The Journal of Systems & Software 174 (2021) 110902

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Concept drift-aware temporal cloud service APIs recommendation for building composite cloud systems☆

Lei Wang [a,*], Yunqiu Zhang [a], Xiaohu Zhu [b]

[a] Department of Management Science and Engineering, Nanjing Forestry University, Longpan Road 159, Nanjing 210037, China
[b] Center for Safe AGI, Lingshi Road 695, Shanghai, 200000, China

ARTICLE INFO

Article history:
Received 19 August 2020
Received in revised form 16 December 2020
Accepted 31 December 2020
Available online 7 January 2021

Keywords:
Application Programming Interfaces (APIs)
Cloud service recommendation
Preference drift
Temporal recommendation
User behavior
User preference

ABSTRACT

The booming advances of cloud computing promote rapid growth of the number of cloud service Application Program Interfaces (APIs) published at the large-scale software cloud markets. Cloud service APIs recommendation remains a challenging issue for a composite cloud system construction, due to massively available candidate component cloud services with similar (or identical) functionalities in the cloud markets. As for a specific user, the probability distribution of the data indicating his/her preferences to the cloud service APIs may change with time, resulting in concept drifting preferences. To adapt users' preference drifts and provide effective recommendation results to composite cloud system developers, we propose a concept drift-aware temporal cloud service APIs recommendation approach for composite cloud systems (or CD-APIR) in this paper. First, we track users temporal preferences through users' behavior-aware information analysis. Second, we utilize Singular Value Decomposition (SVD) method to predict the missing values in the user–service matrices. Third, we identify the degree of users preference drifts by Jensen–Shannon (or JS) divergence. Finally, we recommend cloud service APIs by presenting a piecewise trading-off equation. Experimental evaluations conducted on WS-Dream dataset demonstrate that the CD-APIR approach can effectively improve the accuracy of cloud service APIs recommendation comparing with 7 representative approaches.

1. Introduction

With the rapid advent of the cloud service industry, the fast growth of public available SOAP-based and RESTful cloud service Application Program Interfaces (APIs) has inspired the development of large-scale cloud service API markets (Bouguettaya et al., 2017). To name a few, there are ProgrammableWeb,[1] api-platform,[2] Alibaba Cloud Market,[3] etc. With the help of service composition techniques, developers can select, rent, and then assemble cloud service APIs to build new and value-added composite cloud systems based on these cloud markets. Consequently, API mashups and composite cloud system applications are more and more popular recently (Lemos et al., 2016; Niu et al., 2017; He et al., 2017).

There are massive APIs with similar (or identical) functionalities but different Quality of Service (QoS) in the cloud markets (Zheng et al., 2010a; Wang, 2019). The non-functional requirements of different users (i.e., the composite cloud system developers) to the APIs are diverse. With the fast increase of the number of cloud service APIs and their information (i.e., service descriptive information, price, and QoS, etc.), information overload in the distribution and retrieval of cloud service APIs is a critical problem that should be investigated (Cao et al., 2020; Zhang et al., 2018b; Wang et al., 2018). Service selection decision should be made over the big data-space APIs' non-functional information to match the user preference and the heterogeneous APIs information. To this end, an effective cloud service APIs recommendation based on data science techniques is urgently desired by users, cloud service APIs providers and the cloud market platforms (Wang et al., 2016; Ding et al., 2018).

Different from the traditional service provider-driven Web and/or Cloud service markets, the large-scale cloud service APIs market is more open and dynamic. Under the large-scale cloud APIs markets, the non-functional information of each API dynamically changes. For budget changes, application requirement changes, interest changes, etc., the users' preferences to the APIs' non-functional attributes may also change with time. Suppose that the users' preferences for different APIs are expressed as a

☆ Editor: [J.C. DUENAS].

* Corresponding author.
E-mail addresses: leiwangchn@163.com (L. Wang), eitherzhangyq@163.com (Y. Zhang), xhzhu.nju@gmail.com (X. Zhu).

[1] https://www.programmableweb.com/.
[2] https://api-platform.com/.
[3] https://market.aliyun.com/products/.

https://doi.org/10.1016/j.jss.2020.110902
0164-1212/

user–service rating matrix, where each row represents the user's ratings to the APIs and each column represents an API's ratings by different users. Changes, gradual or sudden, of the APIs information and the user' preferences make the distribution of the users' rating information to the cloud service APIs change with time, resulting in concept drifts of user preferences (Barddal et al., 2017; Lu et al., 2019; Gama et al., 2014). Traditional Web and/or Cloud service recommendation methods do not consider the issue that user preferences may drift, resulting in reduced accuracy of the traditional recommender models (Cao et al., 2009). Cloud service APIs recommendation for large-scale cloud markets while considering the preference drifts is a challenging issue.

First, it is difficult to track the users' preference drifts. User preferences change over time. Concept drifts may take place in user preference data streams. Moreover, preference drifts are caused by hidden variables that cannot be directly measured. The direction of drifts (positive or negative) is unforeseen and different for each user. When the recommender model cannot adapt to the users' preference drifts in time, the accuracy of the recommendation results will reduce accordingly.

Second, the traditional recommender models assume that user preferences are static. That is, users' history information, regardless of the time of occurrence, plays an identical role in predicting current user preferences. However, the concept drifts of users' temporal preferences are commonly seen during a period of time (Zhang et al., 2016). When user preferences drift, traditional recommender models use the original data blindly, resulting in suffering a significant reduction in performance.

Third, APIs' QoS evaluation information collected via users' feedbacks or reported by API providers may be biased. Identification of user preferences by requirement elicitation techniques may also be incomplete or incorrect. More useful information for constructing the recommender model should be analyzed based on users' historical behavior data.

To the best of our knowledge, this is the first work that considers the effect of concept drifting preferences for cloud service APIs recommendation. Traditional Web and/or Cloud service recommendation approaches mainly predict quality of service (QoS) and user preferences through collaborative filtering (CF) algorithm, including user-based recommendation, item-based recommendation and integrated users and services recommendation (Zheng and Lyu, 2013; Zhang et al., 2018a,b). User-based CF recommends services by identifying similar users. It relies on users' ratings of services to calculate similarity, or relies on purchasing the same services to cluster users. However, the users' reported rating information may be not reliable and complete, and users may maliciously submit feedback information (Somu et al., 2018; Yao et al., 2015). Clustering users through the same service purchased may suffer from the cold-start and data sparsity problem. Item-based CF is also not applicable to the cloud service APIs recommendation. In the large-scale cloud service APIs markets, users often purchase composite cloud service APIs when developing composite cloud system applications. Moreover, users seldom repeatedly purchase cloud service APIs that are extremely similar to the purchased ones. Therefore, the recommendation of the cloud service APIs should make some changes in the service-based CF to avoid recommending similar APIs repeatedly (Qi et al., 2019a). Some recommendation approaches integrate users and services. They mainly combine QoS data of similar users and similar services to predict user preferences. However, due to the data sparsity problem in the large-scale cloud markets, the recommendation results will be inaccurate.

To summarize, existing service recommendation methods predict user preferences through users' feedback data and/or purchased data. The identified preferences for specific users are static. However, as for a composite cloud systems developer under a large-scale cloud service APIs market, when renting cloud service APIs, his/her preferences may drift over time. The recommended services obtained by traditional approaches cannot accurately track preference drifts and meet the users' dynamic changing demands, which will deteriorate the accuracy of recommendation approaches over time (Zafari et al., 2019). For the uncertainty of preference drifts, the degree of a preference drift impacts the accuracy of recommender model in different degree. A minor drift has little impact on the recommendation results, while a major drift impacts significantly. A model adaptation decision must be carefully made. An online learning mechanism is essential for the recommender system to stay with the current user preferences.

To deal with this above issue, we propose a concept drift-aware temporal cloud service APIs recommendation approach for building composite cloud systems (or CD-APIR) under the large-scale cloud market environments. CD-APIR tracks changes in user preferences through the temporal behavior-aware information, and combines the results of preference drift detection with cloud service APIs recommendation to generate recommendation results. The contributions of this paper are summarized as follows.

- We model the degree of preference drift of a specific user by calculating the distance of two preference data stream distributions in neighboring time windows.
- We analyze the behavior-aware information including users' selecting, tracking, purchasing, and evaluation records and mark the time at which users' behaviors occur with timestamps to predict the missing values in the user–service matrices based on Singular Value Decomposition (SVD).
- We propose CD-APIR, a concept drift-aware temporal cloud service APIs recommendation approach for building composite cloud systems. Jensen–Shannon (or JS) divergence is utilized to identify the degree of preference drift for the users. A piecewise equation is presented to make trade-off decisions between the previous user preferences and the drifted up-to-date user preferences to make recommendations.
- We conduct extensive experiments on WS-Dream dataset to verify the effectiveness of the proposed approach. Experimental results compared with 7 respective approaches show that our approach takes advantage of the temporal behavior-aware information and improves the recommendation accuracy, which sheds some light on a new way to make cloud service APIs recommendations.

The remainder of the paper is organized as follows. We give a motivating example of preference drift in Section 2. We summarize the related works done on service recommendation and interest drift in Section 3. We introduce the modeling approach of preference drift in Section 4. In Section 5, we present the CD-APIR approach in detail. Section 6 gives the experimental evaluations. Finally, Section 7 concludes the paper with some future directions.

2. Motivating example

Suppose that an application developer, say Alice, wants to develop an advanced composite cloud system app for weather forecasting by integrating open cloud service APIs. As illustrated in Fig. 1, during the early stage of the development, Alice prefers to the following four basic APIs to build the system. (1) Address identification APIs (s_1), which are used to identify the location of the user to provide default weather forecasting result; (2) Search APIs (s_2), which are used to search for the target city to provide the city's weather forecasting result; (3) Map APIs (s_3); and (4)

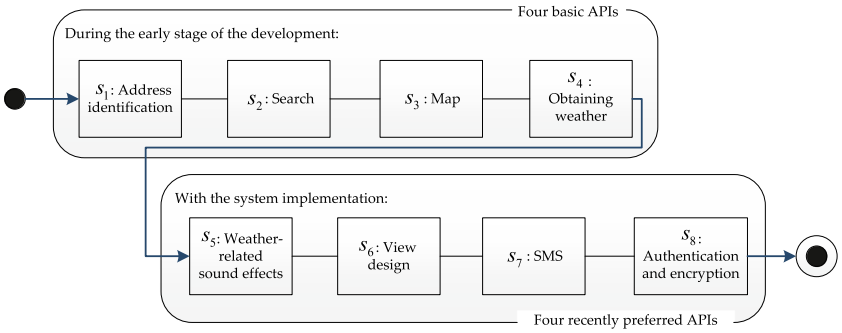


Fig. 1. A demonstrating application of preference drifts.

Obtaining weather APIs (s_4). During this stage, the recommender system should only recommend the above four types of APIs to Alice. With the system implementation, Alice may want to enhance the user experience of the system. Her interests to the APIs may drift. She may prefer the following four APIs. (1) Weather-related sound effects APIs (s_5), which are used to provide some auditory stimulations to the users; (2) View design APIs (s_6), which are used to enrich the users' visual stimulation; (3) Short message services (or SMS) APIs (s_7), which are used to send reminders of weather warning to the users; (4) Authentication and encryption APIs (s_8), which are used to protect user privacy and solve the security problems. To satisfy Alice's drifted preference, the recommender system needs to recommend these recently preferred APIs to her. If the recommendation is still based on the previous preference model, the drifted new preference of Alice may be ignored. The recommender system needs to identify the preference drifts in real-time to enhance recommendation accuracy.

3. Related work

In this section, we review some relevant existing works that significantly inspired our proposed CD-APIR approach which include service recommendation and interest drift.

3.1. Service recommendation

CF is one of the current popular service recommendation methods. After calculating the similarity, CF mainly use the similarity to filter out the items with low similarity by a fixed threshold. Then it predicts and fills in the missing values of the user–service matrix based on the existing ratings, QoS values, context and other data of users or services with high similarity (Zhang et al., 2018b; Xu et al., 2016).

Content-based recommendation mainly recommends services similar to those selected in the past. Knowledge-Based recommendation uses a similar function to construct a knowledge network to estimate how much the user's demands match the recommendations. The combination of machine learning and collaborative filtering is a relatively popular recommendation approach recently (Zhang et al., 2019; Qin et al., 2020; Xia et al., 2015; Batmaz et al., 2019).

To solve the problem of data credibility, Su et al. (2017) paid attention to the problem of unreliable QoS data contributed by dishonest users, and proposed a trust-aware approach, namely TAP. It is used to predict reliable personalized QoS, which combines QoS data of trustworthy similar users and similar services to make prediction for active users. Somu et al. (2018) focused on cloud service providers (CSP), and proposed a Hypergraph Binary Fruit Fly Optimization based service ranking Algorithm (HBFFOA) to identify suitable service and trustworthy CSPs that meet user demands by similar CSPs.

To improve the accuracy and diversity of APIs recommendation results, Cao et al. (2020) proposed an integrated content and network-based service cluster and Web APIs recommendation approach, which utilizes the implicit co-invocation relationship between Web APIs inferred from the historical invocation between Mashups and Web APIs to recommend diverse Web APIs for each Mashups clusters. Qi et al. (2019a) paid attention to the efficiency of APIs recommendation and compatibilities between different Web APIs. They analyzed the input keywords describing the functions expected by application developers, and defined a weighted APIs correlation graph (W-ACG), and proposed a Keywords-based and Compatibility-aware APIs Recommendation (K-CAR) approach.

To address the problem of QoS instantaneity, that is, the QoS changes frequently over time, Ding et al. (2018) combined the CF and the ARIMA model to capture the temporal feature of user similarity and solve the data sparsity in the existing PITs (point in time) approach, and then predict the QoS values in the future PIT under QoS instantaneity. The work in Qi et al. (2019b) combined time factor and privacy protection for service recommendation.

Existing service recommendation approaches rarely consider the occurrence of preference drifts. User preferences change dynamically, and it may drift with some unmeasured variables. Solely filling missing values based on user similarity is not applicable in the cloud service API market, which cannot address the data sparsity. If users do not purchase similar services, the recommender system cannot cluster them. In addition, the cloud market will have increasing users without sophisticated programming knowledge (Yao et al., 2015). This may make users' self-description information unreliable, which will reduce the accuracy of user-based recommendation.

3.2. Interest drift

The preference drift researches in traditional recommender systems mainly focus on reasoning the changes of user preferences to predict users' future interests. Yin et al. (2016) focused on the phenomenon of user interest drifts across geographic regions, and proposed Spatial–Temporal LDA (ST-LDA) to enhance the inference of region-dependent personal interests through effectively exploiting the social and spatial correlation information. Zafari et al. (2019) considered the dynamics of preferences and the reasons for changes in user preferences, and proposed a latent factor model to capture the domain-dependent component-specific temporal patterns in preferences. Wangwatcharakul and Wongthanavasu (2018) used item clustering and linear regression techniques to predict the future interests of users by categories and Gaussian mixture models to fill the data matrix to solve data sparsity problem. To detect

interest drifts, Cao et al. (2009) constructed a rating graph and a rating chain through the similarities between rated items to identify the type of a given user's interest pattern for improving recommender systems.

Considering the changes of user preferences, especially the problem of concept drift, the learning model for cloud service APIs recommendation faces a hidden technical debt issue, i.e., the recommender system needs to continuously make new technical efforts to update the model to learn from new data (Sculley et al., 2015). Training new recommender models results in wasting model learning time and raising the operation and maintenance costs of the recommender system. In contrast to the code level issues, this debt may be difficult to be resolved because it is based on the system level data analysis.

To summarize, the existing recommender models assume that the users' behavior information is static. Users' historical behavior information is utilized to predict user preferences (Cao et al., 2020; Huang et al., 2018). However, they ignore that users' preferences may change over time. When predicting the users' preferences, the latest behavior information may play a more important role. To deal with the incomplete and vague item features and user-interest drifts, the existing interest drift-aware approaches perform fuzzy recommendations based on preference drift detection (Zhang et al., 2016; Zenebe et al., 2010; Cornelis et al., 2007). For the dynamics of user interests, users' preferences to the APIs may change to different degrees over time. It is particularly important to study online adaptive recommendation methods based on accurately measuring the degree of preference drifts in real-time.

In this paper, we propose CD-APIR, which is an online adaptation approach to deal with preference drifts. The model adaptation is realized by the degree of preference drift detection and the trading-off piecewise equation. Online capturing the degree of preference drifts and updating the learning model for cloud service APIs recommendation may be one of the possible ways to address the technical debt problem of machine learning models. The proposed CD-APIR approach combines degree of preference drifts detection and drifts adaptation to predict the users' real-time ratings to the APIs for cloud service APIs recommendation. CD-APIR contributes to more accurately predict the user preferences in real-time for temporal cloud service APIs recommendation.

4. Preference drift modeling

Given $U = \{u_1, \ldots, u_i, \ldots, u_m\}$ as the set of users, and m is the total number of users registered in the large-scale cloud market ($1 \leq i \leq m$), we define $S = \{s_1, \ldots, s_j, \ldots, s_n\}$ as the cloud service API set, where n is the total number of cloud service APIs published in the market ($1 \leq j \leq n$).

When users select, track, purchase, and evaluate services, they will generate temporal behavior-aware information about certain services. Each behavior establishes a connection between the user and the service, which can be recorded in the website's server logs. These connections turn into the basic component of the user–service matrix.

Preferences indicate the users' willingness to purchase services, which include both the selected and unselected cloud service APIs. The ratings are usually used to quantify the user preferences for them. Hence, we define the user preference to a specific API as the rating of the service s_j evaluated by the user u_i, which is denoted by $r_{i,j}$. The preference of a user u to all the APIs is defined as

$$P(S, u) = r(S, u) = [r_{u,1}, r_{u,2}, \ldots, r_{u,n}], \quad (1)$$

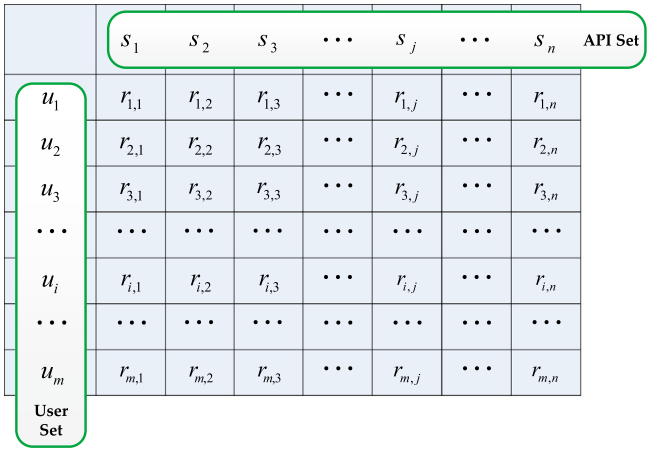


Fig. 2. A user–service matrix.

where S represents the n-dimensional feature vector of all the cloud service APIs in a cloud market, and u represents the label, that is, the user u_i ($u_i \in \{u_1, u_2, \ldots, u_m\}$).

In particular, the user–service matrix is depicted in Fig. 2. It should be noted that each user only buys a few cloud service APIs from the market. For instance, during the early stage of developing the weather forecasting composite cloud system presented in Section 2, Alice may only prefer to and search for the APIs of s_1, s_2, s_3, and s_4. There is no browsing or usage information for other APIs in Alice's historical behavior information. As a result, the user–service matrix is generally sparse (Cao et al., 2020; Yu et al., 2015).

However, user preferences for services are not static and usually change over time. As a result, we need to distinguish preferences during different periods.

Definition 1 (*Temporal User Preference*)**.** We define the temporal user preference of user u at a given time window t as

$$P_t(S, u) = r_t(S, u). \quad (2)$$

Temporal behavior-aware information usually changes over time, thus we use a preference data stream to describe the sequence of ratings changing over time, where the order in which the preference appears is specified by a timestamp (Zafari et al., 2019).

Definition 2 (*User Preference Data Stream*)**.** Given a time period $\{0, 1, \ldots, t\}$, user u's preference data stream is defined as $P_{0,t}(S, u) = \{P_0(S, u), P_1(S, u), \ldots, P_t(S, u)\}$.

We give the following example to demonstrate the preference drift. Suppose that, in the early stage of the composite cloud system development, Alice formerly preferred to the APIs s_1, s_2, s_3, and s_4. In this stage, the temporal user preference of Alice is described as $P_t(S, u)$. When time goes to the system implementation stage, the temporal user preference of Alice would change to $P_{t+1}(S, u)$ (i.e., more prefers to s_5, s_6, s_7, and s_8) and $D(P_t(S, u) \parallel P_{t+1}(S, u))$ is higher than the fixed threshold θ. At this time, we can say that Alice's preference has drifted. This is a realistic example of preference drift. We give the following definition of preference drift.

Definition 3 (*Preference Drift*)**.** Given a user u and its preference data streams $P_{0,t}(S, u)$, for $i \in \{0, 1, \ldots, t-1\}$, if $D(P_i(S, u) \parallel P_{i+1}(S, u)) \geq \theta$, it is identified that there is a preference drift of user u at time window $i+1$, where $D(\cdot \parallel \cdot)$ measures the distance of two preference data stream distributions in different time windows, and θ is a threshold value.

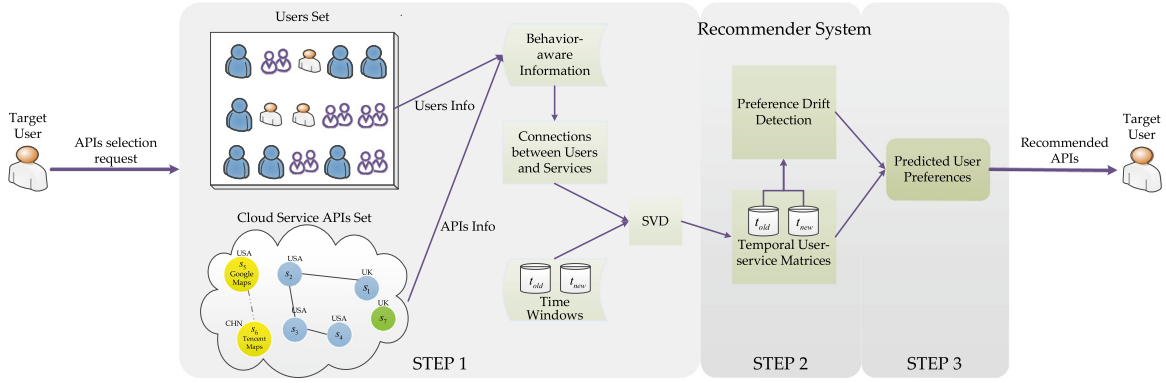


Fig. 3. An overview of CD-APIR approach.

Preference drift means that user preferences for different APIs change in an arbitrary manner over time. This change may be caused by some hidden variables that cannot be measured directly (Liu et al., 2017).

For example, suppose the user originally preferred to the services with high QoS, but the budget became tight due to some unknown reasons that recommender systems cannot measure. Then the user changed to focus on APIs' prices and relax requirements for services' performance. Preference drifts may not only change from preferring QoS to price, but also from preferring the type of interface to location, etc.

Definition 4 (*KL Divergence of Neighboring Temporal Preferences*)**.** We use Kullback–Leibler (KL) divergence (or relative entropy) to measure the dissimilarity between the distributions of corresponding user preferences in each neighboring time window's matrix (Wang and Zhou, 2013; Dasu et al., 2006; Sebastião and Gama, 2007; Chen et al., 2017). The KL divergence of neighboring temporal preferences is defined as

$$D_{PKL}(P_t(S,u) \parallel P_{t+1}(S,u)) = \sum_{S} P_t(S,u) \log \frac{P_t(S,u)}{P_{t+1}(S,u)}, \quad (3)$$

where $P_{t+1}(S,u)$ is the distributions of user u's preference at time window $t+1$, that is, u's ratings of all services, which is generated by the approach proposed in Section 5.2. t and $t+1$ are allocated by timestamps. S is the set of cloud service APIs. $D_{PKL}(P_t(S,u) \parallel P_{t+1}(S,u))$ is the distance between these two distributions, and $D_{PKL}(P_t(S,u) \parallel P_{t+1}(S,u)) \geq 0$.

The larger value of D_{PKL} indicates that the distributions of preferences in two time windows (t and $t+1$) are more different. If $P_t(S,u) = P_{t+1}(S,u)$, $D_{PKL}(P_t(S,u) \parallel P_{t+1}(S,u)) = 0$ (MacKay and Mac Kay, 2003).

Noting that, the evaluation criteria provided by different users are different. This makes the distribution of different user preferences data stream $P_{0,t}(S,u)$ varies. The data variety results in a heterogeneous distribution of D_{PKL} values, which is not conducive to the comparison of the degree of preference drifts in a uniform metric. To address this issue, we utilize the min–max normalization to transform the D_{PKL} values to a uniform dimension with the range of [0, 1]. In this way, we will ensure that a larger value can indicate more serious preference drifts. We define the following Δ function to measure the degree of preference drift.

Definition 5 (*The Degree of Preference Drift*)**.** The degree of preference drift aims at providing a normalized value to measure the divergence between the preference distributions (Lu et al., 2019). Formally, the degree of preference drift $\Delta(P_t(S,u) \parallel P_{t+1}(S,u))$ is defined as

$$\Delta(P_t(S,u) \parallel P_{t+1}(S,u)) = \frac{D_{PKL}(P_t(S,u) \parallel P_{t+1}(S,u)) - D_{\min}}{D_{\max} - D_{\min}}, \quad (4)$$

where $D_{\max}$ and $D_{\min}$ are the minimum and maximum D_{PKL} values of all users, respectively. The larger the value of $\Delta(P_t(S,u) \parallel P_{t+1}(S,u))$, the larger degree of preference drift.

5. CD-APIR approach

In this section, we present the concept drift-aware temporal cloud service APIs recommendation approach for building composite cloud systems (or CD-APIR).

5.1. Overview of the recommender model

The recommender system needs to recommend the users' preferred cloud service APIs by mining the hidden demands of users. The CD-APIR tracks changes in user preferences through temporal behavior-aware information, models the evolution of users preference, and then combines preference drifts with cloud service APIs recommendations (Liu, 2015).

As illustrated in Fig. 3, CD-APIR utilizes the behavior-aware information such as selecting, tracking, purchasing and evaluating to establish the connections between users and services. At the same time, timestamps are used to mark the time of user behaviors, and different time windows are allocated to form multiple time-based user–service matrices. Missing values are filled with Singular Value Decomposition (SVD) in user–service matrices. Subsequently, the recommender approach detects whether the user preference distributions have changed in different time windows, and measures the degree of preference drifts. The results of the preference drift detection are combined with the user–service matrices of different time windows to predict user preferences. Finally, recommended cloud service APIs are returned to target user.

5.2. Temporal behavior-aware information

To build the CD-APIR model, we need to collect and preprocess the user data and service data in the cloud vendor-driven API market. The recommender system needs to predict user preferences, that is, the APIs' users may be interested in.

First of all, we extract user purchase, evaluation, QoS values and other records to establish user–service matrices. Every time a user purchases a certain cloud service API, a connection will be established between the user and this cloud service API. As a result, the ratings will be filled into the user–service matrix. If the user has not yet provided a rating on the website, recommender system uses the QoS value ($q_{u,s}$) evaluated by the user u of the invoked service s to fill in the user–service matrix. $q_{u,s}$ needs to be normalized and converted into a rating, which can eliminate dimensions of each value and unify tendencies. We utilize the

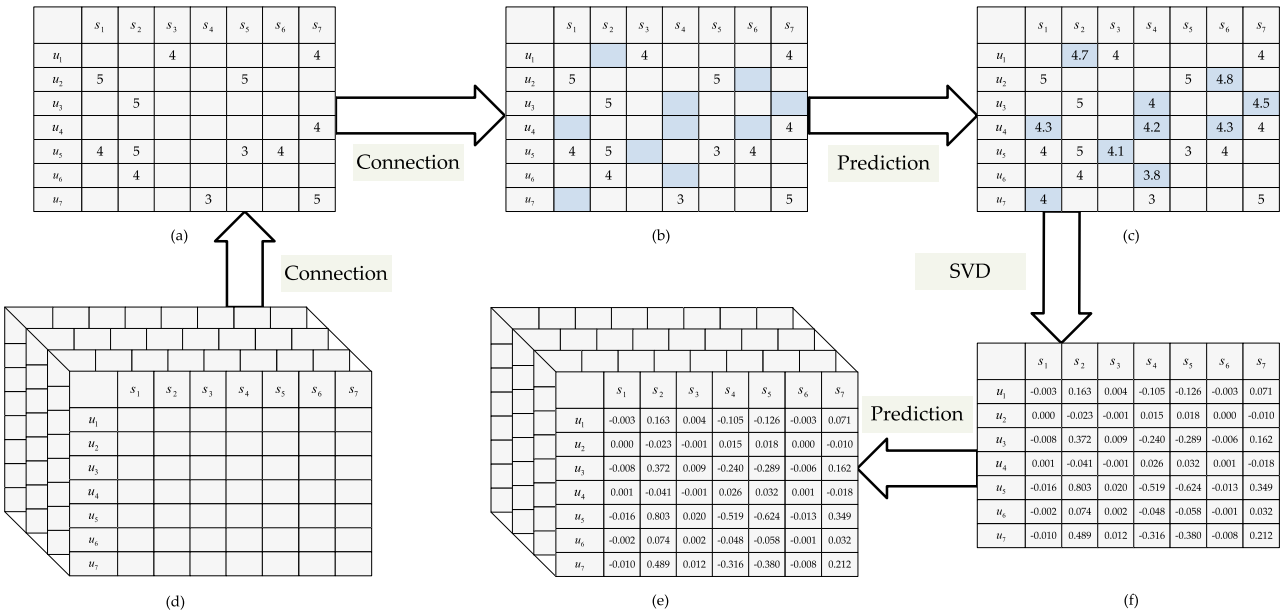


Fig. 4. Temporal user–service matrices formation process.

min–max normalization to map the values into the range of [0, 5]. For the indicators that bigger is better

$$r_{u,s} = 5 \times \frac{q_{u,s} - q_{\min}}{q_{\max} - q_{\min}}. \tag{5}$$

As for the indicators that smaller is better

$$r_{u,s} = 5 \times \frac{q_{\max} - q_{u,s}}{q_{\max} - q_{\min}}, \tag{6}$$

where $r_{u,s}$ is the normalized rating user u gives to the service s (this equation is only applicable when the user has not provided an actual rating). $q_{\max}$ and $q_{\min}$ are the minimum and maximum QoS values of the services evaluated by all users, respectively. The user–service matrix is shown in Fig. 4(a). However, the user–service matrix is often sparse, and there are a lot of missing values in it.

We need to obtain more user–service connections through users' behavior-aware information, such as selecting, searching and tracking (other than purchase and evaluation). Taking ProgrammableWeb as an example, when a user clicks on the "track API" in the website, this API will be added to the user's watchlist. Through the watchlist, we can know what services the user may be interested in and build more user–service connections.

However, unlike the user's purchase and evaluation records, when the user tracked an API, we cannot know the user's evaluation or the QoS value of the API, because the service has not been purchased and invoked. The user–service connection (the colored part in Fig. 4(b)) can be established by behavior-aware information. Since the users' usage evaluation is unknown, we need to predict the users' ratings.

We can deal with this problem through associated services. It is worth noting that, as mentioned above, the large-scale cloud markets are different from other markets. Users may seldom purchase cloud service APIs with the same functions. Unless for some fault-tolerance applications, user may need to bind some redundant APIs. However, this situation is rare.

We can extract service information from the server log or crawl them from the Web pages of the software cloud market to construct a service matrix. We associate APIs through combinations of services purchased by users. As shown in Fig. 5, $\{s_1, s_2, s_3, s_4\}$ are clustered composite APIs. We take the user's

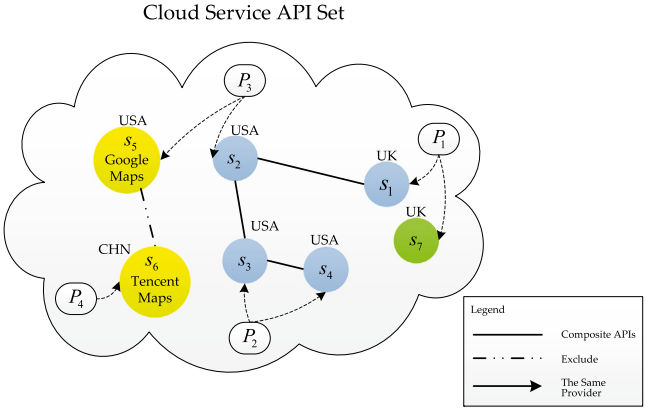


Fig. 5. Associated cloud service APIs.

evaluation of the purchased service s_1 to predict the user's evaluation of s_2, s_3 and s_4. The user may track a service at time t, but has not purchased any service in the composite APIs. For example, the user tracked s_2, but the user has not yet purchased s_1, s_3 or s_4. We cannot get the relevant ratings from composite APIs. But we can predict the rating by

$$r_{u_i,s} = \mathbf{N} \sum_{v \in \hat{C}} sim(u_i, u_j) \times r_{u_j,s}, \tag{7}$$

where $sim(u_i, u_j)$ is the similarity between u_i and u_j, which is calculated by Pearson Correlation Coefficient (PCC) (Elahi et al., 2016), $r_{u_j,s}$ is the u_j's evaluation of service s, $\hat{C}$ is a group of users that are the most similar user u_i who have purchased service s, $\mathbf{N}$ is a normalization factor, and

$$\mathbf{N} = \frac{1}{\sum_{v \in \hat{C}} sim(u_i, u_j)}. \tag{8}$$

Here the user–service matrix is shown as Fig. 4(c), some missing values are filled in.

Then, through the established user–service connections, we predict user preferences for all unselected services. The matrix decomposition algorithm is cast as the state-of-the-art approach

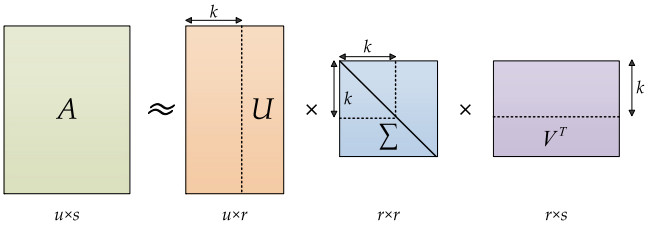


Fig. 6. Singular Value Decomposition for user–service matrix.

for its performance of filling the missing values for a large-scale sparse matrix and solving the problem of data sparsity. The advantages of SVD are to save computing resources and reduce noise. As illustrated in Fig. 6, in the recommender system, we do not have to use all the singular values, but select the largest k singular value to re-construct the matrix, which can basically express the original matrix.

Hence, we perform SVD to fill the user–service matrix ($[r_{u,s}]$). Before SVD, the average value of each user's ratings is subtracted for each service. Suppose $A_{u\times s} = [r_{u,s}]$ is the user–service matrix, where u represents the user, s represents the candidate service. Each value $r_{u,s}$ in matrix $A_{u\times s}$ represents the preference of the user for each service in a certain time window.

Specifically, we decompose $A_{u\times s}$ by solving the following equation.

$$A_{u\times s} \approx U_{u\times k}\Sigma_{k\times k}{V^T}_{k\times s}, \tag{9}$$

where k is much smaller than u and s. Thus, a large matrix $A_{u\times s}$ can be represented by three small matrices $U_{u\times k}$, $\Sigma_{k\times k}$ and ${V^T}_{k\times s}$, respectively.

In this way, SVD takes only the first few singular values to re-construct. It saves storage usage and calculation time, and reduces noise and server's computing load for the cloud service APIs recommender system. The user–service matrix is shown in Fig. 4(f).

Now we can only identify the user's preference during a period. However, when Alice's preference change from preferring s_1, s_2, s_3 and s_4 to preferring s_5, s_6, s_7, and s_8, the change need to be extracted by the recommender system in real-time. Those new APIs should be recommended to Alice based on her drifted preference.

To this end, we need to analyze user preferences in different time windows. As illustrated in Fig. 7, there are many ways to allocate the time windows, and *US* represents the user–service connections at different times. Old time window represents the matrix constructed by user–service connections, occurring in certain period (from $t+1$ to $t+6$), followed by the construction of multidimensional matrices over time. Each matrix represents user preferences at each time window. The occurrence of preference drifts are detected by analyzing user preferences in different window periods.

In Fig. 7(a), the starting points of two windows are fixed, while the ending point of two windows will be extended over time. We can also perform drift detection by comparing the most recent time window with the overall time window (as shown in Fig. 7(b)). Here the starting point of the old window is fixed. The ending points of the old window, the starting point and the ending point of the new window extend over time. In addition, when analyzing users' preference drifts, multiple windows can also be allocated, this method can accurately obtain the time at which the preference drifts occur. As shown in Fig. 7(c), if $D(P_{new(1)}(S,u) \parallel P_{new(2)}(S,u)) \le \theta \le D(P_{new(2)}(S,u) \parallel P_{new(3)}(S,u))$, it indicates a greater degree of preference drift for user u at time $t+10$.

The window allocation method can be decided according to the characteristics of the large-scale cloud markets. The window

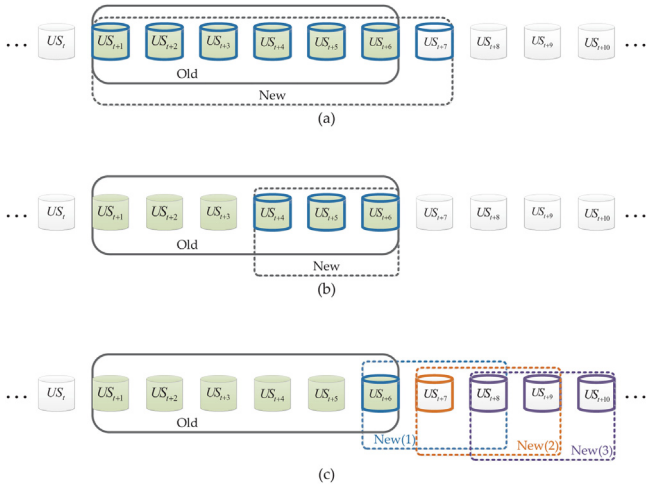


Fig. 7. Time windows for preference drift detection.

duration should not be too short, because a preference drift may not only take place at an exact timestamp but also last for a long period (Lu et al., 2019). Given the analysis of the problems studied in this paper, concept drift-aware recommendations need to address data sparsity and amplify the role of new window's user preferences (Koren, 2009). Therefore, the second method (i.e., Fig. 7(b)) is more appropriate for solving these problems. After choosing this method to allocate the windows, we obtain the user–service matrices to describe the user preferences (see Fig. 4(e)).

5.3. Preference drift detection

Preference drift detection is mainly used to evaluate the degree of preference drifts. The recommender system needs to detect the distribution changes of user preferences in different periods. When the recommender system finds that there is a degree of preference drift for some users, it need to update the predicted user preference to adapt to the drift.

After obtaining the user–service matrices, we need to calculate the dissimilarity of ratings in different time windows to measure the degree of drifts. When we allocate the two windows (old and new), preference drift detection can be understood as a two-sample detection problem. That is, it examines whether the population of two given matrices (also known as the sample sets) are from the same distribution.

Preference drift detection aims at identifying a shift in user preference distribution (Lu et al., 2014). According to Definition 5, $\Delta(u_{old} \parallel u_{new})$ is the degree of preference drift between the distributions of the user's old preference and new preference, which is calculated by Eq. (4).

The advantage of KL divergence is that it can not only report a warning when a preference drift occurs, but also provide the degree of the drift. However, KL divergence cannot be used as

a strict distance measurement. Noting that KL divergence is not symmetric under the interchange of the distributions u_{old} and u_{new}: in general $\Delta(u_{old} \parallel u_{new}) \neq \Delta(u_{new} \parallel u_{old})$ (MacKay and Mac Kay, 2003), which does not satisfy the symmetry requirements for distance measurement. We make some improvements and introduce Jensen–Shannon (JS) divergence to calculate the distance between the temporal user preference distributions. Unlike KL divergence, JS divergence is symmetrical and it can be used to measure the distance between distributions (Lin, 1991). At this time, we can obtain the degree of preference drift for each user by JS divergence as

$$D_{JS}(u_{old} \parallel u_{new}) = \frac{1}{2}\Delta\left(u_{old}\middle\|\frac{u_{old}+u_{new}}{2}\right) + \frac{1}{2}\Delta\left(u_{new}\middle\|\frac{u_{old}+u_{new}}{2}\right), \tag{10}$$

where $D_{JS}(u_{old} \parallel u_{new})$ is the distance between these two distributions (its range is [0, 1]). The larger it is, the larger the degree of preference drift. A larger degree of preference drift means that the user's preference has a larger degree of change. When we predict users' ratings, we need to pay attention to the role of new preference distribution with larger JS divergence, because they are likely to be more in accord with the current user preference distribution.

5.4. Preference prediction

Users' preference drifts can be tracked through preference drift detection. The recommender system amplifies the role of the users' preference in the most recent period, and then performs user preference prediction, finally recommends APIs with high prediction ratings to the user. In this paper, we do not need to update the recommender model in the case of preference drifts. CD-APIR can online adapt to the new preference by a trade-off decision.

Specifically, the JS divergence is combined with the user–service matrices of different time windows to generate recommendation results. The recommender system compares the distance of user preference distributions in different time windows. According to Definition 3, when $D_{JS}(u_{old} \parallel u_{new})$ is not lower than a fixed threshold θ, it indicates that the user preferences have drifted, and the role of the preferences at new time window needs to be considered.

In particular, when $\theta \leq D_{JS}(u_{old} \parallel u_{new}) \leq 1-\theta$, we make a trade-off to combine the two time windows' preference distributions and predict the user preference by

$$P_{u,s} = D_{JS}(u_{old} \parallel u_{new})R^{new}_{u,s} + \left[1 - D_{JS}(u_{old} \parallel u_{new})\right]R^{old}_{u,s}, \tag{11}$$

where $P_{u,s}$ is the user u's final predicted rating for service s, $R^{old}_{u,s}$ and $R^{new}_{u,s}$ represent the ratings in user–service matrices $A^{old}_{u\times s}$ and $A^{new}_{u\times s}$ in two time windows, respectively. θ is a fixed threshold, and the range of its value is [0, 0.5].

Furthermore, when $D_{JS}(u_{old} \parallel u_{new})$ is less than the fixed threshold θ, it implies a minor drift. In general, this preference drift may be caused by abnormal data due to inaccurate user feedbacks, measurement errors, etc. In fact, user preference has not really drifted in this situation, and the drift need to be further observed and verified. It will be more accurate to use the original predicted preference.

In contrast, when $D_{JS}(u_{old} \parallel u_{new}) > 1-\theta$, it indicates that user preferences drift dramatically, and the predicted old time window's preferences have expired, that is, it cannot meet user demands at present. Recommendation should consider the users' interest in the most recent period.

To ensure the stability of the recommender system, we further define the following piecewise equation for the above two conditions.

$$P_{u,s} = \begin{cases} R^{old}_{u,s}, & D_{JS}(u_{old} \parallel u_{new}) < \theta \\ R^{new}_{u,s}, & D_{JS}(u_{old} \parallel u_{new}) > 1-\theta \end{cases}. \tag{12}$$

To summarize, given the complex user preference drifts, the two time windows' preference distributions are combined in Eq. (11) in order to more accurately tap the hidden demands of users. To further improve the accuracy of recommendations, we define Eq. (12). In Section 6.7, we will further verify the recommendation effectiveness for the piecewise equation.

Finally, through the predicted user preference, the Top-N cloud service APIs are recommended to users. It is worth to note that, to avoid repeatedly recommending APIs that are the same as the user have purchased, the recommender system should exclude the purchased APIs from the recommended results.

6. Experiments

In this section, a set of experiments are conducted to evaluate the CD-APIR approach. To assess the effectiveness of the proposed approach for large-scale cloud service API markets, experiments were performed on a large-scale open service dataset. All the experiments were implemented using Python 3.7.3 on win32, with CPUs of 2.2 Ghz and 8 GB RAM. The operating system is Windows 10. To help other researchers repeat the experiments in this paper, we have published our implementation of CD-APIR and the competitors on Github, which is available at https://github.com/InBSLab/CD-APIR.

6.1. Dataset

In the experiments, to simulate actual API application scenarios, WS-Dream[4] dataset is adopted. The dataset collects 5825 services' performance data in 74 countries. All the services can be cast as RESTful or SOAP-based cloud service APIs. These services are accessed by 339 users in 31 countries. These data have actual response time (RTT) and throughput.

We calculate QoS values through RTT and generate user–service matrices. We normalize them by Eq. (6) to convert them into ratings. In this way, the maximum value of the user–service matrix is 5, which represents that service s is the most satisfactory for user u. The minimum value is 0, which is the unsatisfactory service. Since there is no timestamps in the dataset, to simulate the real purchase environment, we selected ratings' timestamps from the Movielens[5] dataset and then mapped them to WS-Dream dataset. Preference drifts are caused by hidden variables that cannot be directly measured, therefore, we use the most recent 20% ratings as the user preferences for the new time window.

However, the matrix generated by the WS-Dream dataset is dense. To verify the recommendation effectiveness of our proposed approach in the face of data sparsity, we sorted and removed 50% of the data with lower ratings due to users will not choose unsatisfactory services. The experimental results were obtained by comparing the actual services of the test data with the predicted recommended services (Zheng et al., 2014, 2010b).

6.2. Metrics

We define the following metrics to evaluate the effectiveness of the recommendation approaches.

(1) **Accuracy.** Accuracy is the most important evaluation criterion for recommender systems. This metric can directly reflect

[4] http://wsdream.github.io/dataset/.

[5] http://files.grouplens.org/datasets/movielens/.

the quality of cloud service APIs recommendation approaches. In this paper, we define the following metrics (precision, recall and F-measure) to evaluate the accuracy of the recommender results.

$$Precision = \frac{|L_t \cap L_r|}{L_r}, \tag{13}$$

$$Recall = \frac{|L_t \cap L_r|}{L_t}, \tag{14}$$

$$F - measure = \frac{2PR}{P + R}, \tag{15}$$

where L_t represents the list of cloud service APIs actually selected by the user, and L_r represents the list of cloud service APIs generated by the recommender system.

(2) **Coverage.** This metric measures the coverage of all the cloud service APIs by a recommender system. It describes the ability to discover services in the long tail. Coverage is often used in conjunction with accuracy, as the recommender system cannot give a poor accuracy in order to improve coverage. It is the ratio of cloud service APIs list $R(u)$ that the recommender system can recommend in the total API set S. Coverage is calculated by

$$Coverage = \frac{|R(u)|}{S}. \tag{16}$$

(3) **Reaction time.** The reaction time describes the average recommendation time to return the generated recommender list to a single user. The recommender system needs to present cloud service APIs to the user in the shortest possible time.

6.3. Approaches under comparison

Relevant existing works of cloud service recommendations mainly rely on Singular Value Decomposition, Collaborative Filtering, Hot services, PersonalRank, etc., as the primary approaches. Moreover, Euclidean distance and Wasserstein distance are representative data distribution-based concept drift detection approaches. These methods are all available for the concept drift-aware temporal cloud service APIs recommendation. Accordingly, to evaluate the effectiveness of the proposed approach, we implemented the following 7 respective approaches for cloud service APIs recommendation for comparison purpose. Specifically, the 7 approaches under comparison include:

- SVD: The matrix decomposition algorithm is considered as a popular approach for its excellence in filling missing values for large-scale sparse matrices (Qin et al., 2020; Ba et al., 2013). SVD reduces the dimensionality of the user–service matrix, which can alleviate the problem of data sparsity. We perform SVD on the matrix to predict user ratings, then exclude services that are same as purchased services, and finally recommend the services with the highest ratings.
- User-based CF: CF recommends services by predicting user preferences. User-based CF recommends services by identifying similar users. We rely on purchasing the same services to cluster users. It then predicts the APIs that users are interested in through the choices of similar users (Lin et al., 2014).
- Item-based CF: In the large-scale cloud markets, users tend to assemble and build composite cloud system. Item-based CF measures similarity between APIs and predicts the user's interest in them by the attributes of the user who selects them. The key to this approach is to mine and recommend composite APIs (Zheng et al., 2010a).
- Hot: Recommendation based on popular cloud service APIs for keyword-based search is also a essential approach. According to the statistical results, a small number of popular APIs are frequently used in mashups by many users (Cao et al., 2020; Yao et al., 2015). That is, certain popular cloud service APIs can meet the demands of most users, such as maps and SMS verification codes. In addition, on the homepage of the website, it is necessary to present some popular services to mine the potential demands of users. If users find satisfactory services on the homepage, it will benefit the users' search efficiency and increase the users' loyalty to the large-scale cloud market. In this paper, Hot sorts cloud service APIs by the popularity and return them to users.
- PersonalRank: The PersonalRank algorithm is proposed to map the relationship between users and services, such as selecting, purchasing and evaluating records, into an undirected bipartite graph (Hu et al., 2018). From the users' nodes, it uses the graph link structure to recursively calculate the importance of each node. The relevance and importance of network nodes can be calculated based on the source node. The higher the PersonalRank value, the higher the importance of the source node.
- ED-CDD: After allocating time windows, Euclidean distance-based concept drift detection approach (ED-CDD) is used to detect the distribution changes of user preferences. Euclidean distance is commonly used to calculate users' similarity or services' similarity in recommendation algorithms (Auch et al., 2020; Dhawan et al., 2015; Gao et al., 2017). In this paper, we use the Euclidean distance to measure the degree of preference drifts as a comparative approach.
- WD-CDD: Wasserstein distance is commonly known as earth mover's distance (EMD) (and also referred to as the "transportation metric") (Goldenberg and Webb, 2019). Existing study has combined it with CF to address item cold-start problem (Meng et al., 2020). In this paper, after SVD predicts ratings and fills the missing values, Wasserstein distance-based concept drift detection (WD-CDD) is used to calculate the similarity of user preferences in different time windows to detect the degree of preference drifts.

6.4. Performance comparison

In the experiment, we compared the proposed CD-APIR approach with 7 respective approaches, through measuring precision, recall, F-measure, coverage and reaction time, respectively. We set the $k = 60\%$, Top-$N = 10$, and $\theta = 0.1$. The experimental results are shown in Table 1.

Before running the algorithm, we subtract the average value of each user's ratings from their ratings for each service selected. When the matrix is decomposed, the missing values in the matrix will be filled. However, we find that if the rating interval is $[0, 5]$ and the missing value is 0, the missing values will be replaced by values close to 0. After we subtract the average value, 0 becomes the average rating, and the processing of the output results will also become convenient. Ratings for services unselected (i.e. the null values) in the matrix are not subtracted.

From the results, we can know that, the CD-APIR is superior to other methods in accuracy (precision, recall and F-measure) and coverage. The precision of the CD-APIR has 1.09% improvement over two concept drift detection approaches of ED-CDD and WD-CDD, 1.33% improvement over SVD, 8.97% improvement over Item-based CF, 45.60% improvement over User-based CF, 63.48% improvement over PersonalRank, and 73.69% improvement over Hot. After taking preference drifts into consideration, allocating time windows and detecting the degree of drifts, all the data distribution-based concept drift detection approaches (CD-APIR, ED-CDD and WD-CDD) perform better. For drift detection, JS divergence is superior to Euclidean distance and Wasserstein distance. This indicates that JS divergence is more suitable

Table 1
The results of performance comparison.

Metrics	CD-APIR	SVD	User-based CF	Item-based CF	Hot	Personal-Rank	ED-CDD	WD-CDD
Recommendation time	0.00307	0.00570	0.27906	17.02120	**0.00037**	0.00181	0.00320	0.00326
Precision	**0.82035**	0.80708	0.36431	0.73068	0.08348	0.18555	0.80944	0.80943
Recall	**0.02780**	0.02735	0.01369	0.02746	0.00283	0.00629	0.02742	0.02742
F-Measure	**0.05377**	0.05290	0.02639	0.05294	0.00547	0.01216	0.05305	0.05305
Coverage	**0.20670**	0.18884	0.05428	0.07088	0.00172	0.18238	0.19158	0.19193

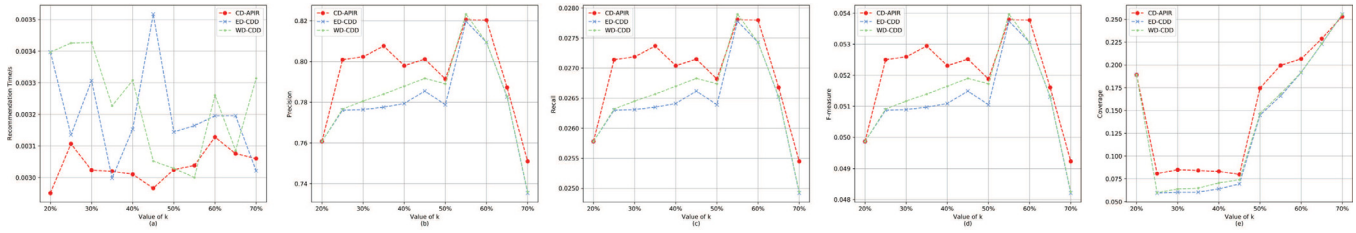


Fig. 8. The impact of parameter k.

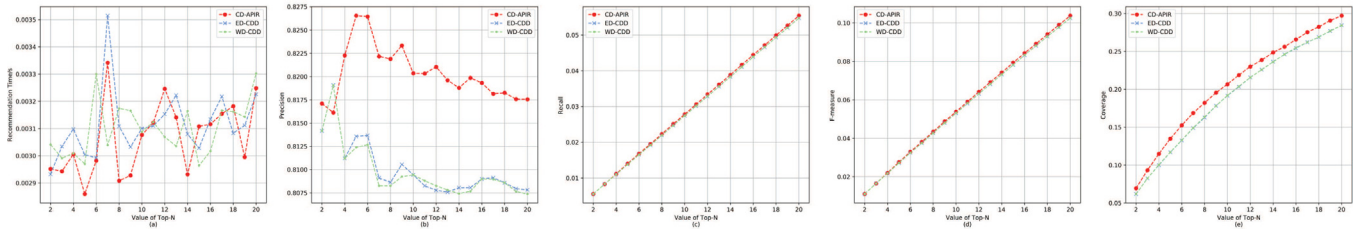


Fig. 9. The impact of parameter Top-N.

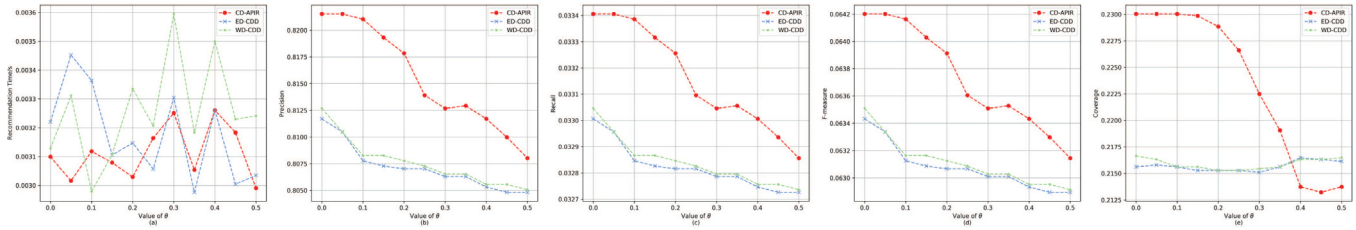


Fig. 10. The impact of parameter θ.

for cloud service APIs recommendation. Furthermore, Item-based CF is more applicable to recommend cloud service APIs than User-based CF.

The recall, F-measure and coverage are also superior to the other 7 comparative approaches. The reason why these metrics is low in all experiments is that the number of recommended services (Top-N) is few. In the parameter analysis in Section 6.6, we find that as the number of recommended services increases, recall, F-measure and coverage gradually increase.

In terms of coverage, the CD-APIR is more practical, as it can guarantee the effectiveness of the results on the basis of recommending more services.

In addition, the CD-APIR is superior in recommendation time, reducing the time to return the recommender list to a single user. When the CD-APIR shows superlative performance in terms of reaction time, it maintains excellent accuracy. Although Hot, User-based CF and PersonalRank performs better in recommender time, their accuracies are obviously lower than CD-APIR. Before recommending cloud service APIs, User-based CF spends more time calculating users' similarity than Item-based CF.

The CD-APIR utilizes user behavior-aware information and SVD to solve the data sparsity problem and predict preference systematically. It focuses on the changes of user preferences over time, and generates recommender lists based on the results of preference drift detection. The experimental results show that the CD-APIR approach improves the accuracy and coverage of recommendation.

6.5. Impact of parameter k

When performing SVD, the choice of parameter k will impact the recommendation performance. In the recommender system, the information of the original matrix can be approximated by only taking a few singular values. Therefore, we do not have to use all the singular values, but select the largest k singular value to re-construct the matrix.

We change the value of the parameter k, and investigate its impact on different recommendation approaches. The value of k is determined according to the proportion of singular values. In general, we believe that when k is small, it can significantly reduce the pressure on online storage, but it may cause the user–service matrix to lack the necessary recommendation information. If the k is large, the matrix can be better restored, but the recommender results may be affected by a few abnormal data, reducing the accuracy of recommendation.

In this experiment, we set Top-$N = 10$, $\theta = 0.1$, and k to gradually increase from 20% to 70% with a step value of 5%. We

Table 2
Comparison of recommendation results.

Alice's drifted preference			CD-APIR			SVD		
IDs	APIs	Providers	IDs	APIs	Providers	IDs	APIs	Providers
3367	Permissions	SP20	**3374**	Views	SP20	**3376**	Forms	SP20
3253	Phone number	SP19271	**3376**	Forms	SP20	**3813**	Interface	SP3356
3374	Views	SP20	**4328**	SMS	SP34401	**4276**	SMS	SP11798
3384	Slide library	SP20	**3680**	Music catalog	SP20	3756	Weather	SP18990
3807	Phone notify	SP3356	**3813**	Interface	SP3356	**4328**	SMS	SP34401
4328	SMS	SP34401	**3367**	Permissions	SP20	**3384**	Slide library	SP20
3813	Interface	SP3356	**3253**	Phone number	SP19271	3727	Authentication	SP32369
3376	Forms	SP20	**3807**	Phone notify	SP3356	4124	Recruitment	SP10694
3680	Music catalog	SP20	3365	DWS	SP20	3356	Search	SP20
4276	SMS	SP11798	3727	Authentication	SP32369	3359	Area service	SP20

use the normalized user–service matrix as a basis for recommendation. ED-CDD and WD-CDD were performed as comparative data distribution-based concept drift detection approaches.

To investigate the optimal value of k in the cloud service APIs recommendation, we conducted experiments on the WS-Dream dataset. Fig. 8 presents the relationship between metrics and parameter k for different approaches.

With the gradual increase of k, precision, recall and F-measure generally show a trend of increasing first and then decreasing. When k increase from 20% to 60%, the accuracy metrics (precision, recall, and F-measure) keep stable. When $k = 55\%$, CD-APIR's precision, recall, and F-measure are the best, which are 82.06%, 2.78% and 5.38%, respectively. When the k is 70%, the coverage is best, which is about 31.36%. The accuracy and recommend time of CD-APIR are superior to ED-CDD and WD-CDD.

After the accuracy and coverage are taken into consideration, we believe that when k is 60%, it is suitable for WS-Dream dataset for cloud service APIs recommendation.

6.6. Impact of parameter Top-N

The number of services in the recommender result also has impact on the recommender system. To ensure effectiveness of the recommendation, the more services mean the more possible to satisfy user demands, which can include all the appropriate cloud service APIs comprehensively. However, the purpose of service recommendation is to firstly filter services for users. When the number of cloud service APIs recommended on the homepage is too large, it will not only reduce the user's search efficiency, but also be not conducive to cultivating users' recognition of the cloud service API market.

In this experiment, we investigate the best recommended number by adjusting the parameter Top-N. We set $k = 60\%$ and $\theta = 0.1$. The number of recommended APIs is gradually increased from 2 to 20 with the step size of 1. Fig. 9 presents the impacts of Top-N to the recommendation performance for different approaches.

CD-APIR keeps stable in precision. As N gradually increases, precision tends to increase first and then decrease. When $N = 5$, the precision of CD-APIR reaches a maximum of 82.65%. When $N = 3$, the precision of both ED-CDD and WD-CDD reaches 81.91%, respectively.

As the number of recommended services increases, other metrics increase steadily. When $N = 20$, recall, F-measure and coverage of CD-APIR are the best, which are 5.54%, 10.38% and 29.72%, respectively. After the various metrics are taken into consideration, we believe that $N = 12$ is suitable for cloud service APIs recommendation for CD-APIR.

6.7. Impact of parameter θ

We also investigated the impacts of fixed threshold θ on recommendation performance, that is, the impacts of the relationship between the degree of users' preference drifts and the user–service matrices in different time windows on recommendation performance. In this experiment, we set $k = 60\%$, Top-$N = 12$, and change θ to gradually increase from 0 to 0.5 with a step size of 0.05. When $\theta = 0$, it means all the $D_{JS}(u_{old} \parallel u_{new})$ in the range of $[\theta, 1-\theta]$. Fig. 10 presents the impacts of θ on recommendation performance for different approaches.

We can see that, as θ increases, accuracy and coverage gradually declines. When we focus on the accuracy and coverage of the recommender system, θ should be set as 0 or 0.05. In this way, precision reaches 82.15%, recall reaches 3.34%, F-measure reaches 6.42%, and coverage reaches 23.00%. Coverages of ED-CDD and WD-CDD is about 21.70%. When θ is in the range of [0, 0.15], the metrics remain stable. The reason is that the number of users' $D_{JS}(u_{old} \parallel u_{new})$ in this range is few.

6.8. Case study

To show the exploitability and helpfulness of CD-APIR in the real-world application, we extracted the ratings of a target user (say Alice) from the dataset presented in Section 6.1. CD-APIR and SVD were used for cloud service APIs recommendations, respectively. The results are compared and analyzed in Table 2.

The first three columns of Table 2 present the top 10 cloud service APIs that Alice preferred at the new time window in the dataset. We will compare the results obtained by CD-APIR and SVD with the first three columns of Table 2 to show the effectiveness of CD-APIR. As stated in Section 2, Alice is developing a composite cloud system app for weather forecasting. She preferred to the four basic APIs of s_1, s_2, s_3, and s_4 at the old time window. Then, she would be at the stage of system implementation and prefer APIs of s_5, s_6, s_7, and s_8 at the new time window. That is, Alice's preference may drift in the direction of enhancing the user experience. Her interests to the APIs would change into developing systems' sound, view, SMS and encryption features.

As shown at the 4 to 6 columns of Table 2, CD-APIR recommended the first 8 APIs accurately, including Views, Music Catalog, SMS, Permissions, etc (see the bolded IDs for the APIs). As can be seen from the last three columns of Table 2, 10 APIs were recommended by SVD. The recommendation result of SVD deviated more from Alice's new preferences. It recommended APIs such as weather, search and area service, etc. They are the APIs that Alice preferred in the early stages of development, but they cannot meet Alice's demands at the new time window. It is because SVD solely uses the old time window's model to perform recommendations. CD-APIR approach is combined with preference drifts. It recommends the users' recently preferred APIs. By CD-APIR, the online model adaptation is realized based on the degree of preference drift detection and the trading-off piecewise equation.

7. Conclusion and future work

In this paper, we focus on the challenging issue of users' preference drifts in APIs recommendation, and propose a concept drift-aware temporal cloud service APIs recommendation approach for building composite cloud systems (or CD-APIR). This approach first utilizes the users' behavior-aware information to establish connections between users and cloud service APIs. Then, SVD is utilized to predict the missing values in the user–service matrix. The JS divergence is utilized to detect the degree of concept drift of users preferences. Finally, we combine the results of preference drift detection with user–service matrices in different time windows by proposing a trading-off piecewise equation to generate APIs recommendation results for each user. We conducted comparison experiments based on WS-Dream dataset to verify the effectiveness of the proposed approach. Experimental results demonstrate that CD-APIR approach improves precision, recall, F-measure, coverage of service recommendation and shortens the recommendation time.

In the future, we plan to explore the feasibility of recommending through dividing more time windows by timestamps. We will also investigate different types of drifts and their impacts. Due to new preferences may occur suddenly, gradually, incrementally, and iteratively, we will further research suitable cloud service APIs recommendation approaches in face of different types of preference drifts.

CRediT authorship contribution statement

Lei Wang: Conceptualization, Methodology, Writing - review & editing. **Yunqiu Zhang:** Data curation, Software, Writing - original draft. **Xiaohu Zhu:** Software, Writing - original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors thank Dr. Tao Zhu for his constructive suggestions. This work was partially supported by the Humanity and Social Science Youth Fund of Ministry of Education of China (No. 18YJCZH170), Six talent peaks project in Jiangsu Province in 2019 (No. RJFW-029), Innovative training program for College Students in Jiangsu Province (Nos. 201810298016Z, 2019NFUSPITP0250). L. Wang is the corresponding author.

References

Auch, M., Weber, M., Mandl, P., Wolff, C., 2020. Similarity-based analyses on software applications: A systematic literature review. J. Syst. Softw. 168, 110669.

Ba, Q., Li, X., Bai, Z., 2013. Clustering collaborative filtering recommendation system based on svd algorithm. In: 2013 IEEE 4th International Conference on Software Engineering and Service Science. IEEE, pp. 963–967.

Barddal, J.P., Gomes, H.M., Enembreck, F., Pfahringer, B., 2017. A survey on feature drift adaptation: Definition, benchmark, challenges and future directions. J. Syst. Softw. 127, 278–294.

Batmaz, Z., Yurekli, A., Bilge, A., Kaleli, C., 2019. A review on deep learning for recommender systems: challenges and remedies. Artif. Intell. Rev. 52 (1), 1–37.

Bouguettaya, A., Singh, M., Huhns, M., Sheng, Q.Z., Dong, H., Yu, Q., Neiat, A.G., Mistry, S., Benatallah, B., Medjahed, B., et al., 2017. A service computing manifesto: the next 10 years. Commun. ACM 60 (4), 64–72.

Cao, H., Chen, E., Yang, J., Xiong, H., 2009. Enhancing recommender systems under volatile userinterest drifts. In: Proceedings of the 18th ACM Conference on Information and Knowledge Management, pp. 1257–1266.

Cao, B., Liu, X., Rahman, M., Li, B., Liu, J., Tang, M., 2020. Integrated content and network-based service clustering and web apis recommendation for mashup development. IEEE Trans. Serv. Comput. 13 (1), 99–113.

Chen, C., Wang, Y., Zhang, J., Xiang, Y., Zhou, W., Min, G., 2017. Statistical features-based real-time detection of drifted Twitter spam. IEEE Trans. Inf. Forensics Secur. 12 (4), 914–925.

Cornelis, C., Lu, J., Guo, X., Zhang, G., 2007. One-and-only item recommendation with fuzzy logic techniques. Inform. Sci. 177, 4906–4921.

Dasu, T., Krishnan, S., Venkatasubramanian, S., Yi, K., 2006. An information-theoretic approach to detecting changes in multi-dimensional data streams. In: Proc. Symp. on the Interface of Statistics, Computing Science, and Applications. Citeseer, pp. 1–24.

Dhawan, S., Singh, K., et al., 2015. High rating recent preferences based recommendation system. Procedia Comput. Sci. 70, 259–264.

Ding, S., Li, Y., Wu, D., Zhang, Y., Yang, S., 2018. Time-aware cloud service recommendation using similarity-enhanced collaborative filtering and ARIMA model. Decis. Support Syst. 107, 103–115.

Elahi, M., Ricci, F., Rubens, N., 2016. A survey of active learning in collaborative filtering recommender systems. Comp. Sci. Rev. 20, 29–50.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A., 2014. A survey on concept drift adaptation. ACM Comput. Surv. (CSUR) 46 (4), 1–37.

Gao, T., Cheng, B., Chen, J., Chen, M., 2017. Enhancing collaborative filtering via topic model integrated uniform euclidean distance. China Commun. 14 (11), 48–58.

Goldenberg, I., Webb, G.I., 2019. Survey of distance measures for quantifying concept drift and shift in numeric data. Knowl. Inf. Syst. 60, 591–615.

He, Q., Zhou, R., Zhang, X., Wang, Y., Ye, D., Chen, F., Grundy, J.C., Yang, Y., 2017. Keyword search for building service-based systems. IEEE Trans. Softw. Eng. 43 (7), 658–674.

Hu, J., Liu, L., Zhang, C., He, J., Hu, C., 2018. Hybrid recommendation algorithm based on latent factor model and personalrank. J. Internet Technol. 19 (3), 919–926.

Huang, J., Ding, S., Wang, H., Liu, T., 2018. Learning to recommend related entities with serendipity for web search users. ACM Trans. Asian Low Resour. Lang. Inf. Process. 17 (3), 25:1–25:22.

Koren, Y., 2009. Collaborative filtering with temporal dynamics. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 447–456.

Lemos, A.L., Daniel, F., Benatallah, B., 2016. Web service composition: A survey of techniques and tools. ACM Comput. Surv. 48 (3), 33:1–33:41.

Lin, J., 1991. Divergence measures based on the Shannon entropy. IEEE Trans. Inf. Theory 37 (1), 145–151.

Lin, S., Lai, C., Wu, C., Lo, C., 2014. A trustworthy QoS-based collaborative filtering approach for web service discovery. J. Syst. Softw. 93, 217–228.

Liu, X., 2015. Modeling users' dynamic preference for personalized recommendation. In: Twenty-Fourth International Joint Conference on Artificial Intelligence. pp. 1785–1791.

Liu, A., Song, Y., Zhang, G., Lu, J., 2017. Regional concept drift detection and density synchronized drift adaptation. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence, pp. 2280–2286.

Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., Zhang, G., 2019. Learning under concept drift: A review. IEEE Trans. Knowl. Data Eng. 31 (12), 2346–2363.

Lu, N., Zhang, G., Lu, J., 2014. Concept drift detection via competence models. Artificial Intelligence 209, 11–28.

MacKay, D.J., Mac Kay, D.J., 2003. Information Theory, Inference and Learning Algorithms. Cambridge university press.

Meng, Y., Yan, X., Liu, W., Wu, H., Cheng, J., 2020. Wasserstein collaborative filtering for item cold-start recommendation. In: Proceedings of the 28th ACM Conference on User Modeling, Adaptation and Personalization, pp. 318–322.

Niu, H., Keivanloo, I., Zou, Y., 2017. API usage pattern recommendation for software development. J. Syst. Softw. 129, 127–139.

Qi, L., He, Q., Chen, F., Dou, W., Wan, S., Zhang, X., Xu, X., 2019a. Finding all you need: web APIs recommendation in web of things through keywords search. IEEE Trans. Comput. Soc. Syst. 6 (5), 1063–1072.

Qi, L., Wang, R., Hu, C., Li, S., He, Q., Xu, X., 2019b. Time-aware distributed service recommendation with privacy-preservation. Inform. Sci. 480, 354–364.

Qin, D., Zhou, X., Chen, L., Huang, G., Zhang, Y., 2020. Dynamic connection-based social group recommendation. IEEE Trans. Knowl. Data Eng. 32 (3), 453–467.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., Dennison, D., 2015. Hidden technical debt in machine learning systems. In: Advances in Neural Information Processing Systems. pp. 2503–2511.

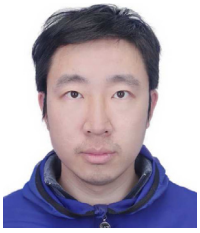
Sebastião, R., Gama, J., 2007. Change detection in learning histograms from data streams. In: Portuguese Conference on Artificial Intelligence. Springer, pp. 112–123.

Somu, N., MR, G.R., Kirthivasan, K., VS, S.S., 2018. A trust centric optimal service ranking approach for cloud service selection. Future Gener. Comput. Syst. 86, 234–252.

Su, K., Xiao, B., Liu, B., Zhang, H., Zhang, Z., 2017. TAP: A personalized trust-aware qos prediction approach for web service recommendation. Knowl.-Based Syst. 115, 55–65.
Wang, L., 2019. Architecture-based reliability-sensitive criticality measure for fault-tolerance cloud applications. IEEE Trans. Parallel Distrib. Syst. 30 (11), 2408–2421.
Wang, S., Ma, Y., Cheng, B., Yang, F., Chang, R.N., 2016. Multi-dimensional QoS prediction for service recommendations. IEEE Trans. Serv. Comput. 12 (1), 47–57.
Wang, H., Yu, C., Wang, L., Yu, Q., 2018. Effective bigdata-space service selection over trust and heterogeneous qos preferences. IEEE Trans. Serv. Comput. 11 (4), 644–657.
Wang, W., Zhou, Z.-H., 2013. Co-training with insufficient views. In: Asian Conference on Machine Learning. pp. 467–482.
Wangwatcharakul, C., Wongthanavasu, S., 2018. Improving dynamic recommender system based on item clustering for preference drifts. In: 2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE). IEEE, pp. 1–6.
Xia, B., Fan, Y., Tan, W., Huang, K., Zhang, J., Wu, C., 2015. Category-aware API clustering and distributed recommendation for automatic mashup creation. IEEE Trans. Serv. Comput. 8 (5), 674–687.
Xu, Y., Yin, J., Deng, S., Xiong, N.N., Huang, J., 2016. Context-aware qos prediction for web service recommendation and selection. Expert Syst. Appl. 53, 75–86.
Yao, L., Wang, X., Sheng, Q.Z., Ruan, W., Zhang, W., 2015. Service recommendation for mashup composition with implicit correlation regularization. In: 2015 IEEE International Conference on Web Services. IEEE, pp. 217–224.
Yin, H., Zhou, X., Cui, B., Wang, H., Zheng, K., Nguyen, Q.V.H., 2016. Adapting to user interest drift for POI recommendation. IEEE Trans. Knowl. Data Eng. 28 (10), 2566–2581.
Yu, Q., Wang, H., Chen, L., 2015. Learning sparse functional factors for large-scale service clustering. In: 2015 IEEE International Conference on Web Services. IEEE, pp. 201–208.
Zafari, F., Moser, I., Baarslag, T., 2019. Modelling and analysis of temporal preference drifts using a component-based factorised latent approach. Expert Syst. Appl. 116, 186–208.
Zenebe, A., Zhou, L., Norcio, A.F., 2010. User preferences discovery using fuzzy models. Fuzzy Sets and Systems 161 (23), 3044–3063.
Zhang, Q., Lu, J., Wu, D., Zhang, G., 2018a. A cross-domain recommender system with kernel-induced knowledge transfer for overlapping entities. IEEE Trans. Neural Netw. Learn. Syst. 30 (7), 1998–2012.
Zhang, Q., Wu, D., Zhang, G., Lu, J., 2016. Fuzzy user-interest drift detection based recommender systems. In: 2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). IEEE, pp. 1274–1281.
Zhang, S., Yao, L., Sun, A., Tay, Y., 2019. Deep learning based recommender system: A survey and new perspectives. ACM Comput. Surv. 52 (1), 1–38.
Zhang, Y.-w., Zhou, Y.-y., Wang, F.-t., Sun, Z., He, Q., 2018b. Service recommendation based on quotient space granularity analysis and covering algorithm on spark. Knowl.-Based Syst. 147, 25–35.
Zheng, Z., Lyu, M.R., 2013. Personalized reliability prediction of web services. ACM Trans. Softw. Eng. Methodol. 22 (2), 12:1–12:25.
Zheng, Z., Ma, H., Lyu, M.R., King, I., 2010a. Qos-aware web service recommendation by collaborative filtering. IEEE Trans. Serv. Comput. 4 (2), 140–152.
Zheng, Z., Zhang, Y., Lyu, M.R., 2010b. Distributed qos evaluation for real-world web services. In: 2010 IEEE International Conference on Web Services. IEEE, pp. 83–90.
Zheng, Z., Zhang, Y., Lyu, M.R., 2014. Investigating QoS of real-world web services. IEEE Trans. Serv. Comput. 7 (1), 32–39.

Lei Wang received the Ph.D. degree in computer science from Southeast University, China. He is an associate professor with Nanjing Forestry University, China. He visited the Department of Computer Science and Engineering, The Chinese University of Hong Kong, during 2019. His research interests mainly include service computing, software reliability engineering, and data mining. His publications have appeared in well-known journals and popular conferences, e.g., the IEEE Transactions on Software Engineering, the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Services Computing, the Journal of Parallel and Distributed Computing, ICSOC, ICWS, SCC, etc.

Yunqiu Zhang is currently pursuing the master's degree with Nanjing Forestry University, P.R. China. Her research interests mainly include cloud service APIs recommendation, concept drift detection, and software reliability engineering.



Xiaohu Zhu received the Master degree in software engineering from Nanjing University, P.R. China. He founded University AI and Center for Safe AGI. His research interests mainly include deep reinforcement learning, game theory and AI safety. He has translated four books on artificial intelligence and deep learning frameworks into Chinese. He is a Google ML Developer Expert (GDE) and Baidu Deep Learning Advocator.