Outlier Resilient Collaborative Web Service QoS Prediction

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

Zhiwei Lin  
Sun Yat-sen University  
linzgw25@mail2.sysu.edu.cn

Chuan Chen  
Sun Yat-sen University  
chenchuan@mail.sysu.edu.cn

Zibin Zheng  
Sun Yat-sen University  
zhzibin@mail.sysu.edu.cn

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

Emine Yilmaz  
University College London  
emine.yilmaz@ucl.ac.uk

ABSTRACT

Nowadays, more and more Web services are provided by different enterprises and organizations. The proliferation of Web services makes it difficult for users to select the most appropriate Web services among numerous functionally identical or similar service candidates. Quality-of-Service (QoS) is widely employed for describing 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 a robust collaborative 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.

KEYWORDS

Web service, QoS prediction, outlier resilience, collaborative filtering, Cauchy loss

1 INTRODUCTION

Web service is a standard technology for providing interoperability between disparate software applications and plays the key role in service-oriented computing [3]. Over the past few years, a large number of Web services have been published as indicated by the Web service repository—ProgrammableWeb1. 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 [38, 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 [13, 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 [43]. 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.

For QoS prediction, collaborative filtering is one of the most widely adopted methods [47]. Existing collaborative filtering methods can be grouped into memory-based methods and model-based methods. The memory-based methods aim at utilizing similar users (or similar Web services) for a target user (or a target Web service) to predict unknown QoS values [10, 72]. Although memory-based methods are straightforward and easy to implement, they usually suffer from the data sparsity problem [61, 66]. The model-based methods aim at building global prediction models based on existing QoS observations to predict unknown QoS values [61]. Among model-based methods, matrix factorization (MF) has gained the most attention and popularity and it is able to deal with the issue of data sparsity [26]. In addition, MF-based methods usually have greater capability than memory-based methods in capturing the similarity between users and the similarity between Web services, which therefore results in better QoS prediction performance.

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 [32, 44, 48, 50, 53, 57, 66]. It is well-known that $L_2$-norm is sensitive to outliers [8, 35, 58, 63, 64, 69]. 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 [70]. Although $L_1$-norm is more robust to outliers [14, 23, 36, 64], $L_1$-norm-based

1https://www.programmableweb.com/
objective function is much harder to optimize and the solution is also unstable [35, 56]. Moreover, L1-norm is still sensitive to outliers especially when outliers take significantly different values from the normal ones [11, 55]. There are also some methods seeking to identify outliers explicitly by means of clustering algorithms [18, 49, 71], which usually treat all the elements in the smallest cluster as outliers [19, 47]. 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.

To deal with the aforementioned issues, in this paper we propose a novel robust QoS prediction method under the matrix factorization framework. Our method chooses to measure the discrepancy between the observed QoS values and the predicted ones by Cauchy loss [2, 28] 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 [28, 55]. Theoretically, Cauchy loss allows nearly half of the observations to be out of the normal range before it gives incorrect results [37]. Apparently, for a given QoS dataset, it is almost impossible 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 identify outliers explicitly, 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 specific 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 remainder of this paper is organized as follows. Section 2 shows 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 description 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 deprive the performance of their competitors’ Web services. In addition, the QoS 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 Figure 1 (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. 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 Figure 1 (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.

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 $X \in \mathbb{R}^{m \times n}$ whose entry $X_{ij}$ denotes the QoS value of Web service $j$ observed by user $i$. Obviously, it is both 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 $X$ 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 $\Omega$ denote the set of existing QoS observations, that is,

$$\Omega = \{(i, j, X_{ij}) | \text{the QoS value } X_{ij} \text{ between user } i \text{ and Web service } j \text{ has been observed}\}.$$  

Then the problem of QoS prediction is defined as follows.

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

3.2 Matrix Factorization for QoS Prediction

Matrix factorization is arguably the most popular model-based QoS prediction method at present [61]. 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 $X$ into two low-rank factor matrices $U \in \mathbb{R}^{m \times l}$ and $S \in \mathbb{R}^{n \times l}$, i.e.,

$$X \approx U^T S.$$  

Here, each row of $U$ represents the latent feature of a user, and each row of $S$ represents the latent feature of a Web service. The dimensionality of latent features is controlled by parameter $l$ ($l \ll \min(m, n)$). It is apparent that $U^T S$ should be as close to $X$ as possible, which means that in MF-based methods, the QoS values are determined by users’ latent features and Web services’ latent features simultaneously. More specifically, the rows of $U$ and $S$ imply the contribution to the corresponding QoS values from user side and Web service side respectively [19]. In order to make $U^T S$ approximate $X$ well, there is no doubt that similar users and similar Web services should have similar latent features. Compared with memory-based QoS prediction methods that need to calculate similar users and similar Web services directly (e.g., calculating similarity according to Pearson correlation coefficient (PCC) [65, 72]), MF-based methods are able to capture the similarity automatically, which lends more flexibility and generality to themselves.

Based on the discussions above, we can derive the following general objective function for MF-based QoS prediction methods:

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

where $\mathcal{L}$ measures the degree of approximation between $US^T$ and $X$, $\mathcal{L}_{\text{reg}}$ denotes the regularization term to avoid over-fitting, and $\lambda$ 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 [32, 57, 61, 66, 70]. In this setting, the specific objective function can be clearly given as follows:

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

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 $I \in \mathbb{R}^{m \times n}$ denotes the indicator matrix whose entry $I_{ij}$ 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 $j$, $I_{ij}$ 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 [16]. 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 $X$ and their approximation entries in $US^T$ 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 strategy is to replace $L_2$-norm with $L_1$-norm [14, 23, 51, 70]. Based on $L_1$-norm, the objective function is formalized as below:

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

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 [11].

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 which 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, 37, 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:

$$
\min \sum_i g(r_i),
$$

where function $g$ gives the contribution of each residual to the objective function. A reasonable function $g$ should satisfy the following four properties [15]:

- $g(x) \geq 0, \forall x$;
- $g(x) = g(-x), \forall x$;
- $g(0) = 0$;
- $g(x)$ 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:

$$
g'(x) = \frac{dg(x)}{dx}.
$$

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 $g(x) = \frac{1}{2}x^2$, 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-robustness 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 [28, 55] ($|g'(x)| = 1$ even when $x \to \pm \infty$). 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 ($|g'(x)| \to 0$ when $x \to \pm \infty$). Cauchy estimator has been shown to possess such precious characteristics. The $g$ function of Cauchy estimator (i.e., Cauchy loss) is shown as follows:

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

where $\gamma$ is a constant. The influence function is then calculated as

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

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

$$
\min_{U,S} \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} I_{ij} \ln \left( 1 + \frac{(X_{ij} - U_i S_j)^2}{\gamma^2} \right) + \lambda_u \frac{\|U\|^2}{2} + \lambda_s \frac{\|S\|^2}{2},
$$

where $U_i$ and $S_j$ denote the $i$-th row of $U$ and the $j$-th row of $S$ respectively, $\lambda_u$ and $\lambda_s$ represent the regularization coefficients.

The objective function in Eq. (9) can be efficiently optimized by the gradient descent method [16]. Specifically, we choose to optimize $U$ and $S$ row by row. Then, we have the following update rules:

$$
U_i \leftarrow U_i - \eta_u \frac{\partial \mathcal{L}}{\partial U_i},
$$

$$
S_j \leftarrow S_j - \eta_s \frac{\partial \mathcal{L}}{\partial S_j},
$$

where $\eta_u$ and $\eta_s$ denote the learning rates for $U$ and $S$, and

$$
\frac{\partial \mathcal{L}}{\partial U_i} = \lambda_u U_i - \sum_{j=1}^{n} I_{ij} \frac{X_{ij} - U_i S_j^T}{\gamma^2 + (X_{ij} - U_i S_j^T)^2} S_j,
$$

$$
\frac{\partial \mathcal{L}}{\partial S_j} = \lambda_s S_j - \sum_{i=1}^{m} I_{ij} \frac{X_{ij} - U_i S_j^T}{\gamma^2 + (X_{ij} - U_i S_j^T)^2} U_i.
$$

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 $p$ denote the number of available entries in $X$, then the time complexity of Algorithm 1 is $O(rp^2)$.

**Proof.** The main time cost of Algorithm 1 lies in the updates of $U$ and $S$. In each iteration, updating $U$ takes $O(ml + p^2)$ time and updating $S$ takes $O(nl + p^2)$ time. Since both $m$ and $n$ are less than $p$, the time complexity of updating $U$ and $S$ can both be simplified as $O(pl^2)$. Thus, the overall time complexity is of order $O(rp^2)$. □

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 [68]. In this part, we aim to extend our method in order 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
Algorithm 1: Algorithm for Static QoS Prediction

| Input: $X \in \mathbb{R}^{m \times n \times l}$, $I$, $Y$, $\lambda_u$, $\lambda_s$, $\lambda_t$; |
| 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 $U$, $S$; |

Algorithm 2: Algorithm for Time-aware QoS Prediction

| Input: $X \in \mathbb{R}^{m \times n \times l}$, $I$, $Y$, $\lambda_u$, $\lambda_s$, $\lambda_t$; |
| Output: $U \in \mathbb{R}^{m \times l}$, $S \in \mathbb{R}^{n \times l}$, $T \in \mathbb{R}^{l \times l}$; |
| 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 $U$, $S$, $T$; |

QoS prediction as below:

$$
\min_{U,S,T} \mathcal{L}' = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{l} I_{ijk} \ln \left( 1 + \frac{(X_{ijk} - \hat{X}_{ijk})^2}{\gamma^2} \right) + \frac{\lambda_u}{2} \|U\|_F^2 + \frac{\lambda_s}{2} \|S\|_F^2 + \frac{\lambda_t}{2} \|T\|_F^2,
$$

s.t. $U \geq 0, S \geq 0, T \geq 0,$ where $\lambda$ 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 [27] 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:

$$
U_i \leftarrow U_i \odot \frac{\sum_{j=1}^{n} \sum_{k=1}^{l} I_{ijk} \Delta_{ijk} X_{ijk}(S_j \odot T_k)}{\sum_{j=1}^{n} \sum_{k=1}^{l} I_{ijk} \Delta_{ijk} X_{ijk}(S_j \odot T_k) + \lambda_u U_i},
$$

$$
S_j \leftarrow S_j \odot \frac{\sum_{i=1}^{m} \sum_{k=1}^{l} I_{ijk} \Delta_{ijk} X_{ijk}(U_i \odot T_k)}{\sum_{i=1}^{m} \sum_{k=1}^{l} I_{ijk} \Delta_{ijk} X_{ijk}(U_i \odot T_k) + \lambda_s S_j},
$$

$$
T_k \leftarrow T_k \odot \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} I_{ijk} \Delta_{ijk} X_{ijk}(U_i \odot S_j)}{\sum_{i=1}^{m} \sum_{j=1}^{n} I_{ijk} \Delta_{ijk} X_{ijk}(U_i \odot S_j) + \lambda_t T_k},
$$

In the above Equations, $\Delta_{ijk}$ is defined as $\Delta_{ijk} = \frac{1}{y^2 + (X_{ijk} - \hat{X}_{ijk})^2}$. 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 $r'$ denote the number of available entries in $X$, then the time complexity of Algorithm 2 is $O(r' r'^3)$.

Proof. The proof is similar to that of Theorem 1. In each iteration, it takes $O(r'^3)$ time to update $U$, $S$, and $T$. Therefore, the total time complexity of Algorithm 2 is of order $O(r'^3)$. \(\square\)
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).

5.1 Datasets

We conduct all experiments on a publicly available dataset collection-WS-DREAM\(^3\), 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.

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 \( \Pi \) denote the set of QoS values to be predicted (i.e., \( \Pi \) is the testing set) and let \( N = \vert \Pi \vert \), 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 \( \Pi \) 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 small MAE value and RMSE value, the predicted \( \hat{q}_* \) should be close to \( q_* \) rather than the normal QoS value. As thus, smaller MAE or RMSE value may not really indicate better performance. To overcome this limitation, we eliminate outliers from \( \Pi \) 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 iForest (short for isolation forest) method [30, 31] for outlier detection. iForest detects outliers purely based on the concept of isolation without employing any distance or density measure, which makes iForest quite efficient and robust. iForest will calculate an outlier score for each datum. The score takes value in the range of [0, 1] and 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 iForest, 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, iForest 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\(_1\)**: MF\(_1\) 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\(_2\)**: MF\(_2\) is also an MF-based QoS prediction method [70]. However, it utilizes the \( L_1\)-norm loss to construct the objective function. MF\(_2\) is expected to be more robust to outliers. Note that we implement MF\(_2\) a little differently from the original one proposed in [70]. In our implementation, we ignore the privacy and location information.
- **CAP**: CAP is a credibility-aware QoS prediction method [47]. It first employs 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.

| 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|

\(\text{https://github.com/wsdream/wsdream-dataset}\)

**Table 1: Statistics of QoS Data**

![Exemplary outlier detection results by iForest.](image-url)
5.4 Experiments for Static QoS Prediction

We first conduct 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 \( \eta = 3, \lambda_u = \lambda_x = 1 \), and \( \eta_u = \eta_v = 0.003 \). On the throughput dataset, the parameters are set as \( \eta = 3, \lambda_u = \lambda_x = 0.01 \), and \( \eta_u = \eta_v = 0.025 \). For MF\(_2\), CMF, 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 that have 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, 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\(_2\), MF\(_1\), and DALF tend to obtain smaller MAE and RMSE values as the training ratio increases. This is desired because larger training ratios indicate 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 and 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.

### 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\(_1\) | 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\(_2\) | 0.4041 | 0.4037 | 0.4036 | 0.2815 | 0.2786 | 0.2798 | 0.6120 | 0.6106 | 0.6103 | 0.5590 | 0.5544 | 0.5595 |
|                 | 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 | 0.2815 | 0.2786 | 0.2798 |
| Throughput      | MF\(_1\) | 13.9730 | 12.3750 | 10.7783 | 7.8371 | 7.2825 | 7.8073 | 28.9608 | 26.8906 | 24.6608 | 19.6406 | 19.2831 | 18.6451 |
|                 | MF\(_2\) | 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 | 23.7953 | 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 |

- **DALF**: DALF is a data-aware latent factor model for QoS prediction [49]. It utilizes the density peaks based clustering algorithm [41] to detect unreliable QoS data directly.
- **NNCP**: NNCP is a tensor-based time-aware QoS prediction method [60]. It is based on CP decomposition and it imposes nonnegative constraints on all the factor matrices.
- **BNLFT**: BNLFT is a biased nonnegative tensor factorization model [34]. 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 [12]. 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 for CAP, TAP, and DALF) will not detect outliers explicitly when learning the prediction model. The reason for detecting outliers during the testing phase is to make MAE and RMSE 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 3: Performance Comparison with Different Outlier Ratios on Static Dataset (Best Results in Bold Numbers)

| QoS Attributes | Methods | MAE | RMSE |
|----------------|---------|-----|------|
| Response Time  | MF₂     | 0.4080 | 0.3732 | 0.3533 | 0.3445 | 0.3306 | 0.3072 |
|                | MF₁     | 0.3761 | 0.3390 | 0.3185 | 0.2972 | 0.2702 | 0.2525 |
|                | CAP     | 0.4163 | 0.3657 | 0.3311 | 0.2997 | 0.2739 | 0.2413 |
|                | TAP     | 0.4562 | 0.3788 | 0.3268 | 0.2703 | 0.2183 | 0.1649 |
|                | DALF    | 0.3622 | 0.3217 | 0.3071 | 0.2890 | 0.2781 | 0.2480 |
|                | CMF     | 0.2134 | 0.1758 | 0.1545 | 0.1384 | 0.1253 | 0.1019 |
| Throughput     | MF₂     | 11.8832 | 10.7024 | 9.6776 | 9.0889 | 8.6373 | 8.1358 |
|                | MF₁     | 12.3647 | 10.7403 | 9.8674 | 9.2223 | 8.7708 | 8.1667 |
|                | CAP     | 18.2991 | 16.8273 | 15.5975 | 13.8889 | 13.6477 | 12.6762 |
|                | TAP     | 22.0584 | 18.8479 | 16.9577 | 15.9026 | 15.1283 | 14.2151 |
|                | DALF    | 11.8763 | 10.5724 | 9.1783 | 8.9276 | 8.6037 | 8.0449 |
|                | CMF     | 8.3266 | 7.2138 | 6.5143 | 6.0463 | 5.7178 | 5.0177 |

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 MF₂ increases from 48% to 67% over MAE 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 again the robustness of our proposed method.

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 well approximate the original matrix.
We now conduct experiments for time-aware QoS prediction. In this experiment, the training ratio is fixed at 0.5 and the outlier ratio \( \sigma \) 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 \( \gamma \). This is due to that \( \gamma \) implicitly determines which data will be treated as outliers during the training phase. Thus we need to choose a proper \( \gamma \) to achieve the best performance. As shown in Figure 5, \( \gamma \) should take value around 1 on the response time dataset and around 20 on the throughput dataset to obtain accurate prediction results.

5.4.4 Impact of Parameter \( \gamma \). Recall that \( \gamma \) 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 \( \sigma \) is set to 0.02 and 0.1 as well. The results are illustrated in Figure 6. As can be seen, our method is sensitive to \( \gamma \). This is due to that \( \gamma \) implicitly determines which data will be treated as outliers during the training phase. Thus we need to choose a proper \( \gamma \) to achieve the best performance. As shown in Figure 5, \( \gamma \) 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 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 \( \sigma \) 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 now 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 \( \lambda_u = \lambda_s = \lambda_t = 0.1 \). \( \gamma \) is set to 10 when calculating MAE and 35 when calculating RMSE. On the throughput dataset, the parameters are set as \( l = 15 \) and \( \lambda_u = \lambda_s = \lambda_t = 100 \). \( \gamma \) 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

In this section, we 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 MF2 and MF1. 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 it is comparable to NNCP.

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, 42, 67] and model-based methods [40, 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 [42] and hybrid methods [7, 22, 67] that systematically combine the user-based methods and service-based methods. It is well known that memory-based methods usually suffer from the data sparsity problem [61, 66] due to limited number of Web services a single user will invoke. Model-based methods are able to deal with the problem of data
tables:

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

| QoS Attributes | Methods | MAE | RMSE |
|----------------|---------|-----|------|
| Response Time  | NNCPT   | 1.0796 | 2.9225 | 1.0309 |
|                | BNLFT   | 1.0828 | 2.9080 | 1.0391 |
|                | WLRTF   | 1.0560 | 2.9237 | 1.0278 |
|                | PLMF    | 2.6133 | 2.5809 | 1.0271 |
|                | TASR    | 2.8188 | 2.5937 | 1.0147 |
|                | CTF     | 0.9215 | 0.8981 | 0.8860 |

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

| QoS Attributes | Methods | MAE | RMSE |
|----------------|---------|-----|------|
| Response Time  | NNCPT   | 1.1846 | 2.6740 | 1.0204 |
|                | BNLFT   | 1.1647 | 2.6499 | 1.0278 |
|                | WLRTF   | 1.1436 | 2.6483 | 1.0262 |
|                | PLMF    | 2.6379 | 2.5828 | 1.0216 |
|                | TASR    | 3.2125 | 3.4851 | 1.0291 |
|                | CTF     | 1.0292 | 0.9907 | 0.9889 |

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. [33] 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 [32, 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 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 [29]. MF-based methods can also be easily generalized for time-aware QoS prediction under the tensor factorization framework [34, 46, 60, 62]. There are also a few other kinds of time-aware QoS prediction methods like time series model based methods [1, 12] 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 [32, 44, 50, 53, 57, 66]. It is widely accepted that $L_2$-norm is not robust to outliers [8, 35, 58, 63, 64, 69]. 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 [14, 23, 36, 64]. For example, an $L_1$-norm low-rank MF-based QoS prediction method is introduced in [70]. However, $L_1$-norm-based objective function is non-smooth and thus much harder to optimize [35, 56]. Besides, $L_1$-norm is still sensitive to outliers especially when outliers are far beyond the normal range of QoS values [11, 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 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 [41] 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.

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, 136–139.

[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 Bouguetaya, Munindar Singh, Michael Huhns, Quan Z Sheng, Hai Dong, Qi Yu, Azadeh Ghari Neiat, Sajib Mistry, Boualem Benatallah, Ibrahim 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 user similarities using a co-rated items model. 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] Xi'ai Chen, Zhi Han, Yoo Wang, Quan Zhou, 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.

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

[10] Ivan Mizera and Christine H Müller. 2002. Breakdown points of Cauchy distributions. In Proceedings of the IEEE International Conference on Computer Vision (ICCV). 1163–1168.

[11] Shuai Ding, Yeqing Li, Desheng Xu, 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] Xi'ai Chen, Zhi Han, Yoo Wang, Quan Zhou, 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.

[13] Zhen Chen, Limin Shen, Peng Li, Dianlong You, and Jean Pepe Buang Mapetu. 2020. Web service QoS prediction when collaborative filtering meets data fluctuating in big-range. World Wide Web (2020), 1–26.

[14] Weili Li, Jianwei Yin, Ying Li, and Zhaohui Wu. 2015. Efficient computation of robust low-rank matrix approximations in the presence of missing data using the L1 norm. In 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 771–778.

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

[16] Shuo Shang. 2018. Differential private collaborative Web services QoS prediction. World Wide Web 21, 4 (2018), 593–611.

[17] Yaoyang 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] Stephan Rabanser, Oleksandr Shchur, and Stephan Günnemann. 2017. Introduction to tensor decompositions and their applications in machine learning. arXiv.
Ruibin Xiong, Jian Wang, Zhongqiao Li, Bing Li, and Patrick CK Hung. 2018. QoS-based Web service discovery and selection using machine learning. arXiv preprint arXiv:1807.01439 (2018).

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

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

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.

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.

Kai Su, Bin Xiao, Baoping Lin, 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.

Qingxuan 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.

Chen Wu, Wewei 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.

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

Di Wu, Xin Luo, Mingsheng Shang, Yi He, Guoyin Wang, and Xindong Wu. 2020. 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.

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.

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).

Yaming 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.

Qi Xie, Shenglin Zhao, Zibin Zheng, Jierning 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.

Rubin 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.

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.

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.

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.

Shuang Xu, Chunxia Zhang, and Jiangshu Zhang. 2020. Adaptive Quantile Low-Rank Matrix Factorization. Pattern Recognition (2020), 107310.

Yatoa Yang, Zibin Zheng, Xiangdong Niu, Mingdong Tang, Yutong Lu, 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.

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

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.

Qian Zhao, Deyu Sheng, Zonghen Xu, Wangmeng Zuo, and Yan Yan. 2015. L1-norm low-rank matrix factorization by variational Bayesian method. IEEE Transactions on Neural Networks and Learning Systems 26, 4 (2015), 825–839.

Yinqiang Zheng, Guanglei Liu, Shigeki Sugimoto, Shuicheng Yan, and Masatoshi Okutomi. 2012. Practical low-rank matrix approximation under robust 1 l1-norm. In 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 1440–1447.

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.

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.

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

Jierning 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.

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

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).

Zhiyang Zhu, Haitao Yuan, Jie Song, Jing Bi, and Guoqi Liu. 2018. WS-SCAN: An effective approach for Web services clustering. In 2010 International Conference on Computer Application and System Modeling (ICCASM), Vol. 5. IEEE, 618–622.

Guosheng Zou, Ming Jiang, Sen Niu, Hao Wu, Shengye Pang, and Yanglan Gan. 2018. QoS-aware Web service recommendation with reinforced collaborative filtering. In International Conference on Service-Oriented Computing (ICSOC). Springer, 430–445.

Zibin Zheng and Michael R Lyu. 2008. TAP: A personalized trust-aware QoS prediction approach for Web service recommendation. In 2008 IEEE Conference on Computer Application and System Modeling (ICCASM), Vol. 5. IEEE, 618–622.

Zhiliang Zhu, Haitao Yuan, Jie Song, Jing Bi, and Guoqi Liu. 2018. WS-SCAN: An effective approach for Web services clustering. In 2010 International Conference on Computer Application and System Modeling (ICCASM), Vol. 5. IEEE, 618–622.