IJNRD.ORG ISSN : 2456-4184
INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD) | IJNRD.ORG
An International Open Access, Peer-reviewed, Refereed Journal

Qos Prediction for Web Services Based on Deep Learning

***Le Van Thinh*[1], *Nguyen Huu Truc*[2]**
[1]Mientrung industry and trade college, Viet Nam.
[2]Phu Yen Vocational College, Viet Nam

Abstract: With the rapidly growing number of service providers and users, as a result, there are three important issues that we now have to deal with: Firstly, we face challenges with increasingly large data sets. Secondly, many Web service providers have similar functionality but differ in quality properties and services. Finally, there will have a large number of missing QoS values in history records. Therefore, predicting the missing QoS values in the large data sets is becoming more attractive. In this paper, we proposed a new model based on AutoenCoder to solve the above existing problems. It is called AutoenCoder-based Web Service (AWS). The model is performed with two predictions (User-based AWS and Service-based AWS) and the backpropagation algorithm and Stochastic Gradient Descent is used to train the model. The experimental results demonstrated that proposed model has better performance than other approaches.

IndexTerms - Deep learning; AutoenCoder; QoS Prediction.

I. INTRODUCTION

In parallel with improving the performance of Internet connection speed, Web services are becoming more and more important, and indispensable in life. Hence, the number of service providers and service users are growing up too quickly. Leading to many services has similar functionality or identical, this problem can be caused difficulties for users. So, QoS is considered a key issue. Therefore, predicting the QoS values are among the most popular fields to find the best solutions to improve the quality of recommendation to users.

Quality of service (QoS) is usually used for the description of non-functional characteristics of Web services and it is widely applied to Web service composition [1], Web service recommendation [2], service selection [3], [4], and so on. Among different QoS properties, some of the QoS properties are offered by service providers or by third-party registries and some others are offered by different values from different users (e.g., response time, invocation failure rate, etc.). Therefore, the user desires to know how to choose a QoS from service providers such as latency, availability and reliability. However, almost service providers nowadays focus on the profit and growth. Moreover, in e-commerce and other online environments, the QoS properties of each service are pre-determined by the service providers, but because of its promotion purpose, they often exaggerate to attract more users. Hence, the QoS property may have different values to different users (e.g., response time, availability, throughput etc.), to measure the efficiency of QoS properties, it depends on the user who uses that web service. Especially, QoS-based service recommendation has become a hot topic to select the best services from a set of services. But the in real case, there are always many missing QoS values in history records. So, it is necessary to predict the missing QoS values for service recommendation.

Big data is a broad term for data sets so large or complex, which standard algorithms are usually not designed to handle them. Collaborative Filtering (CF) has been used widely in recommender systems. However, CF cannot handle very large data sets [5, 6]. Recently, some of these studies have been extended from different methods to deal large data sets as [7, 8, 9]. Although there are many research papers on big data. However, in fact the data sets in real world are growing day by day, leading to very large datasets. Hence, search algorithm changes almost constantly to meet actual needs. Therefore, this field is still topical issues and is now attracting attention of the research community. So, predicting the QoS values of Web service for larger datasets is a big challenge for researchers.

Recently there has been a resurgence in the field of artificial intelligence after the study of Deep learning, it is a new area of Machine Learning research and has been researched and widely developed such as image processing and speech recognition [10], Google and Baidu have been applied deep learning to search engines in speech recognition since 2013, the collaborative filtering task [11-16]. Inspired by previous works, in this paper, we proposed a new model based on AutoenCoder, it is called AutoenCoder-based Web Service model (AWS). The AWS model is used the backpropagation algorithm and Gradient descent to predict the missing QoS values in two different ways: User-based AWS (is denoted by U-AWS) and Service-based AWS (is denoted by S-AWS). The experimental results showed that proposed model has better performance than other approaches. The contributions of this chapter are summarized as follows:

- We introduce the AWS model based on Autencoder with two predictors: User-based AWS (U-AWS) and Service-based AWS (S-AWS).
- The backpropagation algorithm is used to train for the model.
- The proposed model can effectively deal with large scale datasets.

The remainder of this chapter is organized as follows: Section 2, we present some preliminaries about Autoencoder and Backpropagation. In Section 3, we build the AWS model based on Autoencoder. Predicting the QoS values based on the AWS model is presented in Section 4. Section 5, we perform experiments and report the results. Finally, we conclude our work and some future directions in Section 6.

2. PRELIMINARIES

In this section we introduce some preliminaries related to the paper, such as, AutenCoder, Backpropagation and so on.

2.1. AutoenCoder

An AutoenCoder [17] can be seen as an extended version of artificial neural network with three or more layers (an input layer, one or more hidden layers, and an output layer) and where the output layer should have the same size as the input layer. AutoenCoder is an unsupervised representation learning method to process the representation of the input data, typically used for the purpose of dimensionality reduction and they play a fundamental role in deep learning.

Definition 1 (Autoencoder) *An Autoencoder (Fig 1) is modeled as a 5-tuple* $A = \langle x, x_1, l, A_0, A_1 \rangle$ *where:*

- x *represents the input values of the neural network.*
- x_1 *is the reconstruction values of the same shape as* x.
- l *represents latent features* $(0 < l < x)$.
- A_1 *represents the weights of hidden layers.*
- A_0 *is the weight matrix of the reverse mapping which can be constrained to the transpose of the forward mapping* $A_0 = A_1^T$ *(is referred to tied weights)*

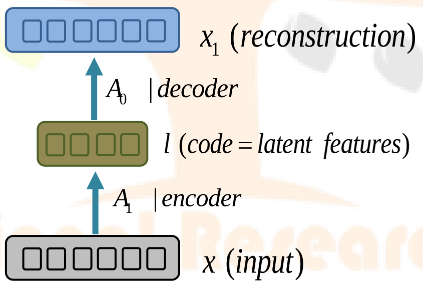

Figure 1: The general architecture of the AutoenCoder

An Autoencoder (Fig 1) consists of two major parts:

Firstly, an Autoencoder takes an input $x \in \Re^d$ and then maps x (as an encoder) to the hidden representation $l \in \Re^k$ using an affine transformation with the weight matrix A_1 of the hidden layer and the bias vector b through *a* deterministic mapping: $a^l = f(A_1.x + b)$, where $f(.)$ is an activation function (Sigmoid activation or softmax activation).

Secondly, the latent representation (latent features) will be mapped back (as a decoder) into a reconstruction x_1 of the same shape as x and through transformation $x_1 = f(A_0.a^l + b_0)$. x_1 is a prediction of x.

Based on the analysis above, we show that the aim of an Autoencoder is to minimize the reconstruction error between the input x and the output x_1. Hence Autoencoder training consists of finding the optimal value of parameters $A_0,\ A_1,\ b,\ b_0$ that minimize the reconstruction error on a training set. The objective function to measure the reconstruction error is used a squared-error cost function $J(x, x_1) = \|x - x_1\|^2$. An autoencoder is often trained the model by using the backpropagation algorithm.

2.2 Backpropagation

Backpropagation is commonly used in the area of neural networks and in conjunction with a Ggradient descent method for minimizing an objective function. The goal of Backpropagation is employed to optimize the weights and the biases in a neural network and learn any arbitrary mapping from the input to the output units.

The Backpropagation algorithm [16] is trained in two phases: Forward pass phase and Backward pass phase. Details will be presented in the following definitions.

***Definition 2* (Forward pass).** *Suppose we have a training set* $\{(x_1,y_1),...,(x_p,y_p)\}$ *representing for p ordered pairs of* ($n\times m$)*-dimensional vectors, where the input value is called input pattern and the corresponding desired output is called target pattern. When the input pattern* x_i *is presented to the network, then it will produce an output* h_i *which is different from the target* y_i *(i=1,...,p). To minimize the error function of the network, defined as*

$$J=\frac{1}{2}\sum_{i=1}^{p}\left\|h_i-y_i\right\|^2 \quad (1)$$

The first step in the minimization process is to compute the error function automatically. Each output unit *j* $(j=1...m)$ is connected to a node which evaluates the function $\frac{1}{2}\left(h_{ij}-y_{ij}\right)^2$, where h_{ij} and y_{ij} represent the *j-th* component of the output pattern h_i and of the target y_i. The outputs of the additional *m* nodes are collected at a node which adds them up and gives the sum J_i as its output. The same network extension has to be built for each pattern y_i. A computing unit collects all quadratic errors and outputs their sum $J_1+...+J_p$. The output of this extended network is the error function *J*.

***Definition 3* (Backwards Pass).** *The* k *weights (parameters)* $w_1,...w_k$ *in the network can be modified to reduce the quadratic error J as small as possible. To minimize J by using an iterative process of gradient descent, we need to compute the gradient* $\nabla J=\left(\frac{\partial J}{\partial w_1},\frac{\partial J}{\partial w_1},...,\frac{\partial J}{\partial w_k}\right)$. *Each weight is updated by summing the following correction* $\Delta w_i=-\alpha\frac{\partial J}{\partial w_i}$ *for* $i=1,...,k$, *where* α *is a learning constant.*

The backward pass starts at the penultimate layer of the feed-forward pass, during the backward pass it updates the weights so that we reach a local minimum of the error function easily by Gradient descent, where $\nabla J=0$

***Definition 4* (Backpropagation).** *Consider a network with the inputs* x *and network function* Q. *The derivative* $Q'(x)$ *is computed in two phases–the training:*

***Forward pass**: The inputs of each training pattern* x_i *is fed into the network. The outputs* h_i *are calculated using the inputs* x_i *and the weights* w_i. *The difference between the target output* y_i *and the output* h_i *is termed the error. The total error is the sum of the errors over all training patterns and provided to the backward pass.*

***Backward pass**: The weights are updated iteratively during training the network to find the minimum of the error function. The result collected at the input unit is the derivative of the network function with respect to x.*

We performed running the forward and backward pass until stopping criterion are satisfied.

3. AUTOENCODER-BASED WEB SERVICE MODEL

In this section, we build the Autoencoder-based Web Service model (AWS) for predicting the missing QoS values based on an Autoencoder, where the input data are partially observed vector of user $x^{(u)}$ or service $x^{(s)}$. Hence, we first generate the partially observed vector inputs $x^{(u)}$, $x^{(s)}$ (presented in definition 5 and 6) and a general model based on Autoencoder was built (Fig 2).

***Definition 5* (Partially observed vector $x^{(u)}$).** *Suppose we have m service users and n Web services, the link between users and services is denoted by the user-service matrix* $R\in\Re^{m\times n}$. *Each user* $u\in U$ $\left(U=\left(u_1,u_2,...,u_m\right)\right)$ *is represented as a partially observed vector* $x^{(u)}=\left(R_{u1},R_{u2},...,R_{un}\right)\in\Re^n$, *where* R_{uj} *represents a QoS value (Response-time, Throughput, etc.,) of the service j observed by the user u. Vector* $x^{(u)}$ *is denoted by a partially observed of user u.*

***Definition 6* (Partially observed vector $x^{(s)}$).** *Suppose we have m service users and n Web services, the link between users and services is denoted by the user-service matrix* $R\in\Re^{m\times n}$. *Each service* $s\in S$ $\left(S=\left(s_1,s_2,...,s_n\right)\right)$ *is represented as a partially observed vector* $x^{(s)}=\left(R_{1s},R_{2s},...,R_{ms}\right)\in\Re^m$, *where* R_{is} *represents a QoS value (Response-time, Throughput, etc.,) of the service s observed by the user i. Vector* $x^{(s)}$ *is denoted by a partially observed of service s.*

***Definition 7* (Autoencoder–Based Web Service (AWS)).** *An AWS (see in Fig 2) is a 10 tuples* $AWS = \langle m, n, l, W_0, W_1, U, S, R, x^{(.)}, F \rangle$, *where*

- m *is the total number of service users*
- *n is the total number of Web services.*
- l *represents the* l *-th layer in the model.*
- W_0, W_1, b_0 and b_1 *represent the transformations* $W_0 \in \Re^{d\times k}, W_1 \in \Re^{k\times d}$, *and biases* $b_0 \in \Re^{k}, b_1 \in \Re^{d}$, where $k \in N_+$.
- *U represents a set of users* $\left(U = \left(u_1, u_2, ..., u_m\right)\right)$ *and S represents a set of services* $\left(S = \left(s_1, s_2, ..., s_n\right)\right)$.
- R *is denoted by the user-service matrix* $R \in \Re^{m\times n}$.
- x *represents the input values of the neural network, where vector* $x^{(u)}$ *is denoted by a partially observed of user u in* F^m *and vector* $x^{(s)}$ *is denoted by a partially observed of service s in* F^n.

Let a^l is the activation of layer *l*, we can compute this neural network given by

$$a^l = f\left(W_1.x + b_1\right) \qquad (2)$$

And the output of layer *l is*

$$\begin{aligned} h_{W_0,b_0}(x) &= f\left(W_0.a^l + b_0\right) \\ &= f\left(W_0.f\left(W_1.x + b_1\right) + b_0\right) \qquad (3) \end{aligned}$$

where $f(.)$ is activation functions (the sigmoid activation*).*

It is noted that we utilize the plate notation to represent *n* copies of the neural network (for each service or user) as follows: Let the weight matrix W_l and the bias vector b_l which are tied across all copies, let n_l is the number of layers and layer L_{n_l} is the output layer.

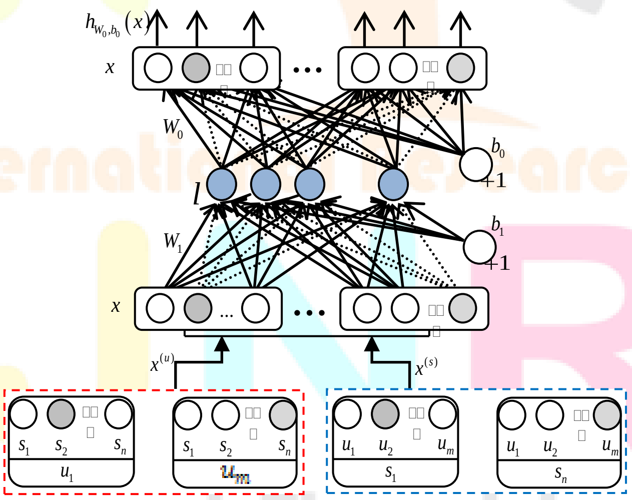

Figure 2: An autoencoder-based Web service model (AWS)

4. AWS-BASED QOS PREDICTION

The AWS model is designed with two predictors for predicting the missing QoS values as follows: User-based AWS (is denoted by U-AWS) and Service-based AWS (is denoted by S-AWS), ///where U-AWS and S-AWS take input from the partially observed vector $x^{(u)}$ and $x^{(s)}$, respectively. Project AWS into a low-dimensional hidden space, and then use Backpropagation algorithm to reconstruct $x^{(u)}$, $x^{(s)}$ in the output space.

4.1 U-AWS-based QoS Prediction

The AWS model has many parameters (links between nodes), so we try to optimize the parameters W_l, b_l by minimizing the reconstruction error as follows.

$$\min_{\{W_l\},\{b_l\}} \sum_{u=1}^{m} \left\| x^{(u)} - h_{W_{L_{nl}}, b_{L_{nl}}}(x^{(u)}) \right\|_2^2 \tag{4}$$

The Backpropagation algorithm is used to learn the set of the parameters W_l, b_l. A U-AWS is presented as per Eq. (4) with two main tasks: *First*, each user $x^{(u)}$ is partially observed, which the weights are updated during Backpropagation that are associated with observed inputs. *Second*, using regularization the learned parameters are to decrease the magnitude of the weights so as to prevent over-fitting. Formally, the objective function of U-AWS to solve the following optimization problem

$$\min_{\{W_l\},\{b_l\}} \sum_{u=1}^{m} \left\| x^{(u)} - h_{W_{L_{nl}}, b_{L_{nl}}}(x^{(u)}) \right\|^2 + \frac{\lambda}{2} \sum_{l} \left\| W_l \right\|_F^2 \tag{5}$$

where λ controls the regularization strength and $\|.\|_F$ denotes the Frobenius norm. In total, *U-AWS* requires the estimation of *2nk+k+n* parameters. To make recommendations, given the learned parameters $\hat{W}_{L_{nl}}$, $\hat{b}_{L_{nl}}$ and *U-AWS* predicted the missing QoS value of user *u* and service *s* as follows:

$$P_U(R_{us}) = \left(h_{\hat{W}_{L_{nl}}, \hat{b}_{L_{nl}}}(x^{(u)}) \right)_s \tag{6}$$

4.2 S-AWS-based QoS Prediction

Similar to U-AWS, optimizing the parameters W_l, b_l, the reconstruction error is computed as follows:

$$\min_{\{W_l\},\{b_l\}} \sum_{s=1}^{n} \left\| x^{(s)} - h_{W_{L_{nl}}, b_{L_{nl}}}(x^{(s)}) \right\|_2^2 \tag{7}$$

And the objective function for S-AWS is given by

$$\min_{\{W_l\},\{b_l\}} \sum_{s=1}^{n} \left\| x^{(s)} - h_{W_{L_{nl}}, b_{L_{nl}}}(x^{(s)}) \right\|^2 + \frac{\lambda}{2} \sum_{l} \left\| W_l \right\|_F^2 \tag{8}$$

S-AWS predicted the missing QoS value of user *u* and service *s* as follows:

$$P_S(R_{us}) = \left(h_{\hat{W}_{L_{nl}}, \hat{b}_{L_{nl}}}(x^{(s)}) \right)_u \tag{9}$$

5. EXPERIMENTS

5.1. Data set description and Evaluation Metric

We use a public real world Web service QoS dataset for our experiments. We conducted experiments the algorithm from two different datasets. This dataset is collected by Zibin Zheng, et al.

Dataset 1: The dataset monitor 100 Web services by using 150 distributed computer nodes located all over the world. The obtained results are contained in 150 files, where each file includes 10,000 Web Services invocations on 100 Web services by a service user, there are totally more than 1.5 million Web service invocations and each line in the file is a web service invocation result. More details can be found in [24].

Dataset 2: The dataset consists two files (seen as two data sets), each dataset contains QoS records of 1,974,675 Web service invocations which are executed by 339 distributed users on 5825 web services and are transformed into two user-service QoS matrices (Response time and Throughput). More details can be found in [18,25].

Each dataset is divided into two parts. A training set (90%) and a test set (10%). We randomly removed entries 10% on the data set as missing QoS values.

In order to measure the prediction accuracy of the model proposed, we use the Mean Absolute Error (MAE) and the Root Mean Square Error (RMSE). The evaluation rules are given by the following forms:

$$MAE = \frac{\sum_{u,s} \left| R_{us} - P\left(R_{us}\right) \right|}{N}$$
$$RMSE = \sqrt{\frac{\sum_{u,s} \left(R_{us} - P\left(R_{us}\right) \right)^2}{N}} \tag{10}$$

where R_{us} is the real QoS value observed by service user *u* on Web service *s*, $P\left(R_{us}\right)$ is the predicted QoS value observed by service user *u* on Web service *s* and N is the sum of the absolute difference over all pairs.

5.2 Approaches to compare

In this paper, we evaluated three AWS-based models: *U-AWS*, a user-based AWS, *S-AWS*, a service-based AWS. To prove their effectiveness, we compare with some well-known models as follows:

- *UPCC*: User-based CF method [19] is the classical method that applies similar users for predicting the missing QoS values.
- *IPCC*: Item-based CF method [20] employs similar Web service items for predicting the missing QoS values.
- *RBM*: RBM-based CF model [11] is a probabilistic graphical model based on Restricted Boltzmann machines (RBM), it estimates parameters of model by maximizing likelihood and training RBM by using Contrastive Divergence.
- *NMF* (Non-negative matrix Factorization): This method is proposed by Lee et al. in [21, 22]. It is used to enforce the constraint that the factorized factors must be non-negative. And it is also widely used in the CF community.
- *PMF* (Probabilistic matrix factorization): This method is proposed by Salakhutdinov and Minh in [23]. It is used a user-item matrix based on PMF for the recommendation.

5.3 Experimental results

To perform experiments, we first set an appropriate latent-variable model *k=400* (the best performance) and tuned $\lambda = \{0.001, 0.91, 0.11, 50, 500\}$

Tables 1 and 2 report the MAE and RMSE values of all methods on two datasets. The results show that the item-based approaches (*IPCC*) have performance better the user-based approaches (*UPCC*). This observation indicates that similar services provide more information than similar users for the prediction in user-service matrix.The results also show that PMF performs better than RMB and NMF. Especially, two predictions (*U-AWS* and *S-AWS*) in the AWS model obtain smaller MAE and RMSE values consistently, which indicates better prediction accuracy. The MAE and RMSE values of Response time in Table 2 are larger than those of the Throughput, since Response time provides more similar values than Throughput.

Table 1: Prediction Quality on dataset 1.

Methods	Response time	
	MAE	*RMSE*
UPCC	1.0975	1.3756
IPCC	1.0237	1.2897
NMF	0.9192	1.1631
RMB	0.9001	1.1391
PMF	0.7875	1.0973
U-AWS	**0.7832**	**0.9037**
S-AWS	**0.7701**	**0.9011**

Table 2: Prediction Quality on dataset 2.

Methods	Response time		Throughput	
	MAE	*RMSE*	*MAE*	*RMSE*
UPCC	0.7551	0.9873	0.6113	0.7457
IPCC	0.7197	0.9521	0.6119	0.7319
NMF	0.6010	0.8979	0.4759	0.6447
RMB	0.5928	0.8731	0.3871	0.6301
PMF	0.5821	0.8765	0.3827	0.6300
U-AWS	**0.5711**	**0.8710**	0.3839	**0.6290**
S-AWS	**0.5613**	**0.8700**	**0.3767**	**0.6155**

We conducted experiments on dataset 2 (Throughput) for comparing two predictions *U-AWS* and *S-AWS*. The results in Figure 3 show that for MAE: The smallest value of *U-AWS* is *0.3939* and *S-AWS* is *0.3767*. For RMSE: The smallest value of *U-AWS* is *0.6290* and *S-AWS* is *0.6155*.

From the above results we can conclude that the model AWS give better results than other models. The experiments also show that *S-AWS* have better performance than *U-AWS*. This result also demonstrates that prediction based on services (*S-AWS*) has the density and thickness more than predicted based on users (*U-AWS*), leading to higher reliability.

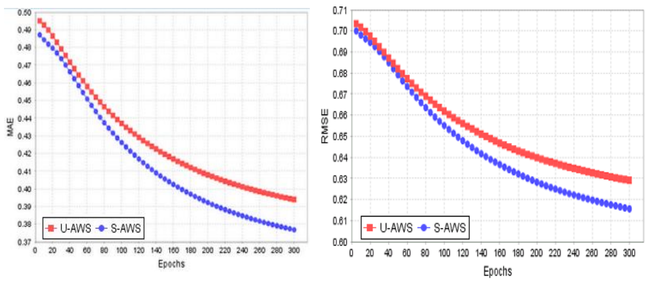

Figure 3: The comparison results of two predictions on dataset 2

6. CONCLUSION

Assessing the quality of web services is one of the most common fields in the research community, especially predicting the missing QoS values to improve the quality for recommendations. Deep Learning is a prospective research field about image recognition, natural language processing, etc. In this paper, we proposed a new model based on Deep learning, the model is constructed based on Autoencoder and used the Backpropagation algorithm to train and the Gradient descent rule for updating the weights. The model can perform on two predictions: U-AWS and S-AWS. The experimental results on two datasets show that the proposed model is more effective than other methods and S-AWS has better performance than U-AWS.

In addition, we also orient some work to be done in the future: We will construct the combined model for two predictions U-AWS and S-AWS.

REFERENCES

[1] Wang H, Qin W, Xin C and Qi Y. Integrating Gaussian Process with Reinforcement Learning for Adaptive Service Composition. Service-Oriented Computing, Springer. 2015, pp.203-217.
[2] Zibin Z, Hao M, Michael R L, and Irwin K. Wsrec: A collaborative filtering based web service recommender system. IEEE International Conference on Web Services (ICWS'09). Los Angeles, CA, 2009, pp.437–444.
[3] Joyce E H, Maude M and Marta R. TQoS: Transactional and qosaware selection algorithm for automatic web service composition. IEEE Transactions on Services Computing. 2010, pp. 73–85.
[4] Ping W. Qos-aware web services selection with intuitionistic fuzzy set under consumer's vague perception. Expert Systems with Applications. Vol.36, No.3, pp. 4460–4466, 2009.
[5] Liu Q, Xiong Y and Huang W. Combining User-Based and Item-Based Models for Collaborative Filtering Using Stacked Regression. Chinese Journal of Electronics, Vol.23, No.4, pp.712-717, 2014.
[6] Zhijun Z and Hong L. Application and Research of Improved Probability Matrix Factorization Techniques in Collaborative Filtering. International Journal of Control and Automation. Vol.7, No.8, pp.79-92, 2014.
[7] Gasbor T, Istvan P, et al. Scalable Collaborative Filtering Approaches for Large Recommender Systems. Journal of Machine Learning Research. Vol.10, pp.623-656, 2009.
[8] Tianqi C, Weinan Z, Qiuxia L, et al. SVDFeature: A Toolkit for Feature-based Collaborative Filtering. Journal of Machine Learning Research, Vol.13, No.1, pp.3619-3622, 2012.
[9] Fabio A. Efficient top-n recommendation for very large scale binary rated datasets. Proceedings of the 7th ACM conference on Recommender systems. 2013, pp.273-280.
[10] Li D. An overview of deep-structured learning for information processing. In proceedings of Asian-Pacific Signal and Information Processing–Annual Summit and Conference. 2011, pp. 1-14.
[11] Salakhutdinov R, Mnih A, Hinton G. Restricted boltzmann machines for collaborative filtering. In proceedings of the 24th International Conference on Machine Learning. 2007, pp. 791–798.
[12] Phung D Q, Venkatesh S, et al. Ordinal boltzmann machines for collaborative filtering. In proceedings of the 25th Conference on Uncertainty in Artificial Intelligence. 2009, pp. 548–556.
[13] Aaron V D O, and Sander D and Benjamin S. Deep content-based music recommendation. In advances in Neural Information Processing Systems 26. pp.2643–2651, 2013.
[14] Asela G and Christopher M. Tied boltzmann machines for cold start recommendations. In proceedings of the 2008 ACM Conference on Recommender Systems. 2008, pp.19–26.
[15] Yuanxin O, Wenqi L, Wenge R and Zhang X. Autoencoder-Based Collaborative Filtering. Neural Information Processing, Springer Link. Vol.8836, pp.284-291, 2014.
[16] Raul R. Neural networks: A systematic introduction. SpringerVerlag Berlin Heidelberg.1996.
[17] Yoshua B. Learning Deep Architectures for AI. Foundations and Trends in Machine Learning. Vol.2, No.1, 1-27, 2009.

[18] Zibin Z, Yilei Z and Michael R L. Distributed qos evaluation for real-world web services. In Proceedings of the 8th International Conference on Web Services (ICWS2010), Miami, Florida, USA, July 5-10, 2010, pp.83-90.
[19] Breese J S, Heckerman D, Kadie C. Empirical analysis of predictive algorithms for collaborative filtering. In proceedings of the 14th annual conference on uncertainty in artificial intelligence (UAI'98), Morgan Kaufmann Publishers, San Francisco. 1998, pp.43–52.
[20] 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 (WWW'01), ACM Press, New York. 2001, pp.285–295.
[21] Lee DD, Seung HS. Learning the parts of objects by Nonnegative Matrix factorization. Natrure 401. pp.788-791, 1999.
[22] Lee DD, Seung HS. Algorithms for Nonnegative Matrix factorization. In proceedings of the advances in neural information processing systems. Denver, USD, 2000, pp.556-562.
[23] Salakhutdinov R, Mnih A. Probabilistic matrix factorization. In proceedings of the advances in neeural information processing systems. 2000, pp. 1257-1264.
[24] Zibin Z, Yilei Z and Michael R L. Collaborative Reliability Prediction for Service-Oriented Systems. In Proceedings of the ACM/IEEE 32nd International Conference on Software Engineering (ICSE2010), Cape Town, South Africa, May 2-8, 2010, pp.35-44.
[25] Yilei Z, Zibin Z and Michael R L. Exploring Latent Features for Memory-Based QoS Prediction in Cloud Computing. In Proceedings of the 30th IEEE Symposium on Reliable Distributed Systems (SRDS ////2011), Madrid, Spain, Oct.4-7, 2011.