International Journal of Data Science and Analytics (2021) 12:1–14
https://doi.org/10.1007/s41060-021-00263-3

REGULAR PAPER

Enhancing personalized modeling via weighted and adversarial learning

Wei Du[1] · **Xintao Wu**[1]

Received: 17 November 2020 / Accepted: 6 May 2021 / Published online: 17 May 2021

Abstract
The data generation sources are increasing in the past few years, such as mobile devices, embedded sensors, various intelligent equipment and so forth. These increasing data sources push the deployment of deep learning models in a distributed manner. However, the traditional distributed deep learning is to build a global model over all collected data and may overlook specific components which are of vital importance to individual users. In this paper, we propose an adversarial learning framework that allows an individual user to build a personalized model. Our framework consists of two stages, including efficient similar data selection from other users and adversarial training. Instead of selecting similar data by computing hand-designed similarity metrics, we train an auto-encoder and a generative adversarial network (GAN) on individual user's data and use them to request similar data from other users. To further improve the personalized model performance, we develop two approaches that combine the requested data and user's own data to build the personalized model. The first approach is that we apply weighted learning to capture the different importance of the requested data. The second approach is that we apply adversarial training to minimize the distribution discrepancy between the requested data and user's own data. Experimental results demonstrate the effectiveness of the proposed framework.

Keywords deep learning · personalized model · weighted learning · adversarial learning

1 Introduction

The past few years have witnessed an increasing role that deep learning plays in various kinds of applications, such as image classification [1], text generation [2], recommendation systems [3] and other artificial intelligence (AI)-related tasks. More recently, data for training deep learning models are increasingly distributed among different users. A traditional approach for the deployment of deep learning model is to collect all these data into a central server and build a global model. An alternative way is to collaboratively learn a global model among distributed users via parameters exchange [4]. The key of previous works is to build a one-fit-all global model and use it to perform classification or prediction for all users. Although a global model captures generic information of training data from all users, it cannot fulfill the personalized demand for each individual user due to the diverse data distribution among different users.

It is necessary to construct a personalized model for each individual user to improve intelligent services. Although the global model learns generic information over all users, the specific features of each individual user's data, which are of vital importance to build the personalized data, are often overlooked by the global model. In many circumstances, building a personalized model can achieve better performance for specific users. For instance, personalized recommendation systems play important roles for many online services [5], such as production advertisement, sales promotion, and information recommendation. In addition, electronic health records [6] are used to predict disease progression and provide treatment plans. Since electronic health records are patient-specific, personalized models are more effective for individual users.

In fact, in many cases, an individual user can only collect a small amount of data, which is insufficient to build an accurate personalized model to achieve satisfactory performance. To improve the performance of personalized model,

✉ Xintao Wu
xintaowu@uark.edu

Wei Du
wd005@uark.edu

[1] Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, Arkansas 72701, USA

an individual user needs to request similar or complementary data from other users. Pioneer works of building personalized model have been done in medical field, e.g., Cheng et al. [6] develop a framework of collecting data from similar patients and then training a personalized model on the combination of these collected data and his own data.

However, there are two challenges in the pipeline of building personalized models. The first challenge is how to efficiently and effectively collect similar data from other users. It is obvious that randomly selecting data from other users may not improve the performance of personalized model as those collected data may be different from the user's own data. Previous research [6] applies similarity metrics to construct similarity index for data selection. However, it is burdensome to construct similarity index and the similarity metrics are often based on manually designed features. The computational cost is high when one compares paired data one by one due to the high operation complexity of $\mathcal{O}(n^2)$ especially when the number of data n is large. Moreover, it is often unclear to choose one appropriate similarity metric so most appropriate data can be collected. Another challenge is how to better train the personalized model with the requested data. Previous research simply combines the collected data with user's own data, which may not build the personalized model with high accuracy. This is because there may exist distribution discrepancy between the collected data and user's own data, especially when the user can only get a small amount of data from other users (e.g., due to privacy concerns or high data collection cost). Hence, it is imperative to develop personalized learning models that can handle the potential distribution discrepancy.

Motivated by the above two challenges, in this paper, we develop a learning framework that enables an individual user to effectively collect data and build a robust model on the combination of the collected data and his own data. For data collection, we propose an approach of using an auto-encoder and a generative adversarial network (GAN) [7]. The auto-encoder is used to obtain data representation that is further used to train the GAN. The trained encoder and the discriminator from GAN are sent to his neighbors. Each neighbor user uses the encoder to obtain the representation of his data and then uses the discriminator to calculate the probability score of his data. Data with high probability scores are combined with the user's original data to train the personalized model. The advantage of using GAN is that it can capture inherent properties of the underlying data without manually specifying features. With the requested data, we develop two approaches to improve the performance of the personalized model. The first approach is weighted learning by assigning a different weight to each record of the requested data. The data record with a high probability score computed by the discriminator is assigned with a high weight. The weighted learning is able to capture the importance of different data. The second approach is the adversarial learning that aims to minimize the distribution discrepancy between the requested data and user's own data. The core idea of the adversarial learning is to map both the requested data and user's own data into the same feature space where the distribution discrepancy is minimized. Our adversarial training is analogous to the discriminator of the GAN. The role of the discriminator is to predict whether the generated features are from user's own data or the requested data.

The main contributions of this paper are summarized as follows. First, we demonstrate that building a global model is not an optimal choice for personalized prediction. Second, we develop a strategy based on auto-encoder and GAN to effectively collect similar data from other users. Third, instead of directly using the requested data for training, we design two approaches, weighted learning and adversarial learning, to further improve the performance of the personalized model. Finally, we conduct extensive experiments and the results demonstrate the efficiency of the proposed framework.

2 Related work

Distributed deep learning: Distributed deep learning from multiple sources has been long investigated in the past few years. In the pioneer work [4], the dataset is distributed in different machines and a global model is learned by exchanging parameters between participating users. Following this work, later researchers focus on how to train distributed deep learning models more efficiently. For instance, Chilimbi et al. [8] propose an efficient and scalable system to allow the training of distributed deep learning. Wen et al. [9] develop a strategy to reduce communication cost in distributed deep learning to accelerate the training process. Moreover, how to improve the performance of distributed deep learning has been investigated extensively [10–12]. Besides distributed deep learning in the data center setting, federated learning [13] is proposed to push deep learning to mobile and edge devices, in which mobile devices only have limited data and users are willing to collaboratively learn a joint model. Recent works focus on achieving security guarantee [14] and solving data statistical challenges [15]. However, the focus of the existing works is to learn a global model from all training data.

Personalized deep learning and recommendation: Personalized model is popular in the areas of medicine and recommendation systems. In these areas, building a personalized model for each individual user is necessary since medical data and recommendation service are user-specific and a global model cannot capture personalized features. In personalized medical prediction area, similarity learning is fundamental for building a personalized patient model. In [16], Che et al. propose a dynamic temporal matching approach to find similar data for individual users and build a

personalized RNN model for each individual user to predict disease. The authors [17] develop a CNN-based similarity method for paired data comparison and perform personalized disease prediction. Other works using different metrics for similarity personalized learning [18–20] are also reported.

Personalized recommendation has also made rapid progress in many areas, such as e-commerce, advertising, audio and video recommendation [21]. Collaborative filtering, which is one of the most popular recommendation approaches, makes predictions about the interests of a user by collecting preferences from many similar users. Various metrics are used to measure the similarity between different users. For instance, Luo et al. [22] apply cosine similarity to build recommendation system for smart grid end users and Kouki et al. [23] use Pearson's correlation to compute data similarity. Recently, deep learning models are widely used to improve the performance of personalized recommendation. For example, Hu et al. [24] propose to diversify personalized recommendation results by leveraging user-session contexts and designing session-based neural networks to efficiently learn session profiles over large number of users and items. Yu et al. [25] design an attention-based LSTM framework to generate users' representations and adaptively to learn and predict according to the specific context.

Different from above works, we propose an efficient similarity approach based on auto-encoder and GAN and apply adversarial training to reduce potential distribution discrepancy. In fact, our work can be integrated into personalized recommendation systems to improve their performance. For example, our auto-encoder and GAN based similarity approach can be used to efficiently group different users in collaborative filtering recommendation. Moreover, the adversarial training can also be applied to learn and differentiate the representations of different users and then improve the personalized prediction in content-specific scenarios.

Representation learning: Representation learning has been a well-studied research area in the past few years [26], especially in computer vision, natural language processing and transfer learning. There are several works that apply representation learning to solve distribution discrepancy between different parties. For example, Tzeng et al. propose to learn domain invariant representations from the source domain and transfer to the target domain for prediction [27]. Liu et al. develop a framework that disentangles domain-invariant and domain-specific features in image translation and manipulation [28]. In [29], Gupta et al. extract invariant features between different agents in the reinforcement learning. Misra et al. propose a multi-task model to learn common representations between different tasks and use it to improve prediction performance [30]. Our work falls into the general area that exploits feature representation among different groups for performance improvement [31,32]. Different from previous works, our work applies adversarial training to reduce the distribution discrepancy and learns the prediction task simultaneously to improve its performance.

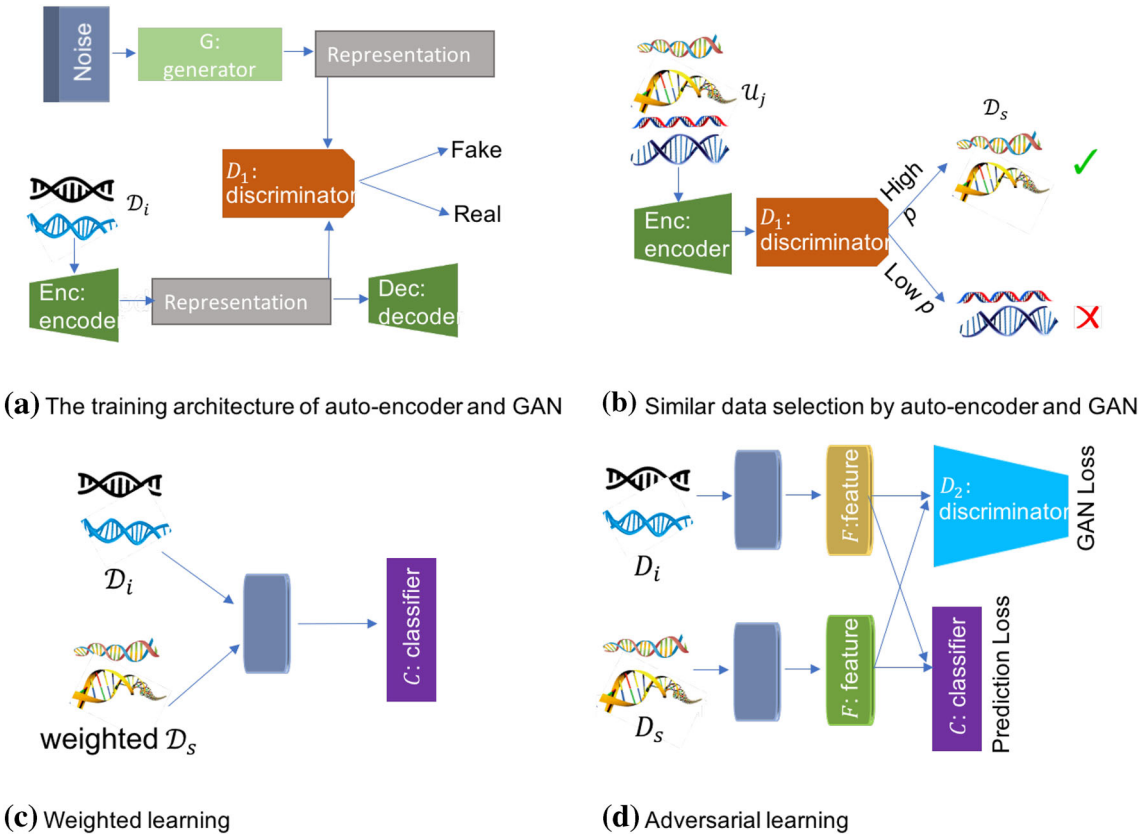
Robust learning: Robust learning is of great importance for many machine learning applications because we usually experience distribution shift from the training data to the testing data. Extensive research has been conducted to correct distribution shift in order to build a robust machine learning model. For example, Zadrozny et al. [33] investigate the sampling selection bias scenario between the training and the testing data and propose a reweighing method to correct the data bias. Wen et al. [34] formulate the robust learning as an adversary game and optimize model performance under the worst case distribution to improve the robustness of the model. Mehran et al. [35] design a robust learning method in the area of domain shift by training the model on noisy labeled data. In [36], Wang et al. propose a flexible and robust framework for transfer learning by transforming the training data distribution to the target data distribution. In [37] and [38], the authors use kernel mean matching method to deal with the covariate shift. Our work also falls into the area of robust learning and we apply the adversarial learning to reduce the distribution shift between user's own data and the collected data from neighbors.

3 Weighted learning and adversarial learning

3.1 Framework overview

Consider there exist N individual users in this distributed setting and each user has his own local dataset that contains features X and label Y. Let $\mathcal{U}$ and $\mathcal{D}$ represent the set of users and datasets, respectively, where each user $\mathcal{U}_i$ is associated with dataset $\mathcal{D}_i$. Suppose $\mathcal{D}_i$ contains M_i data samples, namely $(X_1, Y_1), (X_2, Y_2), \ldots, (X_{M_i}, Y_{M_i})$. The goal of each user $\mathcal{U}_i$ is to build a personalized classification model f_i which takes $\mathcal{D}_i$ as input and minimizes the prediction loss $\mathcal{L}_i = \sum_{m=1}^{M_i} l_i(f_i(X_m), Y_m)$, where l_i is the given loss function. To boost the performance of classifier f_i and reduce prediction loss $\mathcal{L}_i$, user $\mathcal{U}_i$ will request similar data from his neighbors. The neighbors can be defined from different ways. For example, in a sensor network, the neighbors can be sensors within a physical region. In our experiments, we treat the neighbors of $\mathcal{U}_i$ as all other users except $\mathcal{U}_i$.

For the personalized learning, the joint distribution of $\mathcal{U}_i$ could be different from that of its neighbor $\mathcal{U}_j$, i.e., $P(X_{\mathcal{U}_i}, Y_{\mathcal{U}_i}) \neq P(X_{\mathcal{U}_j}, Y_{\mathcal{U}_j})$. The most rigorous assumption is that $P(X_{\mathcal{U}_i}) \neq P(X_{\mathcal{U}_j})$ and $P(Y_{\mathcal{U}_i}|X_{\mathcal{U}_i}) \neq P(Y_{\mathcal{U}_j}|X_{\mathcal{U}_j})$. Although the overall distribution of $\mathcal{D}_i$ and $\mathcal{D}_j$ are not the same, it is possible that a subset of $\mathcal{D}_j$ at user $\mathcal{U}_j$ could be similar to $\mathcal{D}_i$ and hence in our framework $\mathcal{U}_i$ can request



(a) The training architecture of auto-encoder and GAN

(b) Similar data selection by auto-encoder and GAN

(c) Weighted learning

(d) Adversarial learning

Fig. 1 The framework of personalized learning where **a**, **b** and **c** is for weighted learning and **a**, **b** and **d** is for adversarial learning

a subset of similar data, $\mathcal{D}_s$, from its neighbors such that $P(X_{\mathcal{U}_i}) \approx P(X_{\mathcal{U}_s})$ and thus $P(Y_{\mathcal{U}_i}|X_{\mathcal{U}_i}) \approx P(Y_{\mathcal{U}_s}|X_{\mathcal{U}_s})$.

However, there could still exist possible distribution shift between $\mathcal{D}_i$ and $\mathcal{D}_s$, therefore, the second part of our framework is focused on how to better use the requested data to build a personalized model. We propose two approaches on how to combine the requested data $\mathcal{D}_s$ with $\mathcal{D}_i$ to build a personalized model for $\mathcal{U}_i$. The first approach is weighted learning shown as Fig. 1(a–c), while the second approach is adversarial learning shown as Fig. 1a, b and d. We show the workflow of our personalized learning framework in Algorithm 1. The workflow has three subsections.

Train auto-encoder and GAN: The first subsection (lines 2 - 4) is to train auto-encoder and GAN for $\mathcal{U}_i$ based on his own dataset $\mathcal{D}_i$. Line 3 trains a normal auto-encoder for $\mathcal{U}_i$. With the trained auto-encoder, we apply its Enc to compute the representations of $\mathcal{D}_i$ and use the learned representations as the input to train the GAN. For clear illustration, we show the training process in Fig. 1a.

Request $\mathcal{D}_s$ for $\mathcal{U}_i$: The second subsection (lines 6–8) is to request similar data for $\mathcal{U}_i$ from his neighbors. This subsection is also illustrated in Fig. 1b. $\mathcal{U}_i$ first sends his Enc and D_1 to his neighbors. Each neighbor $\mathcal{U}_j$ uses Enc to compute the representation of each sample and then uses discriminator D_1 to obtain a probability score p for each sample. $\mathcal{U}_j$ collects all data samples with score $p > \tau$, puts into $\mathcal{D}_s$ and sends $\mathcal{D}_s$ back to $\mathcal{U}_i$.

Personalized learning: The final subsection is to build a personalized model for $\mathcal{U}_i$ based on $\mathcal{D}_i$ and $\mathcal{D}_s$. We propose two approaches to train a personalized model. The first approach is the weighted learning (Fig. 1c) and lines 10-14 show its training process. $\mathcal{U}_i$ first uses the discriminator D_1 to compute the probability score p of each data sample from $\mathcal{D}_s$ and uses p as the weight for each data from Equation 7. With the probability score, the personalized model can better capture the importance of each requested data sample. The second approach is the adversarial training (Fig. 1d) and lines 15–22 show its training process. The adversarial training can reduce the distribution discrepancy between $\mathcal{D}_i$ and $\mathcal{D}_s$, which can further improve the performance of f_i compared to training f_i directly over $\mathcal{D}_i$ and $\mathcal{D}_s$.

In fact, the proposed personalized learning is a general framework. It can apply to different machine learning models, such as logistic regression, convolutional neural networks and long short-term memory neural networks. In our framework, the goal of the auto-encoder and GAN is used to select similar data from other users. In fact, we do not need to train a perfect GAN using lots of data as previous research on

generating fake images/sentences to fool human. When the user has limited data, the auto-encoder and GAN trained with insufficient data may select data samples that are not similar to the individual's own data. This is the reason why we propose AdvPL to further reduce the distribution discrepancy between the individual user's data and requested data.

3.2 Train auto-encoder and GAN for $\mathcal{U}_i$

The first step of AdvPL is to train the auto-encoder and GAN for $\mathcal{U}_i$. Typically both Enc and Dec are deep neural networks. The Enc takes the data in $\mathcal{D}_i$ and obtains its hidden representation. The hidden representation is then fed into the Dec to output a reconstructed input. The advantage of using representations by the auto-encoder is to make the training of the GAN easier, especially for sequential data. The Enc obtains hidden representation of data in $\mathcal{D}_i$ and GAN takes it as input.

GAN has a very appealing property: its discriminator can implicitly learn hidden similarity metric and use it to discriminate real data and fake data. As a result, the discriminator is able to capture data distribution pattern of $\mathcal{U}_i$.

Encoder: The encoder is a neural network which takes mth data X_m and outputs a hidden representation as:

$$\mathbf{h}_m^{Enc} = Enc(X_m), \tag{1}$$

where $\mathbf{h}_m^{Enc}$ is the hidden representation of X_m and $Enc(\cdot)$ denotes the computation of hidden representation by the encoder neural network. The hidden representation is deemed to capture the information of the input data, which will be the input of the decoder.

Decoder: The decoder is used to reconstruct original input based on $\mathbf{h}_m^{Enc}$ and the reconstruction of X_m is expressed as:

$$X'_m = Dec(\mathbf{h}_m^{Enc}), \tag{2}$$

where X'_m is the reconstructed data of X_m and $Dec(\cdot)$ denotes the computation of reconstructed input by the decoder neural network.

The performance of the auto-encoder is evaluated by the distance between X_m and X'_m, and the loss over $\mathcal{D}_i$ is expressed as:

$$\mathcal{L}_{AE} = \frac{1}{M_i}\sum_{m=1}^{M_i}(X_m - X'_m)^2 \tag{3}$$

After the training process of auto-encoder, $\mathcal{U}_i$ can use the Enc to transform input data into hidden representation and train the GAN based on the representation data. The GAN consists of a generator G and a discriminator D_1. The goal of G is to generate fake representation $\mathbf{h}_{fake}^{Enc}$ and D_1 is to distinguish real representation $\mathbf{h}_{real}^{Enc}$ and fake representation $\mathbf{h}_{fake}^{Enc}$. The objective function of the GAN is the same as the traditional GAN in previous work [7]:

$$\begin{aligned}\min_G \max_{D_1} \quad & \mathbb{E}_{\mathbf{h}_{real}^{Enc}\sim P_{data}} \log D_1(\mathbf{h}_{real}^{Enc}) \\ & + \mathbb{E}_{\mathbf{h}_{fake}^{Enc}\sim P(\mathbf{G})} \log(1 - D_1(G(\mathbf{h}_{fake}^{Enc}))),\end{aligned} \tag{4}$$

where P_{data} ($P_{\mathbf{G}}$) denotes the probability distribution of $\mathcal{D}_i$ (noise).

The data distribution of $\mathcal{U}_i$ is captured by his auto-encoder and GAN. To request similar data, $\mathcal{U}_i$ only needs to send Enc and D_1 to his neighbor users $\mathcal{U}_j$. If a data point of $\mathcal{U}_j$ passes the D_1 with high probability score, then these data are more likely to be sampled from the same distribution of $\mathcal{U}_i$ and can be put into $\mathcal{D}_s$.

Algorithm 1 The framework of personalized learning

Require:
$\mathcal{U}_i$ with dataset $\mathcal{D}_i$ and threshold τ;
Option: [Weighted learning, Adversarial learning]
Ensure:
Well trained personalized model f_i;
/* **Train auto-encoder and GAN** */:
Initialize parameters in auto-encoder (Enc, Dec) and GAN(G, D_1) for $\mathcal{U}_i$;
$\mathcal{U}_i$ trains the auto-encoder using $\mathcal{D}_i$ and updates parameters for Enc, Dec using Eq. 1, 2 and Eq. 3;
$\mathcal{U}_i$ trains the GAN using representations by Enc and updates parameters for G, D_1 using Eq. 4;
/* **Request** $\mathcal{D}_s$ **for** $\mathcal{U}_i$ */:
$\mathcal{U}_i$ sends Enc and D_1 to his neighbor users $\mathcal{U}_j$;
Requested dataset $\mathcal{D}_s = \emptyset$;
For each neighbor user $\mathcal{U}_j$: encode $\mathcal{D}_j$ by Enc, compute probability score p using D_1, and put data in $\mathcal{D}_s$ with $p > \tau$;
If Option = Weighted learning:
    /* **Weighted training** */:
    while not converged:
        Compute probability score for each data from $\mathcal{D}_s$ from Eq. 6;
        Compute weighted loss based on $\mathcal{D}_i$ and $\mathcal{D}_s$ from Eq. 7 and update model parameters;
    end while
Elif Option = Adversarial learning:
    /* **Adversarial training** */:
    Initialize parameters in feature extractor (F), model classifier (C), and discriminator (D_2) for adversarial training;
    $\mathcal{U}_i$ trains C and F to optimal performance using $\mathcal{D}_i$;
    while not converged:
        Fix D_2 and F, and update C with loss Eq. 10 over $\mathcal{D}_i$ and $\mathcal{D}_s$;
        Fix C, update D_2 and F with loss Eq. 9 and 8 over $\mathcal{D}_i$ and $\mathcal{D}_s$;
    end while
return personalized model f_i

3.3 WL: weighted learning for personalized model

A straightforward approach is to directly incorporate $\mathcal{D}_s$ to his personalized model so that we have the combined $\mathcal{D}_s$ and $\mathcal{D}_i$ as the training data. The total loss based on the combination of $\mathcal{D}_s$ and $\mathcal{D}_i$ is expressed as:

$$\mathcal{L}_{unweighted} = \sum_{m=1}^{|\mathcal{D}_i|} l_i(f_i(X_m), Y_m) + \sum_{n=1}^{|\mathcal{D}_s|} l_i(f_i(X_n), Y_n), \quad (5)$$

where l_i is the loss function and f_i is the classifier for $\mathcal{U}_i$. In Equation 5, each data sample in $\mathcal{D}_i$ and $\mathcal{D}_s$ is assigned with the same weight.

However, the importance of data samples in $\mathcal{D}_s$ should not be the same and generally should be less than data samples in $\mathcal{D}_s$ due to the distribution discrepancy. As we discussed, the neighbor user $\mathcal{U}_j$ uses Enc and D_1 to compute the probability score p for each data sample as the following:

$$p_n = D_1(Enc(X_n)), \quad (6)$$

where X_n is the nth data from $\mathcal{D}_s$. As aforementioned, the discriminator D_1 captures the distribution of $\mathcal{D}_i$. Therefore, p_n can be seen as a metric to measure the similarity between X_n and $\mathcal{D}_i$. Higher p_n indicates that X_n is more likely to be sampled from the distribution of $\mathcal{D}_i$. Hence, we take p_n into consideration and assign p_n as the weight of X_n to construct a weighted loss:

$$\mathcal{L}_{weighted} = \sum_{m=1}^{|\mathcal{D}_i|} l_i(f_i(X_m), Y_m) + \sum_{n=1}^{|\mathcal{D}_s|} p_n l_i(f_i(X_n), Y_n), \quad (7)$$

The intuition of Equation 7 is that it assigns a higher (lower) weight for the data sample which is more (less) similar to the distribution of $\mathcal{D}_i$. Consider the following two cases. First, if the probability scores of all requested data from $\mathcal{D}_s$ are approaching to 1, then the weighted loss Equation 7 is reduced to the unweighted loss Equation 5, indicating that the unweighted loss is a special case of the weighted loss. Second, if the probability scores of the requested data are mixed, then the weighted loss can capture the importance difference of different data samples so that it can better improve the performance of the personalized model.

3.4 AdvPL: adversarial learning for personalized model

We explain the motivation and necessity of the adversarial training. Our aim is to reduce the distribution discrepancy between $\mathcal{D}_i$ and $\mathcal{D}_s$. As shown in Fig. 2a, the data in $\mathcal{D}_s$ ($\mathcal{D}_i$) are represented by circular (triangular) dots. In raw data space, $\mathcal{D}_s$ and $\mathcal{D}_i$ are partially overlapped and there exists a distribution discrepancy between them. Consequently, the model performance may not be optimal if $\mathcal{U}_i$ directly uses them to train the personalized model. However, if we can transform the raw data into another space where $\mathcal{D}_i$ and $\mathcal{D}_s$ can be well overlapped as shown in Fig. 2b, then the distribution discrepancy in this new space will be reduced and these new transformed data can be used to build a more accurate model. To minimize the distribution discrepancy between $\mathcal{D}_s$ and $\mathcal{D}_i$, we present an adversarial learning framework as shown in Fig. 1d. The adversarial learning simultaneously reduces the distribution discrepancy between $\mathcal{D}_s$ and $\mathcal{D}_i$ in the feature space and trains the model with these feature data simultaneously.

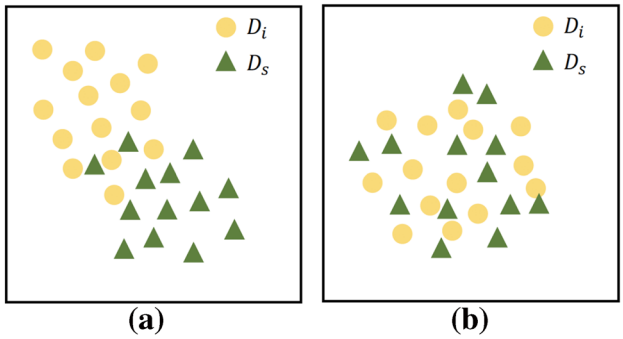


Fig. 2 **a** It depicts raw data distribution using dots and triangles between $\mathcal{D}_i$ and $\mathcal{D}_s$. In raw data space, the distribution between $\mathcal{D}_i$ and $\mathcal{D}_s$ is partly overlapped. **b** By optimizing a loss function that simultaneously maximizes overlap in the feature representation space and improves the model performance, we can reduce the distribution discrepancy between $\mathcal{D}_s$ and $\mathcal{D}_i$

The adversarial training for personalized model is composed of three parts including feature representation extractor (F), discriminator D_2 and model classifier C. It should be mentioned that F and C are two parts of a complete neural network model, so the extracted representations by F can be directly used as intermediate input for C. F is used to extract representations of data in $\mathcal{D}_i$ and $\mathcal{D}_s$. The adversarial training process is analogous to the traditional GAN. The role of F is to mimic the function of the generator in GAN, while the role of the discriminator D_2 is to distinguish the feature representations between $\mathcal{D}_i$ and $\mathcal{D}_s$ created by F. In our proposed AdvPL, F is trained in a manner that maps the data in $\mathcal{D}_i$ and $\mathcal{D}_s$ to feature representations with binary labels, where the label is 1 if feature representation belongs to $\mathcal{D}_i$ and is 0, otherwise. The key of the AdvPL is that F and D_2 are trained together through the adversarial learning process. More specifically, the goal of F is to generate the feature representations of data in $\mathcal{D}_i$ and $\mathcal{D}_s$ into the same space. F is to fool D_2 and makes D_2 unable to distinguish the representations between $\mathcal{D}_i$ and $\mathcal{D}_s$. On the contrary, D_2 is trained to predict whether the feature representations are from $\mathcal{D}_i$ or $\mathcal{D}_s$.

The loss function of F is similar to the generator in traditional GAN and has the following expression:

$$\mathcal{L}_F = -\mathbb{E}_{Z\sim\mathcal{D}_s} \log D_2(F(Z)) \tag{8}$$

where $F(Z)$ is the representation of data in $\mathcal{D}_s$.

The discriminator D_2 is trained to identify whether a data sample is from $\mathcal{D}_i$ or $\mathcal{D}_s$ given the feature representations. It is obvious that if the transformed representation suffers from huge distinction between $\mathcal{D}_i$ and $\mathcal{D}_s$, then D_2 is able to separate them easily. The adversarial loss of D_2 is:

$$\mathcal{L}_{D_2} = -\mathbb{E}_{X\sim\mathcal{D}_i} \log D_2(F(X)) - \mathbb{E}_{Z\sim\mathcal{D}_s} \log(1 - D_2(F(Z))) \tag{9}$$

where $F(X)$ denotes the representations of his own data in $\mathcal{D}_i$. It can be seen that D_2 plays the same role as the traditional discriminator in traditional GAN.

The combination of $\mathcal{L}_F$ and $\mathcal{L}_{D_2}$ can mimic the adversarial training process of traditional GAN. Through the adversarial learning process, the data in $\mathcal{D}_i$ and $\mathcal{D}_s$ are transformed into the hidden representation space. In this space, D_2 cannot tell the difference between them so that the distribution discrepancy between $\mathcal{D}_i$ and $\mathcal{D}_s$ can be minimized.

Different from the traditional GAN which only generates realistic examples, our ultimate goal is to train a more accurate model for individual user. As shown in Fig. 1d, the extracted representations from $\mathcal{D}_i$ and $\mathcal{D}_s$ will be fed into C and the loss function of C with K classes is:

$$\begin{aligned}\mathcal{L}_C = &-\sum_{m=1}^{M_i}\sum_{k=1}^{K} \mathbb{1}_{k=Y_m} \log C(F(X_m)) \\ &-\sum_{m=1}^{|\mathcal{D}_s|}\sum_{k=1}^{K} \mathbb{1}_{k=Y_m} \log C(F(Z_m)),\end{aligned} \tag{10}$$

where C is the final layer of model classifier and Y_m are the corresponding labels. M_i is the data number of $\mathcal{D}_i$ and $|\mathcal{D}_s|$ is the size of $\mathcal{D}_s$. Therefore, the full framework is to minimize the joint loss function $\mathcal{L}$:

$$\mathcal{L} = \mathcal{L}_C + \mathcal{L}_F + \mathcal{L}_{D_2}, \tag{11}$$

where $\mathcal{L}$ denotes the sum loss of the adversarial learning framework. The joint training process of the adversarial learning framework is summarized in Lines 17 - 21 in Algorithm 1. It mainly includes two parts. The first part is that $\mathcal{U}_i$ trains C and F using $\mathcal{D}_i$. Our goal is to build a personalized model and use similar data $\mathcal{D}_s$ to improve model performance, so we first achieve optimal model performance based on $\mathcal{D}_i$. The second step is to alternatively train C, F and D_2. To train C, we fix F and D_2. To train F and D_2, we fix C. In this way, the balance of the adversarial training will be under better control. To be noted here, we first train the personalized model to optimal performance before starting the adversarial training. Compared to training D_2 at the beginning, it is easier to improve the performance of the adversarial training.

4 Experimental results

In this section, we evaluate the performance of WL and AdvPL using three real-world datasets. In Sect. 4.1, we present experimental setup details including dataset descriptions, hyperparameters and baselines. In Sect. 4.2, we present our main results of the accuracy comparison of our proposed framework and other baseline models and the corresponding training efficiency on three datasets. In Sect. 4.3, we present detailed analysis on the performance of the proposed framework using the first two datasets. First, we compare the performance of the personalized model (without requested similar data) and global model to demonstrate the benefits of the personalized model. Second, we measure the performance improvement of the personalized model, WL and AdvPL, with requested similar data. Third, we compare our algorithms with other similarity metrics and demonstrate the advantages of our proposed framework. Fourth, we conduct sensitivity analysis and investigate the effects of the probability distribution of the requested data and the budget size on the performance of WL and AdvPL.

4.1 Experimental setup

Datasets: To evaluate the performance of our algorithm, we conduct our experiments on three real-world datasets, including UNIX Command Sequence, Shakespeare Text and YesiWell health data. The UNIX Command Sequence dataset [39] is composed of 50 files, where each file corresponds to one user's command sequence collected by the UNIX acct auditing mechanism. Each user is recorded with a long sequence consisting of the UNIX command in a period of time, such as *troff, dpost, eqn, sed, cat, ls, gs* and so forth. The sequence length of each user is 15000 and the average number of command types for these 50 users is over 100. We split the long command sequence of each user evenly into 500 sequences. For each sequence, we aim to build a classifier that predicts the final command based on the previous commands in this sequence. More specifically, the input data are a sequence consisting of UNIX commands with length T and the task is to predict the next command at the $T + 1$ step. It is natural that different users have their own typing styles. For example, technical users and non-technical users often have different command sequences. In our experiment, we build one personalized model for each user. The

Shakespeare Text dataset is constructed from The Complete Works of William Shakespeare [13]. This dataset is written in the form of plays and each speaking role in the plays is treated as an individual user. We subsample 40 speaking roles and build one personalized model for each speaking role. For each speaking role, we process its input text data to a sequence list where each sequence has a fixed length 100. The numbers of sequences of each speaking role are within the range between 5000 and 10000. The task is to predict the next character after reading previous characters in each sequence. YesiWell health dataset [40] was collected between 2010 and 2011 as a collaboration study by different institutes, including PeaceHealth Laboratories, SK Telecom Americas, and the University of Oregon, to help people maintain active lifestyles and lose weight. The dataset includes a group of 254 overweight and obese individuals and records the information of various aspects such as physical activities, social activities, biomarkers, and biometric measures. The total distance of walking and running is included in physical activities and measured via a mobile device carried by each user. The distance of each user is reported daily and forms a long sequence. Our task is to build a classifier for each user to predict the daily distance based on the distance sequence of previous days. For this classification task, we divide the distance into 30 ranges and assign a label to each distance range. After preprocessing, we select 69 users by removing sequences with missing values and abnormal patterns due to data reporting error. The sequence length is set as 10 and the data size of each user is between 200 and 500. We summarize the characteristics of UNIX Command Sequence, Shakespeare Text and YesiWell data in Table 1.

Hyperparameters: In our experiment, we use two-layer long short-term memory (LSTM) neural networks as the classification model for both two tasks. Each layer of the LSTM classification model has the dimension 256 and the output of the second LSTM layer is sent to a softmax output layer for prediction. For each user, we select 80% of the sequences as training data and the rest as testing data. We also choose LSTM to build the auto-encoder which is composed of an encoder and a decoder. Both the encoder and a decoder consist of two-layer LSTMs. The dimensions of the hidden layer in the encoder and decoder are both set as 256. Each subsequence is embedded with 512 dimensions before sending to the encoder as input. The last hidden layer of the encoder is taken out to be the input of the decoder. For the GAN model, both the discriminator and generator are feedforward neural networks. More specifically, the generator has two hidden layers with dimensions 50 and 100, respectively. The discriminator also has two hidden layers with dimensions 100 and 50, respectively. The dimension of the Gaussian noise is the same as the size of the hidden representations by the encoder.

Baselines: In our proposed WL (AdvPL), each user collects $\mathcal{D}_s$ from other users and trains his personalized model using $\mathcal{D}_i$ and $\mathcal{D}_s$ in a weighted training (adversarial training) manner. We compare our WL and AdvPL with the following baselines:

- Global: the global model is a one-fit-all model for all of the users, which is built on the collection of all users' training data.
- PL: each user trains his own personalized model (the same structure as the global model) only based on his own training data.
- PL_Rand: each user randomly requests $\mathcal{D}_s$ and trains the model simply with the combination of $\mathcal{D}_i$ and $\mathcal{D}_s$.
- PL_Euc: each user requests $\mathcal{D}_s$ from other users based on the Euclidean similarity metric and trains his model as PL_Rand. The Euclidean similarity metric is computed as follows: for each user, we compute the one-hot-vector of each command and obtain a vector v_i by averaging the one-hot-vector of all commands. Similarly, for each subsequence of other users, we compute its vector v_s. Then, we can compute the Euclidean similarity metric between v_i and v_s.
- PL_Cos: each user requests $\mathcal{D}_s$ from other users based on the cosine similarity metric and trains his model as PL_Rand. The computation of the cosine similarity metric is the same as PL_Euc.
- PL_Multi: each user requests $\mathcal{D}_s$ using our proposed auto-encoder and GAN. We apply the multi-task framework [41] and treat $\mathcal{D}_i$ and $\mathcal{D}_s$ as two different tasks. In this implementation, only the final classification layer for $\mathcal{D}_i$ and that for $\mathcal{D}_s$ are different.

We run our methods and all baselines for five times and report the mean and standard deviation of accuracy in our evaluation.

4.2 Main results

4.2.1 Accuracy comparison

The accuracy comparison of our proposed framework and other baseline models on three datasets is shown in Table 2. The size of requested data $\mathcal{D}_s$ is 1000 for UNIX Command Sequence, 8000 for Shakespeare Text and 300 for YesiWell. From the experimental results, we summarize key observations and leave the detailed comparison in the following sections. First, the average accuracy of PL is higher than that of Global, indicating that one-fit-all model may miss the specific features of individual's data, especially in the scenarios where the distribution of individual's data is rather diverse. Second, our proposed framework outperforms other simi-

Table 1 The characteristics of UNIX Command Sequence, Shakespeare Text and YesiWell data

Dataset	Users	Sequence length	Data size
UNIX command sequence	50	50	500
Shakespeare text	40	100	5000–10000
YesiWell data	69	10	200 - 500

Table 2 Accuracy comparison (mean ± std) of the proposed framework and baselines based on five runs on UNIX Command Sequence, Shakespeare and YesiWell. The scale of the numbers is %

Dataset	PL	Global	PL_Rand	PL_Euc	PL_Cos	PL_Multi	WL	AdvPL
UNIX	54.86 ± 3.42	51.98 ± 2.89	54.46 ± 3.16	55.88 ± 2.77	56.10 ± 3.05	59.46 ± 3.17	59.34 ± 2.96	**61.42 ± 3.26**
Shakespeare	53.67 ± 2.25	51.87 ± 2.37	52.60 ± 2.07	53.91 ± 2.12	53.88 ± 2.27	55.49 ± 1.96	56.19 ± 2.21	**57.45 ± 1.97**
YesiWell	32.79 ± 4.19	30.48 ± 4.37	32.19 ± 3.67	34.21 ± 3.94	34.62 ± 3.77	35.46 ± 4.01	35.82 ± 4.23	**37.84 ± 3.75**

Table 3 Training time (second) of the proposed framework and other baselines

Dataset	PL	Global	PL_Rand	PL_Euc	PL_Cos	PL_Multi	WL	AdvPL
UNIX	118.2	1053.7	211.5	598.2	701.6	543.2	533.5	581.2
Shakespeare	677.7	4762.5	1213.2	4078.2	4983.4	2785.3	2752.4	3042.5
YesiWell	65.3	584.7	118.9	313.5	372.9	252.7	259.3	273.2

larity metrics in selecting similar data. As we explained, the auto-encoder and GAN can capture implicit complex features from the data, whereas other similarity metrics only compute simple statistical information. Third, the proposed AdvPL performs the best among all approaches. It demonstrates that AdvPL can reduce the distribution discrepancy between individual's data and requested data and thus improve the overall prediction accuracy.

We further test the statistical significance of the improvements between our proposed methods and baseline models. We run our methods and all baseline models for five times, use the independent two-sample t test and then calculate the p-value. For UNIX dataset, the p-values of testing AdvPL against Global, PL, PL_Rand, PL_Euc, PL_Cos, PL_Multi, and WL are 0.0006, 0.0073, 0.0045, 0.0102, 0.0143, 0.1819, and 0.1619, respectively. Using the threshold of 0.05, AdvPL has statistically significant improvement over Global, PL, PL_Rand, PL_Euc and PL_Cos. AdvPL can still achieve decent p-values (less than 0.2) when comparing with WL and PL_Multi (using GAN based similarity metric). For Shakespeare and YesiWell datasets, we have similar observations.

4.2.2 Training efficiency

The training efficiency is an important metric to evaluate the performance of the proposed framework. In this section, we compare the training time of our proposed framework and baselines. We report the results in Table 3 and summarize the key findings as the following. First, we can see that the training of auto-encoder and GAN can increase the total completion time from the comparison between PL_Rand and WL. However, this extra training time is worth as we demonstrate that the proposed framework can improve the overall performance significantly. Second, other similarity metrics, PL_Euc and PL_Cos, need to compare paired data one by one, which incurs high computational cost. Although auto-encoder and GAN will increase the total training time in our framework, each data sample only needs one single comparison when using the discriminator of the trained GAN for similar data selection. Third, adversarial learning takes more time to finish the training than WL, but the burden increased by the adversarial learning is not significant.

4.3 Detailed performance analysis

In this section, we conduct detailed evaluations and compare the performance of WL, AdvPL and other baseline models. We study the effects of probability distribution and requested dataset size on the performance of the proposed framework. For demonstration purpose and space limit, we only show the performance analysis on UNIX Command Sequence and Shakespeare Text.

4.3.1 Accuracy comparison of global and PL

In this experiment, we compare the performance of the Global and PL for each user to show the necessity of building the personalized model. The experimental result for UNIX Command Sequence (Shakespeare Text) is shown as Fig. 3a (Fig. 3**c**). To illustrate the result more clearly, we sort the users according to the prediction accuracy of the personalized model. The accuracy of each user using the Global (PL)

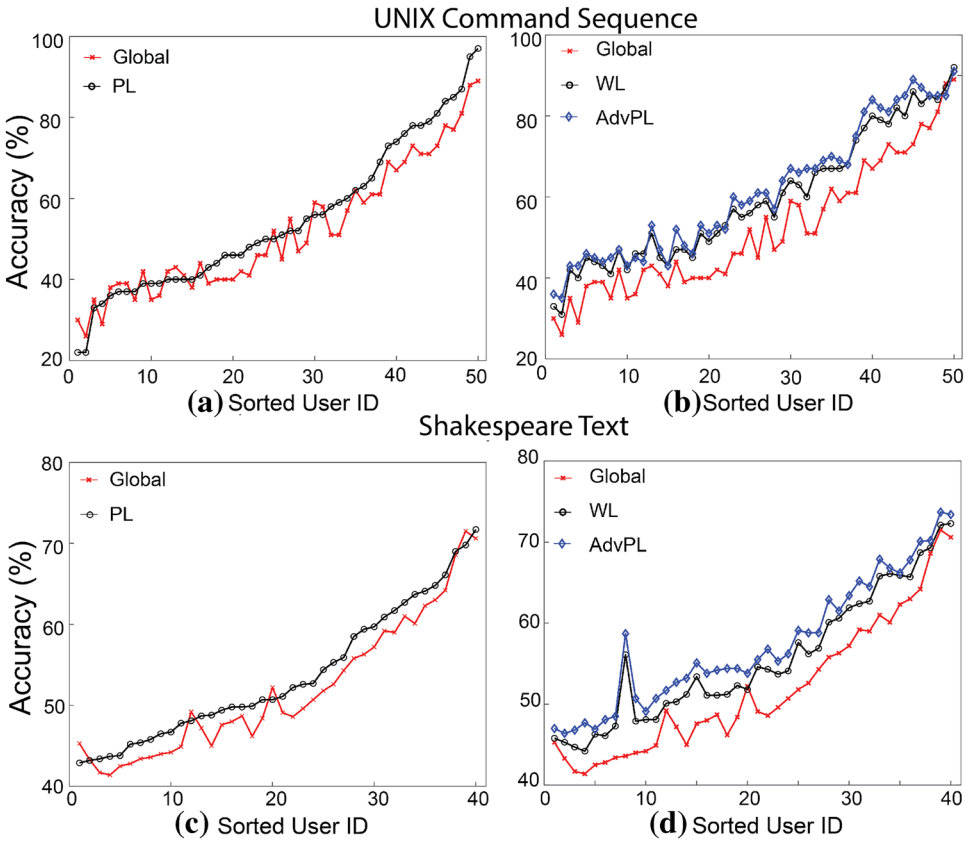


Fig. 3 UNIX Command Sequence **a**: The accuracy of the Global and PL for each user. **b**: The accuracy of the Global, WL and PL_Adv for each user. Shakespeare text **c**: The accuracy of the Global and PL for each user. **d**: The accuracy of the Global, WL and PL_Adv for each user

is shown as the red star line (black dot line). It can be seen that the trend of the PL is above the Global for most users. More specifically, the average accuracy of the Global and PL for UNIX Command Sequence (Shakespeare Text) is 51.98% (51.87%) and 54.86% (53.67%), respectively. The maximum accuracy increment of the PL over the Global for UNIX Command Sequence (Shakespeare Text) is 8.0% (3.7%). As aforementioned, Global captures the overall information of all training samples and overlooks the user-specific information. However, the user-specific information is the key to improve the performance of the PL. Hence the PL can better learn the pattern of each user.

4.3.2 Accuracy comparison of WL and AdvPL

In this experiment, we test the effectiveness of the proposed algorithms and show the accuracy of the WL and AdvPL for UNIX Command Sequence (Shakespeare Text) as Fig. 3b (Fig. 3d). The size of requested $\mathcal{D}_s$ for UNIX Command Sequence (Shakespeare Text) is set as 1000 (8000). The accuracy of the WL (AdvPL) is shown as the black dot line (blue triangular line). For comparison, we also plot the accuracy of the Global as the red star line.

We have the following observations. First, the accuracy of the AdvPL is higher than that of the WL. It demonstrates that adversarial learning can minimize the distribution discrepancy between $\mathcal{D}_i$ and $\mathcal{D}_s$. The reduced distribution discrepancy can help the AdvPL achieve higher accuracy than the WL. Second, both WL and AdvPL have a great advantage over the Global. It shows that our proposed algorithms are effective to select similar data for each user and help improve the performance of the personalized model. More specifically, the average accuracy of the WL and AdvPL for UNIX Command Sequence (Shakespeare Text) is 7.36% (4.32%) and 9.44% (5.58%) higher than that of the Global, respectively. In contrast, the average accuracy of the PL for UNIX Command Sequence (Shakespeare Text) is only 2.88% (1.80%) higher than that of the Global.

We further investigate the effects of the assigned weights in WL and compare its performance with PL_Sim (unweighted learning as shown in Equation 5). From our experimental results, the average performance of WL and PL_Sim over all users are at the same level. This is because for most of the users in our experiments, the probability scores of their requested data $\mathcal{D}_s$ are high, i.e., approaching to 1, and the loss function (Equation 7) of WL is reduced to the loss function (Equation 5) of PL_Sim. However, for a few other users, the probability scores of the requested data $\mathcal{D}_s$ are mixed, i.e., some data samples have lower probability scores while other data samples have higher probability scores. In this case, the performance of WL is better than Pl_Sim. Taking one user (ID = 3) from UNIX Command Sequence as an example, the

Table 4 Accuracy comparison (mean ± std) based on five runs for UNIX Command Sequence using different similarity metrics and $\mathcal{D}_s$ sizes. The average accuracy of the Global and PL is: 51.98% and 54.86%, respectively. The scale of the numbers is %

$\lvert\mathcal{D}_s\rvert$	PL_Rand	PL_Euc	PL_Cos	PL_Multi	WL	AdvPL
400	54.48 ± 3.23	55.12 ± 3.04	55.34 ± 2.91	56.84 ± 3.27	56.82 ± 3.15	**60.14 ± 3.77**
600	54.24 ± 3.19	55.28 ± 3.55	55.62 ± 2.89	57.96 ± 3.02	57.72 ± 3.37	**60.68 ± 2.75**
800	54.62 ± 3.43	55.56 ± 3.24	55.78 ± 2.10	58.62 ± 3.22	58.48 ± 2.88	**61.10 ± 3.73**
1000	54.46 ± 3.16	55.88 ± 2.77	56.10 ± 3.05	59.46 ± 3.17	59.34 ± 2.96	**61.42 ± 3.26**

Table 5 Accuracy comparison (mean ± std) based on five runs for Shakespeare Text using different similarity metrics and $\mathcal{D}_s$ sizes. The average accuracy of the Global and PL is: 51.87% and 53.67%, respectively. The scale of the numbers is %

$\lvert\mathcal{D}_s\rvert$	PL_Rand	PL_Euc	PL_Cos	PL_Multi	WL	AdvPL
2000	52.36 ± 2.04	53.63 ± 2.23	53.45 ± 1.83	54.21 ± 2.19	54.32 ± 2.57	**56.62 ± 2.79**
4000	52.49 ± 2.57	53.65 ± 1.79	53.61 ± 2.18	54.70 ± 2.28	54.95 ± 2.62	**56.91 ± 2.42**
6000	52.35 ± 1.95	53.79 ± 2.17	53.80 ± 2.33	55.24 ± 2.29	55.64 ± 1.93	**57.31 ± 2.14**
8000	52.60 ± 2.07	53.91 ± 2.12	53.88 ± 2.27	55.49 ± 1.96	56.19 ± 2.21	**57.45 ± 1.97**

accuracy of WL is 3% higher than that of PL_Sim. We provide more detailed analysis in 4.3.4 and discuss under what scenarios WL can outperform PL_Sim.

4.3.3 Accuracy comparison using different similarity metrics

Previous section shows the performance between the WL and AdvPL. In this section, we compare with the personalized models using other similarity metrics. For better comparison, we show the average accuracy rather than plotting the accuracy of all users. The comparison result for UNIX Command Sequence (Shakespeare Text) is shown as the 5th row in Table 4 (Table 5) with $|\mathcal{D}_s| = 1000$ ($|\mathcal{D}_s| = 8000$). We have the following interesting observations. First, if the user randomly requests $\mathcal{D}_s$, the average performance of the PL_Rand is slightly decreased compared to the PL. Second, the performance of the PL_Euc and PL_Cos helps improve the performance over the PL, however, the performance increment of our proposed WL is the best among these three similarity metrics. Third, the AdvPL achieves the best performance among all strategies. PL_Multi adopts the general multi-task learning framework [41] to learn common features between $\mathcal{D}_i$ and $\mathcal{D}_s$, however, it only achieves similar performance as WL, which demonstrates that our proposed AdvPL is more effective than the traditional multi-task learning method.

Moreover, we also study the effect of requested dataset $\mathcal{D}_s$ size on the performance of the WL and AdvPL. The result for UNIX Command Sequence (Shakespeare Text) is shown in Table 4 (Table 5) with $\mathcal{D}_s$ size increasing from 400 to 1000 (2000 to 8000). We have the following three observations from the results. First, the performance of the PL_Rand is slightly decreased compared to the PL under different $\mathcal{D}_s$ sizes. It is reasonable that large distribution discrepancy of the randomly requested $\mathcal{D}_s$ and $\mathcal{D}_i$ can deteriorate the personalized model performance. Second, the accuracy values of the PL_Euc, PL_Cos, PL_Multi and WL increase with the increasing size of $\mathcal{D}_s$. Third, we discover that with smaller $\mathcal{D}_s$ size, the accuracy increment from WL to AdvPL is more significant. More specifically, the accuracy increment of the AdvPL over the WL for UNIX Command Sequence (Shakespeare Text) is 3.32% (2.30%), 2.96% (1.96%), 2.62% (1.67%) and 2.08% (1.26%), respectively, with the corresponding $\mathcal{D}_s$ size as 400 (2000), 600 (4000), 800 (6000), 1000 (8000). The reason is that with smaller $\mathcal{D}_s$ size, some less similar data in $\mathcal{D}_s$ impedes the model performance improvement to a greater extent due to the smaller total amount of data. In contrast, if the total data size is larger, then the effects of some less similar data are smaller and the advantage of the AdvPL is also weakened. As a result, the AdvPL plays a more important role for personalized model with smaller budget size of $\mathcal{D}_s$. It is more practical in real scenarios that one user can only request limited amount of data from other users due to the privacy concern or communication cost burden.

To further compare the performance of different strategies, we plot the percentage of users with accuracy increment greater than 5% using the PL_Euc, PL_Cos, PL_Multi, WL and AdvPL over the PL under different $\mathcal{D}_s$ sizes. The result is shown in Fig. 4. It can be seen that the AdvPL greatly outperforms other strategies. In addition, the WL achieves better performance than the PL_Euc and PL_Cos, demonstrating the effectiveness of our proposed method in requesting similar data.

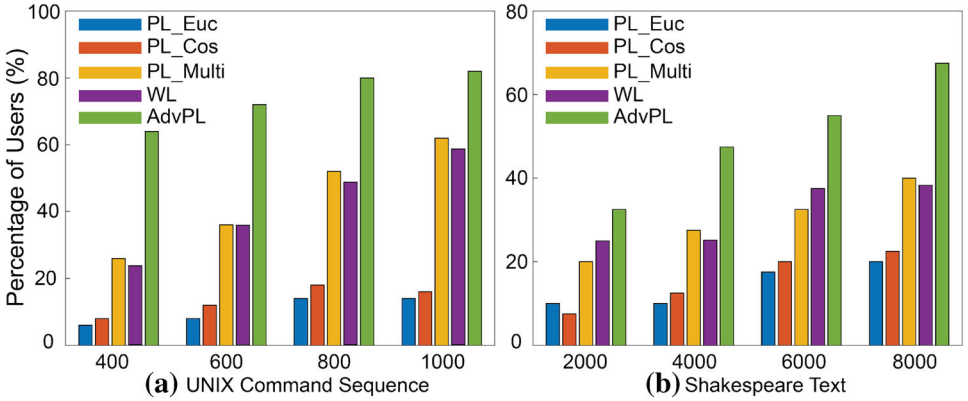


Fig. 4 The percentage of users with accuracy increment over 5% using the PL_Euc, PL_Cos, PL_multi, WL and AdvPL compared to the PL

4.3.4 Effects of probability distribution and budget size on WL, AdvPL and PL_Sim

In this section, we investigate the effects of the probability distribution of the requested data determined by D_2 and the budget size of $\mathcal{D}_s$ for the WL, AdvPL and PL_Sim. We randomly select a single user (ID = 29) in the UNIX Command Sequence dataset and conduct the experiments for demonstration purpose. Table 6 shows the effects of probability distribution of the requested data for the WL, AdvPL and PL_Sim. We divide the range of probability score by D_2 evenly into five regions between 0.5 and 1.0. For each region, we select $\mathcal{D}_s$ containing 1000 samples and test the accuracy of the WL, AdvPL and PL_Sim. For example, if we select the range [0.5, 0.6], it means the probability score of all requested data falls into the range [0.5, 0.6].

We have the following observations. First, the accuracy of the WL is lower if the probability distribution of the data in $\mathcal{D}_s$ is in a low range. For example, if all of the data in $\mathcal{D}_s$ are requested within the range [0.9, 1.0], then the accuracy of the WL is 5% higher than that of the $\mathcal{D}_s$ within the range [0.5, 0.6]. As we know, higher probability by the discriminator indicates the data are more likely to be sampled from $\mathcal{D}_i$. Consequently, lower probability distribution range of $\mathcal{D}_s$ will cause larger distribution discrepancy with $\mathcal{D}_s$ and lower performance improvement of the WL. Second, the accuracy of the AdvPL for $\mathcal{D}_s$ with different probability distribution ranges is stable. The advantage of the AdvPL is that it can reduce the distribution discrepancy between $\mathcal{D}_i$ and $\mathcal{D}_s$, so that the performance of the AdvPL can still be improved greatly under a higher distribution discrepancy. It demonstrates that the AdvPL can better improve the personalized model performance if only less similar data are available. Third, the advantage of the AdvPL is weakened if the distribution discrepancy between $\mathcal{D}_i$ and $\mathcal{D}_s$ is smaller. The reason is that if $\mathcal{D}_s$ is of high similarity compared with $\mathcal{D}_i$, then the user can directly combine them and train the model without considering the effect of distribution discrepancy.

Table 6 Accuracy comparison (mean ± std) based on five runs for a single user (UNIX Command Sequence) based on $\mathcal{D}_s$ with different discriminator scores

Discriminator Score	PL_Sim	WL	AdvPL
0.5–0.6	60.8 ± 1.64	63.0 ± 1.94	66.8 ± 1.79
0.6–0.7	63.0 ± 2.05	63.2 ± 1.48	67.0 ± 2.12
0.7–0.8	64.6 ± 1.67	66.0 ± 1.41	68.2 ± 2.17
0.8–0.9	67.0 ± 1.49	67.2 ± 1.10	67.8 ± 2.16
0.9–1.0	66.8 ± 1.31	67.0 ± 1.58	68.2 ± 1.92

Table 7 Accuracy comparison (mean ± std) based on five runs for a single user (UNIX Command Sequence) based on $\mathcal{D}_s$ with different sizes

$\mathcal{D}_s$ Size	PL_Sim	WL	AdvPL
400	57.6 ± 1.51	59.2 ± 1.48	63.0 ± 2.00
600	58.8 ± 1.64	60.4 ± 1.81	64.0 ± 2.44
800	60.2 ± 1.48	60.2 ± 1.09	65.4 ± 1.87
1000	60.8 ± 1.64	63.0 ± 1.94	66.8 ± 1.79

We also investigate the difference between WL and PL_Sim. In fact, both WL and PL_Sim are sensitive on the probability score. When the probability score is within low range, the accuracy of WL is higher than that of PL_Sim. This is because that WL lowers the importance of the requested data, while PL_Sim treats all requested data with uniform weight. Taking the probability score within the range [0.5, 0.6] as an example, the accuracy values (mean ± std) of PL_Sim and WL are 60.8 ± 1.64 and 63.0 ± 1.94, respectively. Its p-value from the t test is 0.0230, which indicates a significant improvement of WL over PL_Sim. When the probability score is within the high range, WL and PL_Sim have similar performance because WL is reduced to PL_Sim as aforementioned. In short, WL outperforms PL_Sim when the similarity between requested data and original data is relatively low. WL and PL_Sim achieve similar performance when sufficient similar data can be collected from neighbors.

Table 7 shows the effects of $\mathcal{D}_s$ size on the performance of the PL_Sim, WL and AdvPL. To test the sensitivity of the adversarial training process, we request less similar data and set the probability distribution of the data in $\mathcal{D}_s$ within the range [0.5, 0.6]. We have the following observations. First, the accuracy of the WL increases slowly with the increasing size of $\mathcal{D}_s$ as we select less similar data with the probability distribution in the low range [0.5, 0.6]. So although the user requests more data, the performance gain by these increasing less similar data is still insignificant. Second, with the adversarial training, the accuracy of the AdvPL improves greatly over the WL. It can be seen that the performance improvement of the AdvPL is almost stable under different sizes of $\mathcal{D}_s$. It demonstrates that the AdvPL can still improve the model performance greatly even with a limited budget size and less similar data available.

5 Conclusions and future work

In this paper, we proposed the AdvPL framework enabling individual users to effectively collect data from other users and train a robust personalized model. We proposed to use the auto-encoder and GAN to select similar data from other users. The trained auto-encoder and GAN, which capture inherent information of user's personal data, can be efficiently used to choose similar data from others, thereby avoiding tedious process of paired data comparison. We have developed two approaches to combine the requested data and user's own data to improve the performance of personalized learning. The first approach is weighted learning that assigns different weights to different requested data. Then, the model can capture the importance of different requested data. The second approach is adversarial training that maps selected data and user's own data to the same feature space and jointly trains the personalized model. The adversarial training can effectively mitigate potential distribution discrepancy between selected data and user's own data. We conducted extensive experiments to demonstrate the effectiveness of the proposed framework.

There are two major directions for our future work. First, we will study how to achieve privacy in our AdvPL. The auto-encoder and GAN contain private information of the individual user's data. Previous works demonstrate that deep learning models can memorize abundant information of the training data [42]. To protect the privacy of training data in $\mathcal{D}_i$, we can build differential privacy preserving versions of auto-encoder and GAN, e.g., by adopting the ideas of [43] and [44], respectively. Moreover, the data $\mathcal{D}_s$ collected from other users are also private and users may not want to share. We will study the use of local differential privacy [45] for private data comparison and collection. Second, we will investigate how to determine most appropriate data to improve the performance of personalized model. Our current work is based on the idea of selecting similar data to boost the performance. Ideally, we want to determine the properties of new data that can best improve the model performance, e.g., reducing the prediction error of the built model. With that, we only need to collect truly useful data and skip redundant ones, which will greatly reduce the communication burden and improve efficiency. One idea is to use active learning to select new data that improve personalized model performance. However, existing active learning strategies [46] may not be directly applied here because the data determined by the active learning may contain large distribution discrepancy from individual user's own data. It is interesting to investigate how to combine the active learning and personalized model.

Acknowledgements This work was supported in part by NSF 1920920, 1937010, 1940093 and 1946391. This article is an extension of the conference version [47]. We thank anonymous reviewers for their constructive comments.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. He, K., Zhang, X., Ren, S., and Sun, J.: "Deep residual learning for image recognition," in IEEE CVPR, (2016)
2. Lai, S., Xu, L., Liu, K., and Zhao, J.: "Recurrent convolutional neural networks for text classification," in AAAI, (2015)
3. Dong, X., Yu, L., Wu, Z., Sun, Y., Yuan, L., and Zhang, F.: "A hybrid collaborative filtering model with deep structure for recommender systems," in AAAI, (2017)
4. Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le , Q.V. etal.: "Large scale distributed deep networks," in NeurIPS, (2012)
5. Park, D.H., Kim, H.K., Choi, I.Y., and Kim, J.K.: "A literature review and classification of recommender systems research," Expert systems with applications, (2012)
6. Cheng, Y., Wang, F., Zhang, P., and Hu, J.: "Risk prediction with electronic health records: A deep learning approach," in SDM, (2016)
7. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y.: "Generative adversarial nets," in NeurIPS, (2014)
8. Chilimbi, T., Suzue, Y., Apacible, J., and Kalyanaraman, K.: "Project adam: Building an efficient and scalable deep learning training system," in OSDI, (2014)
9. Wen, W., Xu, C., Yan, F., Wu, C., Wang, Y., Chen, Y., and Li, H.: "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in NeurIPS, (2017)
10. Chen, C.-Y., Choi, J., Brand, D., Agrawal, A., Zhang, W., and Gopalakrishnan, K.: "Adacomp: Adaptive residual gradient compression for data-parallel distributed training," in AAAI, (2018)
11. Wang, S., Pi, A., Zhao, X., and Zhou, X.: "Scalable distributed dl training: Batching communication and computation," in AAAI, (2019)

12. Wangni, J., Wang, J., Liu, J., and Zhang, T.: "Gradient sparsification for communication-efficient distributed optimization," in NeurIPS, (2018)
13. McMahan, H.B., Moore, E., Ramage, D., Hampson, S. etal.: "Communication-efficient learning of deep networks from decentralized data," in AISTATS, (2016)
14. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., and Seth, K.: "Practical secure aggregation for privacy-preserving machine learning," in ACM CCS, (2017)
15. Smith, V., Chiang, C.-K., Sanjabi, M., and Talwalkar, A.S.: "Federated multi-task learning," in NeurIPS, (2017)
16. Che, C., Xiao, C., Liang, J., Jin, B., Zho, J., and Wang, F.: "An rnn architecture with dynamic temporal matching for personalized predictions of parkinson's disease," in SDM, (2017)
17. Suo, Q., Ma, F., Yuan, Y., Huai, M., Zhong, W., Zhang, A., and Gao, J.: "Personalized disease prediction using a cnn-based similarity learning method," in IEEE BIBM, (2017)
18. Choi, E., Bahadori, M.T., Searles, E., Coffey, C., Thompson, M., Bost, J., Tejedor-Sojo, J., and Sun, J.: "Multi-layer representation learning for medical concepts," in ACM KDD, (2016)
19. Huai, M., Miao, C., Suo, Q., Li, Y., Gao, J. and Zhang, A.: "Uncorrelated patient similarity learning," in SDM, (2018)
20. Wang, F., Sun, J., Ebadollahi, S.: Composite distance metric integration by leveraging multiple experts' inputs and its application in patient similarity assessment. Stat Anal Data Mining ASA Data Sci J **5**(1), 54–69 (2012)
21. Li, M., and Wang, L.: "A survey on personalized news recommendation technology," IEEE Access, (2019)
22. Luo, F., Ranzi, G., Wang, X., Dong, Z.Y.: Social information filtering-based electricity retail plan recommender system for smart grid end users. IEEE Trans Smart Grid **10**(1), 95–104 (2017)
23. Kouki, P., Fakhraei, S., Foulds, J., Eirinaki, M., and Getoor, L.: "Hyper: A flexible and extensible probabilistic framework for hybrid recommender systems," in ACM RecSys, (2015)
24. Hu, L., Cao, L., Wang, S., Xu, G., Cao, J., and Gu, Z.: "Diversifying personalized recommendation with user-session context." in IJCAI, (2017), pp. 1858–1864
25. Yu, Z., Lian, J., Mahmoody, A., Liu, G., and Xie, X.: "Adaptive user modeling with long and short-term preferences for personalized recommendation." in IJCAI, (2019), pp. 4213–4219
26. Bengio, Y., Courville, A., and Vincent, P.: "Representation learning: A review and new perspectives," IEEE TPAMI, (2013)
27. Tzeng, E., Hoffman, J., Darrell, T., and Saenko, K.: "Simultaneous deep transfer across domains and tasks," in IEEE CVPR, (2015)
28. Liu, A.H., Liu, Y.-C., Yeh, Y.-Y., and Wang, Y.-C.F.: "A unified feature disentangler for multi-domain image translation and manipulation," in NeurIPS, (2018)
29. Gupta, A., Devin, C., Liu, Y., Abbeel, P., and Levine, S.: "Learning invariant feature spaces to transfer skills with reinforcement learning," in ICLR, (2017)
30. Misra, I., Shrivastava, A., Gupta, A., and Hebert, M.: "Cross-stitch networks for multi-task learning," in IEEE CVPR, (2016)
31. Bouchacourt, D., Tomioka, R., and Nowozin, S.: "Multi-level variational autoencoder: Learning disentangled representations from grouped observations," in AAAI, (2018)
32. Narayanaswamy, S., Paige, T.B., Vande Meent, J.-W., Desmaison, A., Goodman, N., Kohli, P., Wood, F., and Torr, P.: "Learning disentangled representations with semi-supervised deep generative models," in NeurIPS, (2017)
33. Zadrozny, B.: "Learning and evaluating classifiers under sample selection bias," in Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004, ser. ACM International Conference Proceeding Series, C.E. Brodley, Ed., vol.69.ACM, 2004. [Online]. Available: https://doi.org/10.1145/1015330.1015425
34. Wen, J., Yu, C.-N., and Greiner, R.: "Robust learning under uncertain test distributions: Relating covariate shift to model misspecification." in ICML, (2014), pp. 631–639
35. Khodabandeh, M., Vahdat, A., Ranjbar, M., and Macready, W.G.: "A robust learning approach to domain adaptive object detection," in Proceedings of the IEEE International Conference on Computer Vision, (2019), pp. 480–490
36. Wang, X., and Schneider, J.: "Flexible transfer learning under support and model shift," in Advances in Neural Information Processing Systems, 2014, pp. 1898–1906
37. Huang, J., Gretton, A., Borgwardt, K., Schölkopf, B., and Smola, A.J.: "Correcting sample selection bias by unlabeled data," in Advances in neural information processing systems, (2007), pp. 601–608
38. Gretton, A., Borgwardt, K., Rasch, M., Schölkopf, B., and Smola, A.J.: "A kernel method for the two-sample-problem," in Advances in neural information processing systems, (2007), pp. 513–520
39. Schonlau, M., DuMouchel, W., Ju, W.-H., Karr, A.F., Theusan, M., Vardi, Y., etal.:, "Computer intrusion: Detecting masquerades," Statistical science, (2001)
40. Phan, N., Ebrahimi, J., Kil, D., Piniewski, B., and Dou, D.: "Topic-aware physical activity propagation in a health social network," IEEE intelligent systems, (2015)
41. Ruder, S.: "An overview of multi-task learning in deep neural networks," arXiv preprint arXiv:1706.05098, (2017)
42. Song, C., Ristenpart, T., and Shmatikov, V.: "Machine learning models that remember too much," in ACM CCS, (2017)
43. Phan, N., Wang, Y., Wu, X., and Dou, D.: "Differential privacy preservation for deep auto-encoders: an application of human behavior prediction," in AAAI, (2016)
44. Xie, L., Lin, K., Wang, S., Wang, F., and Zhou, J.: "Differentially private generative adversarial network," CoRR, (2018)
45. Duchi, J.C., Jordan, M.I., and Wainwright, M.J.: "Local privacy and statistical minimax rates," in IEEE FOCS, (2013)
46. Settles, B.: Active learning literature survey. University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep. (2009)
47. Du, W., and Wu, X.: "Advpl: Adversarial personalized learning," in DSAA, (2020)