TFWT: Tabular Feature Weighting with Transformer
Xinhao Zhang\textsuperscript{1}, Zaitian Wang\textsuperscript{2,3}, Lu Jiang\textsuperscript{4}, Wanfu Gao\textsuperscript{5}, Pengfei Wang\textsuperscript{2,3,*} and Kunpeng Liu\textsuperscript{1,*}
\textsuperscript{1}Portland State University
\textsuperscript{2}Computer Network Information Center, Chinese Academy of Sciences
\textsuperscript{3}University of Chinese Academy of Sciences, Chinese Academy of Sciences
\textsuperscript{4}Dalian Maritime University
\textsuperscript{5}Jilin University
xinhaoz@pdx.edu, wangzaitian23@mails.ucas.ac.cn, jianglei761@dlmu.edu.cn, gaowf@jlu.edu.cn, pfwang@cnic.cn, kumpeng@pdx.edu

Abstract
In this paper, we propose a novel feature weighting method to address the limitation of existing feature processing methods for tabular data. Typically the existing methods assume equal importance across all samples and features in one dataset. This simplified processing methods overlook the unique contributions of each feature, and thus may miss important feature information. As a result, it leads to suboptimal performance in complex datasets with rich features. To address this problem, we introduce Tabular Feature Weighting with Transformer, a novel feature weighting approach for tabular data. Our method adopts Transformer to capture complex feature dependencies and contextually assign appropriate weights to discrete and continuous features. Besides, we employ a reinforcement learning strategy to further fine-tune the weighting process. Our extensive experimental results across various real-world datasets and diverse downstream tasks show the effectiveness of TFWT and highlight the potential for enhancing feature weighting in tabular data analysis.

1 Introduction
Extracting feature information from data is one of the most crucial tasks in machine learning, especially for classification and prediction tasks [Bishop, 1995]. Different features play varied roles in data representation and pattern recognition, thus directly impacting the model’s learning efficiency and prediction accuracy. Effective feature engineering can enhance a model’s ability to handle complex data significantly and help capture critical information in the data.

In feature engineering, a traditional and common assumption is that each feature in a dataset is equally important. Treating all features equally important does simplify the process, but it ignores the fact that each feature contributes uniquely to the downstream task [Daszykowski et al., 2007]. Some features with rich information may significantly impact the model outcome, while others may contribute less or even introduce noise and mislead the model [García et al., 2015]. Treating all features with equal weight may dilute the information of important features and cause less important features to over-influence the model, thus diminishing the overall effectiveness of information extraction. Feature weighting is a technique that assigns weights to each feature in the dataset. The main goal of feature weighting is to optimize the feature space by assigning weights to each feature according to feature importance, thus enhancing the model performance. Feature weighting methods can be categorized based on learning strategies and weighting methods. Supervised feature weighting uses actual data labels to determine the weights of features [Chen and Hao, 2017a; Niño-Adan et al., 2020; Wang et al., 2022]. Unsupervised feature weighting utilizes the intrinsic characteristics of the dataset to assign weight without relying on label information [Zhang et al., 2018].

Feature weighting can also be classified based on feature types or feature information. Weighting by feature types focuses on the inherent properties of features, like discreteness or continuity. This approach is often influenced by data structure [Zhou et al., 2021; Xue et al., 2023; Hashemzadeh et al., 2019; Cardie and Howe, 1997; Lee et al., 2011]. Conversely, weighting by feature information emphasizes the informational content of features. It assesses the importance of features based on their correlation and contribution to the predictive model, typically using statistical measures or machine learning algorithms [Kang et al., 2019; Liu et al., 2004; Druck et al., 2008; Zhu et al., 2023a;}

\textsuperscript{*}Corresponding authors.
Liu et al., 2019]. However, these methods do not effectively capture the complex relationships between features and cause risks like overfitting, local optimality, and noise sensitivity. Moreover, as illustrated in Figure 1, these methods assign the same weight to every sample (row) of each feature, rather than assigning personalized weights to different samples.

In addressing the limitations of traditional feature weighting methods, we adopt a new approach based on the Transformer model. One core feature of the Transformer is its self-attention mechanism [Vaswani et al., 2017; Radford et al., 2019; Luong et al., 2015; Zhou et al., 2023; Siriwardhana et al., 2020; Hashemzadeh et al., 2019]. This mechanism can effectively identify complex dependencies and interactions among data features. In feature weighting, the self-attention mechanism assigns attention weights by considering the relevance and contribution of features, enabling the model to learn and adapt to specific dataset patterns during the training process. Thus, our Transformer-based approach can assign higher weights to features that significantly influence the model output. In this way, the model can focus on the most critical information. With self-attention, the Transformer can effectively capture the contextual information and inter-feature relationships within the tabular data.

The Transformer also employs a multi-head attention mechanism operated with several self-attention components operating in parallel. Each attention head focuses on different aspects of the dataset, and captures diverse patterns and dependencies. Thus, the model can understand the feature relationships from multiple perspectives. This application of multi-head attention significantly enhances the model’s capability in determining feature weights. Thus, by integrating self-attention and multi-head attention mechanisms, our Transformer-based method effectively identifies and assigns feature weights. It adapts to various data patterns and complicated relationships among features.

To further stabilize and enhance this weighting structure, we need an effective fine-tuning method. Adopting reinforcement learning [Schulman et al., 2017; Fan et al., 2020] for fine-tuning is a common strategy [Ziegler et al., 2019; Fickinger et al., 2021; Ouyang et al., 2022; Zhu et al., 2020]. Notably, the Proximal Policy Optimization (PPO) network [Schulman et al., 2017] has the advantage of stability and efficiency in fine-tuning by enhancing policies while ensuring stable updates [Zhu et al., 2023b]. Specifically, the PPO network fits the task of reducing information redundancy within the data. In this scenario, information redundancy refers to the presence of repetitive or irrelevant information. Redundant features lead to possible overfitting in training. By reducing redundancy, the data becomes more concise and focused on the most informative features. By minimizing redundancy, learning becomes more stable and focused, which helps to decrease classification variance.

In summary, we propose a novel feature weighting method aiming to tackle several challenges: first, how to generate appropriate weights of features; second, how to evaluate the effectiveness of feature weights; and finally, how to fine-tune the feature weights according to the feedback from downstream tasks. Hence, we introduce a Transformer-enhanced feature weighting framework in response to the challenges outlined. This framework leverages the strength of the Transformer architecture to assign weights to features by capturing intricate contextual relationships among these features. We evaluate the effectiveness of feature weighting by the improvement of downstream task’s performance. Further, we adopt a reinforcement learning strategy to fine-tune the output and reduce information variance. This adjustment enhances the model’s stability and reliability.

Our contributions are summarized as follows:

- We propose a novel feature weighting method for tabular data based on Transformer called TFWT. This new method can capture the dependencies between features with Transformer’s attention mechanism to assign and adjust weights for features according to the feedback of downstream tasks.
- We propose a fine-tuning method for the weighting process to further enhance the performance. This fine-tuning method adopts a reinforcement learning strategy, reducing the data information redundancy and classification variance.
- We conduct extensive experiments and show that TFWT achieves significant performance improvements under varying datasets and downstream tasks, comparing with raw classifiers and baseline models. The experiments also show the effectiveness of fine-tuning process in reducing redundancy.

2 Related Work

2.1 Feature Weighting

Feature weighting, vital for enhancing machine learning, includes several approaches [Chen and Guo, 2015; Chen and Hao, 2017b; Chowdhury et al., 2023; Wang et al., 2004; Yeung and Wang, 2002]. [Liu et al., 2004], [Druck et al., 2008], and [Raghavan et al., 2006] explored feedback integration, model constraints, and active learning enhancement. [Wang et al., 2013] proposed an active SVM method for image retrieval. Techniques like weighted bootstrapping [Barbe and Bertail, 1995], chi-squared tests, TabTransformer [Huang et al., 2020], and cost-sensitive learning adjust weights through feature changes. These methods have limitations like overfitting or ignoring interactions. Our study focuses on adaptable weight distribution and improvement through feedback.

2.2 Transformer

The Transformer architecture, introduced by [Vaswani et al., 2017], has revolutionized many fields including natural language processing. Instead of relying on recurrence like its predecessors, it utilizes self-attention mechanisms to capture dependencies regardless of their distance in the input data. This innovation has led to several breakthroughs in various tasks. For instance, BERT model [Devlin et al., 2018; Clark et al., 2019], built upon the Transformer, set new records in multiple NLP benchmarks. Later, [Radford et al., 2019] extended these ideas with GPT-2 and GPT-3 [Brown et
al., 2020], demonstrating impressive language generation capabilities. Concurrently, Raffel et al., 2020] proposed a unified text-to-text framework for NLP transfer learning, achieving state-of-the-art results across multiple tasks.

3 Methodology

3.1 Problem Formulation

We consider the problem setting of classification. Let \( D = \{ F, y \} \) be a dataset with \( K \) features and \( N \) samples. We define the feature matrix \( F = \{ f_k \}_{k=1}^{K} \). We use \( f_k = [f_k^1, \ldots, f_k^N]^T \) to denote the \( k \)-th feature, where \( f_k^i \) is the value of \( i \)-th sample on the \( k \)-th feature. \( y = [y_1, \ldots, y_N]^T \) is the label vector. Without loss of generality, we assume the first \( M \) features to be discrete, and the remaining \( K - M \) features to be continuous.

In defining a weighting matrix \( W \in \mathbb{R}^{N \times K} \), each of whose elements corresponds to the elements of the feature matrix \( F \). This weighting matrix \( W \) is applied element-wise to \( F \) to produce a weighted matrix \( F_{rew} = W \odot F \), where \( \odot \) denotes the Hadamard product. In the weighting problem, we aim to find an optimized \( W \), so that \( F_{rew} \) can improve the downstream tasks’ performance when substituting the original feature matrix \( F \) in predicting \( y \).

3.2 Framework

We propose TFWT, a Tabular Feature Weighting with Transformer method for tabular data. We aim to improve downstream tasks’ performance by effectively incorporating the attention mechanism to capture the relations and interactions between features. To achieve this goal, we design a Transformer-based feature weighting pipeline with a fine-tuning strategy. As Figure 2 shows, our method consists of three components: In the Feature Alignment, we align different types of original features so that they are in the same space. In the Feature Weighting, we encode the feature matrix to get its embedding via Transformer encoders, and then decode the embedding into feature weights. In the Fine-Tuning, we design a reinforcement learning strategy to fine-tune the feature weights based on feedback from downstream tasks.

3.3 Feature Alignment

To effectively extract tabular data’s features while maintaining a streamlined computation, we convert both discrete and continuous features into numerical vectors.

Discrete Feature Alignment. We first encode the discrete features into numerical values. The encoded numerical values are then passed to a dense embedding layer, transforming them into vectors for subsequent processes. For each discrete feature \( f_k \) \( (k = 1, \ldots, M) \), the encoded vector is:

\[
\begin{align*}
\nu_k &= \text{Dense}(f_k).
\end{align*}
\] (1)

Continuous Feature Alignment. We normalize all the continuous features with mean of 0 and variance of 1. We then design a linear layer to align their length with discrete features. For each continuous feature \( f_k \) \( (k = M + 1, \ldots, K) \), the encoded vector is:

\[
\begin{align*}
\mathbf{u}_k &= \text{Linear} \left( \frac{f_k - \mu_k}{\sigma_k} \right),
\end{align*}
\] (2)

where \( \mu_k \) and \( \sigma_k \) are the mean and standard deviation of the \( k \)-th feature, respectively. Then the aligned feature matrix \( F' \) is formed by concatenating these vectors:

\[
F' = [v_1, \ldots, v_M, u_{M+1}, \ldots, u_K].
\] (3)

3.4 Feature Weighting

Given aligned feature matrix \( F' \), we aim to explore the relationships between features and assign proper feature weights.

Data Encoding. To enhance the model’s understanding and extract latent patterns and relations from the data, we put \( F' \) into the encoders with a multi-head self-attention mechanism. This mechanism processes the embedded feature matrix \( F' \) by projecting it into query (Q), key (K), and value (V) spaces.

The encoder then applies the self-attention mechanism to capture varying feature relations in the feature matrix and assigns distinct attention weights to them. Assuming \( d_k \) is the dimensionality of the key vectors, the attention mechanism is formulated as:

\[
\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V,
\] (4)

where \( Q = W_Q \cdot F', K = W_K \cdot F', V = W_V \cdot F, W_Q, W_K, W_V \) are parameter matrices.

In our method, we adopt the multi-head attention mechanism, where the results of each head are concatenated and linearly transformed. Assuming \( W^O \) is an output projection matrix and \( Z \) is the feature representation:

\[
\begin{align*}
\text{head}_i &= \text{Attention}(W_i^Q \cdot Q, W_i^K \cdot K, W_i^V \cdot V), \\
\text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W^O, \\
Z &= \text{ResNet}(\text{MultiHead}(Q, K, V)),
\end{align*}
\] (5)

where \( W_i^Q, W_i^K \), and \( W_i^V \) are weights for query, key, and value. Through this process, we obtain the feature representation \( Z \) that captures feature relationships. Specifically, \( Z \) is obtained by passing the input feature matrix through multiple layers of the encoder, where each layer applies self-attention and residual connection-enhanced feedforward networks.

Weight Decoding. In this process, we aim to decode a weighting matrix \( W \) from the embedding \( Z \). This decoding process iteratively updates \( W \) until the downstream task’s performance is satisfied. We initialize the \( W \) by setting all its elements as 1. This is to ensure all features receive equal importance at the beginning. In each decoding layer, we do cross-attention on \( W \) and \( Z \) by:

\[
\begin{align*}
\text{CrossAttention}(Q_w, K_z, V_z) &= \text{softmax} \left( \frac{Q_w K_z^T}{\sqrt{d_z}} \right) V_z,
\end{align*}
\] (6)

where \( Q_w = W_Q \cdot W, K_z = W_K \cdot Z, \) and \( V = W_V \cdot Z, W_Q, W_K, W_V \) are parameter matrices.

By adopting a cross-attention mechanism, we generate a contextual representation that captures various relationships and dependencies in the feature matrix. After several weight decoding layers, we get an updated weighting matrix \( W \):

\[
W = \text{ResNet}(\text{CrossAttention}(Q_w, K_z, V_z)).
\] (7)
We finally use the weighting matrix $W$ to derive a weighted feature matrix $F_{rew}$ by its Hadamard product with the original feature matrix $F$: $F_{rew} = W \odot F$. With this weighted feature matrix, we reorganize the feature space and make features optimized for the downstream task. $F_{rew}$ is then used to substitute $F$ in the downstream tasks.

### 3.5 Fine-Tuning

In the fine-tuning process, our primary goal is to adopt a reinforcement learning strategy to adjust the weighting matrix $W$. This adjustment aims to reduce information redundancy of $F_{rew}$, thereby reducing the variance during training.

#### Weighting Matrix Refinement

We begin by evaluating the redundancy, denoted as $Rdd$, using mutual information as defined by [Shannon, 1948]. $Rdd$ is calculated as follows:

$$ Rdd = \frac{1}{|F|^2} \sum_{f_m, f_n \in F} I(f_m, f_n). $$

In this formula, $F$ represents the feature matrix, with $f_m$ and $f_n$ being the $m$-th and $n$-th features, respectively. The function $I(f_m, f_n)$ measures the mutual information between these two features. We further define $\Delta Rdd$ to represent the change in redundancy, where $\Delta Rdd = Rdd' - Rdd$, $Rdd'$ is the redundancy of the feature matrix after fine-tuning.

Next, we process the input weighting matrix $W$ through a RL model. In this paper, we adopt a Proximal Policy Optimization (PPO) as our RL model, which comprises one actor network and one critic network [Schulman et al., 2017]. While the actor network focuses on determining the actions to take, the critic evaluates how good those actions are, based on the expected rewards. In this context, an action is defined as the output of the PPO network, which is an adjusted weighting matrix $W'$. The state is the weighting matrix $W$ and the reward is the change of redundancy $\Delta Rdd$.

Specifically, the actor network processes $W$ to produce mean and variance values. These values are then used to form a probability distribution matrix $V$, which consists of Gaussian distributions, represented as:

$$ V = (V_k^1, V_k^2, \ldots, V_k^N), $$

$$ V_k^i(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(u_k^i - \mu)^2}{2\sigma^2}}. $$

Here, $V_k^i$ indicates the weight distribution of $u_k^i$, the $i$-th element of the $k$-th feature, with $\mu$ and $\sigma^2$ being the mean and variance, respectively. Here we form $W'$ with each element $w_k^i$ sampled from the probability distribution $V_k^i$.

#### Actor Network Update

After refining the weighting matrix, we update the feature matrix as $F'_{rew} = W' \odot F$, and subsequently calculate the information redundancy $Rdd'$ of $F'_{rew}$. Based on the observed change of redundancy $\Delta Rdd$, we adjust the mean and variance within the probability distribution, following the equations:

$$ \theta' \leftarrow \theta + \alpha \cdot \Delta Rdd \cdot \nabla J(\theta), $$

$$ \nabla J(\theta) = \frac{1}{n} \sum_{i=1}^n \left( \nabla \log \pi_\theta(a_i | s_i) \right) R_i, $$

where $\theta$ is the parameter of the actor network, $\alpha$ is the learning rate, $J(\theta)$ is the objective function to maximize, $\nabla J(\theta)$ is the gradient of the objective function with respect to the mean and variance, $n$ is the number of state-action pairs in the training, $\pi_\theta(a_i | s_i)$ is the policy, and $R_i$ is the reward of state-action pair $(s_i, a_i)$. To ensure a stable fine-tuning process, we implement a clipping mechanism for the updated means. Specifically, we adopt each mean $\mu_i$ using the formula: $\mu_i = \min(\max(\mu_i w_i + \epsilon, w_i - \epsilon)).$ This clipping process is crucial for as it prevents excessive deviations from the current weight $w_i$, thereby maintaining the stability and reducing the variance during downstream training.

#### Critic Network Update

After the update of actor network, we continue to adjust the critic network with the reward. We design the critic network to provide an estimate of the advantage function $A(s, a)$. This advantage function represents the...
expected future advantages under the state \( s \) and action \( a \). We design the function to change the policy gradually based on the current state, so that the policy after adjustment \( \pi'_\theta(a|s) \) is not too biased from the previous policy \( \pi_\theta(a|s) \). We adopt the clipping mechanism with a parameter \( \epsilon \) as well as the advantage function \( A(s, a) \) in the loss function:

\[
L(\theta) = \mathbb{E}_{a,s} \left[ \min \left( \frac{\pi_\theta(a|s)}{\pi'_\theta(a|s)} A(s, a), \right.ight.
\]

\[
\left. \left. \text{clip} \left( \frac{\pi_\theta(a|s)}{\pi'_\theta(a|s)} \right) (1 - \epsilon, 1 + \epsilon) A(s, a) \right) \right]. \tag{15}
\]

By minimizing \( L(\theta) \), we continuously optimize the feature matrix to obtain stable and enhanced performance.

### Algorithm 1: Training of TFWT

**Input:** dataset \( D = \{F, y\} \)

**Output:** weighted feature matrix \( F'_{\text{rew}} \)

for iteration = 0, 1, 2, \ldots, \ do

for each encoder do

\[ Q \leftarrow W_Q \cdot F', K \leftarrow W_K \cdot F', \text{ and} \]

\[ V \leftarrow W_V \cdot F' \]

Compute MultiHead\((Q, K, V)\) by Eq.4 and Eq.6

\[ Z \leftarrow \text{ResNet(MultiHead}(Q, K, V)) \]

for each decoder do

\[ Q_{w} \leftarrow W_Q \cdot W, K_{z} \leftarrow W_K \cdot Z, \text{ and} \]

\[ V \leftarrow W_V \cdot Z \]

Compute CrossAttention\((Q_{w}, K_{z}, V_{z})\) by Eq.8

\[ W \leftarrow \text{ResNet(CrossAttention}(Q_{w}, K_{z}, V_{z})) \]

\[ F'_{\text{rew}} \leftarrow W \odot F \]

for each state-action pairs do

Get \((\mu, \sigma)\) by process \( W \) through PPO network

Estimate \( V \) by \((\mu, \sigma)\) by Eq.11 and Eq.12

Sample \( W' \) from \( V \)

\[ F''_{\text{rew}} \leftarrow W' \odot F \]

Compute \( \text{Rdd} \) for \( F'_{\text{rew}}, \text{Rdd'} \) for \( F''_{\text{rew}} \) by Eq.10

\[ \Delta \text{Rdd} \leftarrow \text{Rdd'} - \text{Rdd} \]

Update actor network parameter by Eq.13

Update critic network parameter by Eq.15

Pretrain a predictive model \( M \) with \( D \)

Get \( y \) by perform \( M \) on \( F'_{\text{rew}} \), and compute cross-entropy loss between \( \hat{y} \) and \( y \).

Backpropagate the loss to update parameters in the encoders and decoders.

### 3.6 Training of TFWT

As Algorithm 1 shows, we first align original features by Eq.1 and Eq.2. Then, we encode the aligned feature matrix \( F' \) into an embedding \( Z \) and decode it into a weighting matrix \( W \).

This encoding-decoding process is accomplished by a designated Transformer. In this way, we get a weighted feature matrix \( F'_{\text{rew}} \). To further fine-tune \( W \), we adopt PPO as a reinforcement learning model to reduce the redundancy of \( F'_{\text{rew}} \). The fine-tuned \( W^T \) is sampled from the actor network of PPO. The PPO networks are trained by the interaction data from the sampling process. The cross-entropy loss derived from the downstream machine learning model is used to update the parameters of the encoders and decoders in the Transformer.

### 4 Experiments

In this section, we present three experiments that validate the strength of our method. First, we demonstrate that our method significantly enhances the performance on various downstream tasks without fine-tuning. Second, we demonstrate the advantages of our TFWT method over the baseline methods. Finally, we demonstrate the effectiveness of fine-tuning comparing with non-fine-tuning version of TFWT in reducing the variance performance metrics. Overall, the results consistently demonstrate the superior performance of our method.

#### 4.1 Experimental Settings

**Datasets.** We evaluate the proposed method with four real-world datasets:

- **Amazon Commerce Reviews Set (AM)** [Liu, 2011] from UCI consists of customer reviews from the Amazon Commerce website. Its purpose is to classify the identities of authors of reviews by analyzing textual patterns. We have randomly divided its 50 labels into two groups, each containing 25 labels, transforming it into a balanced binary classification task.

- **Online Shoppers Purchasing Intention Dataset (OS)** [Sakar and Kastro, 2018] from UCI features multivariate data types including integer and real values. Its purpose is to classify shoppers’ purchasing intentions and predict purchases.

- **MAGIC Gamma Telescope Dataset (MA)** [Bock, 2007] from UCI reflects the simulation of high energy gamma particles registration in a gamma telescope. Its purpose is to classify the primary gammas from cosmic rays.

- **Smoking and Drinking Dataset with body signal (SD)** [Her, 2023] from Kaggle is collected from National Health Insurance Service in Korea. Its purpose is to classify “smoker” or “drinker”.

**Downstream Tasks.** We apply the proposed model across a diverse array of classification tasks, including Random Forests (RF), Logistic Regression (LR), Naive Bayes (NB), K-Nearest Neighbor (KNN) and Multilayer Perceptrons (MLP).

We compare the performance outcomes in these tasks with and without our method.

**Baseline Models.** To demonstrate the effectiveness of our method, We compare our TFWT method with four established baseline techniques, where the Least Absolute Shrinkage and Selection Operator and TabTransformer are used for
Table 1: Overall performance on downstream tasks. The best results are highlighted in bold, and the runner-up results are highlighted in underline. (Higher values indicate better performance.)

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Model</th>
<th>RF</th>
<th>LR</th>
<th>NB</th>
<th>KNN</th>
<th>MLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc</td>
<td>Raw</td>
<td>AM</td>
<td>OS</td>
<td>MA</td>
<td>SD</td>
<td>AM</td>
</tr>
<tr>
<td></td>
<td>USP</td>
<td>0.60</td>
<td>0.62</td>
<td>0.70</td>
<td>0.72</td>
<td>0.60</td>
</tr>
<tr>
<td></td>
<td>TabT</td>
<td>0.62</td>
<td>0.64</td>
<td>0.70</td>
<td>0.72</td>
<td>0.60</td>
</tr>
<tr>
<td></td>
<td>TabTF</td>
<td>0.60</td>
<td>0.62</td>
<td>0.70</td>
<td>0.72</td>
<td>0.60</td>
</tr>
<tr>
<td>Prec</td>
<td>Raw</td>
<td>AM</td>
<td>OS</td>
<td>MA</td>
<td>SD</td>
<td>AM</td>
</tr>
<tr>
<td></td>
<td>USP</td>
<td>0.62</td>
<td>0.64</td>
<td>0.70</td>
<td>0.72</td>
<td>0.60</td>
</tr>
<tr>
<td></td>
<td>TabT</td>
<td>0.62</td>
<td>0.64</td>
<td>0.70</td>
<td>0.72</td>
<td>0.60</td>
</tr>
<tr>
<td></td>
<td>TabTF</td>
<td>0.60</td>
<td>0.62</td>
<td>0.70</td>
<td>0.72</td>
<td>0.60</td>
</tr>
<tr>
<td>Rec</td>
<td>Raw</td>
<td>AM</td>
<td>OS</td>
<td>MA</td>
<td>SD</td>
<td>AM</td>
</tr>
<tr>
<td></td>
<td>USP</td>
<td>0.62</td>
<td>0.64</td>
<td>0.70</td>
<td>0.72</td>
<td>0.60</td>
</tr>
<tr>
<td></td>
<td>TabT</td>
<td>0.62</td>
<td>0.64</td>
<td>0.70</td>
<td>0.72</td>
<td>0.60</td>
</tr>
<tr>
<td></td>
<td>TabTF</td>
<td>0.60</td>
<td>0.62</td>
<td>0.70</td>
<td>0.72</td>
<td>0.60</td>
</tr>
</tbody>
</table>

Table 2: Datasets Description.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Samples</th>
<th>Features</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM</td>
<td>1,500</td>
<td>10,000</td>
<td>2</td>
</tr>
<tr>
<td>OS</td>
<td>12,330</td>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>MA</td>
<td>19,020</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>SD</td>
<td>991,346</td>
<td>23</td>
<td>2</td>
</tr>
</tbody>
</table>
4.2 Experimental Results

Overall Performance. Table 1 illustrates that our TFWT method consistently surpasses baseline methods across a variety of metrics and datasets. For example, when focusing on MLP, TFWT attains an accuracy improvement ranging from 17% to 23% compared to raw downstream tasks, and from 14.6% to 39.8% improvement over the most competitive feature weight adjustment methods. Furthermore, when applying fine-tuning method, TFWT sees an additional accuracy increase from 19% to 27%, and a variance decline from 5% to 18%. Notably, our method also consistently outperforms the TabTransformer model, which also incorporates the Transformer for feature adjustment.

Enhancement over Raw Downstream Tasks. Our evaluation focuses on the improvement that TFWT method brings to various downstream tasks in terms of performance. To ensure a robust and reliable comparison, we execute each model configuration five times and calculate the average metrics. The comparative results, showcased in Figure 3 and Table 1, clearly demonstrate that TFWT consistently boosts performance across all four metrics in four datasets, particularly when applied to MLP. The significant improvement incorporated in the TFWT method enhances the performance of downstream tasks from multiple dimensions.

Superiority over Baseline Models. We examine the impacts of our TFWT approach and conduct comparative analyses against four baseline models across four performance metrics. Our primary metric of representation is overall accuracy, depicted in Figure 3. Our TFWT method consistently achieves the highest accuracy on all four datasets. Particularly noteworthy is the comparison with TabTransformer model. While TabTransformer also integrates a Transformer structure in the feature preprocess, TFWT demonstrates a marked superiority in accuracy and F1, showing in Figure 4,5.

5 Conclusion

In this study, we introduce TFWT, a weighting framework designed to automatically assign weights to features in tabular datasets to improve classification performance. Through this method, we utilize the attention mechanism of Transformers to capture dependencies between features to assign and adjust weights iteratively according to the feedback of downstream tasks. Moreover, we propose a fine-tuning strategy adopting reinforcement learning to refine the feature weights and to reduce information redundancy. Finally, we present extensive testing across various real-world datasets to validate the effectiveness of TFWT in a broad range of downstream tasks. The experimental results demonstrate that our method significantly outperforms the raw classifiers and baseline models.

6 Limitations

While TFWT shows impressive performance, there are limitations including high computational demands and limited scalability with very complex tasks, and there may be inherent limitations in tasks with unique requirements.
Acknowledgements

This work was supported by the Strategic Priority Research Program of the Chinese Academy of Sciences XDB38030300, the National Key R&D Program of China under Grant No. 2021YFE0108400, the Natural Science Foundation of China under Grant No. 61836013, and Informatization Plan of Chinese Academy of Sciences (CAS-WX2023ZX01-11).

References


[Luong et al., 2015] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to


