Outlier-Resilient Web Service QoS Prediction

Fanghua Ye
University College London
London, UK
fanghua.ye.19@ucl.ac.uk

Zhiwei Lin
Sun Yat-Sen University
Guangzhou, China
linzhw25@mail2.sysu.edu.cn

Chuan Chen
Sun Yat-Sen University
Guangzhou, China
chenchuan@mail.sysu.edu.cn

Zibin Zheng
Sun Yat-Sen University
Guangzhou, China
zhzibin@mail.sysu.edu.cn

Hong Huang
Huazhong University of Science and Technology, Wuhan, China
honghuang@hust.edu.cn

ABSTRACT

The proliferation of Web services makes it difficult for users to select the most appropriate one among numerous functionally identical or similar service candidates. Quality-of-Service (QoS) describes the non-functional characteristics of Web services, and it has become the key differentiator for service selection. However, users cannot invoke all Web services to obtain the corresponding QoS values due to high time cost and huge resource overhead. Thus, it is essential to predict unknown QoS values. Although various QoS prediction methods have been proposed, few of them have taken outliers into consideration, which may dramatically degrade the prediction performance. To overcome this limitation, we propose an outlier-resilient QoS prediction method in this paper. Our method utilizes Cauchy loss to measure the discrepancy between the observed QoS values and the predicted ones. Owing to the robustness of Cauchy loss, our method is resilient to outliers. We further extend our method to provide time-aware QoS prediction results by taking the temporal information into consideration. Finally, we conduct extensive experiments on both static and dynamic datasets. The results demonstrate that our method is able to achieve better performance than state-of-the-art baseline methods.

CCS CONCEPTS

• **Information systems → Web services**; *Collaborative filtering*.

KEYWORDS

Web service, QoS prediction, outlier resilience, Cauchy loss

ACM Reference Format:
Fanghua Ye, Zhiwei Lin, Chuan Chen, Zibin Zheng, and Hong Huang. 2021. Outlier-Resilient Web Service QoS Prediction. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3442381.3449938

1 INTRODUCTION

Web services provide interoperability among disparate software applications and play a key role in service-oriented computing [3]. Over the past few years, numerous Web services have been published as indicated by the Web service repository–ProgrammableWeb[1]. The proliferation of Web services brings great benefits in building versatile service-oriented applications and systems.

It is apparent that the quality of service-oriented applications and systems relies heavily on the quality of their component Web services. Thus, investigating the quality of Web services is an important task to ensure the reliability of the ultimate applications and the entire systems. The quality of Web services can be characterized by their functional and non-functional attributes. Quality-of-Service (QoS) represents the non-functional aspect of Web services, such as response time, throughput rate and failure probability [37, 47]. Since there are many functionally equivalent or similar services offered on the Web, investigating non-functional QoS properties becomes the major concern for service selection [12, 62]. However, the QoS value observed by users depends heavily on the Web service invocation context. Hence, the quality of the same Web service experienced by different users may be relatively different [42]. For this reason, it is important to acquire personalized QoS values for different users. Considering that users cannot invoke all Web services to obtain personalized QoS values on their own due to high time cost and huge resource overhead [47, 61], predicting missing QoS values based on existing observations plays an essential role in obtaining approximate personalized QoS values.

Matrix factorization (MF) is arguably the most popular technique adopted for QoS prediction [16, 61, 68]. However, most existing MF-based QoS prediction methods directly utilize L_2-norm to measure the difference between the observed QoS values and the predicted ones [31, 44, 48, 50, 53, 57, 66]. It is well-known that L_2-norm is sensitive to outliers [8, 58, 63, 70]. That is, the objective function value may be dominated by outliers during the L_2-norm minimization process, which will lead to severe approximation deviation between the observed normal values and the predicted ones. As a result, without taking outliers into consideration, existing MF-based methods may not achieve satisfactory performance. In recent years, there are some explorations on enhancing the robustness of MF-based QoS prediction methods by replacing L_2-norm with L_1-norm [71]. Although L_1-norm is more robust to outliers [13, 35, 64], L_1-norm-based objective function is much harder to optimize and the solution is also unstable [34, 56]. Moreover, L_1-norm is still sensitive to outliers, especially when outliers take significantly different values from the normal ones [10, 55]. There are also some

[1]https://www.programmableweb.com/

methods seeking to identify outliers explicitly by means of clustering algorithms [18, 49, 72], which usually treat all the elements in the smallest cluster as outliers [19, 47]. However, it is difficult to choose the proper number of clusters. Consequently, these methods usually suffer from the misclassification issue. That is, either some outliers may not be eliminated successfully or some normal values may be selected as outliers falsely.

In this paper, we propose a robust QoS prediction method under the matrix factorization framework to deal with the aforementioned issues. Our method chooses to measure the discrepancy between the observed QoS values and the predicted ones by Cauchy loss [2, 27] instead of the L_1-norm loss or L_2-norm loss. It has been shown that Cauchy loss is much more robust to outliers than the L_1-norm loss and L_2-norm loss [27, 55]. Theoretically, Cauchy loss allows nearly half of the observations to be out of the normal range before it gives incorrect results [36]. For a given QoS dataset, it is unlikely that nearly half of the observations are outliers. Thus, Cauchy loss is sufficient for outlier modeling and has the potential to provide better prediction results. Note also that our method does not explicitly identify outliers, which reduces the risk of misclassification and makes our method more general and more robust. In other words, our method is resilient to outliers. Considering that the QoS value of a Web service observed by a particular user may change over time, it is essential to provide time-aware personalized QoS prediction results. To achieve this goal, we further extend our method under the tensor factorization framework by taking the temporal information into consideration.

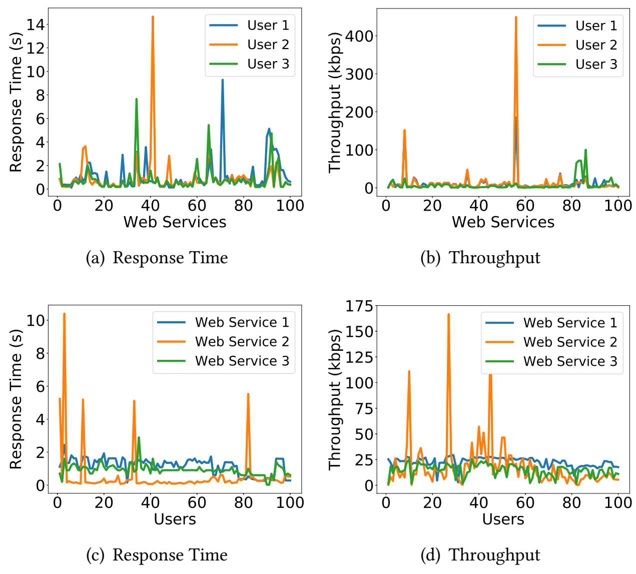
In summary, the main contributions of this paper include:

- First, we propose a robust Web service QoS prediction method with outlier resilience. Our method measures the discrepancy between the observed QoS values and the predicted ones by Cauchy loss, which is robust to outliers.
- Second, we extend our method to provide time-aware QoS prediction results under the tensor factorization framework by taking the temporal information into consideration.
- Third, we conduct extensive experiments on both static and dynamic datasets to evaluate the performance of our method. The results demonstrate that our method can achieve better performance than state-of-the-art baseline methods.

The rest of this paper is organized as follows. Section 2 explores the unavoidability of outliers in QoS observations. Section 3 provides an overview on how matrix factorization can be employed for QoS prediction. Section 4 presents the detailed descriptions of our method and its extension for time-aware QoS prediction. Section 5 reports the experimental results. Section 6 gives a brief review of the related work. This paper is finally concluded in Section 7.

2 UNAVOIDABILITY OF QOS OUTLIERS

Most existing QoS prediction methods assume that the QoS observations are reliable and rational. However, this assumption may not hold in the real world. This is because the observed QoS data can be affected by many factors. For example, there may be some malicious users submitting wrong QoS values deliberately. The service providers may also pretend to be service users and thus exaggerate the performance of their own Web services and depreciate the performance of their competitors' Web services. In addition, the QoS



(a) Response Time (b) Throughput

(c) Response Time (d) Throughput

Figure 1: The distribution of response time and throughput of randomly selected Web services observed by different users from the publicly available dataset WS-DREAM.

values observed by users are largely dependent on the invocation environment such as network latency and server overload, which may lead some of the QoS values to deviate far from the normal range. In consideration of these complicated factors, we argue that it is highly possible that some of the QoS observations are outliers.

However, we are in lack of an oracle showing in advance which QoS observations are outliers. Here, we treat the rare extreme values which significantly differ from the remaining ones as outliers by following the definition in [24]. To be more intuitive, in Figure 1 (a) and (b), we show the distribution of both response time and throughput of 100 Web services invoked by 3 randomly selected users from a publicly available dataset–WS-DREAM_dataset1[2]. As can be seen, although a user tends to have different usage experiences on different Web services, most QoS values of these Web services observed by the three users fall into a normal range. For example, the response time mainly falls in the interval of $[0, 2]$. However, there are also some observations deviating far from the normal range. As shown in Figure 1 (a), the response time experienced by user 2 even reaches up to 14 seconds, which is far beyond the normal range. Needless to say, such kind of observations should be treated as outliers. We further demonstrate the distribution of response time and throughput of 3 Web services experienced by 100 different users in Figure 1 (c) and (d). It can be observed that although the usage experiences of a Web service can vary widely among different users, the QoS values of the same Web service observed by the majority of users tend to fall into a normal range. Whereas, there are also some observations taking values far beyond the normal range. These phenomena verify the rationality of treating extreme values as outliers and also reveal the unavoidability of outliers in QoS observations.

[2]https://github.com/wsdream/wsdream-dataset/tree/master/dataset1

3 PRELIMINARIES

Suppose that we are provided with a set of m users and a set of n Web services, then the QoS values between all users and Web services can be represented by a matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ whose entry $\boldsymbol{X}_{ij}$ denotes the QoS value of Web service j observed by user i. Obviously, it is time-consuming and resource-consuming for each user to invoke all Web services to get the personalized QoS values. As a consequence, we typically have only partial observations between users and Web services, which means that lots of entries in $\boldsymbol{X}$ are null. The goal of QoS prediction is to predict these null entries by exploiting the information contained in existing observations.

3.1 Problem Definition

Let Ω denote the set of existing QoS observations, that is,

$$\Omega = \{(i, j, \boldsymbol{X}_{ij}) \mid \text{the QoS value } \boldsymbol{X}_{ij} \text{ between user } i \text{ and Web service } j \text{ has been observed}\}. \tag{1}$$

Then the problem of QoS prediction is defined as follows.

Problem Statement: Given a set of QoS observations Ω, QoS prediction aims at predicting unknown QoS values by utilizing the information contained in Ω.

3.2 Matrix Factorization for QoS Prediction

Generally speaking, matrix factorization tries to factorize a given matrix into the product of several low-rank factor matrices. In the context of QoS prediction, the basic framework of MF-based methods is to factorize matrix $\boldsymbol{X}$ into two low-rank factor matrices $\boldsymbol{U} \in \mathbb{R}^{m \times l}$ and $\boldsymbol{S} \in \mathbb{R}^{n \times l}$, i.e., $\boldsymbol{X} \approx \boldsymbol{U}\boldsymbol{S}^T$. Here, each row of $\boldsymbol{U}$ represents the latent feature of a user, and each row of $\boldsymbol{S}$ represents the latent feature of a Web service. The dimensionality of latent features is controlled by a parameter l ($l \ll \min(m, n)$). Apparently, $\boldsymbol{U}\boldsymbol{S}^T$ should be as close to $\boldsymbol{X}$ as possible. As thus, the general objective function for MF-based QoS prediction methods can be derived as:

$$\min_{\boldsymbol{U},\boldsymbol{S}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{U}\boldsymbol{S}^T) + \lambda \mathcal{L}_{reg}, \tag{2}$$

where $\mathcal{L}$ measures the degree of approximation between $\boldsymbol{U}\boldsymbol{S}^T$ and $\boldsymbol{X}$, $\mathcal{L}_{reg}$ denotes the regularization term to avoid over-fitting, and λ represents the regularization coefficient.

The most widely adopted loss function in matrix factorization is the least square loss (i.e., L_2-norm loss), which is also the most commonly used loss function in MF-based QoS prediction methods [31, 57, 61, 66, 71]. In this setting, the specific objective function can be clearly given as follows:

$$\min_{\boldsymbol{U},\boldsymbol{S}} \frac{1}{2} \|\boldsymbol{I} \odot (\boldsymbol{X} - \boldsymbol{U}\boldsymbol{S}^T)\|_2^2 + \lambda \mathcal{L}_{reg}, \tag{3}$$

where $\|\cdot\|_2$ denotes the L_2-norm which is calculated as the square root of the sum of squares of all entries, $\odot$ denotes the Hadamard product (i.e., entry-wise product), and $\boldsymbol{I} \in \mathbb{R}^{m \times n}$ denotes the indicator matrix whose entry $\boldsymbol{I}_{ij}$ indicates whether the QoS value of Web service j has been observed by user i or not. If user i has the record of Web service j, $\boldsymbol{I}_{ij}$ is set to 1; otherwise, it is set to 0.

The objective function based on L_2-norm as in Eq. (3) is smooth and can be optimized by the gradient descent method [15]. However, the L_2-norm is sensitive to outliers (i.e., rare extreme values) [63]. When the given observations contain outliers, the residuals between these outliers' corresponding entries in $\boldsymbol{X}$ and their approximation entries in $\boldsymbol{U}\boldsymbol{S}^T$ become huge due to the square operation. Therefore, when minimizing the objective function in Eq. (3), more priorities are given to these outliers, which unfortunately causes severe approximation deviation of the normal QoS values. As a result, the QoS prediction performance may degrade dramatically.

To make the model more robust to outliers, a common stategy is to replace L_2-norm with L_1-norm [13, 23, 51, 71]. Based on L_1-norm, the objective function is formularized as below:

$$\min_{\boldsymbol{U},\boldsymbol{S}} \|\boldsymbol{I} \odot (\boldsymbol{X} - \boldsymbol{U}\boldsymbol{S}^T)\|_1 + \lambda \mathcal{L}_{reg}, \tag{4}$$

where $\|\cdot\|_1$ denotes the L_1-norm which is calculated as the sum of the absolute values of all entries. Although L_1-norm is to some extent more robust to outliers than L_2-norm, the objective function based on L_1-norm as in Eq. (4) is a non-smooth function and it is much harder to optimize. What's more, although the large residuals due to outliers are not squared in L_1-norm, they may still be quite large relative to the normal ones and thus one would expect that they would influence the objective function as well [10].

4 OUR METHOD

As stated in the previous section, both L_1-norm and L_2-norm are sensitive to outliers. In order to make the MF-based methods more robust to outliers, we propose a novel QoS prediction method that utilizes Cauchy loss [2] as the measurement of the discrepancy between the observed QoS values and the predicted ones. It has been shown that Cauchy loss is resistant to outliers [17, 36, 55]. Thus, our method is expected to be robust to outliers.

4.1 M-Estimator

Before presenting the details of our method, we first introduce the concept of M-estimator. In robust statistics, M-estimators are a broad class of estimators, which represent the minima of particular loss functions [21]. Let r_i denote the residual of the i-th datum, i.e., the difference between the i-th observation and its approximation. Then M-estimators try to optimize the following objective function:

$$\min \sum_i g(r_i), \tag{5}$$

where function g gives the contribution of each residual to the objective function. A reasonable function g should satisfy the following four properties [14]:

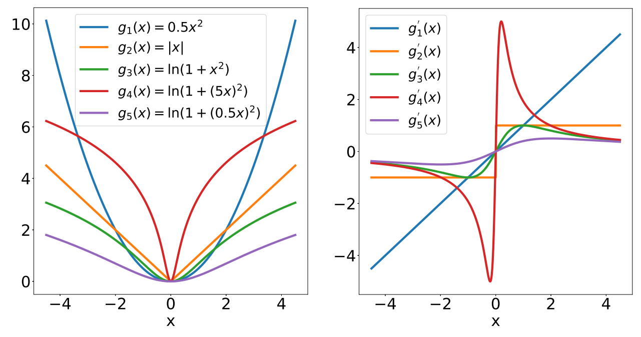
- $g(x) \geq 0, \forall x$;
- $g(x) = g(-x), \forall x$;
- $g(0) = 0$;
- $g(x)$ is non-decreasing in $|x|$, i.e., $g(x_1) \leq g(x_2), \forall |x_1| < |x_2|$.

The influence function of g is defined as its first-order derivative:

$$g'(x) = \frac{\mathrm{d}g(x)}{\mathrm{d}x}. \tag{6}$$

The influence function g' measures the influence of each datum on the value of the parameter estimate. For a robust M-estimator, it would be inferred that the influence of any single datum is insufficient to yield any significant offset [55]. Ideally, a robust M-estimator should have a bounded influence function.

Both L_2-norm loss and L_1-norm loss satisfy the four properties required by M-estimators. For the L_2 estimator with $g(x) = \frac{1}{2}x^2$,



(a) The Loss Function (b) The Influence Function

Figure 2: Comparison of different M-estimators.

the influence function is $g'(x) = x$, which means that the influence of a datum on the parameter estimate grows linearly as the error increases. This confirms the non-robusteness of L_2 estimator to outliers. Although the L_1 estimator with $g(x) = |x|$ can reduce the influence of large errors due to its bounded influence function, it will still be affected by outliers since its influence function has no cut off point [27, 55] ($|g'(x)| = 1$ even when $x \longrightarrow \pm\infty$). Besides, L_1 estimator is not stable because $g(x) = |x|$ is not strictly convex in x. It follows that the influence function of a robust M-estimator should not only be bounded but also be insensitive to the increase of errors ($|g'(x)| \longrightarrow 0$ when $x \longrightarrow \pm\infty$). Cauchy estimator has been shown to possess such precious characteristics. The g function of Cauchy estimator (i.e., Cauchy loss) is shown as follows:

$$g(x) = \ln\left(1 + \frac{x^2}{\gamma^2}\right), \tag{7}$$

where γ is a constant. The influence function is then calculated as:

$$g'(x) = \frac{2x}{\gamma^2 + x^2}, \tag{8}$$

which takes value in the range of $[-\frac{1}{\gamma}, \frac{1}{\gamma}]$. Moreover, $g'(x)$ tends to zero when x goes to infinity. This indicates that the influence function of Cauchy estimator is insensitive to the increase of errors. Therefore, Cauchy estimator is robust to outliers. A comparison of different M-estimators is illustrated in Figure 2.

4.2 Model Formulation

In view of Cauchy estimator's robustness, we choose Cauchy loss to construct the objective function of our method. Based on Cauchy loss, the objective function is derived as:

$$\min_{\boldsymbol{U},\boldsymbol{S}} \mathcal{L} = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n} \boldsymbol{I}_{ij} \ln\left(1 + \frac{(\boldsymbol{X}_{ij} - \boldsymbol{U}_i\boldsymbol{S}_j^T)^2}{\gamma^2}\right) + \frac{\lambda_u}{2}\|\boldsymbol{U}\|_2^2 + \frac{\lambda_s}{2}\|\boldsymbol{S}\|_2^2, \tag{9}$$

where $\boldsymbol{U}_i$ and $\boldsymbol{S}_j$ denote the i-th row of $\boldsymbol{U}$ and the j-th row of $\boldsymbol{S}$ respectively, λ_u and λ_s represent the regularization coefficients.

The objective function in Eq. (9) can be efficiently optimized by the gradient descent method [15]. Specifically, we choose to optimize $\boldsymbol{U}$ and $\boldsymbol{S}$ row by row. Then, we have the following update rules:

$$\boldsymbol{U}_i \leftarrow \boldsymbol{U}_i - \eta_u \frac{\partial \mathcal{L}}{\partial \boldsymbol{U}_i}, \tag{10}$$

$$\boldsymbol{S}_j \leftarrow \boldsymbol{S}_j - \eta_s \frac{\partial \mathcal{L}}{\partial \boldsymbol{S}_j}, \tag{11}$$

where η_u and η_s denote the learning rates for $\boldsymbol{U}$ and $\boldsymbol{S}$, and

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{U}_i} = \lambda_u \boldsymbol{U}_i - \sum_{j=1}^{n} \boldsymbol{I}_{ij} \frac{\boldsymbol{X}_{ij} - \boldsymbol{U}_i\boldsymbol{S}_j^T}{\gamma^2 + (\boldsymbol{X}_{ij} - \boldsymbol{U}_i\boldsymbol{S}_j^T)^2}\boldsymbol{S}_j, \tag{12}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{S}_j} = \lambda_s \boldsymbol{S}_j - \sum_{i=1}^{m} \boldsymbol{I}_{ij} \frac{\boldsymbol{X}_{ij} - \boldsymbol{U}_i\boldsymbol{S}_j^T}{\gamma^2 + (\boldsymbol{X}_{ij} - \boldsymbol{U}_i\boldsymbol{S}_j^T)^2}\boldsymbol{U}_i. \tag{13}$$

Algorithm 1 Algorithm for Static QoS Prediction

```
Input: $X \in \mathbb{R}^{m\times n}$, $l, \gamma, \lambda_u, \lambda_s, \eta_u, \eta_s$;
Output: $U \in \mathbb{R}^{m\times l}$, $S \in \mathbb{R}^{n\times l}$;
1: Randomly initialize $U$ and $S$;
2: repeat
3:     for $i = 1$ to $m$ do
4:         Update $U_i$ according to Eq. (10);
5:     for $j = 1$ to $n$ do
6:         Update $S_j$ according to Eq. (11);
7: until Convergence
8: return $U, S$;
```

The overall optimization procedure of our method is presented in Algorithm 1, whose time complexity is shown in Theorem 1.

Theorem 1. *Let r denote the number of iterations for Algorithm 1 to achieve convergence and let ρ denote the number of available entries in $\boldsymbol{X}$, then the time complexity of Algorithm 1 is $O(r\rho l)$.*

Proof. The main time cost of Algorithm 1 lies in the updates of $\boldsymbol{U}$ and $\boldsymbol{S}$. In each iteration, updating $\boldsymbol{U}$ takes $O(ml + \rho l)$ time and updating $\boldsymbol{S}$ takes $O(nl + \rho l)$ time. Since both m and n are less than ρ, the time complexity of updating $\boldsymbol{U}$ and $\boldsymbol{S}$ can both be simplified as $O(\rho l)$. Thus, the overall time complexity is of order $O(r\rho l)$. □

4.3 Extension for Time-Aware QoS Prediction

As pointed out in [62], the QoS performance of Web services is highly related to the invocation time because the service status (e.g., number of users) and the network environment (e.g., network speed) may change over time. Thus, it is essential to provide time-aware personalized QoS information to help users make service selection at runtime [69]. In this part, we aim to extend our method to make it suitable for time-aware personalized QoS prediction.

To achieve the goal, we choose to extend our method under the tensor factorization framework. A tensor is a multidimensional or N-way array [25]. An N-way tensor is denoted as $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$, which has N indices $(i_1, i_2, \cdots, i_N)$ and its entries are denoted by $\boldsymbol{\mathcal{X}}_{i_1 i_2\cdots i_N}$. In this sense, a tensor can be treated as a generalized matrix and a matrix can also be treated as a two-way tensor.

For time-aware QoS prediction, we need to take the temporal information of QoS values into consideration. According to the definition of tensors, it is clear that we can model QoS observations with temporal information as a three-way tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{m\times n\times t}$

Algorithm 2 Algorithm for Time-Aware QoS Prediction

```
Input: $\mathcal{X} \in \mathbb{R}^{m\times n\times t}$, $l$, $\gamma$, $\lambda_u$, $\lambda_s$, $\lambda_t$;
Output: $U \in \mathbb{R}^{m\times l}$, $S \in \mathbb{R}^{n\times l}$, $T \in \mathbb{R}^{t\times l}$;
1: Randomly initialize $U \geq 0$, $S \geq 0$ and $T \geq 0$;
2: repeat
3:     for $i = 1$ to $m$ do
4:         Update $U_i$ according to Eq. (17);
5:     for $j = 1$ to $n$ do
6:         Update $S_j$ according to Eq. (18);
7:     for $k = 1$ to $t$ do
8:         Update $T_k$ according to Eq. (19);
9: until Convergence
10: return $U, S, T$;
```

whose each entry $\mathcal{X}_{ijk}$ represents the QoS value of Web service j observed by user i at time k. Here, t denotes the total number of time intervals. Accordingly, we can use an indicator tensor $\mathcal{I} \in \mathbb{R}^{m\times n\times t}$ to show whether the QoS values have been observed or not. If user i has the record of Web service j at time k, $\mathcal{I}_{ijk}$ is set to 1; otherwise, its value is set to 0. To predict the unknown QoS values in $\mathcal{X}$, similar to MF-based methods, we first factorize $\mathcal{X}$ to learn the latent features of users, Web services and contexts, respectively, and then leverage the interaction among them to predict QoS values. Specifically, we adopt the canonical polyadic (CP) decomposition method [38] to factorize $\mathcal{X}$ into three low-rank factor matrices $U \in \mathbb{R}^{m\times l}$, $S \in \mathbb{R}^{n\times l}$ and $T \in \mathbb{R}^{t\times l}$. Then, $\mathcal{X}$ is approximated in the following way:

$$\mathcal{X} \approx \hat{\mathcal{X}} = \sum_{\ell=1}^{l} U^{(\ell)} \circ S^{(\ell)} \circ T^{(\ell)}, \tag{14}$$

where $U^{(\ell)} \in \mathbb{R}^m$, $S^{(\ell)} \in \mathbb{R}^n$ and $T^{(\ell)} \in \mathbb{R}^t$ denote the ℓ-th column of U, S and T respectively, and $\circ$ represents the vector outer product. In this way, each entry $\mathcal{X}_{ijk}$ is approximated by:

$$\mathcal{X}_{ijk} \approx \hat{\mathcal{X}}_{ijk} = \sum_{\ell=1}^{l} U_{i\ell} S_{j\ell} T_{k\ell}. \tag{15}$$

For tensor factorization, nonnegative constraints are usually enforced on the factor matrices to promote the model's interpretability [43]. We also add nonnegative constraints to all the factor matrices U, S and T. Then together with the Cauchy loss, we derive the objective function for time-aware QoS prediction as below:

$$\begin{aligned} \min_{U,S,T} \mathcal{L}' = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{t} \mathcal{I}_{ijk} \ln\left(1 + \frac{(\mathcal{X}_{ijk} - \hat{\mathcal{X}}_{ijk})^2}{\gamma^2}\right) \\ + \frac{\lambda_u}{2}\|U\|_2^2 + \frac{\lambda_s}{2}\|S\|_2^2 + \frac{\lambda_t}{2}\|T\|_2^2, \\ s.t.\ U \geq 0, S \geq 0, T \geq 0, \end{aligned} \tag{16}$$

where λ_t denotes the regularization coefficient for matrix T.

Due to the nonnegative constraints, we cannot adopt the gradient descent method to optimize the objective function in Eq. (16) any more. Alternatively, we use the multiplicative updating (MU) algorithm [26] to solve Eq. (16). To be more specific, MU alternately updates U, S and T with the other two being fixed in each iteration. Although the objective function in Eq. (16) is nonconvex over U, S and T simultaneously, it is a convex function in each variable when the other two are fixed. Thus we can derive a closed-form update rule for each variable under the Karush-Kuhn-Tucker (KKT) conditions [5]. The detailed update rules are listed as follows:

$$U_i \leftarrow U_i \odot \frac{\sum_{j=1}^{n}\sum_{k=1}^{t} \mathcal{I}_{ijk}\Delta_{ijk}\mathcal{X}_{ijk}(S_j \odot T_k)}{\sum_{j=1}^{n}\sum_{k=1}^{t} \mathcal{I}_{ijk}\Delta_{ijk}\hat{\mathcal{X}}_{ijk}(S_j \odot T_k) + \lambda_u U_i}, \tag{17}$$

$$S_j \leftarrow S_j \odot \frac{\sum_{i=1}^{m}\sum_{k=1}^{t} \mathcal{I}_{ijk}\Delta_{ijk}\mathcal{X}_{ijk}(U_i \odot T_k)}{\sum_{i=1}^{m}\sum_{k=1}^{t} \mathcal{I}_{ijk}\Delta_{ijk}\hat{\mathcal{X}}_{ijk}(U_i \odot T_k) + \lambda_s S_j}, \tag{18}$$

$$T_k \leftarrow T_k \odot \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} \mathcal{I}_{ijk}\Delta_{ijk}\mathcal{X}_{ijk}(U_i \odot S_j)}{\sum_{i=1}^{m}\sum_{j=1}^{n} \mathcal{I}_{ijk}\Delta_{ijk}\hat{\mathcal{X}}_{ijk}(U_i \odot S_j) + \lambda_t T_k}. \tag{19}$$

In the above equations, Δ_{ijk} is defined as $\Delta_{ijk} = \frac{1}{\gamma^2 + (\mathcal{X}_{ijk} - \hat{\mathcal{X}}_{ijk})^2}$.

Algorithm 2 summarizes the overall optimization procedure of our time-aware QoS prediction method. Since Algorithm 2 updates U, S and T alternately and each update decreases the objective function value monotonically, it is guaranteed to converge to a local minimal solution. The time complexity of Algorithm 2 is shown in Theorem 2.

Theorem 2. *Let r' denote the number of iterations for Algorithm 2 to achieve convergence and let ρ' denote the number of available entries in $\mathcal{X}$, then the time complexity of Algorithm 2 is $O(r'\rho' l)$.*

Proof. The proof is similar to that of Theorem 1. In each iteration, it takes $O(\rho' l)$ time to update U, S and T. Therefore, the total time complexity of Algorithm 2 is of order $O(r'\rho' l)$. □

It is worth mentioning that in both Algorithm 1 and Algorithm 2, we do not detect outliers explicitly. Thus our method will not suffer from the problem of misclassification, which indicates that our method is more resilient and more robust to outliers.

5 EXPERIMENTS

In this section, we conduct a set of experiments on both static QoS prediction and time-aware QoS prediction to evaluate the efficiency and effectiveness of our method by comparing it with several state-of-the-art QoS prediction methods. We implement our method and all baseline methods in Python 3.7. And all the experiments are conducted on a server with two 2.4GHz Intel Xeon CPUs and 128GB main memory running Ubuntu 14.04.5 (64-bit)[3].

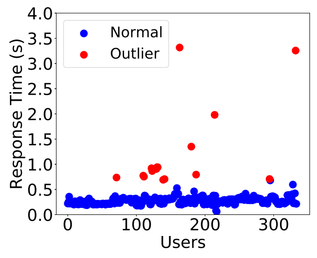
5.1 Datasets

We conduct all experiments on a publicly available dataset collection–WS-DREAM[4], which was collected from real-world Web services. WS-DREAM contains both static and dynamic QoS datasets. The static dataset describes real-world QoS measurements, including both response time and throughput values, obtained from 339 users on 5825 Web services. The dynamic dataset describes real-world QoS measurements from 142 users on 4500 Web services over 64 consecutive time slices (at 15-minute interval). The dynamic dataset also includes records of both response time and throughput values. The statistics of the datasets are presented in Table 1.

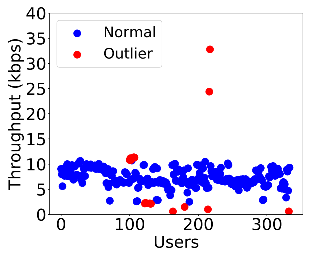
[3]The source code is available at https://github.com/smartyfh/CMF-CTF
[4]https://github.com/wsdream/wsdream-dataset

Table 1: Statistics of QoS Data

Type	QoS Attributes	#User	#Service	#Time	Range	Mean
Static	Response Time (s)	339	5825	-	0-20	0.9086
	Throughput (kbps)	339	5825	-	0-1000	47.5617
Dynamic	Response Time (s)	142	4500	64	0-20	3.1773
	Throughput (kbps)	142	4500	64	0-6727	11.3449



(a) Response Time



(b) Throughput

Figure 3: Exemplary outlier detection results by *i*Forest.

5.2 Evaluation Metrics

The most commonly used evaluation metrics for QoS prediction include mean absolute error (MAE) [67] and root mean square error (RMSE) [67]. Let Π denote the set of QoS values to be predicted (i.e., Π is the testing set) and let $N = |\Pi|$, then MAE is calculated as:

$$MAE = \frac{\sum_{q\in\Pi} |q - \hat{q}|}{N}, \tag{20}$$

and RMSE is calculated as:

$$RMSE = \sqrt{\frac{\sum_{q\in\Pi} (q - \hat{q})^2}{N}}, \tag{21}$$

where $\hat{q}$ denotes the predicted value for the observation q. For both MAE and RMSE, smaller values indicate better performance.

However, according to the definition of MAE and RMSE, we can see that both MAE and RMSE are sensitive to outliers, which means that if Π contains outliers, then MAE and RMSE cannot truly reflect the QoS prediction performance. For example, suppose that $q_* \in \Pi$ is an outlier, then in order to get a small MAE value and RMSE value, the predicted $\hat{q}_*$ should be close to q_* rather than the normal QoS value. As thus, a smaller MAE or RMSE value may not really indicate better performance. To overcome this limitation, we eliminate outliers from Π when calculating MAE and RMSE. Note that we do not have groundtruth labels for outliers. Therefore, we need to detect outliers from scratch. To achieve this goal, we employ the *i*Forest (short for isolation forest) method [29, 30] for outlier detection. *i*Forest detects outliers purely based on the concept of isolation without employing any distance or density measure, making *i*Forest quite efficient and robust. *i*Forest will calculate an outlier score for each datum. The score takes value in the range of $[0, 1]$ and a larger value indicates more possibility to be outliers. Based on the outlier score, we can set the number of outliers flexibly. To intuitively show the effectiveness of *i*Forest, we report the outlier detection results of a randomly selected Web service from the static dataset in Figure 3, where the outlier ratio is set to 0.05. As can be seen, *i*Forest demonstrates good performance in outlier detection. It can detect both overinflated and underinflated outliers.

5.3 Baseline Methods

For ease of presentation, we name our method for static QoS prediction as **CMF** and our method for time-aware QoS prediction as **CTF** hereafter. For static QoS prediction, we compare CMF with the following five methods:

- **MF$_2$**: MF$_2$ denotes the basic MF-based QoS prediction method [66] and it measures the discrepancy between the observed QoS values and the predicted ones by L_2-norm.
- **MF$_1$**: MF$_1$ is also an MF-based QoS prediction method [71]. However, it utilizes the L_1-norm loss to construct the objective function. MF$_1$ is expected to be more robust to outliers. Note that we implement MF$_1$ a little differently from the original one proposed in [71]. In our implementation, we ignore the privacy and location information.
- **CAP**: CAP is a credibility-aware QoS prediction method [47]. It first employs a two-phase k-means clustering algorithm to identify untrustworthy users (i.e., outliers), and then predicts unknown QoS values based on the clustering information contributed by trustworthy users.
- **TAP**: TAP is a trust-aware QoS prediction method [45]. It aims to provide reliable QoS prediction results via calculating the reputation of users by a beta reputation system, and it identifies outliers based on k-means clustering as well.
- **DALF**: DALF is a data-aware latent factor model for QoS prediction [49]. It utilizes the density peaks based clustering algorithm [40] to detect unreliable QoS data directly.

For time-aware QoS prediction, we compare our CTF with the following five methods:

- **NNCP**: NNCP is a tensor-based time-aware QoS prediction method [60]. It is based on CP decomposition and imposes nonnegative constraints on all the factor matrices.
- **BNLFT**: BNLFT is a biased nonnegative tensor factorization model [33]. It incorporates linear biases into the model for describing QoS fluctuations, and it adds nonnegative constraints to the factor matrices as well.
- **WLRTF**: WLRTF is an MLE (maximum likelihood estimation) based tensor factorization method [9]. It models the noise of each datum as a mixture of Gaussian (MoG).
- **PLMF**: PLMF is an LSTM (long short-term memory) [20] based QoS prediction method [54]. PLMF can capture the dynamic latent representations of users and Web services.
- **TASR**: TASR is a time-aware QoS prediction method [11]. It integrates similarity-enhanced collaborative filtering model and the ARIMA model (a time series analysis model) [4].

Although MF$_1$, CAP, TAP and DALF are able to deal with outliers to some extent for static QoS prediction, to our best knowledge, our method CTF is the first to take outliers into consideration for time-aware QoS prediction. It is also worth emphasizing that our method and all baseline methods (except CAP, TAP and DALF) will not explicitly detect outliers when learning the prediction model. The reason for detecting outliers during the testing phase is to make MAE and RMSE be able to truly reflect the QoS prediction performance. For all methods, outliers will be removed when calculating MAE and RMSE. In addition, in the experiments, we run each method 10 times and report the average results for fair comparison.

Table 2: Performance Comparison with Different Training Ratios on Static Dataset (Best Results in Bold Numbers)

QoS Attributes	Methods	MAE						RMSE					
		10%	20%	30%	70%	80%	90%	10%	20%	30%	70%	80%	90%
Response Time	MF_2	0.5334	0.4103	0.3534	0.3044	0.2921	0.2848	0.8407	0.6978	0.6195	0.5742	0.5565	0.5438
	MF_1	0.4041	0.4037	0.4036	0.2815	0.2786	0.2798	0.6120	0.6106	0.6103	0.5590	0.5544	0.5505
	CAP	0.3603	0.3521	0.3312	0.2282	0.2112	0.1881	0.6439	0.6640	0.6789	0.5815	0.5987	0.5821
	TAP	0.3385	0.2843	0.2449	0.2477	0.2812	0.3189	0.5512	0.4985	0.4589	0.4687	0.5155	0.5665
	DALF	0.3955	0.3439	0.3081	0.2496	0.2492	0.2397	0.7466	0.6779	0.5974	0.5471	0.5403	0.5388
	CMF	**0.1762**	**0.1524**	**0.1408**	**0.1153**	**0.1102**	**0.1085**	**0.3705**	**0.3599**	**0.3504**	**0.3106**	**0.2877**	**0.2699**
Throughput	MF_2	13.9730	12.3750	10.7753	7.8371	7.8255	7.8071	28.9608	26.8906	24.6608	19.6406	19.2831	18.6451
	MF_1	16.5509	13.1105	10.7200	7.5736	7.3263	7.1115	33.8889	27.9648	23.6611	18.1316	17.9458	17.3698
	CAP	16.4269	16.3125	16.1946	9.7147	8.6984	7.8516	32.9558	32.9334	32.9540	23.7955	22.2425	21.3711
	TAP	22.1419	19.8273	17.8388	14.5786	14.8380	15.4028	43.4987	40.9533	38.8371	33.3052	32.4076	32.0935
	DALF	13.1968	11.9619	10.6882	7.8156	7.7902	7.7771	27.8531	26.0299	24.4506	19.3523	18.9886	18.2965
	CMF	**8.4573**	**7.2501**	**6.4300**	**5.1865**	**5.1241**	**5.0078**	**24.9137**	**20.8927**	**18.8985**	**17.2916**	**17.1433**	**16.9388**

Table 3: Performance Comparison with Different Outlier Ratios on Static Dataset (Best Results in Bold Numbers)

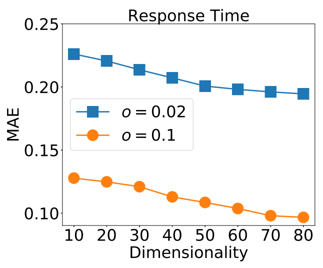
QoS Attributes	Methods	MAE						RMSE					
		2%	4%	6%	8%	10%	20%	2%	4%	6%	8%	10%	20%
Response Time	MF_2	0.4080	0.3732	0.3533	0.3445	0.3306	0.3072	0.8040	0.7210	0.6711	0.6508	0.6021	0.5810
	MF_1	0.3761	0.3390	0.3185	0.2972	0.2702	0.2525	0.7935	0.6903	0.6398	0.5918	0.5575	0.3744
	CAP	0.4163	0.3657	0.3311	0.2997	0.2739	0.2413	0.9789	0.8375	0.7616	0.6817	0.6191	0.5258
	TAP	0.4562	0.3788	0.3268	0.2703	0.2183	0.1649	1.1536	0.9393	0.8148	0.6475	0.4294	0.2478
	DALF	0.3622	0.3217	0.3071	0.2890	0.2781	0.2480	0.7695	0.6728	0.6346	0.5975	0.5701	0.5132
	CMF	**0.2134**	**0.1758**	**0.1545**	**0.1384**	**0.1253**	**0.1019**	**0.6582**	**0.5001**	**0.4452**	**0.3811**	**0.3195**	**0.2347**
Throughput	MF_2	11.8832	10.7024	9.6776	9.0889	8.6373	8.1358	32.9795	28.5992	25.3608	22.9710	21.1042	18.5597
	MF_1	12.3647	10.7403	9.8674	9.2223	8.7708	8.1667	32.9672	27.7982	24.4438	22.1691	20.2018	17.4015
	CAP	18.2991	16.8273	15.5975	13.8889	13.6477	12.6762	45.9353	39.6390	35.5944	31.4784	29.2029	25.2830
	TAP	22.0584	18.8479	16.9577	15.9026	15.1283	14.2151	58.5192	47.7490	41.6689	38.5700	35.2813	31.2779
	DALF	11.8763	10.5724	9.1783	8.9276	8.6037	8.0449	32.8586	28.5797	24.8752	22.7428	20.9789	18.3713
	CMF	**8.3266**	**7.2138**	**6.5143**	**6.0463**	**5.5718**	**5.0177**	**30.5885**	**26.0933**	**22.9529**	**20.7105**	**17.8538**	**14.7925**

5.4 Experiments for Static QoS Prediction

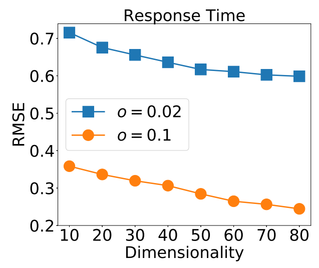
5.4.1 Parameter Settings. In the experiments, for all baseline methods, we tune the corresponding parameters following the guidance of the original papers. As for our method CMF, on the response time dataset, the parameters are set as $l = 30$, $\gamma = 1$, $\lambda_u = \lambda_s = 1$, and $\eta_u = \eta_s = 0.003$. On the throughput dataset, the parameters are set as $l = 30$, $\gamma = 20$, $\lambda_u = \lambda_s = 0.01$, and $\eta_u = \eta_s = 0.025$. For MF_2, MF_1 and DALF, the feature dimensionality is also set to 30.

5.4.2 Experimental Results. We first report the results by varying the training ratios in the range of $\{0.1, 0.2, 0.3, 0.7, 0.8, 0.9\}$. This is to simulate various prediction scenarios with different data sparsity. For example, when the training ratio is set to 0.1, then 10% of the dataset will be used as training data and the rest will be used as testing data. As aforementioned, during the testing phase, outliers should be eliminated explicitly. Here we set the outlier ratio to 0.1, which means 10% of the testing data with large outlier scores will be removed when calculating MAE and RMSE. The detailed comparison results are presented in Table 2. As can be seen, our method CMF consistently shows better performance than all baseline methods. Moreover, the MAE and RMSE values obtained by our method are much smaller than those of baseline methods, especially on the response time dataset. For instance, CMF achieves more than 30% performance promotion on response time over both MAE and RMSE. From Table 2, we can also see that MF_1, MF_2, DALF and CMF tend to obtain smaller MAE and RMSE values as the training ratio increases. This is desired because a larger training ratio indicates that more QoS observations (i.e., more information) will be used to train the prediction model. However, CAP and TAP do not show this pattern, especially on the response time dataset. We can also observe that although CAP, TAP and DALF explicitly take outliers into consideration during the training phase, their performance is not satisfactory. The reason may be the misclassification of outliers. Since our method does not detect outliers directly during the training phase, it will not suffer from the misclassification issue. The resilience of our method to outliers makes it more robust.

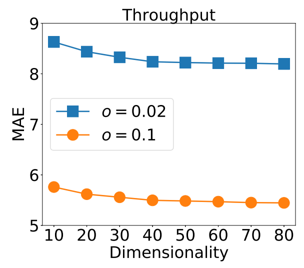
We then report the results by varying the outlier ratios in the range of $\{0.02, 0.04, 0.06, 0.08, 0.1, 0.2\}$. In this experiment, the training ratio is fixed at 0.5. The results are shown in Table 3. From Table 3, we can see that our method still shows the best performance under different outlier ratios. It can also be observed that the MAE and RMSE values of all methods become smaller as the outlier ratio increases. This is reasonable because the larger the outlier ratio is, the more testing data with large outlier scores will be removed. Thus, the effects of outliers on the calculation of MAE and RMSE will be reduced accordingly. From Table 3, we can further obtain that with the increasing of outlier ratios, the performance promotion of CMF relative to MF_2 increases from 48% to 67% over MAE



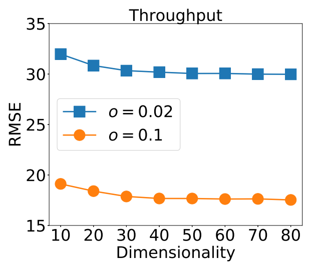
(a) MAE



(b) RMSE

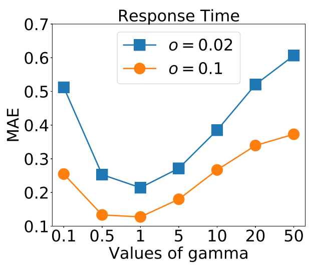


(c) MAE

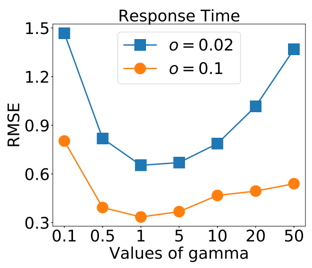


(d) RMSE

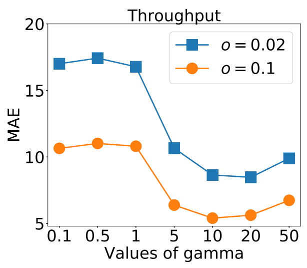
Figure 4: Impact of dimensionality l on CMF (with outlier ratio set to 0.02 and 0.1).



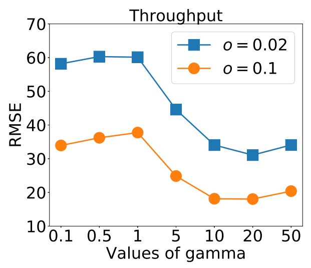
(a) MAE



(b) RMSE

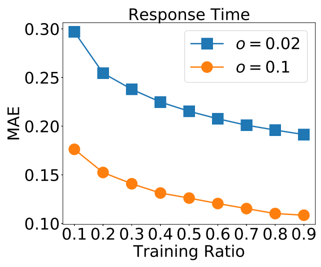


(c) MAE

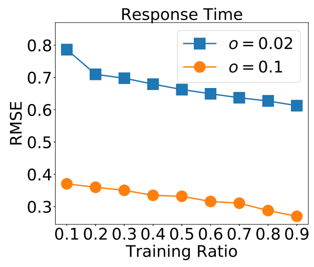


(d) RMSE

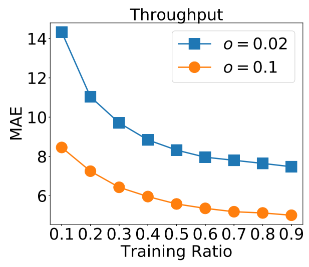
Figure 5: Impact of parameter γ on CMF (with outlier ratio set to 0.02 and 0.1).



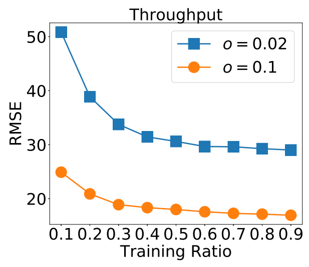
(a) MAE



(b) RMSE



(c) MAE



(d) RMSE

Figure 6: Impact of data sparsity on CMF (with outlier ratio set to 0.02 and 0.1).

and from 18% to 60% over RMSE on the response time dataset. On the throughput dataset, the performance promotion also increases from 30% to 38% over MAE and from 7% to 20% over RMSE. The increase of performance promotion verifies the necessity of removing outliers during the testing phase. It also verifies the robustness of our proposed method again.

5.4.3 Impact of Dimensionality. The parameter dimensionality l controls the dimension of latent features in the factor matrices. To study the impact of l, we vary its value from 10 to 80 with a step size of 10. In this experiment, the training ratio is fixed at 0.5 and the outlier ratio (denoted as o) is set to 0.02 and 0.1. The results are illustrated in Figure 4. As we can see, both MAE and RMSE take smaller values when dimensionality l grows. This is because when l takes larger values, more features of users and Web services will be captured, thus resulting in more accurate prediction results. We also observe on the throughput dataset that the performance tends to be stable when $l \geq 40$, which indicates that $l = 40$ is sufficient for the factor matrices to approximate the original matrix well.

5.4.4 Impact of Parameter γ. Recall that γ denotes the constant in the Cauchy loss. Here we study its impact on the performance of our method by varying its value in the range of $\{0.1, 0.5, 1, 5, 10, 20, 50\}$. In this experiment, the training ratio is fixed at 0.5, and the outlier ratio o is set to 0.02 and 0.1 as well. The results are illustrated in Figure 5. As can be seen, our method is sensitive to γ. This is due to that γ implicitly determines which data will be treated as outliers during the training phase. Thus we need to choose a proper γ to achieve the best performance. As shown in Figure 5, γ should take value around 1 on the response time dataset and around 20 on the throughput dataset to obtain accurate prediction results.

5.4.5 Impact of Data Sparsity. To evaluate the performance of our method comprehensively, it is also necessary to investigate the

Table 4: Performance Comparison with Different Training Ratios on Dynamic Dataset (Best Results in Bold Numbers)

QoS Attributes	Methods	MAE						RMSE					
		10%	20%	30%	70%	80%	90%	10%	20%	30%	70%	80%	90%
Response Time	NNCP	1.0796	1.0536	1.0550	1.0424	1.0406	1.0392	2.6401	2.5797	2.5809	2.5574	2.5668	2.5616
	BNLFT	1.0828	1.0575	1.0467	1.0368	1.0556	1.0403	2.6181	2.5809	2.5682	2.5559	2.5731	2.5582
	WLRTF	1.0560	1.0437	1.0288	1.0299	1.0218	1.0274	2.6009	2.5706	2.5642	2.5566	2.5571	2.5491
	PLMF	2.6133	2.5932	2.4054	2.2097	2.1266	2.0247	4.5582	4.3536	4.3294	4.1843	3.9818	3.8542
	TASR	2.8188	2.7120	2.5591	2.1184	2.0066	1.8854	6.3872	6.1807	5.9552	5.0212	4.8000	4.5447
	CTF	**0.9215**	**0.8981**	**0.8890**	**0.8860**	**0.8766**	**0.8750**	**2.5865**	**2.5579**	**2.5548**	**2.5529**	**2.5517**	**2.5401**
Throughput	NNCP	1.5079	1.4342	1.4287	1.3761	1.3708	1.3761	4.9207	4.7019	4.6404	4.5080	4.4484	4.4968
	BNLFT	1.4241	1.3935	1.3791	1.3856	1.3695	1.3613	4.6031	4.4685	4.4537	4.4128	4.3595	4.3493
	WLRTF	2.9576	2.9568	2.9564	2.9561	2.9562	2.9537	4.9161	4.9160	4.9165	4.9153	4.9159	4.9095
	PLMF	2.4712	2.2602	2.4459	2.3328	2.4655	2.2329	3.6705	3.8363	3.8455	3.8209	3.7541	3.5119
	TASR	4.3265	3.6419	3.4736	2.8803	2.8258	2.7417	5.9152	5.1844	5.0034	4.3744	4.3142	4.2709
	CTF	**1.3567**	**1.1945**	**1.1225**	**0.9907**	**0.9889**	**0.9782**	**3.0436**	**2.9225**	**2.8576**	**2.7732**	**2.6978**	**2.6178**

Table 5: Performance Comparison with Different Outlier Ratios on Dynamic Dataset (Best Results in Bold Numbers)

QoS Attributes	Methods	MAE						RMSE					
		2%	4%	6%	8%	10%	20%	2%	4%	6%	8%	10%	20%
Response Time	NNCP	1.1846	1.1451	1.1069	1.0692	1.0521	1.0204	2.6740	2.6393	2.6023	2.5826	2.5805	2.5799
	BNLFT	1.1647	1.1253	1.0871	1.0654	1.0475	0.9936	2.6499	2.6149	2.5771	2.5687	2.5659	2.5646
	WLRTF	1.1436	1.1036	1.0562	1.0435	1.0261	0.9758	2.6438	2.6079	2.5696	2.5609	2.5584	2.5581
	PLMF	2.6379	2.6011	2.5798	2.4241	2.3315	2.3162	5.2828	5.0571	4.7704	4.5917	4.2765	4.0729
	TASR	2.5125	2.4326	2.3589	2.3363	2.3292	2.3019	5.4851	5.4510	5.4229	5.4018	5.3942	5.3877
	CTF	**1.0292**	**0.9813**	**0.9357**	**0.9105**	**0.8879**	**0.8448**	**2.6369**	**2.6015**	**2.5627**	**2.5564**	**2.5541**	**2.5503**
Throughput	NNCP	2.4853	2.0339	1.7508	1.5419	1.3926	1.0231	9.8925	7.6471	6.2757	5.2096	4.5026	2.8916
	BNLFT	2.4335	1.9909	1.7163	1.5137	1.3693	1.0117	9.7267	7.4979	6.1664	5.1236	4.4319	2.8376
	WLRTF	6.4309	4.8846	3.9224	3.3382	2.9562	2.0911	17.9461	11.6611	7.9882	5.9178	4.9156	3.1509
	PLMF	5.3105	4.1556	3.0467	2.9632	2.3924	2.1807	13.7985	8.7995	6.6349	4.9651	3.8347	3.7906
	TASR	5.7661	4.5595	3.8264	3.3965	3.1322	2.6317	14.8241	9.7143	6.8450	5.2958	4.6089	3.5886
	CTF	**2.2624**	**1.6385**	**1.3421**	**1.1437**	**1.0193**	**0.7323**	**9.5370**	**6.0759**	**4.3650**	**3.2415**	**2.7989**	**1.8020**

impact of the sparsity of training data. To this end, we vary the training ratio from 0.1 to 0.9 with a step size of 0.1. Apparently, different training ratio implies different data sparsity. In this experiment, we also set the outlier ratio o to 0.02 and 0.1. The results are reported in Figure 6. From Figure 6, we see that as the training ratio increases (i.e., the sparsity of data decreases), more accurate results are obtained.

5.5 Experiments for Time-Aware QoS Prediction

We further conduct experiments for time-aware QoS prediction.

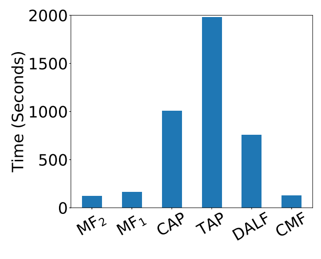
5.5.1 Parameter Settings. In the experiments, we tune the parameters of all baseline methods following the guidance of the original papers. As for our method CTF, on the response time dataset, the parameters are set as $l = 15$ and $\lambda_u = \lambda_s = \lambda_t = 0.1$. γ is set to 10 when calculating MAE and 35 when calculating RMSE. On the throughput dataset, the parameters are set as $l = 15$ and $\lambda_u = \lambda_s = \lambda_t = 100$. γ is fixed at 5 for both MAE and RMSE. As for NNCP, BNLFT, WLRTF and PLMF, the feature dimensionality is also set to 15.

5.5.2 Experimental Results. We first report the results by varying the training ratios in the range of $\{0.1, 0.2, 0.3, 0.7, 0.8, 0.9\}$ and fixing the outlier ratio at 0.1. The results are presented in Table 4. From Table 4, we can see that our method consistently shows better performance than all baseline methods on both datasets. The results verify the robustness of our method in the time-aware extension.

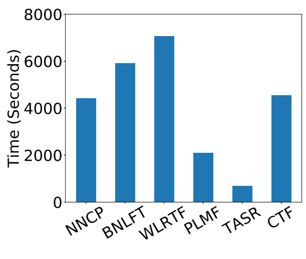
We then report the results by varying the outlier ratios in the range of $\{0.02, 0.04, 0.06, 0.08, 0.1, 0.2\}$ and fixing the training ratio at 0.5. The results are shown in Table 5, from which we observe that our method achieves better performance under different outlier ratios, which is similar to the results on the static dataset.

5.6 Efficiency Analysis

Here, we further investigate the runtime efficiency of our method. In this experiment, we fix the training ratio at 0.5 and the outlier ratio at 0.1. The runtime of different methods on the response time dataset is reported in Figure 7. On the static dataset, we can observe that CMF is very efficient. Its runtime is comparable to that of MF_2 and MF_1. It also runs much faster than CAP, TAP and DALF. On the dynamic dataset, although CTF runs slower than PLMF and TASR, it is faster than BNLFT and WLRTF and is comparable to NNCP.



(a) Static



(b) Dynamic

Figure 7: Runtime comparison of different methods on the response time dataset.

6 RELATED WORK

6.1 Collaborative QoS Prediction

Most existing QoS prediction methods fall into collaborative filtering methods [47], which can be further divided into two categories: memory-based methods [6, 7, 22, 41, 67] and model-based methods [39, 44, 51, 53, 59]. Memory-based methods predict unknown QoS values by employing the neighbourhood information of similar users and similar Web services [65], which further leads to user-based methods [6], service-based methods [41] and hybrid methods [7, 22, 67] that systematically combine the user-based methods and service-based methods. Memory-based methods usually suffer from the data sparsity problem [61, 66], due to the limited number of Web services a single user will invoke. Model-based methods can deal with the problem of data sparsity, thus they have gained the most popularity [61]. Model-based methods usually train a predefined prediction model based on existing QoS observations and then predict missing QoS values. For example, Wu et al. [52] propose to train a factorization machine model for QoS prediction. Luo et al. [32] introduce fuzzy neural networks and adaptive dynamic programming to predict QoS values. Matrix factorization is also a model-based technique and it has obtained the most attention [31, 44, 53, 61, 66]. MF-based methods factorize the user-service matrix into two low-rank factor matrices with one factor matrix capturing the latent representations of users and another revealing the latent representations of Web services. Therefore, MF-based methods are able to automatically model the contributions to a specific QoS value from the user side and service side simultaneously, which usually results in better prediction performance. In addition, MF-based methods possess high flexibility of incorporating side information such as location [19], contexts [50, 51] and privacy [28]. MF-based methods can also be easily generalized for time-aware QoS prediction under the tensor factorization framework [33, 46, 60, 62]. There are also a few other kinds of time-aware QoS prediction methods like time series model-based methods [1, 11] and neural networks-based methods [54].

6.2 Reliable QoS Prediction

Although there are various QoS prediction methods, few of them have taken outliers into consideration. However, as analyzed in Section 2, some QoS observations indeed should be treated as outliers. Thus, the performance of existing methods may not be reliable. For example, most existing MF-based QoS prediction methods directly utilize L_2-norm to measure the discrepancy between the observed QoS values and the predicted ones [31, 44, 50, 53, 57, 66]. It is widely accepted that L_2-norm is not robust to outliers [8, 58, 70]. As a consequence, the performance of MF-based methods may be severely influenced when QoS observations contain outliers.

In order to obtain reliable QoS prediction results, it is necessary to take outliers into consideration. One popular method to reduce the effects of outliers is replacing L_2-norm with L_1-norm because L_1-norm is more robust to outliers [13, 23, 35, 64]. For example, an L_1-norm low-rank MF-based QoS prediction method is introduced in [71]. However, L_1-norm-based objective function is non-smooth and thus much harder to optimize [34, 56]. Besides, L_1-norm is still sensitive to outliers, especially when outliers are far beyond the normal range of QoS values [10, 55].

Another line of reliable QoS prediction is detecting outliers explicitly based on clustering algorithms. In [47], Wu et al. propose a credibility-aware QoS prediction method, which employs a two-phase k-means clustering algorithm to identify untrustworthy users (i.e., outliers). Su et al. [45] propose a trust-aware QoS prediction method, which provides reliable QoS prediction results via calculating the reputation of users by a beta reputation system and identifies outliers based on k-means clustering as well. In [49], a data-aware latent factor model is introduced, which utilizes the density peaks-based clustering algorithm [40] to detect unreliable QoS values. However, it is difficult to choose a proper number of clusters, thus either some outliers may not be eliminated successfully or some normal values may be selected as outliers falsely. Our method does not detect outliers explicitly. Therefore it will not suffer from the misclassification issue.

7 CONCLUSION

In this paper, we have proposed a novel robust QoS prediction method, which utilizes Cauchy loss to measure the discrepancy between the observed QoS values and the predicted ones. Owing to the robustness of Cauchy loss, our method is resilient to outliers. That is, there is no need to detect outliers explicitly. Therefore, our method will not suffer from the problem of misclassification. Considering that the QoS performance may change over time, we have further extended our method to make it suitable for time-aware QoS prediction. To evaluate the efficiency and effectiveness of our method, we have conducted extensive experiments on both static and dynamic datasets. Experimental results have demonstrated that our method can achieve better performance than existing methods.

ACKNOWLEDGMENTS

The work described in this paper was supported by the Guangdong Basic and Applied Basic Research Foundation (No. 2019A1515011043).

REFERENCES

[1] Ayman Amin, Lars Grunske, and Alan Colman. 2012. An automated approach to forecasting QoS attributes based on linear and non-linear time series modeling. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 130–139.
[2] Jonathan T Barron. 2019. A general and adaptive robust loss function. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4331–4339.
[3] Athman Bouguettaya, Munindar Singh, Michael Huhns, Quan Z Sheng, Hai Dong, Qi Yu, Azadeh Ghari Neiat, Sajib Mistry, Boualem Benatallah, Brahim Medjahed,

et al. 2017. A service computing manifesto: the next 10 years. *Commun. ACM* 60, 4 (2017), 64–72.

[4] George EP Box and David A Pierce. 1970. Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *J. Amer. Statist. Assoc.* 65, 332 (1970), 1509–1526.

[5] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex optimization.* Cambridge University Press.

[6] John S Breese, David Heckerman, and Carl Kadie. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI)*. Morgan Kaufmann Publishers Inc., 43–52.

[7] Jie Cao, Zhiang Wu, Youquan Wang, and Yi Zhuang. 2013. Hybrid collaborative filtering algorithm for bidirectional Web service recommendation. *Knowledge and Information Systems* 36, 3 (2013), 607–627.

[8] Xiangyong Cao, Yang Chen, Qian Zhao, Deyu Meng, Yao Wang, Dong Wang, and Zongben Xu. 2015. Low-rank matrix factorization under general mixture noise distributions. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 1493–1501.

[9] Xi'ai Chen, Zhi Han, Yao Wang, Qian Zhao, Deyu Meng, and Yandong Tang. 2016. Robust tensor factorization with unknown noise. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5213–5221.

[10] Chris Ding and Bo Jiang. 2017. L1-norm error function robustness and outlier regularization. *arXiv preprint arXiv:1705.09954* (2017).

[11] Shuai Ding, Yeqing Li, Desheng Wu, Youtao Zhang, and Shanlin Yang. 2018. Time-aware cloud service recommendation using similarity-enhanced collaborative filtering and ARIMA model. *Decision Support Systems* 107 (2018), 103–115.

[12] Joyce El Haddad, Maude Manouvrier, and Marta Rukoz. 2010. TQoS: Transactional and QoS-aware selection algorithm for automatic Web service composition. *IEEE Transactions on Services Computing* 1 (2010), 73–85.

[13] Anders Eriksson and Anton Van Den Hengel. 2010. Efficient computation of robust low-rank matrix approximations in the presence of missing data using the L 1 norm. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 771–778.

[14] John Fox. 2002. *An R and S-Plus companion to applied regression.* Sage.

[15] Rainer Gemulla, Erik Nijkamp, Peter J Haas, and Yannis Sismanis. 2011. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. ACM, 69–77.

[16] Seyyed Hamid Ghafouri, Seyyed Mohsen Hashemi, and Patrick CK Hung. 2020. A Survey on Web Service QoS Prediction Methods. *IEEE Transactions on Services Computing* (2020).

[17] Naiyang Guan, Tongliang Liu, Yangmuzi Zhang, Dacheng Tao, and Larry S Davis. 2017. Truncated Cauchy non-negative matrix factorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41, 1 (2017), 246–259.

[18] Fitriyah Hasny, Samuel Mensah, Deliang Yi, Chune Li, and Richong Zhang. 2016. Predicting the quality of Web services based on user stability. In *2016 IEEE International Conference on Services Computing (SCC)*. IEEE, 860–863.

[19] Pinjia He, Jieming Zhu, Zibin Zheng, Jianlong Xu, and Michael R Lyu. 2014. Location-based hierarchical matrix factorization for Web service recommendation. In *2014 IEEE International Conference on Web Services (ICWS)*. IEEE, 297–304.

[20] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[21] Peter J Huber. 2011. *Robust statistics.* Springer.

[22] Yechun Jiang, Jianxun Liu, Mingdong Tang, and Xiaoqing Liu. 2011. An effective Web service recommendation method based on personalized collaborative filtering. In *2011 IEEE International Conference on Web Services (ICWS)*. IEEE, 211–218.

[23] Qifa Ke and Takeo Kanade. 2005. Robust L/sub 1/norm factorization in the presence of outliers and missing data by alternative convex programming. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Vol. 1. IEEE, 739–746.

[24] Minjung Kim, Byungkook Oh, Jooik Jung, and Kyong-Ho Lee. 2016. Outlier-robust Web service selection based on a probabilistic QoS model. *International Journal of Web and Grid Services* 12, 2 (2016), 162–181.

[25] Tamara G Kolda and Brett W Bader. 2009. Tensor decompositions and applications. *SIAM Rev.* 51, 3 (2009), 455–500.

[26] Daniel D Lee and H Sebastian Seung. 2001. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems (NIPS)*. 556–562.

[27] Xuelong Li, Quanmao Lu, Yongsheng Dong, and Dacheng Tao. 2018. Robust subspace clustering by Cauchy loss function. *IEEE Transactions on Neural Networks and Learning Systems* 30, 7 (2018), 2067–2078.

[28] An Liu, Xindi Shen, Zhixu Li, Guanfeng Liu, Jiajie Xu, Lei Zhao, Kai Zheng, and Shuo Shang. 2018. Differential private collaborative Web services QoS prediction. *World Wide Web* (2018), 1–24.

[29] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining (ICDM)*. IEEE, 413–422.

[30] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2012. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 6, 1 (2012), 3.

[31] Wei Lo, Jianwei Yin, Ying Li, and Zhaohui Wu. 2015. Efficient Web service QoS prediction using local neighborhood matrix factorization. *Engineering Applications of Artificial Intelligence* 38 (2015), 14–23.

[32] Xiong Luo, Yixuan Lv, Ruixing Li, and Yi Chen. 2015. Web service QoS prediction based on adaptive dynamic programming using fuzzy neural networks for cloud services. *IEEE Access* 3 (2015), 2260–2269.

[33] Xin Luo, Hao Wu, Huaqiang Yuan, and MengChu Zhou. 2019. Temporal pattern-aware QoS prediction via biased non-negative latent factorization of tensors. *IEEE Transactions on Cybernetics* (2019), 1–12.

[34] Deyu Meng and Fernando De La Torre. 2013. Robust matrix factorization with unknown noise. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 1337–1344.

[35] Deyu Meng, Zongben Xu, Lei Zhang, and Ji Zhao. 2013. A cyclic weighted median method for l1 low-rank matrix factorization with missing entries. In *Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI)*. 704–710.

[36] Ivan Mizera and Christine H Müller. 2002. Breakdown points of Cauchy regression-scale estimators. *Statistics & Probability Letters* 57, 1 (2002), 79–89.

[37] Michael P Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. 2007. Service-oriented computing: State of the art and research challenges. *Computer* 40, 11 (2007), 38–45.

[38] Stephan Rabanser, Oleksandr Shchur, and Stephan Günnemann. 2017. Introduction to tensor decompositions and their applications in machine learning. *arXiv preprint arXiv:1711.10781* (2017).

[39] Sarathkumar Rangarajan. 2018. Qos-based Web service discovery and selection using machine learning. *arXiv preprint arXiv:1807.01439* (2018).

[40] Alex Rodriguez and Alessandro Laio. 2014. Clustering by fast search and find of density peaks. *Science* 344, 6191 (2014), 1492–1496.

[41] Badrul Munir Sarwar, George Karypis, Joseph A Konstan, John Riedl, et al. 2001. Item-based collaborative filtering recommendation algorithms. *WWW* 1 (2001), 285–295.

[42] Lingshuang Shao, Jing Zhang, Yong Wei, Junfeng Zhao, Bing Xie, and Hong Mei. 2007. Personalized QoS prediction for Web services via collaborative filtering. In *IEEE International Conference on Web Services (ICWS)*. IEEE, 439–446.

[43] Amnon Shashua and Tamir Hazan. 2005. Non-negative tensor factorization with applications to statistics and computer vision. In *Proceedings of the 22nd international conference on Machine learning*. 792–799.

[44] Kai Su, Liangli Ma, Bin Xiao, and Huaiqiang Zhang. 2016. Web service QoS prediction by neighbor information combined non-negative matrix factorization. *Journal of Intelligent & Fuzzy Systems* 30, 6 (2016), 3593–3604.

[45] Kai Su, Bin Xiao, Baoping Liu, Huaiqiang Zhang, and Zongsheng Zhang. 2017. TAP: A personalized trust-aware QoS prediction approach for Web service recommendation. *Knowledge-Based Systems* 115 (2017), 55–65.

[46] Qingxian Wang, Minzhi Chen, Mingsheng Shang, and Xin Luo. 2019. A momentum-incorporated latent factorization of tensors model for temporal-aware QoS missing data prediction. *Neurocomputing* 344 (2019), 299–307.

[47] Chen Wu, Weiwei Qiu, Zibin Zheng, Xinyu Wang, and Xiaohu Yang. 2015. QoS prediction of Web services based on two-phase k-means clustering. In *2015 IEEE International Conference on Web Services (ICWS)*. IEEE, 161–168.

[48] Di Wu, Qiang He, Xin Luo, Mingsheng Shang, Yi He, and Guoyin Wang. 2019. A Posterior-neighborhood-regularized Latent Factor Model for Highly Accurate Web Service QoS Prediction. *IEEE Transactions on Services Computing* (2019).

[49] Di Wu, Xin Luo, Mingsheng Shang, Yi He, Guoyin Wang, and Xindong Wu. 2019. A data-aware latent factor model for Web service QoS prediction. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*. Springer, 384–399.

[50] Hao Wu, Kun Yue, Bo Li, Binbin Zhang, and Ching-Hsien Hsu. 2018. Collaborative QoS prediction with context-sensitive matrix factorization. *Future Generation Computer Systems* 82 (2018), 669–678.

[51] Hao Wu, Zhengxin Zhang, Jiacheng Luo, Kun Yue, and Ching-Hsien Hsu. 2018. Multiple attributes QoS prediction via deep neural model with contexts. *IEEE Transactions on Services Computing* (2018).

[52] Yaoming Wu, Fenfang Xie, Liang Chen, Chuan Chen, and Zibin Zheng. 2017. An embedding based factorization machine approach for Web service QoS prediction. In *International Conference on Service-Oriented Computing (ICSOC)*. Springer, 272–286.

[53] Qi Xie, Shenglin Zhao, Zibin Zheng, Jieming Zhu, and Michael R Lyu. 2016. Asymmetric correlation regularized matrix factorization for Web service recommendation. In *2016 IEEE International Conference on Web Services (ICWS)*. IEEE, 204–211.

[54] Ruibin Xiong, Jian Wang, Zhongqiao Li, Bing Li, and Patrick CK Hung. 2018. Personalized LSTM based matrix factorization for online QoS prediction. In *2018 IEEE International Conference on Web Services (ICWS)*. IEEE, 34–41.

[55] Chang Xu, Dacheng Tao, and Chao Xu. 2015. Multi-view intact space learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37, 12 (2015), 2531–2544.

[56] Huan Xu, Constantine Caramanis, and Shie Mannor. 2012. Sparse algorithms are not stable: A no-free-lunch theorem. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34, 1 (2012), 187–193.

[57] Jianlong Xu, Zibin Zheng, and Michael R Lyu. 2016. Web service personalized quality of service prediction via reputation-based matrix factorization. *IEEE Transactions on Reliability* 65, 1 (2016), 28–37.

[58] Shuang Xu, Chunxia Zhang, and Jiangshe Zhang. 2020. Adaptive Quantile Low-Rank Matrix Factorization. *Pattern Recognition* (2020), 107310.

[59] Yatao Yang, Zibin Zheng, Xiangdong Niu, Mingdong Tang, Yutong Lu, and Xiangke Liao. 2018. A location-based factorization machine model for Web service QoS prediction. *IEEE Transactions on Services Computing* (2018).

[60] Wancai Zhang, Hailong Sun, Xudong Liu, and Xiaohui Guo. 2014. Temporal QoS-aware Web service recommendation via non-negative tensor factorization. In *Proceedings of the 23rd International Conference on World Wide Web (WWW)*. ACM, 585–596.

[61] Yiwen Zhang, Kaibin Wang, Qiang He, Feifei Chen, Shuiguang Deng, Zibin Zheng, and Yun Yang. 2019. Covering-based Web service quality prediction via neighborhood-aware matrix factorization. *IEEE Transactions on Services Computing* (2019).

[62] Yilei Zhang, Zibin Zheng, and Michael R Lyu. 2011. WSPred: A time-aware personalized QoS prediction framework for Web services. In *2011 IEEE 22nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 210–219.

[63] Qian Zhao, Deyu Meng, Zongben Xu, Wangmeng Zuo, and Yan Yan. 2015. *L*_{1}-norm low-rank matrix factorization by variational Bayesian method. *IEEE Transactions on Neural Networks and Learning Systems* 26, 4 (2015), 825–839.

[64] Yinqiang Zheng, Guangcan Liu, Shigeki Sugimoto, Shuicheng Yan, and Masatoshi Okutomi. 2012. Practical low-rank matrix approximation under robust l 1-norm. In *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 1410–1417.

[65] Zibin Zheng and Michael R Lyu. 2010. Collaborative reliability prediction of service-oriented systems. In *2010 ACM/IEEE 32nd International Conference on Software Engineering (ICSE)*, Vol. 1. IEEE, 35–44.

[66] Zibin Zheng and Michael R Lyu. 2013. Personalized reliability prediction of Web services. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 22, 2 (2013), 12.

[67] Zibin Zheng, Hao Ma, Michael R Lyu, and Irwin King. 2011. Qos-aware Web service recommendation by collaborative filtering. *IEEE Transactions on Services Computing* 4, 2 (2011), 140–152.

[68] Zibin Zheng, Li Xiaoli, Mingdong Tang, Fenfang Xie, and Michael R Lyu. 2020. Web Service QoS Prediction via Collaborative Filtering: A Survey. *IEEE Transactions on Services Computing* (2020).

[69] Jieming Zhu, Pinjia He, Zibin Zheng, and Michael R Lyu. 2017. Online QoS prediction for runtime service adaptation via adaptive matrix factorization. *IEEE Transactions on Parallel and Distributed Systems* 28, 10 (2017), 2911–2924.

[70] Rui Zhu, Di Niu, and Zongpeng Li. 2017. Robust Web service recommendation via quantile matrix factorization. In *IEEE 2017 International Conference on Computer Communications (INFOCOM)*. IEEE, 1–9.

[71] Xiaoke Zhu, Xiao-Yuan Jing, Di Wu, Zhenyu He, Jicheng Cao, Dong Yue, and Lina Wang. 2018. Similarity-maintaining privacy preservation and location-aware low-rank matrix factorization for QoS prediction based Web service recommendation. *IEEE Transactions on Services Computing* (2018).

[72] Zhiliang Zhu, Haitao Yuan, Jie Song, Jing Bi, and Guoqi Liu. 2010. WS-SCAN: A effective approach for Web services clustering. In *2010 International Conference on Computer Application and System Modeling (ICCASM)*, Vol. 5. IEEE, 618–622.