviation values varying from 0.001 to 0.003), which suggests that primary capsules model some semantically meaningful data patterns and our BoW Routing succeeds in finding out clear decision paths for classification.

4.6 ABLATION STUDIES

We conduct ablation studies to examine the effects of the routing algorithms and the decoder (in a missing value imputation test) on the Diabetes dataset. We compare our BoW Routing with the dynamic routing \(\text{(Sabour et al., 2017)}\) on our architecture. See rows (1), (2), and (3) in Table 3. It is obvious that TABCAPS with our proposed BoW Routing outperforms that with the dynamic routing \(\text{(Sabour et al., 2017)}\) on the Diabetes data classification task by around 2.5 × 10⁻² on log-loss. We also compute the fps of the BoW Routing and dynamic routing, and find that BoW Routing is nearly 25% faster than the dynamic routing, suggesting the superiority of our straightforwardness design. Compared with the simple projection (without entmax), our sparse projection is witnessed to be beneficial and is compatible with the BoW Routing, which shows that the use of entmax benefits in finding informative features and excludes the useless ones.

To inspect the capability of our proposed decoder, we conduct a test on missing value imputation, and employ an XGBoost classifier to evaluate the effectiveness. We randomly mask out 20% of the feature values of the Diabetes training data, and simply fill the missing values with the means of feature values within the same classes. We train three models on the training set: (i) our TABCAPS with the proposed decoder (using the loss \(L\) in Eq. (8) and MSE), (ii) our TABCAPS with the fully-connection-based decoder of the Vector CapsNet \(\text{(Sabour et al., 2017)}\) (using \(L\) in Eq. (8) and MSE), and (iii) an auto-encoder with an auxiliary classifier that takes the hidden states as input (using MSE and cross-entropy loss). After training, we take the reconstructed feature values provided by these 3 decoders to refill the missing values respectively, and obtain 3 different training sets.

Figure 3: Real-time validation performances (log-loss, the lower the better) on Click dataset. A “step” (on the \(x\)-axis) indicates an “epoch” for neural networks and is a “boosting step” for XGBoost and CatBoost.
fairly compare the qualities of the reconstructed values, we train 3 XGboost models with identical hyper-parameters on these 3 training sets, and report their performances on the original test set. A better performance indicates that the reconstructed values are more trustworthy.

As reported in Table 3, XGboost that is trained on the training set provided by our decoder is markedly superior to the other two methods, while the fully-connection-based decoder (Sabour et al., 2017) performs the worst. Notably, we train another XGboost on the training set in which the masked values remain missing, and find that this XGboost version still obtains 26.174 on the test set. Thus, one can see that our proposed decoder is able to yield significant performance gain in filling the missing values, while the fully-connection-based decoder predicts noisy values.

The effects of capsule layers were tested. See rows (6), (7), and (8). The 3-layer and 4-layer TABCAPS obtain better performances on the Diabetes dataset compared with the default 2-layer model. On the Heart dataset, similar conclusion is attained that TABCAPS with 2, 3, 4 capsule layers obtained log-loss values of 0.34047, 0.33992, and 0.33673, respectively. In general, we consider TABCAPS with different capsule layers perform competitive, and a 2-capsule-layer model is sufficient to attain a good performance. We also inspected the effects of the “tabular feature extraction” module at the bottom of TABCAPS (see Fig. 1), and the effects of the ensemble of the senior capsules. Comparing row (9) with (10), and row (11) with (12), it is evident that both of the tabular feature extraction and the senior capsule ensemble are beneficial, but are not the core reasons of the superior performances of TABCAPS. We consider it is the novel architecture makes TABCAPS perform well.

5 Conclusions and Future Work

This paper proposed a novel and simple paradigm for supervised tabular learning, which encapsulated all features of a record into vectorial features as operating units and thus avoids interactions among heterogeneous tabular features. Consequently, our proposed new CapsNet (called TABCAPS), with a new primary capsule synthesis approach, a novel sparse projection, and a novel BoW Routing, coped with vectorial features and obtained superior performances on tabular data classification tasks at low computational costs. Specifically, the proposed BoW Routing performs straightforwardly without iterative clustering processes of previous CapsNets. This work is a beneficial attempt in the research areas of capsule neural networks, routing algorithms, and supervised tabular learning. Following previous works, our proposed method only considered classification tasks, and it would be interesting to explore how to use TABCAPS for tabular data regression. Moreover, it is also interesting to design other approach for directly learning data-level semantics without feature-wise interactions.

Acknowledgements. This research was partially supported by the National Key R&D Program of China under grant No. 2018AAA0102102 and National Natural Science Foundation of China under grants No. 62132017.

Reproducibility Statement. To ensure the reproducibility, we include the settings of hyper-parameter tuning for various methods in Appendix A.1. The classification performances measured by “accuracy” and “ROC-AUC” are reported in Appendix A.2 for reference. Besides, an additional discussion between our proposed “multivariate Gaussian embedding” and “linear feature embedding” approaches are given in Appendix A.3.
REFERENCES


KDD Cup. Predict the click-through rate of ads given the query and user information, 2012. https://www.kaggle.com/c/kddcup2012-track2/


A APPENDIX

A.1 HYPERPARAMETER-TUNING SETTINGS

As mentioned in the main paper, some hyperparameters of methods were turned for datasets by grid search or the HyperOpt library with 50 steps of the Tree-structured Parzen Estimator (TPE), similar to the configurations in (Popov et al., 2019). The details are listed in the subsections below. As for XGBoost, CatBoost, FCNN, TabNet, DANets, Vector CapsNet, and our TABCAPS, we used the HyperOpt hyperparameter search library; as for NODE and Net-DNF, we used grid search (following the settings in their papers). For FT-Transformer (Gorishniy et al., 2021), we used their official open source.

The hyperparameter search spaces and search algorithms of XGBoost (Chen & Guestrin, 2016), CatBoost (Prokhorenkova et al., 2018), NODE (Popov et al., 2019), and FCNN (Nair & Hinton, 2010) were set identical as in (Popov et al., 2019). We set the maximal number of the trees in XGBoost and CatBoost up to 4096 (instead of 2048 used in NODE (Popov et al., 2019)), since we found that 2048 is not enough for XGBoost and CatBoost to obtain good performances. We also set the maximal layer number in FCNN up to 9. Besides, the hyperparameter search settings of Net-DNF (Abutbul et al., 2021) following its original paper. The architectures of FCNN with lasso regularization or mixup were constructed following the FCNN in (Popov et al., 2019). For both of DANet-24 (Chen et al., 2022) and TabNet (Arik & Pfister, 2020), we set the hyperparameter search settings to contain all the possible values provided by the main papers.

A.1.1 XGBOOST

For easy re-implementation, the hyperparameter variable names followed the sklearn package settings.

- \textit{eta} (learning rate): Log-Uniform distribution $[e^{-7}, 1]$;
- \textit{max_depth}: Discrete uniform distribution $[2, 10]$;
- \textit{subsample}: Uniform distribution $[0.5, 1]$;
- \textit{colsample_bytree}: Uniform distribution $[0.5, 1]$;
- \textit{colsample_bylevel}: Uniform distribution $[0.5, 1]$;
- \textit{min_child_weight}: Uniform distribution $[e^{-16}, e^{5}]$;
- \textit{alpha}: Uniform choice $\{0, \text{Log-Uniform distribution } [e^{-16}, e^{2}]\}$;
- \textit{lambda}: Uniform choice $\{0, \text{Log-Uniform distribution } [e^{-16}, e^{2}]\}$;
- \textit{gamma}: Uniform choice $\{0, \text{Log-Uniform distribution } [e^{-16}, e^{2}]\}$.

A.1.2 CATBOOST

For easy re-implementation, the hyperparameter variable names followed the sklearn package settings.

- \textit{learning_rate}: Log-Uniform distribution $[e^{-5}, 1]$;
- \textit{random_strength}: Discrete uniform distribution $[1, 20]$;
- \textit{one_hot_max_size}: Discrete uniform distribution $[0, 25]$;
- \textit{i2.leaf_reg}: Log-Uniform distribution $[1, 10]$;
- \textit{bagging_temperature}: Uniform distribution $[0, 1]$;
- \textit{leaf_estimation_iterations}: Discrete uniform distribution $[1, 10]$.

\url{https://github.com/hyperopt/hyperopt}
\url{https://scikit-learn.org/}
A.1.3 NODE

- number of layers: \{2, 4, 8\};
- total number of trees: \{1024, 2048\};
- tree depth: \{6, 8\};
- output dimension of trees: \{2, 3\};
- learning rate: $10^{-3}$.

A.1.4 NET-DNF

Following the original setting in (Abutbul et al., 2021), the learning rate is initialized as 0.05 and is reduced by monitoring the training loss (check its official codes for details).

- number of formulas: \{64, 128, 256, 512, 1024, 2048, 3072\};
- feature selection beta: \{1.6, 1.3, 1.0, 0.7, 0.4, 0.1\}.

A.1.5 TabNet

- $\lambda_{\text{sparse}}$: \{0.0, 10^{-6}, 10^{-4}, 10^{-3}, 10^{-2}, 0.1\};
- $\gamma$: \{1.0, 1.2, 1.5, 2.0\};
- batch size $B$: \{256, 512, 1024, 2048, 4096, 8192, 16384, 32768\};
- $N_d$ and $N_a$: \{8, 16, 24, 32, 64, 128\};
- $N_{\text{steps}}$: \{3, 4, 5, 6, 7, 8, 9, 10\};
- $B_V$: \{256, 512, 1024, 2048, 4096\};
- $m_B$: \{0.6, 0.7, 0.8, 0.9, 0.95, 0.98\};
- learning rate: \{0.002, 0.01, 0.02, 0.025\}.

A.1.6 FCNN

- number of layers: Discrete uniform distribution [2, 9];
- output dimensions of full-connections: Discrete uniform distribution \{128, 256, 512, 1024\};
- learning rate: Uniform distribution \{10^{-4}, 10^{-2}\};
- dropout rate: Uniform distribution \{0, 0.5\};
- lasso coefficient (optional): Discrete uniform distribution uniform \{10^{-7}, 10^{-4}\};
- learning rate: Uniform distribution \{10^{-4}, 0.01\}.

A.1.7 FT-Transformer

- the number of layers: Discrete uniform distribution [1, 4];
- feature embedding size: Discrete uniform distribution \{64, 512\};
- residual dropout: Uniform distribution \{0, 0.2\};
- attention dropout: Uniform distribution \{0, 0.5\};
- FFN dropout: Uniform distribution \{0, 0.5\};
- FFN factor: Uniform distribution \{\frac{2}{3}, \frac{5}{6}\};
- learning rate: Log-Uniform distribution \{10^{-5}, 10^{-3}\};
- weight decay: Log-Uniform distribution \{10^{-6}, 10^{-3}\};
- the number of iterations: 100.
A.1.8 DANet-24
The learning rate of DANet-24 is initially set 0.008 and is decayed by 5% in every 20 epochs.

- $k_0$: $\{5, 8\}$;
- $d_0$: $\{32, 48\}$;
- $d_1$: $\{64, 96\}$.

A.1.9 Vector CapsNet
- learning rate: Uniform distribution $[0.02, 0.1]$;
- primary capsule size: Discrete uniform distribution $[64, 128]$;
- the number of primary capsules: Discrete uniform distribution $[4, 32]$;
- senior capsule size: Discrete uniform distribution $[4, 32]$.

A.1.10 TabCaps
- learning rate: Uniform distribution $[0.02, 0.1]$;
- the number of senior capsules for a class: Discrete uniform distribution $[1, 5]$;
- primary capsule size: Discrete uniform distribution $[64, 128]$;
- the number of primary capsules: Discrete uniform distribution $[4, 32]$;
- senior capsule size: Discrete uniform distribution $[4, 32]$;
- the number of learnable “words”: Discrete uniform distribution $[16, 64]$.
A.2 Classification Performances on “Accuracy” and “ROC-AUC”

Here we report the model performances measured by the “accuracy” in Table 4 and the binary classification performances measured by “ROC-AUC” in Table 5. These classification performances are obtained in 5 runs with regard to random seeds. One can see that our TABCAPS obtains better or competitive performances in comparison with previous works.

### Table 4: Classification Performances measured by “accuracy”.

<table>
<thead>
<tr>
<th>Method</th>
<th>Click</th>
<th>Diabetes</th>
<th>EEG</th>
<th>Gas</th>
<th>Heart</th>
<th>Hill</th>
<th>Hig</th>
<th>Epsilon</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGboost</td>
<td>0.6666</td>
<td>0.9575</td>
<td>0.9342</td>
<td>0.9957</td>
<td>0.8833</td>
<td>0.5761</td>
<td>0.7262</td>
<td>0.8884</td>
</tr>
<tr>
<td>Catboost</td>
<td>0.6599</td>
<td>0.9250</td>
<td>0.9322</td>
<td>0.9935</td>
<td>0.8501</td>
<td>0.4856</td>
<td>0.7256</td>
<td>0.8878</td>
</tr>
<tr>
<td>TabNet</td>
<td>0.6684</td>
<td>0.9725</td>
<td>0.8815</td>
<td>0.9949</td>
<td>0.8166</td>
<td>0.9012</td>
<td>0.7190</td>
<td>0.8980</td>
</tr>
<tr>
<td>Net-DNF</td>
<td>0.6694</td>
<td>0.9775</td>
<td>0.9379</td>
<td>0.9942</td>
<td>0.8333</td>
<td>0.7819</td>
<td>0.7268</td>
<td>0.8922</td>
</tr>
<tr>
<td>NODE</td>
<td>0.6688</td>
<td>0.7903</td>
<td>0.7871</td>
<td>0.8606</td>
<td>0.8667</td>
<td>0.6214</td>
<td>0.7260</td>
<td>0.8952</td>
</tr>
<tr>
<td>FT-Transformer</td>
<td>0.6606</td>
<td>0.9725</td>
<td>0.9826</td>
<td>0.9953</td>
<td>0.9166</td>
<td>0.8971</td>
<td>0.7290</td>
<td>0.8976</td>
</tr>
<tr>
<td>DANet-24</td>
<td>0.6698</td>
<td>0.9687</td>
<td>0.9776</td>
<td>0.9949</td>
<td>0.8333</td>
<td>0.9095</td>
<td>0.7284</td>
<td>0.8962</td>
</tr>
<tr>
<td>FCNN mixup</td>
<td>0.6674</td>
<td>0.9375</td>
<td>0.9746</td>
<td>0.9956</td>
<td>0.7333</td>
<td>0.9136</td>
<td>0.7075</td>
<td>0.8934</td>
</tr>
<tr>
<td>FCNN lasso</td>
<td>0.6670</td>
<td>0.9403</td>
<td>0.9639</td>
<td>0.9863</td>
<td>0.6677</td>
<td>0.8949</td>
<td>0.6677</td>
<td>0.8958</td>
</tr>
<tr>
<td>Vector CapsNet</td>
<td>0.6678</td>
<td>0.9025</td>
<td>0.9726</td>
<td>0.9939</td>
<td>0.6833</td>
<td>0.6419</td>
<td>0.7102</td>
<td>0.8905</td>
</tr>
<tr>
<td>TABCAPS (Ours)</td>
<td>0.6685</td>
<td>0.9725</td>
<td>0.9776</td>
<td>0.9968</td>
<td>0.9023</td>
<td>0.9012</td>
<td>0.7300</td>
<td>0.8975</td>
</tr>
</tbody>
</table>

### Table 5: Binary classification Performances measured by “ROC-AUC”. The Gas dataset is not included because it is not for binary classification.

<table>
<thead>
<tr>
<th>Method</th>
<th>Click</th>
<th>Diabetes</th>
<th>EEG</th>
<th>Heart</th>
<th>Hill</th>
<th>Hig</th>
<th>Epsilon</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGboost</td>
<td>0.7216</td>
<td>0.9859</td>
<td>0.9884</td>
<td>0.9447</td>
<td>0.5939</td>
<td>0.8054</td>
<td>0.9542</td>
</tr>
<tr>
<td>Catboost</td>
<td>0.7199</td>
<td>0.9878</td>
<td>0.9859</td>
<td>0.9330</td>
<td>0.4965</td>
<td>0.8048</td>
<td>0.9524</td>
</tr>
<tr>
<td>TabNet</td>
<td>0.7122</td>
<td>0.9853</td>
<td>0.9535</td>
<td>0.9095</td>
<td>0.9632</td>
<td>0.7911</td>
<td>0.9562</td>
</tr>
<tr>
<td>Net-DNF</td>
<td>0.7169</td>
<td>0.9932</td>
<td>0.9838</td>
<td>0.9412</td>
<td>0.8707</td>
<td>0.8018</td>
<td>0.9532</td>
</tr>
<tr>
<td>NODE</td>
<td>0.7146</td>
<td>0.8411</td>
<td>0.8173</td>
<td>0.9506</td>
<td>0.6626</td>
<td>0.7444</td>
<td>0.9497</td>
</tr>
<tr>
<td>FT-Transformer</td>
<td>0.7180</td>
<td>0.9883</td>
<td>0.9978</td>
<td>0.9764</td>
<td>0.9833</td>
<td>0.8078</td>
<td>0.9586</td>
</tr>
<tr>
<td>DANet-24</td>
<td>0.7189</td>
<td>0.9825</td>
<td>0.9981</td>
<td>0.8931</td>
<td>0.9710</td>
<td>0.8042</td>
<td>0.9582</td>
</tr>
<tr>
<td>FCNN mixup</td>
<td>0.7178</td>
<td>0.9653</td>
<td>0.9962</td>
<td>0.8978</td>
<td>0.9531</td>
<td>0.7844</td>
<td>0.9598</td>
</tr>
<tr>
<td>FCNN lasso</td>
<td>0.7128</td>
<td>0.9702</td>
<td>0.9958</td>
<td>0.8343</td>
<td>0.5809</td>
<td>0.7247</td>
<td>0.9323</td>
</tr>
<tr>
<td>Vector CapsNet</td>
<td>0.7168</td>
<td>0.9902</td>
<td>0.9965</td>
<td>0.9036</td>
<td>0.5724</td>
<td>0.7754</td>
<td>0.9545</td>
</tr>
<tr>
<td>TABCAPS (Ours)</td>
<td>0.7199</td>
<td>0.9907</td>
<td>0.9970</td>
<td>0.9436</td>
<td>0.9911</td>
<td>0.8081</td>
<td>0.9606</td>
</tr>
</tbody>
</table>

A.3 Performances with Other Embedding Approaches

In this paper, we devise multivariate Gaussian kernels to synthesize vectorial features. Here we compare our proposed multivariate Gaussian embedding approach with the commonly used linear feature embedding approach (like the embedding approach in FT-Transformer). On the Diabetes dataset, TABCAPS with the linear embedding layer achieves 0.1328 on log-loss (↓) while TABCAPS with multivariate Gaussian kernels achieves 0.1204. The comparison results suggest that our multivariate Gaussian kernels is beneficial, which might be due to the non-linearity of the multivariate Gaussian embedding approach.