An Encoder-Decoder Architecture for the Prediction of Web Service QoS

Mohammed Ismail Smahi[1], Fethellah Hadjila[1], Chouki Tibermacine[2], Mohammed Merzoug[1], and Abdelkrim Benamar[1]

LRIT, University of Tlemcen, Algeria
LIRMM, CNRS and University of Montpellier, France
{i_smahi, f_hadjila, mohamed.merzoug, a_benamar}@mail.univ-tlemcen.dz
and Chouki.Tibermacine@lirmm.fr

Abstract. Quality of Service (QoS) prediction is an important task in Web service selection and recommendation. Existing approaches to QoS prediction are based on either Content Filtering or Collaborative Filtering. In the two cases, these approaches use external data or past interactions between users and services to predict missing or future QoS scores. One of the most effective techniques for QoS prediction is Matrix Factorization (MF), with Latent Factor Models. The key idea of MF consists in learning a compact model for both users and services. Thereafter QoS prediction is simply computed as a dot product between the user's latent model and the service's latent model. Despite the successful results of MF in the recommendation area, there are still a set of problems that should be handled, like: i) the sparsity of the input models, and ii) the learning of the latent factors which is prone to over-fitting. In this paper, we propose an approach to solve these two problems by using a simple neural network, an auto-encoder, and by exploiting cross-validation on a well-known dataset, in order to select the ideal number of latent factors, and thereby reduce the over-fitting phenomenon.

1 Introduction

Web service recommendation and selection have attracted much attention in the service computing community these last years [3][21][23][13][22]. With the rapid increase of the number of services over the Internet and Cloud computing platforms, the task of recommendation became more important. One of the most important criteria taken into consideration in recommending services is their Quality of Service (QoS). Recommendation systems are based, among other artifacts, on large collections of QoS scores related to different service users and invocations over different time periods. However, these collections contain sometimes missing QoS scores. In addition, QoS scores vary substantially in time. These two facts induce an additional complexity in building recommendation systems [11][20].

To deal with the aforementioned complexity, recommendation systems leverage either Content Filtering or Collaborative Filtering (denoted CF) techniques

to predict the QoS score of a given service [9]. The first technique (*i.e.* content filtering) requires external data to build the profile of users or items (the services), which is not always available. However the second one is mainly based on the past interactions between the users and the items. Through these interactions and transactions, recommendation systems can infer the missing values. Roughly speaking, there are two types of CF techniques: Nearest Neighbors approaches and Matrix Factorization approaches (also known as Latent Factor Models). The key idea of matrix factorization consists in learning a compact model for both users and services, thereafter the QoS prediction is simply computed as a dot product between the user's latent model and the service's latent model. According to [9][20] Matrix Factorization techniques (MF) are more effective (in terms of accuracy) than the nearest neighbors schemes. Despite the successful results of MF in the recommendation area, there are still a set of problems that should be handled, as mentioned in [20][19]; there are two major issues: i) the sparsity of the service invocation matrix, which is the input of the recommendation system, can largely affect the predicted QoS; ii) the learning of the latent factors is prone to over-fitting; as a result the consistency of the latent factors can be compromised. To deal with this situation, MF techniques should adopt additional mechanisms to reduce this side effect.

In this paper, we enhance MF techniques to alleviate the aforementioned issues. The main contributions of our paper are summarized as follows:

- We leveraged auto-encoders [14][8] to build the latent models of both users and services. This choice is mainly motivated by the consistent mathematical foundation of this neural network (in fact, the auto-encoder can learn the optimal decomposition of any real service invocation matrix). In addition, we divide the input data set into a set of clusters in order to reduce the data sparsity **(see the discussion of the first algorithm of section 3)**.
- To reduce the over-fitting phenomenon, we selected the ideal number of latent factors (the size of the hidden layer) according to the cross-validation principle;
- To evaluate the proposed approach, we conducted a set of experiments on a public dataset. These experiments are related to different sizes of the dataset and different levels of sparsity.

The remaining of the paper is organized as follows. Section 2 introduces some background material and motivates our work. Section 3 details the proposed approach for Web service QoS prediction. Section 4 exposes the conducted experiments and discusses the obtained results. Before concluding and presenting some perspectives at the end of the paper, we present in Section 5 the related work.

2 Background & Motivations

Auto-encoder [14][8] is an unsupervised neural network that aims to learn a representation of the inputs that produces the least deformation. In general,

this representation (or code) must be compact and meaningful. In terms of architecture, the auto-encoder is designed as a feed-forward non recurrent neural network (see figure 1), where the size of the input layer is equal to the size of the output layer (which is denoted as n). Additionally, the auto-encoder can have one or more hidden layers, among which the central hidden layer, representing the code of the inputs (its size is denoted as $code\_size$).

In terms of dynamics, the auto-encoder can be viewed as a composition of two functions: the encoding function F_1 (which produces the code) and the decoding function F_2 (which produces the reconstruction), where $F_1 : D_1^n \rightarrow D_2^{code\_size}$ and $F2 : D_2^{code\_size} \rightarrow D_1^n$

The class of functions having D_1^n as domain and $D_2^{code\_size}$ as range is termed A. The class of functions having $D_2^{code\_size}$ as domain and D_1^n as range is termed B. Thus, the output will be : $x' = F_2(F_1(x))$, and the code is $z = F_1(x)$.

If the auto-encoder contains only one hidden layer, then the encoding/decoding functions will be defined as: $z = f(Wx + b)$ and $x' = f'(W'z + b')$, such that: f and f' are transfer functions which can be linear or non-linear (sigmoid, for instance). W and W' are two matrices having the dimensions $(code\_size, n)$ and $(n, code\_size)$ respectively. b and b' represent the bias vectors of dimension $code\_size$ and n respectively.

The auto-encoder is called linear if the transfer functions are linear, otherwise it is non-linear.

In terms of learning, the auto-encoder has to produce the closest reconstructions with respect to the inputs. To do so, it minimizes a dissimilarity function (referred to as $error$). The latter dissimilarity may leverage either the L_p norm, the *Hamming* distance, or another elementary function. Formally, the aim is to find $F_1 \in A$, $F_2 \in B$ such that:

$$error(F_1, F_2) = \sum_{t=1}^{m} \Delta(F_2(F_1(x_t)), x_t) \tag{1}$$

where: Δ : is the L_p norm, the Hamming distance or another dissimilarity function. X_t: is an example that belongs to the learning data set. m: is the size of the data set.

It can be proven that, in case where the decoder is linear and the loss function uses the sum of the squared Euclidean distances, then the linear auto-encoder has the same performance as the non-linear auto-encoder [18] (which means that they reach the same optimum).

If we assume that the auto-encoder is linear and contains a unique hidden layer, and the delta function is the squared Euclidian distance, then the optimal encoding matrix W and the optimal decoding matrix W' will be given as follows:

$$W = \Sigma^{-1}_{\leq p, \leq p}.(U_{.,\leq p})^t \; and \; W' = (U_{.,\leq p}).\Sigma \leq p, \leq p \tag{2}$$

where U and Σ are derived from the singular value decomposition [7] of the input matrix X (*i.e.* $X = U\Sigma V^t$). Equation 2 means that we keep the p largest singular values, where X is a real matrix of size (m, n), U is an orthonormal matrix of size (m, m), V is an orthonormal matrix of size (n, n).

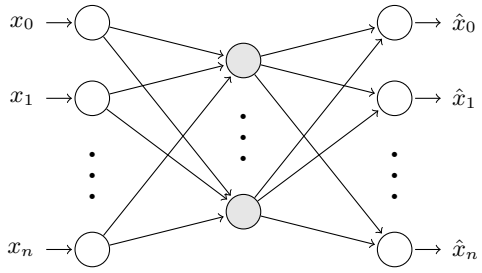

Fig. 1. Auto-encoder architecture (with one hidden layer)

3 Proposed Approach

Figure 2 depicts the global architecture of the proposed approach. First, we assume that the QoS data is collected from various sources such as social networks, third party monitoring systems, or direct feed-backs. The collected QoS data set is viewed as a matrix that contains n services on columns and m users on lines, each cell is modeled as a vector of r realizations of the corresponding QoS criterion with respect to a given user and service. We assume that these QoS realizations contain missing values which need to be predicted. To do so, the QoS data set will undergo a set of steps which are described as follows:

- Firstly (Step 1) we cluster the lines of the initial matrix according to the service location, more specifically we will perform a clustering based on the service country property and another clustering based on the service provider property. According to the works in [20] and [17] the services on the same country are likely to have the same infrastructure and thus similar QoS. Each cluster contains a set of lines that have the same service provider or the same country. We noticed that the sparsity of the entire data set is 26%. However the sparsity of USA's cluster is 24%. In addition the sparsity of other clusters is less than 20% (like Australia, Argentina and others). The aim of this step is to reduce the sparsity of the input matrix. The more the matrix is dense the better the results are. In summary this step will produce a set of clusters that have the same property (either the provider ID or the country ID). Each cluster is represented with a reduced matrix that has less columns and lines with respect to the initial dataset.
- Secondly (Step 2) we perform the learning of latent factors of each reduced matrix (or cluster) by leveraging an auto-encoder. During the auto-encoder training, we also perform a cross validation in order to infer the best size of the hidden layer (the number of latent factors), we assume that all the clusters are trained with the same number of latent factors. This step will provide the hidden layer size that ensures the best validation error (the lowest error).
- At last (Step 3), the learned auto-encoder produces the missing QoS values and stores them in the initial data set.

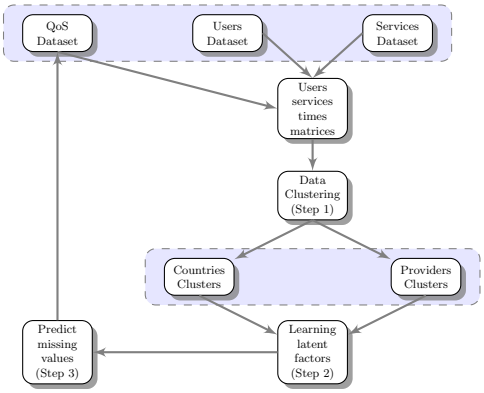

Fig. 2. Overview of the prediction system architecture

Algorithm 1 allows the clustering of the services according to the country ID or the provider ID. In step 7 we initialize the set of clusters according to the provider criterion, the same thing is done for the country criterion in step 8. In step 9 up to 16, we update each cluster with its corresponding service.

Algorithm 2 learns the latent variables as well as their optimal size (denoted $code\_size^{\star}$). In step 4, we explore six possibilities for the code size (20, 40, 60, 80, 100 and 120), the first possibility is initialized in step 1. Thereafter, we extract the current cluster $Cluster_i$ (step 7) and we learn the optimal encoding/decoding matrices (in the next step). This learning function is explained later (algorithm 3). Afterward, we average the previous validation error over the $clusters\_number$ available clusters (step 11). In step 12 up to 16, we update the minimal validation error as well as the optimal encoding/decoding matrices (W_{11}^{*} up to W_{p2}^{*}), additionally the optimal code size is also updated. In step 17, we increment the code size and we repeat the same process for the other values.

Algorithm 3, Autoencoder Cross Validation (AeCV), infers the best encoding/decoding matrices $W_{i\ 1}^{\star}/W_{i\ 2}^{\star}$ for a given $Cluster_i$ and predefined $code\_size$.

The learned auto-encoder (AE) leverages a linear transfer function in the output layer and a sigmoid transfer function in the hidden layer. In step 7, we divide the current $Cluster_i$ into eight parts. After that we perform eight learnings by training each part as a validation set (step 10) and the remaining parts as a training set (step 9). In step 11, we learn the optimal encoding/decoding matrices $W_{k\ i\ 1}^{\star}$ and $W_{k\ i\ 2}^{\star}$ (they represent the best encoding/decoding matrices related to $cluster_i$ and $folder_k$), which are related to the K^{th} part of the $Cluster_i$. The cost function represents the squared error between the auto-encoder output and the desired value. In step 12, we compute the auto-encoder error performed on the validation set. In step 13 and 19 we sum the validation errors related to all folders and we take the mean. The statements 14 up to 17 retain the minimal validation error (*i.e.* v_error_{min}) and its corresponding encoding/decoding matrices (*i.e.*

Algorithm 1: Data clustering according to countries and providers

Input : $\mathcal{P}, \mathcal{C}$ (Providers and Countries from dataset)
Output: $\mathcal{C}luster\_Set\_P$,▷ Countries clusters $\mathcal{C}luster\_Set\_C$ ▷ Providers clusters

foreach $id \in Providers(\mathcal{P})$ **do**
  $Cluster\_P_{id} = \emptyset$
end
foreach $id \in Countries(\mathcal{C})$ **do**
  $Cluster\_C_{id} = \emptyset$
end
$Cluster\_Set\_P =< Cluster\_P_1, Cluster\_P_2, \ldots, Cluster\_P_{|P|} >$
$Cluster\_Set\_C =< Cluster\_C_1, Cluster\_C_2, \ldots, Cluster\_C_{|C|} >$
foreach $service_i \in \mathcal{D}ataset$ **do**
  $provider\_id = get\_provider(service_i)$
  $country\_id = get\_country(service_i)$
  $Cluster\_P_{p\_id} = Cluster\_P_{p\_id} \cup \{service_i\}$
  $Cluster\_C_{c\_id} = Cluster\_C_{c\_id} \cup \{service_i\}$
  $update(\mathcal{C}luster\_Set\_P, Cluster\_P_{p\_id})$
  $update(\mathcal{C}luster\_Set\_C, Cluster\_C_{c\_id})$
end

$W^{\star}_{k\ i\ 1}$,$W^{\star}_{k\ i\ 2}$) also termed *best_weights*. Finally, we return the best weight as well as the mean validation error (*mve*).

4 Experimental Evaluation

We conducted an experimental evaluation of the performance of the proposed approach. In this section, we will show the execution time of the learning algorithm, the prediction accuracy, and the impact of the technical parameters on the prediction quality.

4.1 Experimental Setup

To evaluate the proposed approach, we conducted experiments on a real-world Web service QoS performance repository[1] [24]. This data set contains series of QoS data (for both response time and throughput) which are collected from 142 users (distributed over 57 countries). These users invoke 4500 web services that are located all over the world. Each sequence of QoS data contains at most 64 values which are measured once after a time interval in a time slot. The time slot takes 15 minutes and the duration of the time interval between two consecutive time slots is 15 minutes. In summary we have 142 × 4500 × 64 QoS records for each criterion (response time or throughput). These QoS records contain missing or invalid information that are about 26%. Table 1 summarizes the properties of

[1] WS-DREAM: http://www.wsdream.net

Algorithm 2: Learning latent factors

Input : $Cluster\_Set\_j$ ▷ $j \in \{P, C\}$
Output: $Best\ Configuration$:
$error_{min}$ ▷ the best validation error
$code\_size^{\star}$ ▷ the optimal code size
$best\_weights$ ▷ the best encoding/decoding matrices

$code\_size = 20$
$cs\_max = 6$
$error_{min} = \infty$
for $code\_size\_value : 1 \rightarrow cs\_max$ **do**
  $error = 0$
  for $i : 1 \rightarrow |Cluster\_Set\_j|$ **do**
    $Cluster_i = getCluster(i, Cluster\_Set\_j)$
    $<< W_{i1}^{\star}, W_{i2}^{\star} >, er_i >= AeCV(Cluster_i, code\_size)$
    $error = error + er_i$
  end
  $error = error / |Cluster\_Set\_j|$
  if $error < error_{min}$ **then**
    $error_{min} = error$
    $best\_weights =<< W_{11}^{\star}, W_{12}^{\star} >, < W_{21}^{\star}, W_{22}^{\star} >, \ldots < W_{p1}^{\star}, W_{p2}^{\star} >>$
    $code\_size^{\star} = code\_size$
  end
  $code\_size = code\_size + 20$
end

our dataset. All the learning algorithms are implemented in Tensorflow Python[2]. The experiments run on 3 different machines: i3-380M 3 GHz with 4 GB RAM, i5-4200U 1.6GHz with 4GB RAM and i7-3840QM 2.8 GHz with 16 GB RAM.

In what follows, we present some important elements that are primordial to understand the rest of the section:

– To evaluate the approach performance, we employ two metrics: Mean Absolute Error (MAE) (formula 3) and Root Mean Square Error (RMSE) (formula 4). Since we used the cross-validation, we focus on the average of MAE (formula 5) and the average of RMSE (formula 6). The score of the last formula (6) is used as the output of Algorithm 3.
The standard MAE is specified as follows:

$$MAE_V = \frac{1}{|V|} \sum_{(u,s) \in V} \left| X_{u,s} - \hat{X}_{u,s} \right| \tag{3}$$

where V represents the validation set. $X_{u,s}$ represents the real QoS score for service s given by user u, and $\hat{X}_{u,s}$ the predicted one.

[2] Source code :https://github.com/imsld/Auto-encoder-QoS-Prediction

Algorithm 3: Autoencoder cross validation (AeCV)

Inputs : $Cluster_i, code\_size$
Outputs: $< W_{i\ 1}^{\star}, W_{i\ 2}^{\star} >, mve$

1 $v\_error_{min} = \infty$ ▷ the best validation error
2 $err_k = \infty$
3 $k\_fold = 8$
4 $mve = 0$
5 $T = Cluster_i$ ▷ training set
6 $V = \emptyset$ ▷ validation set
7 $< folder_1, ..., folder_{k\_fold} >= division(Cluster_i)$
8 **for** $k : 1 \rightarrow k\_fold$ **do**
9 $\quad T = T - folder_k$
10 $\quad V = folder_k$
11 $\quad < W_{k\ i\ 1}^{\star}, W_{k\ i\ 2}^{\star} >= \underset{\substack{\text{all possible matrices} \\ W_{k\ i\ 1}, W_{k\ i\ 2}}}{\text{argmin}} \sqrt{\frac{1}{|T|} \sum_{m=1}^{|T|} (AE_{W_{k\ i\ 1}, W_{k\ i\ 1}}(S_m) - S_m)^2}$
$\quad$ ▷ $S_m \in T$
12 $\quad err_k = \frac{1}{|V|} \sum_{m=1}^{|V|} (AE_{W_{k\ i\ 1}^{\star}, W_{k\ i\ 1}^{\star}}(S_m) - S_m)^2$
13 $\quad mve = mve + err_k$
14 $\quad$ **if** $(err_k < v\_error_{min})$ **then**
15 $\quad\quad v\_error_{min} = err_k$
16 $\quad\quad best\_weight =< W_{k\ i\ 1}^{\star}, W_{k\ i\ 2}^{\star} >$
17 $\quad$ **end**
18 **end**
19 $mve = mve/k\_fold$

The standard RMSE is specified as follows:

$$RMSE_V = \sqrt{\frac{1}{|V|} \sum_{(u,s) \in V} \left(X_{u,s} - \hat{X}_{u,s} \right)^2} \tag{4}$$

The average MAE is specified as follows:

$$AverageMAE = \frac{1}{\left| \bigcup_{allV_h} \right|} \sum_{allV_h} (MAE_{V_h}.|V_h|) \tag{5}$$

Where V_h represents a validation set. The average RMSE (which is also denoted as mve in step 19 of Algorithm 3) is specified as follows:

$$AverageRMSE = \frac{1}{\left| \bigcup_{allV_h} \right|} \sum_{allV_h} (RMSE_{V_h}.|V_h|) \tag{6}$$

Table 1. Information of Web Service QoS Values

Statistics	response time	throughput time
Scale	0-20 s	0-20 s
Mean all values	3.165 s	9.608 s
Num. of users	142	142
Num. of web services	4500	4500
Num. of times slices	64	64
Num. all values	30 287 611	30 287 611
Num. missing values	10 609 313	10 609 313

- Before launching the clustering and the learning steps of the framework, we prepare our data set in order to improve its density. Initially the data set contains around 26% of invalid values. After the execution of the two following rules, the percentage of invalid values becomes 23%.
  - R1: For each $Qos_{i,u,s,t}$ of the data set, such as $i \in \{response\_time, throughput\}$, u is the *user ID*, s is the *service ID*, and t is the *time slot* of the QoS realization ($t \in \{1, \ldots, 64\}$). If $Qos_{i,u,s,t}$ is invalid (missing or zero) then we replace it by the average of the valid QoS values of the previous time slots: $Qos_{i,u,s,t} = \frac{1}{(\sum_{t \in T} valid(i,u,s,t))}(\sum_{t \in T} Qos_{i,u,s,t}.valid(i,u,s,t))$
    where
    $$valid(i,u,s,t) = \begin{cases} 1 \; if \; Qos_{i,u,s,t} \; is \; available \\ 0 \; if \; Qos_{i,u,s,t} \; is \; missing \end{cases}$$
  - R2: For each $Qos_{i,u,s,t}$ of the data set, if $Qos_{i,u,s,t}$ is invalid and all its previous values (regarding time slots) are invalid, then $Qos_{i,u,s,t} = 0$.

  In what follows, we assume that this prepared data set is the official input of the proposed approach.
- We assume that the learning phase of each cluster lasts for 5000 iterations (at most). Additionally the learning will be stopped if the auto-encoder error is less than 0.01.
- We perform the clustering step by grouping the services having the same country ID (the clustering based on provider ID will be handled in future works), we obtain 70 clusters which can involved up to 1404 services per group. By doing so, we divide the initial matrix into several sub-matrices that have the same number of lines (142) and different columns. The number of columns corresponds to the size of the cluster. This step aims to reduce the number of invalid entries, and thus it improves the prediction accuracy.
- Concerning the cross validation, we divide each cluster of the data set into eight parts ($k\_fold = 8$). Each part consists of eight consecutive QoS data (*i.e.* Part=$\{Qos_{iust}, Qos_{ius(t+1)}, \ldots, Qos_{ius(t+7)}\}$. In addition, we examine 6 code size values: $code\_size \in \{20, 40, 60, 80, 100, 120\}$. This means that we perform eight trainings for each code size. Thereafter we retain the code size that provides the best validation error. We also notice that the density factor (Ds) of this scheme is $Ds = 80\%$ (density of 80% means that 80% of the

entries data set are retained as training set, while the other 20% are used to test the performance of our model for easch code size).

4.2 Research questions

Our experiments aim to answer the following questions:

- Q1: Does the proposed approach provide a prediction accuracy better than well-known state of the art methods? The compared methods are the following:
  - User-based CF using PCC (UPCC) [2]: this model is a user-based prediction model.
  - Item-based CF using PCC (IPCC) [15]: this model uses similar services for the QoS prediction.
  - WSRec [26][25]: an approach which combines both UPCC and IPCC
  - AVG: this approach takes the mean of the three valid QoS data at the most recent time slots.
  - IPCC* [26]: a linear aggregation of IPCC and AVG (we take equitable weights for both sub models).
  - UPCC* [26]: a linear aggregation of UPCC and AVG (we take equitable weights for both sub models).
  - WSRec* [26]: a linear aggregation of WSRec and AVG (we take equitable weights for both sub models).
  - ARIMA [6]: this is a well-known statistical method adapted to QoS web service prediction.
  - Lasso-k20 [19]: this approach optimizes the recommendation problem by adapting the lasso penalty function.
- Q2: What is the impact of the code size on the prediction accuracy?
- Q3: What is the impact of the code size on the sensitivity to over-fitting?

4.3 Results & Discussion

Table 2 presents the MAE and the RMSE of different prediction algorithms for the response time. We assume that the density of the state of the art methods is Ds= 80%.

From these results, we derive the following findings:

- According to the MAE metric, UPCC, IPCC, WSRec are less effective than the remaining approaches since they do not use the past QoS data.
- The Lasso-K20 out-performs ARIMA, AVG , UPCC*, IPCC*,and WSRec* in terms of MAE and RMSE. In addition it presents less variation of RMSE when we change the data set [19]. However ARIMA and AVG approaches show larger RMSE variations when we change the data set. We notice that, the more the variation of the model error is low the better the generalization.

Table 2. Accuracy Comparison of Prediction methods on Response Time

Approaches	MAE	RMSE
AVG	1.159	3.206
IPCC	1.467	3.032
IPCC*	1.242	2.753
UPCC	1.372	2.925
UPCC*	1.200	2.714
WSRec	1.372	2.925
WSRec*	1.200	2.716
ARIMA	1.028	2.986
LassoK20	0.893	2.572
Autoencoder-100	**0.704**	**1.422**
Autoencoder-120	**0.681**	**1.369**

– The models auto-encoder-100 and auto-encoder-120 present the highest scores for both MAE and RMSE. For instance, the auto-encoder-100 achieves about 57% improvements in MAE accuracy compared to ARIMA. Likewise it achieves about 51% improvements in MAE accuracy compared to the lassoK20 method .

To explain the impact of the code size (denoted as code_size in Algorithm 2) we show in Figure 3 the variation of RMSE (the output of Algorithm 3) according to the code size and the cluster size. It can be clearly seen that the code sizes 100 and 120 out-perform the remaining values for all clusters. Furthermore, we observe that the performance of code size 100 is greater or equal than the performance of code size 120 for almost all clusters with less than 100 services. (These clusters represent the majority of services). Therefore, according to the Occam's razor principle [1], we should use a code size equal to 100 for test sets with less than 100 services. In addition, when the test set size is larger than 100 we should use 120 as code size. Broadly speaking, we can say that a model

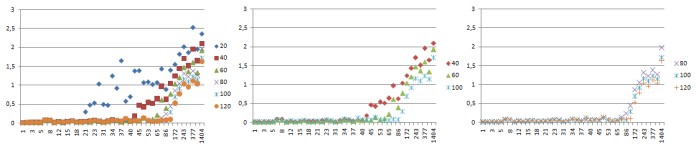

Fig. 3. The RMSE variation for response time metric

is prone to over-fitting if the prediction error highly changes when we change the data set (this model is qualified as a high variance model). Consequently, if we aim to confirm that our approach (*i.e.* the auto-encoder with 100 hidden neurons) is less sensitive to over-fitting, we should compute the prediction error (RMSE) on a new test set and derive the deviation between the validation error

and the new test set error. The larger the deviation is, the higher the over-fitting sensitivity we obtain. In Table 3 we show the mean validation error (also termed the average RMSE) as well as the RMSE related to the entire data set (*i.e.* 142 $\times$ 4500 $\times$ 64 entries) for some code sizes.

Table 3. Over-fitting sensitivity

Code size	20	40	60	80	100	120
Average RMSE	1.939	1.684	1.564	1,487	1.422	1.369
RMSE Dataset	2.818	2.521	2.337	2.211	2.116	2.040
Deviation (absolute difference)	0.879	0.837	0.773	0.724	0.694	0.671

It is clearly shown in this table that the code sizes 100 and 120 are the least sensitive to over-fitting, compared to the other possibilities. We also notice that the code sizes 20, 40 are less sensitive to over-fitting but they also suffer from under-fitting, since their corresponding auto-encoders have a large average RMSE.

4.4 Threats to Validity

The way we measured RMSE may be a threat to **construct validity**. To some extent, the measurement is biased by the fact that the data set that we considered is the one that we have slightly modified (by changing some of the invalid values), instead of the original one from WS-Dream. But since there is only 3 % of values which have been changed, we are quite confident that the impact of this threat on the validity of the results is really marginal. We have started another measurement of this metric, but this takes several weeks of training and prediction. The preliminary results (on all clusters by using the auto-encoder with a code size of 20) showed that RMSE is higher, but with only 0.01 (1.79 instead of 1.78).

A potential threat to **internal validity** concerns the extent to which we may be confident with the conclusions, on sensitivity to over-fitting, which have been made from variance in prediction error when data sets are changed. We could have taken one code size and then measure the deviation on several test data sets. The comparison we have made is between the average RMSE and the RMSE of the whole data set. The evaluation on this single test data set may influence the results. But the fact that this was made on several code sizes (20, 40, 60, 80, 100 and 120) mitigates this risk.

A threat to the **external validity** may be the generalization of the presented results to other contexts, and more precisely to other data sets. The presence of such a large data set (WS-Dream) enabled us to train correctly our predictor. The use of another data set may give lower accuracy. However, the use in our prediction of an auto-encoder together with a cross-validation to identify the best

code size, helps in reducing the impact of using another data set on prediction error.

5 Related Works

In this section, we present some related works based on Collaborative Filtering (CF) algorithms that were proposed recently.

Many existing CF works are based on neighborhood methods. This kind of methods leverages the most similar neighbors of a service/a user to predict the missed QoS. However, this category mainly suffers from the data sparsity, small coverage and cold start problems [25][16].

The work presented in [19] assumes that QoS values depend on service invocation time. In order to make an accurate service recommendation, a time-aware prediction approach is brought forth. Specifically, the authors make a zero-mean Laplace prior distribution assumption on the residuals of the QoS prediction, which corresponds to a Lasso regression problem. To reduce the search range while improving the prediction accuracy,the approach uses the geo-localization of web services to handle the sparse representation of temporal QoS values.

The system proposed in [12] uses the liner regression to predict unknowns QoS data from known QoS values. The work in [10] constructs a recommendation system by inferring the satisfaction probability of the user with respect to a given service. This inference is based on a Bayesian network.

To alleviate the limits of the neighborhood methods, the community has designed another type of approaches which is based on matrix factorization. In fact, this category reduces the sparsity of the invocation matrix by inferring a low dimension model for both services and users [5][11]. In [20] the authors propose three contributions for solving the recommendation problem. The first one combines the matrix factorization with the QoS data provided by the user's neighbors. This data is derived from the user's context (like, the latitude and the longitude of geographical position, and the IT infrastructure). The second one, combines the matrix factorization with the QoS data provided by the service's neighbors. This data is derived from the service context (such as the country, the autonomous system), we also notice that the matrix factorization is solved as an optimization problem with a regularization term. The third contribution combines both the matrix factorization, the user's context and the service context. The experimental results show that the third approach is more effective than the first two.

The work presented in [4], leverages both matrix factorization and service clustering, first of all, the authors build a set of service clusters through the use of context information (like the country and service provider) and the Pearson Correlation Similarity. This hybridization is mainly motivated by the fact that the services which belong to the same geographical region tend to have correlated ratings or QoS data. Therefore the authors add a neighborhood based term to the service latent model.

In [17], the authors develop an enhanced matrix factorization approach by identifying the users' or the web services' neighborhoods. The users' neighbors are selected by measuring the network map distance between them. It is empirically proven that the users with smaller distances are likely to have more similar QoS values on a common set of web services.

In the field of cloud services, the system presented in [23], considers all the software/hardware characteristics of the Cloud Computing architecture. The authors propose a cloud service QoS prediction approach based on Bayesian Networks. The entire process is divided into three steps: data collection and pretreatment, Bayesian model training and prediction of QoS values.

Compared to these works, our approach refines clustering approaches based on service neighborhood by considering a training step that uses a neural network. Thanks to cross-validation, this network, an auto-encoder, is customized with the code size (number of hidden layers) that minimizes the prediction error. The ultimate goal of our work is to solve problems that are complementary to those addressed by the previous works, like data sparsity and over-fitting.

6 Conclusion and Perspectives

We have presented an auto-encoder for predicting unknown QoS scores of Web services based on their history. To achieve the best scores of accuracy, we leveraged the country ID for dividing the whole data set of QoS scores into clusters. Thereafter we have learned the latent factors by using the auto-encoder neural network. In addition we have derived the best code size through the use of cross validation. The comparison results between this prediction system and the state-of-the-art systems, showed the effectiveness of the proposed model.

As future works, we aim to enhance the prediction accuracy by addressing two particular points. First, we plan to develop other clustering alternatives (such as Expectation-Maximization) on user or provider properties. Second, our project is to develop more elaborated learning models such as stacked auto-encoders or denoising auto-encoders.

References

1. Blumer, A., Ehrenfeucht, A., Haussler, D., Warmuth, M.K.: Occam's razor. Information processing letters 24(6), 377–380 (1987)
2. Breese, J.S., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In: Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence. pp. 43–52. Morgan Kaufmann Publishers Inc. (1998)
3. Chen, S., Fan, Y., Tan, W., Zhang, J., Bai, B., Gao, Z.: Service recommendation based on separated time-aware collaborative poisson factorization. Journal of Web Engineering 16(7-8), 595–618 (2017)
4. Chen, Z., Shen, L., Li, F.: Exploiting web service geographical neighborhood for collaborative qos prediction. Future Generation Computer Systems 68, 248–259 (2017)

5. Deng, S., Huang, L., Xu, G.: Social network-based service recommendation with trust enhancement. Expert Systems with Applications 41(18), 8075–8084 (2014)
6. Godse, M., Bellur, U., Sonar, R.: Automating qos based service selection. In: IEEE International Conference on Web Services (ICWS). pp. 534–541. IEEE (2010)
7. Golub, G.H., Reinsch, C.: Singular Value Decomposition and Least Squares Solutions, pp. 134–151. Springer Berlin Heidelberg, Berlin, Heidelberg (1971)
8. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science 313(5786), 504–507 (2006)
9. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer 42(8), 30–37 (Aug 2009)
10. Kuang, L., Xia, Y., Mao, Y.: Personalized services recommendation based on context-aware qos prediction. In: Proceedings of the IEEE 19th International Conference on Web Services (ICWS'12). pp. 400–406. IEEE Computer Society, Washington, DC, USA (June 2012)
11. Lo, W., Yin, J., Li, Y., Wu, Z.: Efficient web service qos prediction using local neighborhood matrix factorization. Engineering Applications of Artificial Intelligence 38, 14 – 23 (2015)
12. Ma, Y., Wang, S., Hung, P.C.K., Hsu, C.H., Sun, Q., Yang, F.: A highly accurate prediction algorithm for unknown web service qos values. IEEE Transactions on Services Computing 9(4), 511–523 (July 2016)
13. Rong, W., Peng, B., Ouyang, Y., Liu, K., Xiong, Z.: Collaborative personal profiling for web service ranking and recommendation. Information Systems Frontiers 17(6), 1265–1282 (2015)
14. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. In: Rumelhart, D.E., McClelland, J.L., PDP Research Group, C. (eds.) Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1, pp. 318–362. MIT Press, Cambridge, MA, USA (1986)
15. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: Proceedings of the 10th international conference on World Wide Web. pp. 285–295. ACM (2001)
16. Shao, L., Zhang, J., Wei, Y., Zhao, J., Xie, B., Mei, H.: Personalized qos prediction forweb services via collaborative filtering. In: Proceedings of the IEEE International Conference on Web Services (ICWS 2007). pp. 439–446. IEEE Computer Society, Washington, DC, USA (July 2007)
17. Tang, M., Zheng, Z., Kang, G., Liu, J., Yang, Y., Zhang, T.: Collaborative web service quality prediction via exploiting matrix factorization and network map. IEEE Transactions on Network and Service Management 13(1), 126–137 (March 2016)
18. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of Machine Learning Research 11, 3371–3408 (Dec 2010)
19. Wang, X., Zhu, J., Zheng, Z., Song, W., Shen, Y., Lyu, M.R.: A spatial-temporal qos prediction approach for time-aware web service recommendation. ACM Transactions on the Web 10(1), 7:1–7:25 (Feb 2016)
20. Xu, Y., Yin, J., Deng, S., Xiong, N.N., Huang, J.: Context-aware qos prediction for web service recommendation and selection. Expert Systems with Applications 53, 75 – 86 (2016)
21. Xu, Y., Yin, J., Li, Y.: A collaborative framework of web service recommendation with clustering-extended matrix factorisation. International Journal of Web and Grid Services 12(1), 1–25 (2016)

22. Yin, J., Xu, Y.: Personalised qos–based web service recommendation with service neighbourhood–enhanced matrix factorisation. International Journal of Web and Grid Services 11(1), 39–56 (2015)
23. Zhang, P., Han, Q., Li, W., Leung, H., Song, W.: A novel qos prediction approach for cloud service based on bayesian networks model. In: 2016 IEEE International Conference on Mobile Services (MS). pp. 111–118 (June 2016)
24. Zhang, Y., Zheng, Z., Lyu, M.R.: Wspred: A time-aware personalized qos prediction framework for web services. In: Proceedings of the IEEE 22nd International Symposium on Software Reliability Engineering (ISSRE'11). pp. 210–219 (Nov 2011)
25. Zheng, Z., Ma, H., Lyu, M., King, I.: Qos-aware web service recommendation by collaborative filtering. IEEE Transactions on Services Computing 4(2), 140–152 (April 2011)
26. Zheng, Z., Ma, H., Lyu, M.R., King, I.: Wsrec: A collaborative filtering based web service recommendation system. In: Web Services, 2009. ICWS 2009. IEEE International Conference on Web Services (ICWS 2009). pp. 437–444. IEEE Computer Society (2009)