Estimating Web Service Quality of Service Parameters using Source Code Metrics and LSSVM

Lov Kumar
NIT Rourkela, India
lovkumar505@gmail.com

Santanu Rath
NIT Rourkela, India
skrath@nitrkl.ac.in

Ashish Sureka
Ashoka University, India
ashish.sureka@ashoka.edu.in

***Abstract*—We conduct an empirical analysis to investigate the relationship between thirty seven different source code metrics with fifteen different Web Service QoS (Quality of Service) parameters. The source code metrics used in our experiments consists of nineteen Object-Oriented metrics, six Baski and Misra metrics, and twelve Harry M. Sneed metrics. We apply Principal Component Analysis (PCA) and Rough Set Analysis for feature extraction and selection. The different sets of metrics are provided as input to the predictive model generated using Least Square Support Vector Machine (LSSVM) with three different types of kernel functions: RBF, Polynomial, and Linear. Our experimental results reveal that the prediction model developed using LSSVM method with RBF kernel function is more effective and accurate for prediction of QoS parameters than the LSSVM method with linear and polynomial kernel functions. Furthermore, we also observe that the predictive model created using object-oriented metrics achieves better results in comparison to other sets of source code metrics.**

***Index Terms*—LSSVM, Machine Learning, Service Oriented Computing, Source Code Metrics, Web Services, Quality of Service (QoS)**

I. Introduction

Service Oriented Computing and Architecture (SOA) paradigm consists of assembling and combining loosely coupled software components called as services for developing distributed system. Prediction of Web Service QoS parameters is important for both the developers and consumers of the service [7]. One of the major objectives of a Web Service provider is the ability to estimate and subsequently improve the QoS parameters associated with the given Web Services. One of the approaches for estimating and improving the QoS parameters is to compute source-code metrics during the development phase. Predicting quality of Object-Oriented (OO) Software System using different kinds of source code metrics is an area which has attracted several researchers' attention in the past [2][22][11][5]. However, predicting QoS parameters for Web Services using source code metrics is a relatively unexplored area. In the study presented in this paper, we conduct an experiment on fifteen different quality of service parameters such as Availability, Best Practices, Compliance, Conformity, Documentation, Interoperability, Latency, Maintainability, Modularity, Response Time, Reusability, Reliability, Successability, Throughput, and Testability, using thirty seven different source code metrics on a dataset consisting of two hundred real-world Web Services. We compute thirty seven source code metrics and use them as input to develop a model using LSSVM method with three different types of kernel functions: linear kernel, polynomial kernel and RBF kernel. LSSVM method is a least square version of support vector machine (SVM) and is based on statistical learning theory [18].

The overall effectiveness and performance of the QoS parameter prediction models depends on the subset of source code metrics used as input to develop the statistical models. In our work, six different sets of source code metrics: all metrics (AM) for source code (thirty seven metrics), Baski and Misra Metrics suite (BMS), Harry M. Sneed Metrics suite (HMS), Object-Oriented source code metrics (OOM), metrics extracted using Principal Component Analysis (PCA) method and metrics selected using Rough Set Analysis (RSA) are considered as input to develop a QoS prediction model. The study presented in this paper is an extension of our previous work on predicting QoS parameters using Extreme Learning Machines [7]. While ELM has been used in the past for QoS parameter prediction, the application of LSSVM is novel and unique in context to exiting work. This research contributions of the study presented in this paper are the following:

1) Application of 37 source-code metrics for prediction of 15 different Web Service QoS parameters by using LSSVM machine learning classifier with three different variants of kernel functions.
2) Application of two feature selection techniques i.e., PCA and RSA to select suitable set of source code metrics for building a predictive model.

II. Related Work

Several researchers have investigated the impact of Object-Oriented (OO) source code metrics on software quality and observed that OO metrics have a strong influence on software quality attributes. Research shows that the quality of OO software can be estimated using several source code metrics [5] [10] [3] [9][8]. Bingu Shim et al. have defined five different quality parameters i.e., effectiveness, flexibility, discoverability, reusability and understandability for service oriented applications [16]. Mikhail et al. have defined SCMs in order to measure the structural coupling & cohesion of service-oriented systems [14][15]. Vuong Xuan Tran et al. proposed a novel approach to design and develop QoS systems and describe an algorithm to evaluate its ranking in order to compute the quality of Web services [19]. Cristian Mateos et al. analyzed

the available approaches to remove undesirable anti-patterns using code-first [12]. Ping Wang proposed another decision model under obscure data to choose a Web Service [21].

III. Research Framework

A. Dependent Variables- QoS Parameters

Al-Masri et al. define 9 quality of service parameters of Web Services. They compute the QoS parameters using Web service benchmark tools. The QoS parameters are: Availability (AV), Best Practices (BP), Compliance (CP), Documentation (DOC), Latency (LT), Response Time (RT), Reliability (REL), Successability (SA), Throughput (TP), Maintainability, Modularity, Reusability, Testability, Interoperability and Conformity. These QoS parameters are the dependent variables for the machine learning model.

B. Predictor Variables: Source Code Metrics

Three different types of metrics suite are considered as independent or predictor variables

1) Object-Oriented Source Code Metrics: We compute nineteen different Object-Oriented source code metrics from the bytecode of the compiled Java files of the Web Services in our experimental dataset using CKJM extended tool[1] [5]. CKJM extended is an extended version of tool for calculating Chidamber and Kemerer Java source code metrics and many other metrics such as average method complexity, McCabe's Cyclomatic Complexity, lack of cohesion among the classes. Java class files from the WSDL file are generated using WSDL2Java Axis2 code generator[2], which is available as an Eclipse plug-in. We then compiled the Java files to generate the bytecode for computing the size and structure of software metrics using the CKJM extended tool.

2) Henry M. Sneed WSDL Metric Suite: Sneed et al. develop a tool for measuring Web Service interfaces [17][6]. The suite primarily consists of six different source code metrics to measure complexity of service interfaces: Data Flow Complexity, Interface Relation Complexity, Interface Data Complexity, Interface Structure Complexity, Interface Format Complexity and Language Complexity. These metrics are statically computed from a service interface in WSDL. The metrics are based on analyzing the WSDL schema elements [17][6].

3) Baski and Misra Metrics: Baski and Misra proposed a tool to compute six different complexity metrics of WSDL file [4]. These metrics are based on the analysis of the structure of the exchanged messages described in WSDL file which becomes the basis for computing the data complexity. These metrics are based on analyzing the WSDL and XSD schema elements [4].

C. Experimental Dataset

In our study, the Web Service dataset collected by Al-Masri et al. [3] is used to measure the performance of the proposed LSSVM based approach. The Web Service dataset provides the quality of service parameters values such as response time, availability, throughput, compliance, latency for 2507 Web Services. The QoS parameters values by the dataset provider are computed using Web service benchmark tools [1]. In this study, we use 200 Web Services for the analysis. The reason for selection of 200 web-services is stated in our earlier work [7] as the study presented in this paper is an extension of the previous work.

[1]http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/

[2]https://sourceforge.net/projects/wsdl2javawizard/

[3]http://www.uoguelph.ca/~qmahmoud/qws/

D. Feature Extraction using Principal Component Analysis (PCA)

We perform feature extraction using Principal Component Analysis (PCA). The main motivation of using PCA is for transforming high dimension data space into lower dimension data space. The lower dimension data consists of the most significant features [20]. We label the new metrics (or features) after applying PCA as principal component domain metrics. Figure 1 displays the steps followed by us to extract the feature set using PCA.

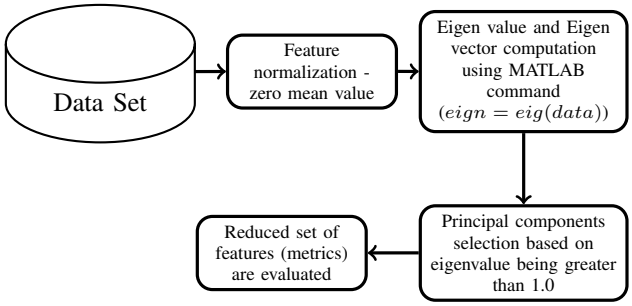

Fig. 1: Sequence of Steps for Applying PCA

We apply PCA with varimax rotation technique on all the software metrics. Table I shows the result and outcome of PCA with varimax rotation method. Table I reveals the relationship between domain metrics and original software metrics. For each principal component (PC), % variance, % cumulative and interpreted metrics set are presented in Table I.

TABLE I: Principle Component Analysis (PCA) Results

PC	Eigenvalue	variance %	% Cumulative	Interpreted Metrics
PC1	6.40	17.30	17.30	Ce, Ca, RFC, CBO, LCO, LCOM3, CAM, DAM
PC2	5.8	15.76	33.06	DP, FP, OP, MRS, OPS, IDFC, IRC
PC3	3.67	9.94	43.00	CE, MiRV, MDC, MeRV, DW, MR
PC4	3.39	9.16	52.17	ILC, DMR, ISC, IDC
PC5	3.34	9.03	61.2	MOA, CBM, IC
PC6	2.50	6.77	67.98	MFA, NOC, DIT, IFC
PC7	2.23	6.02	74.00	NPM, WMC
PC8	2.14	5.79	79.79	AMC, MRV
PC9	1.36	3.7	83.5	LCOM

E. Feature Selection using Rough Set Analysis (RSA)

Before the application of RSA, the input data need to be categorized. In our study, K-means clustering approach is applied for the purpose of data categorization. In the approach, the data belonging to a particular cluster are grouped under a single category or class. After the application of K-means clustering approach, we obtain 3 clusters and the data

were categorized into three groups: High, Medium, and Low correlation. Figure 1 shows the steps followed to identify the best set (in-terms of relevance) of features using RSA.

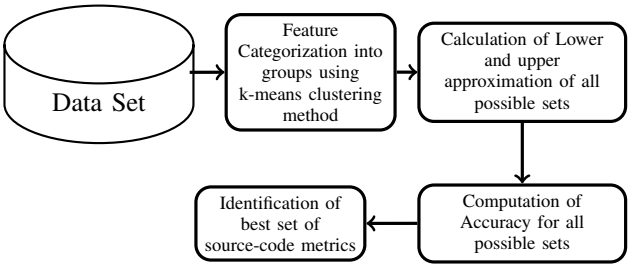

Fig. 2: Rough Set Analysis (RSA) based Feature Selection

Table II displays the selected set of source-code metrics for every QoS parameter. The resultant set is derived from the original set of 37 source-code metrics using RSA approach.

TABLE II: Source Code Metrics Identified using Rough Set Analysis (RSA)

QoS	Selected Metrics
Availability	MiRV, Ca, CC, CAM, IC, MFA, LC, SC, LCOM3, WMC, FC, MeRV
Response Time	Ca, DMR, LC, SC, WMC, MFA, CC, IC, CAM, LCOM3
Successability	CAM, LCOM3, DAM, FC, LC, DFC, MRV, ME, SC, WMC, LCO, MOA
Throughput	MiRV, Ce, CC, CAM, ME, MFA, LC, SC, CBM, MRV, FC, MeRV, MOA
Compliance	MiRV, NPM, CC, WMC, CAM, MOA, SC, LC, FC, DFC, ME, Ca, MRV, DAM
Reliability	LCOM3, MFA, FC, LC, DFC, CAM, SC, WMC, LCO, MOA
Latency	IC, FC, LC, DMR, MRV, MOA, ME, CAM, DC, DFC, NOC, LCO, NPM
Best Practices	MiRV, Ca, CC, CAM, ME, MFA, LC, SC, MRV, FC, MOA, WMC, DFC, NPM
Maintainability	CBM, DP, LCOM3, MFA, Ce, CAM, MOA
Documentation	CC, LC, ME, IC, SC, CAM, Ca, DFC, MRV, WMC, MeRV, FC, NPM
Reusability	LCOM3, FC, MDC, LCOM, DMR, SC, LC, DFC
Modularity	AMC, LC, DMR, SC, Ca, IC, DFC, ME, DP, MiRV, MOA, MRV, WMC
Interoperability	MiRV, CC, SC, MeRV, LC, WMC, MFA, DIT, CBO
Testability	FC, CC, RFC, ME, NOC, MiRV, DIT, SC, LC
Conformity	CAM, Ca, ME, DFC, FC, WMC, MRV, LC

F. Effectiveness of Metrics

Once we have the QoS data, the relationship or degree of association between source code metrics and QoS parameters can be determined. The set of source code metrics are considered as independent variables and QoS parameters are considered as a dependent variables. In our experiments, six different set of source metrics (all metrics (AM), Baski and Misra metrics suite, Henry M. Sneed WSDL metric suite, object-oriented metrics suite, selected set of metrics principal component analysis (PCA), and selected set of metrics using rough set analysis (RSA) are considered as input to develop fifteen QoS parameters (Response Time Availability, Throughput, Successability, Reliability, Compliance, Best Practices, Latency, Documentation, Maintainability, Modularity, Reusability, Testability, Interoperability, and Conformity) prediction models. Figure 3 shows the independent and dependent variables used for QoS parameter prediction model. From Figure 3, we infer that a total of eight different sets of independent variables are possible for each QoS parameter.

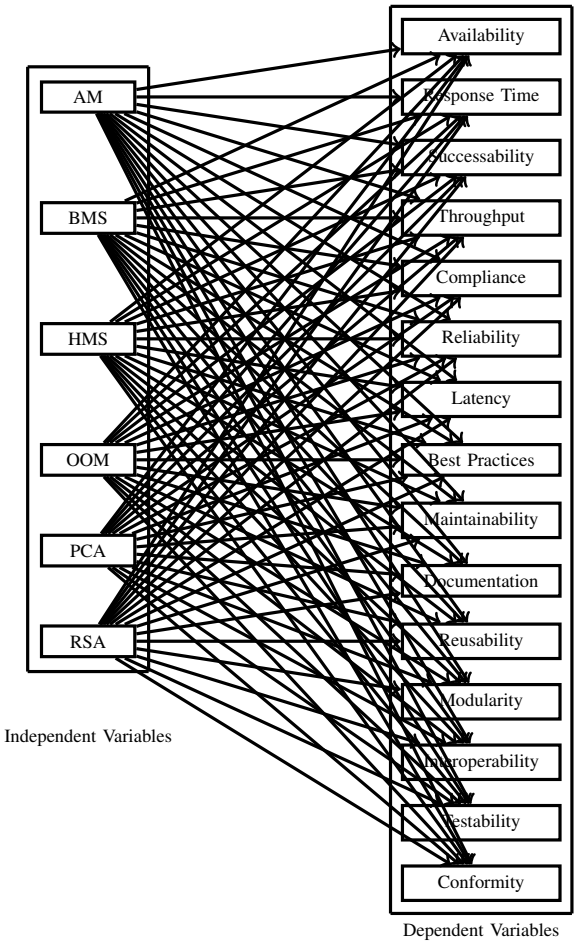

Fig. 3: Dependent and Independent Variables

IV. Proposed Machine Learning Based Approach

Least Square Support Vector Machines (LSSVM) are supervised learning methods having wide range of applications doe classification, regression and outliers detection problems [18]. In our experiments, we use LSSVM as regression technique to generate models for predicting QoS parameters. The different set of source code metrics are used a input of the models. We also examine LSSVM different kernel functions to investigate if we can achieve better result and compare the performance of various kernel functions.

The block diagram displayed in Figure 4, illustrates the sequence of steps used to determine the predicted quality of service (QoS) parameters using the LSSVM method with three different types of kernel functions. In our work, the following steps are performed to generate quality of service prediction models:

1) Source code metrics computation for all the Web Services in the data-set as described in Section III-B.
2) Selection of suitable set of source code metrics using PCA and RSA feature extraction and selection techniques.
3) Predictive model generation by considering source code metrics as input to estimate fifteen different quality of service parameters.
4) Identification of performance measures to evaluate the predictive ability and effectiveness of quality of service prediction models.
5) Application of validation methods to determine the true

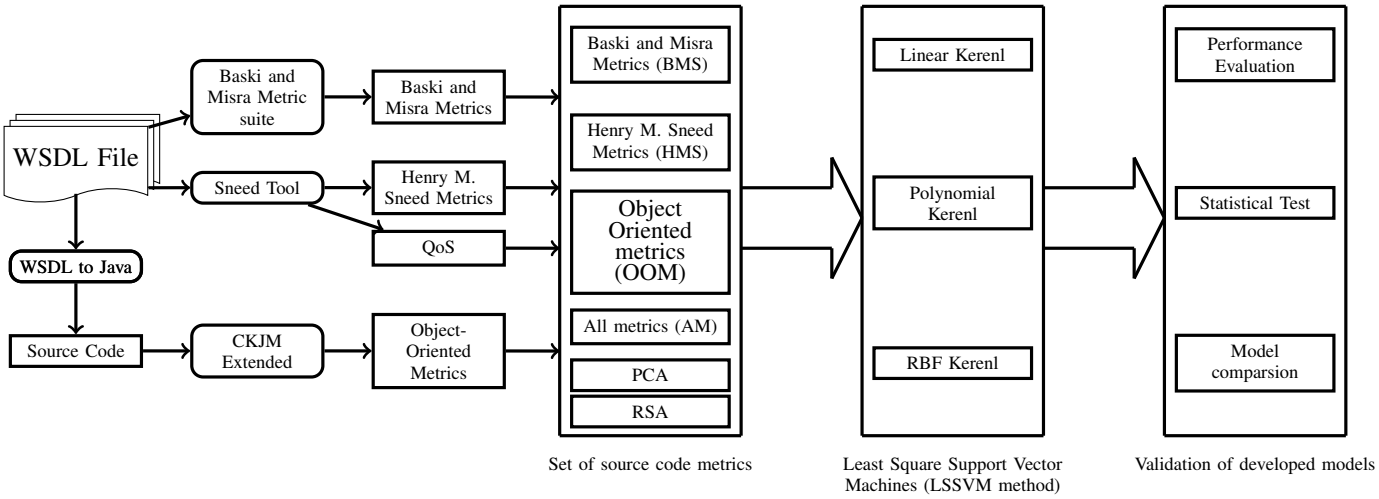

Fig. 4: Proposed Steps Used for the QoS Prediction (A Multi-Step Process)

predictive applicability of the predictive models.

6) Application of rigorous statistical significance tests to compare the performance of one prediction technique over other approaches and also determine the superiority of one set of source code metrics over the other sets.

A. Computation of Source Code Metrics

During our experimental analysis, we consider 200 Web Services as the experimental dataset (same as used by Kumar et al. [7]). In this work, WSDL interface complexity metrics and WSDL complexity metrics are computed using Henry Sneed metrics tool and Baski and Misra metrics tool respectively. Then we use the wsimport [4] tool to parse WSDL document file of a Web Service and generate its corresponding Java class. This involves extracting the Java source code implementing the service. As shown in Figure 4, we compiled the Java files to generate the bytecode for computing the size and structure software metrics using the CKJM extended tool.

B. Feature Selection Method

In our experiments, we examine 37 different software metrics (Chidamber and Kemerer, Harry M. Sneed, Baski & Misra)) to predict 15 different QoS attributes. It is very essential to remove irrelevant and unimportant source code metrics out of these source code metrics so that only relevant source code metrics are included in the construction of QoS prediction models. In order to achieve the stated objective, we consider two different features selection techniques: principal component analysis and rough set analysis for feature selection.

C. Prediction Techniques

In this work, we have use LSSVM with three different kernel functions to develop QoS prediction model.

[4]http://docs.oracle.com/javase/6/docs/technotes/tools/share/wsimport.html

D. Performance Parameter

In order to evaluate the QoS prediction model, various performance parameters are defined in the machine learning literature to measure the effectiveness of the QoS prediction models. In this work, we consider three different performance parameters: Mean Magnitude Relative Error (MMRE), Mean Absolute Error (MAE), and Root Mean Square Error (RMSE) to evaluate the QoS prediction model [13]. A lower value for these performance parameters denotes an effective prediction model.

E. Validation Method

The objective of the study presented in this paper is to build and apply statistical models to predict different QoS parameters for future releases and unseen similar natured projects. Hence, it is necessary to validate the developed QoS model on a different data-sets than the dataset on which the training is done. In our experiments, we consider the standard k-fold cross validation approach (we take k = 10) to validate the proposed QoS model. In our analysis, we also perform outlier detection analysis to eliminate the extreme values which may add to the noise & bias in the model performance and accuracy results. The outlier analysis is doen based on the following equations:

$$e_i = \begin{cases} \text{if } |y_{ji} - \hat{y_j}| > 3 * \sigma & \text{for Effective outliers} \\ \text{if } |y_{ji} - \hat{y_j}| <= 3 * \sigma & \text{for Non Effective outliers} \end{cases} \quad (1)$$

F. Statistical Significance Tests and Procedures

In order to bring rigour and mitigate threats to validity in our analysis of the results, we apply pairwise t-test approach. We conduct t-test to determine which prediction method and feature selection techniques performs relatively better or does the models perform equally well. We analyze all the results based on the 0.05 significance level, i.e. two models are significantly different (null hypothesis rejected) if the p-value

is less than 0.05 (the cut-off valie) else there is no significant different between them (null hypothesis accepted).

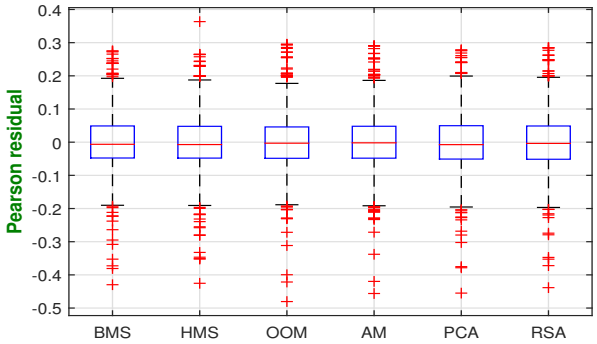

Fig. 5: Box-Plot Visual Analysis (Linear Kernel)

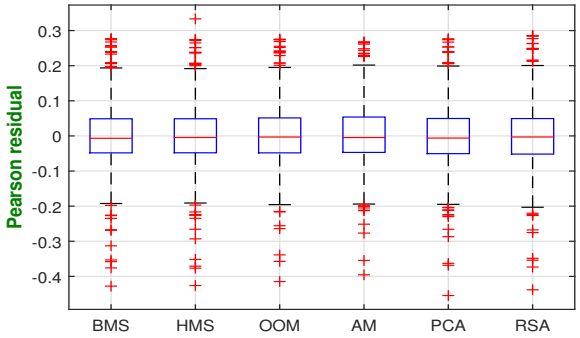

Fig. 6: Box-Plot Visual Analysis (Polynomial Kernel)

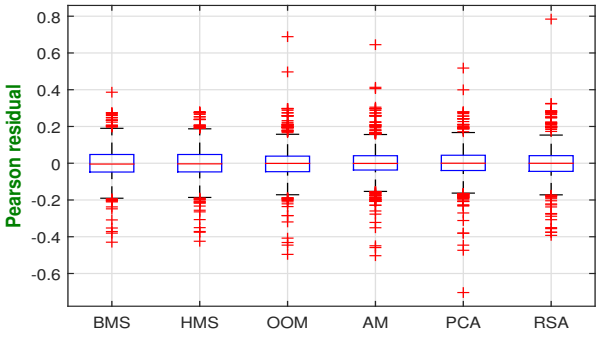

Fig. 7: Box-Plot Visual Analysis (RBF Kernel)

V. Experimental Analysis

In this paper, LSSVM method with three different types of kernel functions have been considered to develop a model to predict fifteen different QoS parameters by considering six different set of source code metrics as input. The detailed descriptions of these source code metrics were already presented in section III. The performance of the developed QoS prediction models are compared using three different types of performance parameters i.e., MMRE, MAE and RMSE.

Table III displays the MMRE, MAE, and RMSE values obtained after applying LSSVM method with linear kernel, polynomial kernel and RBF kernel functions. Table III provides complete and detailed results for all the QoS parameters, feature extraction and selecion techniques, metrics suites and performance evaluation methods. From Table III, we make the following observations:

1) In case of linear kernel function, we observe that the model built by considering selected set of metrics using RSA as input has low values of MMRE, MAE and RMSE in comparison with other sets of metrics. This clearly implies that the performance of the model developed using RSA is much better than the performance of other models i.e., low vales of MMRE, MAE, and RMSE for QoS prediction as compared to a model developed using other sets of metrics.
2) In case of polynomial kernel function, we observe that the model built by considering all metrics has low value of MMRE, MAE and RMSE in comparison to other sets of metrics. We thus infer that the performance of the prediction model developed using all metrics (AM) is much better than other models i.e., low vales of MMRE, MAE, and RMSE for QoS prediction as compared to a model developed using other sets of metrics.
3) In case of RBF kernel function, we notice that the model developed by considering Baski and Misra Metric has low value of MMRE, MAE, and RMSE in comparison with other sets of metrics. This implies that the performance of the model developed using BMS is much better than other models i.e., low vales of MMRE, MAE, and RMSE for QoS prediction as compared to a model developed using other sets of metrics.

In order to have a visual comparion (visual analytics) of the results, we draw the Pearson residual boxplots of models developed using LSSVM method with linear kernel function. The plots are displayed in Figure 5. The middle line of each box in Figure 5 shows the median value of Pearson residual. From Figure 5, we observe that all models built have a median residual value close to zero and the model developed by considering selected set of metrics using RSA has smallest whiskers, narrowest box as well as few numbers of outliers. This shows that model developed by considering selected set of metrics using RSA as input results in better performance as compared to other metrics. Figure 6 shows the Pearson residual boxplots of models developed using LSSVM method with polynomial kernel function. From Figure 6, we infer that all models have a median residual value close to zero and the model developed based on selected set of metrics using AM has smallest whiskers, narrowest box as well as few numbers of outliers. This result shows that model developed by considering AM as input results in better performance as compared to others.

Figure 7 shows the Pearson residual boxplots of models developed using LSSVM method with RBF kernel function. From Figure 7, it is observed that all developed models have

TABLE III: Performance Matrix for LSSVM method with Linear, Polynomial and RBF Kernel and All Different Metrics Suites

	Availability	*Response Time*	*Successability*	*Throughput*	*Compliance*	*Reliability*	*Latency*	*Best Practices*	*Maintainability*	*Documentation*	*Reusability*	*Modularity*	*Interoperability*	*Testability*	*Conformity*
Linear Kernel															
	MMRE														
AM	0.32	0.32	0.35	0.74	0.60	0.68	0.56	0.46	0.32	0.90	0.29	0.14	0.16	0.13	0.06
OOM	0.31	0.33	0.34	0.73	0.59	0.69	0.51	0.36	0.39	0.89	0.46	0.44	0.42	0.49	0.25
WIM	0.32	0.34	0.35	0.74	0.61	0.67	0.56	0.38	0.49	0.92	0.42	0.22	0.20	0.18	0.06
WSM	0.31	0.33	0.35	0.75	0.60	0.68	0.56	0.39	0.72	0.91	0.66	0.41	0.35	0.43	0.21
RSA	0.31	0.34	0.34	0.65	0.60	0.69	0.56	0.34	0.84	0.94	0.62	0.32	0.38	0.30	0.10
PCA	0.31	0.34	0.35	0.74	0.61	0.66	0.55	0.37	0.54	0.92	0.61	0.33	0.31	0.33	0.13
	MAE														
AM	0.13	0.09	0.16	0.19	0.25	0.24	0.11	0.19	0.08	0.21	0.09	0.04	0.08	0.04	0.03
OOM	0.13	0.09	0.16	0.18	0.25	0.24	0.1	0.17	0.10	0.21	0.13	0.12	0.18	0.13	0.14
WIM	0.14	0.10	0.17	0.18	0.26	0.24	0.11	0.18	0.11	0.22	0.11	0.06	0.10	0.04	0.03
WSM	0.14	0.10	0.17	0.18	0.26	0.24	0.11	0.18	0.17	0.22	0.16	0.11	0.15	0.11	0.13
RSA	0.13	0.10	0.16	0.16	0.25	0.24	0.11	0.17	0.20	0.23	0.15	0.10	0.18	0.10	0.06
PCA	0.14	0.10	0.17	0.18	0.26	0.23	0.11	0.18	0.13	0.22	0.15	0.10	0.14	0.09	0.08
	RMSE														
AM	0.20	0.13	0.22	0.22	0.29	0.27	0.14	0.23	0.11	0.26	0.11	0.05	0.1	0.05	0.04
OOM	0.19	0.14	0.21	0.22	0.29	0.27	0.13	0.21	0.13	0.27	0.17	0.17	0.22	0.18	0.17
WIM	0.20	0.15	0.22	0.21	0.30	0.27	0.15	0.22	0.16	0.28	0.15	0.08	0.13	0.07	0.05
WSM	0.20	0.15	0.22	0.21	0.29	0.27	0.15	0.22	0.22	0.28	0.21	0.15	0.20	0.16	0.16
RSA	0.19	0.15	0.22	0.2	0.29	0.27	0.14	0.2	0.25	0.29	0.20	0.15	0.23	0.14	0.10
PCA	0.20	0.15	0.22	0.21	0.30	0.26	0.14	0.21	0.17	0.28	0.19	0.14	0.18	0.13	0.11
Polynomial Kernel															
	MMRE														
AM	0.32	0.33	0.35	0.74	0.61	0.68	0.56	0.48	0.73	0.90	0.70	0	0.37	0	0.25
OOM	0.31	0.33	0.34	0.73	0.59	0.70	0.55	0.38	1.02	0.89	0.61	0.59	0.51	0.53	0.37
WIM	0.32	0.34	0.35	0.74	0.61	0.67	0.56	0.38	0.5	0.92	0.62	0.11	0.19	0.19	0.54
WSM	0.31	0.33	0.34	0.73	0.60	0.67	0.56	0.39	0.70	0.91	0.71	0.54	0.32	0.48	0.15
RSA	0.3	0.34	0.34	0.74	0.61	0.69	0.56	0.36	0.85	0.94	0.62	0.47	0.47	0.32	0.60
PCA	0.31	0.34	0.35	0.74	0.61	0.68	0.55	0.38	1.03	0.92	0.65	0.45	0.36	0.38	0.14
	MAE														
AM	0.13	0.09	0.16	0.19	0.26	0.24	0.11	0.20	0.18	0.21	0.17	0	0.16	0	0.11
OOM	0.13	0.09	0.16	0.18	0.25	0.24	0.11	0.18	0.24	0.22	0.15	0.16	0.23	0.14	0.20
WIM	0.14	0.10	0.17	0.18	0.26	0.24	0.11	0.18	0.12	0.22	0.15	0.03	0.09	0.05	0.24
WSM	0.14	0.10	0.17	0.18	0.26	0.24	0.11	0.18	0.16	0.22	0.16	0.14	0.13	0.12	0.09
RSA	0.13	0.10	0.16	0.18	0.26	0.24	0.11	0.18	0.21	0.23	0.15	0.14	0.22	0.10	0.27
PCA	0.14	0.10	0.17	0.18	0.26	0.24	0.11	0.18	0.25	0.22	0.15	0.13	0.16	0.11	0.08
	RMSE														
AM	0.20	0.14	0.22	0.22	0.29	0.28	0.14	0.23	0.22	0.26	0.22	0	0.20	0	0.14
OOM	0.19	0.14	0.21	0.22	0.29	0.28	0.14	0.22	0.30	0.27	0.20	0.22	0.27	0.20	0.22
WIM	0.20	0.15	0.22	0.21	0.30	0.27	0.15	0.22	0.16	0.28	0.19	0.05	0.12	0.08	0.25
WSM	0.20	0.14	0.22	0.21	0.29	0.27	0.15	0.22	0.22	0.28	0.22	0.19	0.18	0.18	0.12
RSA	0.19	0.15	0.22	0.21	0.30	0.28	0.14	0.21	0.26	0.29	0.20	0.19	0.26	0.14	0.29
PCA	0.20	0.15	0.22	0.21	0.30	0.27	0.14	0.21	0.30	0.28	0.20	0.17	0.19	0.15	0.1
RBF Kernel															
AM	0.32	0.32	0.35	0.71	0	0.67	0.38	0.45	0.09	0.38	0.08	0.03	0.08	0.05	0.02
OOM	0.3	0.32	0.34	0.57	0.13	0.68	0.28	0.36	0.22	0.01	0.40	0.42	0.20	0.44	0.13
WIM	0.32	0.34	0.35	0.27	0.42	0.39	0.45	0.29	0.08	0.92	0.16	0.03	0.07	0.02	0.02
WSM	0.3	0.33	0.33	0.50	0.60	0.67	0.56	0.37	0.39	0.89	0.30	0.35	0.16	0.24	0.05
RSA	0.3	0.34	0.34	0.60	0	0.68	0.28	0.35	0.58	0.34	0.10	0	0.16	0.13	0.09
PCA	0.31	0.33	0.35	0.73	0	0.67	0.49	0.36	0.17	0.29	0.27	0.18	0.21	0.22	0.05
	MAE														
AM	0.13	0.09	0.16	0.18	0	0.23	0.07	0.18	0.03	0.09	0.03	0.01	0.04	0.02	0.01
OOM	0.13	0.09	0.16	0.14	0.06	0.24	0.05	0.17	0.06	0	0.11	0.12	0.09	0.11	0.07
WIM	0.14	0.10	0.17	0.07	0.18	0.14	0.09	0.14	0.03	0.22	0.04	0.01	0.04	0.01	0.01
WSM	0.13	0.10	0.16	0.12	0.26	0.23	0.11	0.18	0.10	0.22	0.07	0.09	0.06	0.06	0.02
RSA	0.13	0.10	0.16	0.15	0	0.24	0.06	0.17	0.14	0.08	0.03	0	0.07	0.04	0.05
PCA	0.14	0.10	0.17	0.18	0	0.23	0.09	0.17	0.05	0.07	0.07	0.05	0.09	0.06	0.03
	RMSE														
AM	0.20	0.14	0.22	0.21	0	0.27	0.10	0.22	0.04	0.11	0.04	0.01	0.05	0.02	0.02
OOM	0.19	0.14	0.21	0.17	0.07	0.27	0.07	0.21	0.08	0.01	0.15	0.17	0.12	0.16	0.10
WIM	0.20	0.15	0.22	0.08	0.20	0.16	0.12	0.17	0.04	0.28	0.06	0.02	0.05	0.01	0.02
WSM	0.19	0.15	0.21	0.15	0.29	0.27	0.15	0.21	0.14	0.27	0.10	0.14	0.10	0.09	0.04
RSA	0.19	0.15	0.22	0.18	0	0.27	0.07	0.20	0.19	0.11	0.03	0	0.09	0.07	0.08
PCA	0.20	0.15	0.22	0.21	0	0.27	0.13	0.20	0.06	0.09	0.10	0.08	0.12	0.09	0.04

TABLE IV: Result of t-test: Among Different Metrics Set

P-Value																		
	MMRE						MAE						RMSE					
	AM	OOM	HMS	BMS	RSA	PCA	AM	OOM	HMS	BMS	RSA	PCA	AM	OOM	HMS	BMS	RSA	PCA
AM	1.000	0.063	0.031	0.063	0.031	0.031	1.000	0.031	0.031	0.031	0.031	0.031	1.000	0.031	0.031	0.031	0.031	0.031
OOM	0.063	1.000	0.031	0.031	0.031	0.031	0.031	1.000	0.031	0.031	0.031	0.031	0.031	1.000	0.031	0.031	0.031	0.031
HMS	0.031	0.031	1.000	0.031	0.094	0.031	0.031	0.031	1.000	0.063	0.031	0.031	0.031	0.031	1.000	0.313	0.031	0.031
BMS	0.063	0.031	0.031	1.000	0.031	0.031	0.031	0.031	0.063	1.000	0.031	0.031	0.031	0.031	0.313	1.000	0.031	0.031
RSA	0.031	0.031	0.094	0.031	1.000	0.031	0.031	0.031	0.031	0.031	1.000	0.031	0.031	0.031	0.031	0.031	1.000	0.031
PCA	0.031	0.031	0.031	0.031	0.031	1.000	0.031	0.031	0.031	0.031	0.031	1.000	0.031	0.031	0.031	0.031	0.031	1.000
Mean Difference																		
	MMRE						MAE						RMSE					
	AM	OOM	HMS	BMS	RSA	PCA	AM	OOM	HMS	BMS	RSA	PCA	AM	OOM	HMS	BMS	RSA	PCA
AM	0.000	0.020	-0.392	-0.013	-0.345	-0.268	0.000	0.038	-0.082	-0.068	-0.142	-0.158	0.000	-0.052	-0.067	-0.073	-0.123	-0.148
OOM	-0.020	0.000	-0.412	-0.033	-0.365	-0.288	-0.038	0.000	-0.043	-0.030	-0.103	-0.120	0.052	0.000	-0.015	-0.022	-0.072	-0.097
HMS	0.392	0.412	0.000	0.378	0.047	0.123	0.082	0.043	0.000	0.013	-0.060	-0.077	0.067	0.015	0.000	-0.007	-0.057	-0.082
BMS	0.013	0.033	-0.378	0.000	-0.332	-0.255	0.068	0.030	-0.013	0.000	-0.073	-0.090	0.073	0.022	0.007	0.000	-0.050	-0.075
RSA	0.345	0.365	-0.047	0.332	0.000	0.077	0.142	0.103	0.060	0.073	0.000	-0.017	0.123	0.072	0.057	0.050	0.000	-0.025
PCA	0.268	0.288	-0.123	0.255	-0.077	0.000	0.158	0.120	0.077	0.090	0.017	0.000	0.148	0.097	0.082	0.075	0.025	0.000

TABLE V: t-test: Among different Kernel

P-Value									
	MMRE			MAE			RMSE		
	Lin	Poly	RBF	Lin	Poly	RBF	Lin	Poly	RBF
Lin	1.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000	0.000
Poly	0.000	1.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000
RBF	0.000	1.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000
Mean Difference									
	MMRE			MAE			RMSE		
	Lin	Poly	RBF	Lin	Poly	RBF	Lin	Poly	RBF
Lin	0.000	-0.051	0.155	0.000	-0.015	0.047	0.000	-0.016	0.057
Poly	0.051	0.000	0.206	0.015	0.000	0.063	0.016	0.000	0.074
RBF	-0.155	-0.206	0.000	-0.047	-0.063	0.000	-0.057	-0.074	0.000

a median residual value being close to zero and the model developed by considering selected set of metrics using BMS has smallest whiskers, narrowest box as well as few numbers of outliers. This shows that model developed by considering BMS as input obtained better performance as compared to others.

VI. Comparison of Various Kernel Functions and Metric Suites

We apply Wilcoxon signed rank test to compare the performance of the models using LSSVM method with three different types of kernel functions and different sets of source code metrics. We use Wilcoxon test with Bonferroni correction for comparative analysis.

A. Kernel Functions

Three different types of kernel functions have been applied to develop QoS prediction models. Hence for each of the kernel functions, a total number of three sets (one for each performance parameter) are used. Each set has 90 data points (15 QoS parameters * 6 sets of metrics). The results of Wilcoxon test with Bonferroni correction for all performance parameters are shown in Table V. The upper portion of the Table V shows the p-value between kernel functions and the lower portion shows the mean difference value of performance parameters between different kernels. The Bonferroni correction sets the significance cut-off at $\frac{\alpha}{n}$, where n is number of different pairs (3 kernel functions; n=$^{3technique}C_2 = 3 * 2/2 = 3$) and all results are analyzed at a 0.05 significance level. Hence, null hypothesis is rejected only if the p-value is less than $\frac{0.05}{3} = 0.0167$. The null hypothesis while applying the Wilcoxon test is that there is no significant difference between the two classification techniques. From the result Table V, we observe that **there is a significant difference between the kernel functions**. This interpretation is due to the fact that the p-value is lower than 0.0167 (rejecting the null hypothesis and accpeting the alternate hypothesis). However by closely examining the value of mean difference, **RBF kernel function yields better result as compared to other kernel functions**.

B. Source Code Metrics Sets

In this work, six different set of source code metrics are used as input to develop QoS prediction models. Hence for each set of source code metrics, a total of three sets (one for each performance parameter) are used. Each set has 45 data points (15 QoS parameters * 3 kernel functions). Results of Wilcoxon test with Bonferroni correction for all performance parameter is shown in Table V. The Bonferroni correction sets the significance cutoff at $\frac{\alpha}{n}$, where n is number of different pairs (here 6 sets of metrics; n=$^{6technique}C_2 = 6 * 5/2 = 15$) and all results are analyzed at a 0.05 significance level. Hence, null hypothesis is rejected only if the p-value is less than $\frac{0.05}{15} = 0.0033$. From Table IV, we infer that **there is no significant difference between sets of metrics**. We arrive at this conclusion due to the fact that the p-value is greater than 0.0033 (accepting the null hypothesis). However, by closely examining the value of mean difference, we infer that the **object-oriented Metrics are yields better performance results in comparison to other sets of metrics**.

VII. Threats to Validity

One threat to validity is that the impact on the dependent variable may not be completely attributed to the changes in the independent variable due to overfitting of the predictive model. Another threat to validity is that we have conducted experiments on a limited dataset and the answer result can be biased to the specific dataset.

VIII. Conclusion

Our main conclusions and takeaways are the following:

- Based on our correlation analysis of the metrics, we conclude that there exists a high correlation between Object-Oriented metrics and WSDL metrics.
- From the result of Wilcoxon test with Bonferroni correction, we conclude that there is a statistically significant difference between the performance of the predictive models built using three different LSSVM kernel functions.
- From the rest of Wilcoxon test with Bonferroni correction, we conclude that there is no statistically significant difference between different sets of source code metrics.
- From Table III, we conclude that no one set of source-code metrics dominate the other sets for any QoS parameter and vice-versa.
- By assessing the value of mean difference shown in Table V, we conclude that that the RBF kernel for LSSVM method yields better performance results compared to other kernel functions.
- By assessing the value of mean difference shown in Table IV, we conclude that the object-oriented metrics yields better result compared to other sets of source code metrics.
- Our analysis shows evidences that it is possible to estimate the QoS parameters of Web Services using source code metrics and LSSVM based method.

References

[1] E. Al-Masri and Q. H. Mahmoud. Investigating web services on the world wide web. In *Proceedings of the 17th international conference on World Wide Web*, pages 795–804. ACM, 2008.

[2] V. R. Basili, L. C. Briand, and W. L. Melo. How reuse influences productivity in object-oriented systems. *Communications of the ACM*.

[3] V. R. Basili, L. C. Briand, and W. L. Melo. A validation of Object-Oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, October 1996.

[4] D. Baski and S. Misra. Metrics suite for maintainability of extensible markup language web services. *IET Software*, 5(3):320–341, 2011.

[5] S. R. Chidamber and C. F. Kemerer. A metrics suite for Object-Oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, June 1994.

[6] J. L. O. Coscia, M. Crasso, C. Mateos, and A. Zunino. Estimating web service interface quality through conventional object-oriented metrics. *CLEI Electron. J*, 16(1), 2013.

[7] L. Kumar, S. K. Rath, and A. Sureka. Predicting quality of service (qos) parameters using extreme learning machines with various kernel methods. In *Quantitative Approaches to Software Quality*, page 11, 2016.

[8] L. Kumar, S. K. Rath, and A. Sureka. Using source code metrics and multivariate adaptive regression splines to predict maintainability of service oriented software. In *High Assurance Systems Engineering (HASE), 2017 IEEE 18th International Symposium on*, pages 88–95. IEEE, 2017.

[9] L. Kumar, S. K. Rath, and A. Sureka. Using source code metrics to predict change-prone web services: A case-study on ebay services. In *Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE), IEEE Workshop on*, pages 1–7. IEEE, 2017.

[10] W. Li and S. Henry. Maintenance metrics for the Object-Oriented paradigm. In *International Software Metrics Symposium*, pages 52–60, 1993.

[11] R. Malhotra and Y. Singh. On the applicability of machine learning techniques for object oriented software fault prediction. *Software Engineering: An International Journal*.

[12] C. Mateos, M. Crasso, A. Zunino, and J. L. O. Coscia. Detecting wsdl bad practices in code–first web services. *International Journal of Web and Grid Services*, 7(4):357–387, 2011.

[13] T. Menzies, Z. Chen, J. Hihn, and K. Lum. Selecting best practices for effort estimation. *IEEE Transactions on Software Engineering*, 32(11):883–895, 2006.

[14] M. Perepletchikov, C. Ryan, and K. Frampton. Cohesion metrics for predicting maintainability of service-oriented software. In *QSIC*, pages 328–335. IEEE, 2007.

[15] M. Perepletchikov, C. Ryan, K. Frampton, and Z. Tari. Coupling metrics for predicting maintainability in service-oriented designs. In *ASWEC*, pages 329–340. IEEE, 2007.

[16] B. Shim, S. Choue, S. Kim, and S. Park. A design quality model for service-oriented architecture. In *2008 15th Asia-Pacific Software Engineering Conference*, pages 403–410. IEEE, 2008.

[17] H. M. Sneed. Measuring web service interfaces. In *Web Systems Evolution (WSE)*, pages 111–115. IEEE, 2010.

[18] J. A. Suykens, J. De Brabanter, L. Lukas, and J. Vandewalle. Weighted least squares support vector machines: robustness and sparse approximation. *Neurocomputing*.

[19] V. X. Tran, H. Tsuji, and R. Masuda. A new qos ontology and its qos-based ranking algorithm for web services. *Simulation Modelling Practice and Theory*, 17(8):1378–1398, 2009.

[20] D. Wang and J. Romagnoli. Robust multi-scale principal components analysis with applications to process monitoring. *Journal of Process Control*, 15(8):869–882, 2005.

[21] P. Wang. Qos-aware web services selection with intuitionistic fuzzy set under consumer's vague perception. *Expert Systems with Applications*, 36(3):4460–4466, 2009.

[22] Y. Zhou and H. Leung. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. 32(10):771–789, 2006.