Application of LSSVM and SMOTE on Seven Open Source Projects for Predicting Refactoring at Class Level

Lov Kumar
Thapar University, India
lovkumar505@gmail.com

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

***Abstract*—Source code refactoring consisting of modifying the structure of the source code without changing its functionality and external behavior. We present a method to predict refactoring candidates at class level which can help developers in improving their design and structure of source code while preserving the behavior. We propose a technique to predict refactoring candidates based on the application of a machine learning based framework. We use Least Squares Support Vector Machines (LS-SVM) as the learning algorithm, Principal Component Analysis (PCA) as a feature extraction technique and Synthetic Minority Over-sampling Technique (SMOTE) as a technique for handling imbalanced data.**

We start with 102 source code metrics as input features which are then reduced to 31 features after removing irrelevant and redundant features through statistical tests. We conduct a series of experiments on publicly available software engineering dataset consisting of seven open-source software systems in which the refactored classes are manually validated. We apply LS-SVM with three different functions: linear, polynomial and Radial Basis Function (RBF). Statistical significance test demonstrate that RBF kernel outperforms linear and polynomial kernel but there is no statistically significant difference between the performance of linear and polynomial kernel. Statistical significance test reveals that with-SMOTE technique outperforms without-SMOTE and all metrics outperforms PCA based metrics. The mean value of Area Under Curve (AUC) for LS-SVM RBF kernel is 0.96.

***Index Terms*—Empirical Software Engineering, Machine Learning, Predictive Modeling, Refactoring, Software Maintenance, Source Code Metrics**

I. Research Motivation and Aim

Refactoring of object-oriented applications consists of changing the internal structure of the software system or its design to decrease the complexity of the software without changing its external behavior or functionality [1][2]. Refactoring is performed to transform the code so that it is more maintainable and it is done without changing the semantics of the code [1][2]. Some of the refactoring types from the catalog[1] of refactoring are: Add Parameter, Collapse Hierarchy, Dynamic Method Definition, Inline Method, Preserve Whole Object, Remove Control Flag, Replace Conditional with Polymorphism, Replace Parameter with Explicit Methods and Split Temporary Variable. As the name of various types of refactoring implies, the process of refactoring involves modifying classes, methods and variables. It is also non-trivial for developers to identify all the code elements or segments of legacy software or in a large complex system which requires refactoring.

Which code segment to refactor? is a problem encountered by software practitioners. The work presented in this paper is motivated by the need to build tool support for software developers for identifying methods and classes which requires refactoring. Automatic identification of refactoring opportunities in source code is an area that has attracted several researchers attention (refer to the Related Work Section of the paper: Section II). However, a large scale study (involving several software systems) on the application of machine learning based techniques and using object oriented source code metrics as features is relatively unexplored. Our literature survey reveals lack of empirical studies on the evaluation of a machine learning algorithm on multiple real-world accurate and manually validated ground-truth dataset. Furthermore, there are lack of statistical analysis based studies examining the correlation between object-oriented metrics and refactoring. We frame several research questions aimed at filling some of the research gaps. Our work utilizes SMOTE, LS-SVM and PCA. SMOTE is Synthetic Minority Over-Sampling Technique which is an approach to construct classifier for imbalanced dataset [3]. For our problem domain, the dataset is imbalanced as the number of classes which are refactored is much less than the number of classes which are not refactored. LS-SVM (Least Squares Support Vector Machines) are least squares versions of support vector machines (SVM) and is a supervised learning method for classification and regression analysis [4]. Principal component analysis (PCA) is a statistical method that applies orthogonal transformation to create a set of observations of possibly correlated variables (predictors in a classification problem) into a set of values of linearly uncorrelated variables called as the principal components[2]. The work presented in this paper is paper is guided by the following research questions:

RQ 1: Is there a statistical significant difference in mean value of the source code metrics between the two groups consisting of classes which were refactored in the subsequent release

[1]https://refactoring.com/catalog/

[2]https://en.wikipedia.org/wiki/Principal_component_analysis

and classes which were not refactored in the subsequent release of the software system?

RQ 2: What is the correlation between the set of source code metrics identified as features and predictors affecting the need for refactoring?

RQ 3: What is the impact of PCA based feature extraction technique on the classifier performance?

RQ 4: What is the impact of SMOTE synthetic minority over-sampling technique for handling imbalanced data technique on the LS-SVM classifier performance?

RQ 5: What is the relative performance of the three different types of LS-SVM kernels?

II. Related Work

Zhao et al. conduct a study which reveals that complexity and size of a class are indicators that can be used to predict classes which are need of refactoring [5]. They conduct their study on an application written as part of a software engineering course in a university. They use metrics such as Halstead metrics and Weighted Methods per Class (WMC) and do not use any machine learning algorithm for prediction [5]. Alkhalid et al. use three clustering techniques Single Linkage algorithm (SLINK), the Complete Linkage algorithm (CLINK) and the Weighted Pair-Group Method using Arithmetic averages (WPGMA) along with Adaptive K-Nearest Neighbour (A-KNN) algorithm for identifying refactoring candidates at class level [6]. Dallal et al. investigate the application of several size, cohesion and coupling metrics for identifying need of refactoring at class-level [7]. Their focus is on extract subclass refactoring and they use univariate as well as multivariate logistic regression technique to build a predictive model [7].

Liu et al. describe a tool which uses conceptual relationship, implementation similarity, structural correspondence, and inheritance hierarchies to identify potential refactoring opportunities in the source code of open-source software systems [8]. Fokaefs et al. describe a tool called as JDeodorant which is an Eclipse plugin capable of recognizing opportunities for extracting cohesive classes from God classes (extract class refactoring) [9]. Tsantalis et al. describe a method to identify refactoring suggestions introducing polymorphism [10]. Their focus is on detection and removal of state-checking problems in Java projects implemented and deployed as an Eclipse add-on or plug-in [10]. Tourwe et al. present a method for identifying refactoring opportunities using logic meta-programming [11]. Their approach consists of first detecting bad smells using logic meta-programming and then using the gathered information to detect refactoring opportunities [11].

III. Research Contributions

In context to existing work, the study presentation in this paper makes the following novel and unique research contributions:

1) The study presented in this paper is the first study on the application of LS-SVM based learning algorithm (using three different types of kernel methods), PCA based feature extraction technique, SMOTE technique for handling imbalanced data and several source code based metrics for identifying refactoring opportunities in object-oriented software systems.
2) We conduct a series of experiments on seven open-source software systems. We conduct several statistical significance test to demonstrate the effectiveness of our proposed approach. We frame several research questions on the effectiveness of the proposed approach, relative comparison of the various LS-SVM kernel methods, impact of PCA and SMOTE, correlation between the features, correlation between the source code metrics and the target class. We collect data to answer the stated research questions. The results are presented using descriptive statistics of the AUC performance metrics and visualization using ROC curves and box-plots.

IV. Experimental Dataset

We use a publicly available dataset available at tera-PROMISE Repository[3] for our experiments. The tera-PROMISE website is a well-known repository consisting of several software engineering research datasets on code analysis, defects, effort, refactoring and test generation [12]. We use a dataset from the tera-PROMISE repository so that our experiments can be easily replicated and can be used for benchmarking and comparison by other researchers. We use the manually validated dataset on refactoring uploaded by Kadar et al. [13][14]. Kadar et al. create a refactoring and source code metrics dataset of two subsequent releases of seven open-source Java applications [13][14]. The seven Java

[3]http://openscience.us/repo/

TABLE I: Experimental Data Set Description

Project	Without SMOTE			With SMOTE (ALL Metrics)			With SMOTE (PCA)		
	# NOC	# NORC	% RC	# NOC	# NORC	% RC	# NOC	# NORC	% RC
antlr4	408	23	5.64	423	92	21.75	434	92	21.20
junit	655	9	1.37	660	36	5.45	664	36	5.42
MapDB	419	4	0.95	419	16	3.82	420	16	3.81
mcMMO	89	4	4.49	92	15	16.30	94	15	15.96
mct	2028	15	0.74	2042	60	2.94	2047	60	2.93
oryx	504	15	2.98	526	60	11.41	524	60	11.45
titan	1158	13	1.12	1163	52	4.47	1174	52	4.43

software systems are available on GitHub[4], the refactoring extraction by Kadar et al. is done using RefFinder tool [15] and the source code metrics are computed using the SourceMeter tool[5]. All the metrics are class level metrics and hence no aggregation of metrics is applied at system level. The dataset is of high quality as all the false positive instances generated by the RefFinder tool are removed by manually validating the dataset.

Table I displays the list of Java projects, Number of Classes (# NOC), Number of Refactored Classes (# NOCR) and Percentage of Refactored Classes (% RC) for three variations of the dataset: without SMOTE, with SMOTE (All Metrics), with SMOTE (PCA). The dataset without SMOTE is the version which is available at the tera-PROMISE Repository whereas the other two versions with SMOTE (All Metrics) and with SMOTE (PCA) are created by us by applying a technique called as SMOTE (Synthetic Minority Over-sampling Technique) [3] for the purpose of increasing the sensitivity of the classifier for the minority class. Principal Component Analysis (PCA) is a feature extraction technique used for dimensionality reduction. We apply SMOTE and PCA as the dataset is imbalanced and the number of input features is high. Table I shows that after applying SMOTE (over-sampling the minority class and under-sampling the majority class) changes the distribution of the target class. For example, in the without SMOTE version, the % RC for the antlr4 project changes from 5.64% to 21.75% and 21.20% respectively.

[4]https://github.com/

[5]https://www.sourcemeter.com/

V. Research Framework and Solution Approach

Figure 1 illustrates our research framework and proposed approach. As shown in Figure 1, our approach is a multi-step process consisting of relevant feature analysis using statistical significance test, feature extraction using PCA, application of min-max normalization technique for feature scaling, usage of SMOTE approach for handling imbalanced data, employing LS-SVM learning algorithm with three different types of kernel methods and evaluating the proposed approach using AUC and ROC curves. The dataset consists of 102 source code metrics computed using the SourceMeter[6] tool. The target class is a binary variable denoting whether the class was refactored in the subsequent release or not. We create two groups: classes which are refactored and classes which were not refactored.

Step 1 in our proposed approach is a preprocessing step consisting of a statistical hypothesis test to identify a subset of source code metrics affecting refactoring. Step 2 of our approach is to apply PCA based feature extraction technique for dimensionality reduction. One of our objectives is to compare the performance of the predictive model with two set of features: all features (obtained after Step 1) and PCA based features. Our experimental dataset is not evenly distributed and highly imbalanced. Training on imbalanced dataset can lead to bias towards the minority class and thus we use a well-known technique called as SMOTE to handle imbalanced data. One of the aspects of SMOTE is oversampling of the minority class. Table I shows the characteristics of the experimental dataset

[6]https://www.sourcemeter.com/

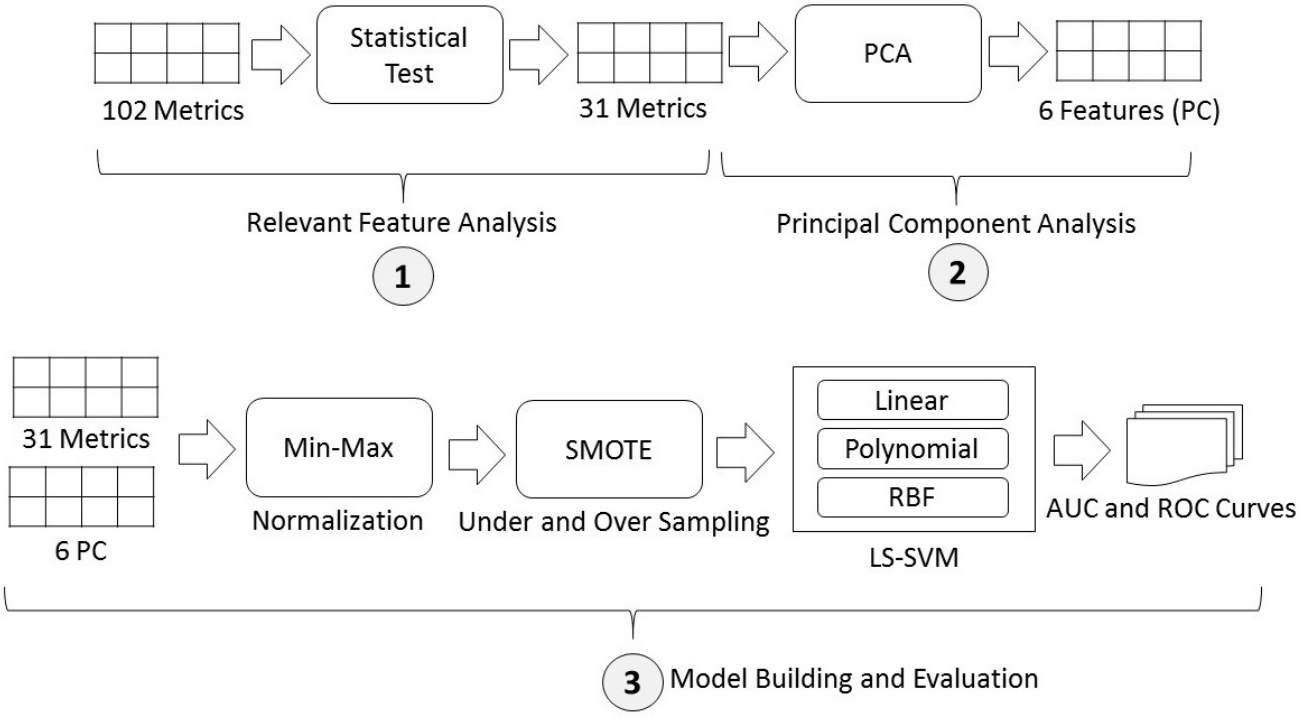


Fig. 1: Research Framework and Solution Approach

after application of the SMOTE technique. One of our research objectives is to investigate the impact of SMOTE technique on the classifier performance in terms of the AUC values and ROC curves. The range of the features are different and hence we apply a feature scaling technique to rescale and standardize the features.

We apply a min-max normalization approach and scale all the features between a fixed range of 0 to 1. We apply LS-SVM learning algorithm which has been used in several classification and pattern recognition problems in software engineering domain. LS-SVM is a variant of SVM and can be viewed as a least squares version of SVM [4][16]. SVM and LS-SVM is a nonlinear machine learning technique and consists of mapping the input feature space into a higher dimensional feature space [4][16]. LS-SVM is a kernel based learning technique and as shown in Figure 1, we examine the performance of tree different types of kernels (linear, polynomial and RBF) on the experimental dataset. We finally evaluate the performance of the various combinations of the classifier using AUC and ROC Curves. We take the mean accuracy in our measurements by applying the 10-fold cross validation technique. The dataset is split into ten sets of equal sizes. The model is trained on a dataset consisting of nine sets and evaluated on the tenth set. We repeat the experiment ten times to remove bias and take the average accuracy.

We follow a research methodology in which we formulate *five grounded research questions* which guides our study. We apply the Sandberg et al. approach of formulating research questions based on existing literature and theories [17]. Our primary object is to formulate research questions and answer them to facilitate development of tools and creation of methods on tool support for automatic identification of refactoring opportunities at class-level.

VI. Experimental Results and Research Findings

RQ 1: *Is there a statistical significant difference in mean value of the source code metrics between the two groups consisting of classes which were refactored in the subsequent release and classes which were not refactored in the subsequent release of the software system?*

We create two datasets for each of the seven projects. One dataset consists of all the classes on which refactoring is performed and the other dataset consists of classes on which refactoring is not performed. In this test the samples are independent as there is no common file between the two datasets. The files come from the same project but both the groups have different files. We compute the mean for each of the 102 metrics for both the datasets. Mean of a metric summarizes the value of the variable. We frame the null hypothesis H_0 as a claim that there is no statistically significant difference in the mean value of the metric (independent variable) for the two datasets i.e. the metric has no influence on the dependent variable (dependent variable representing whether a class being refactored or not). The alternate hypothesis H_a is the mathematical opposite the null hypothesis which states that there is a statistically significant difference between the means of the two datasets and hence the metric is potentially a predictor of the dependent variable. We compute the mean 1428 times as there are 102 metrics, 2 datasets and 7 projects.

We apply the Wilcoxon rank-sum test to determine whether the null hypothesis should be accepted or rejected. We applied Wilcoxon rank-sum test as it does not assume that the population follows a normal distribution and is much more efficient than parametric tests which assumes that the data is normally distributed. Hence we do not make any parametric assumptions while conducting the null test. We set the significance level as 0.05 and compare the p-value with

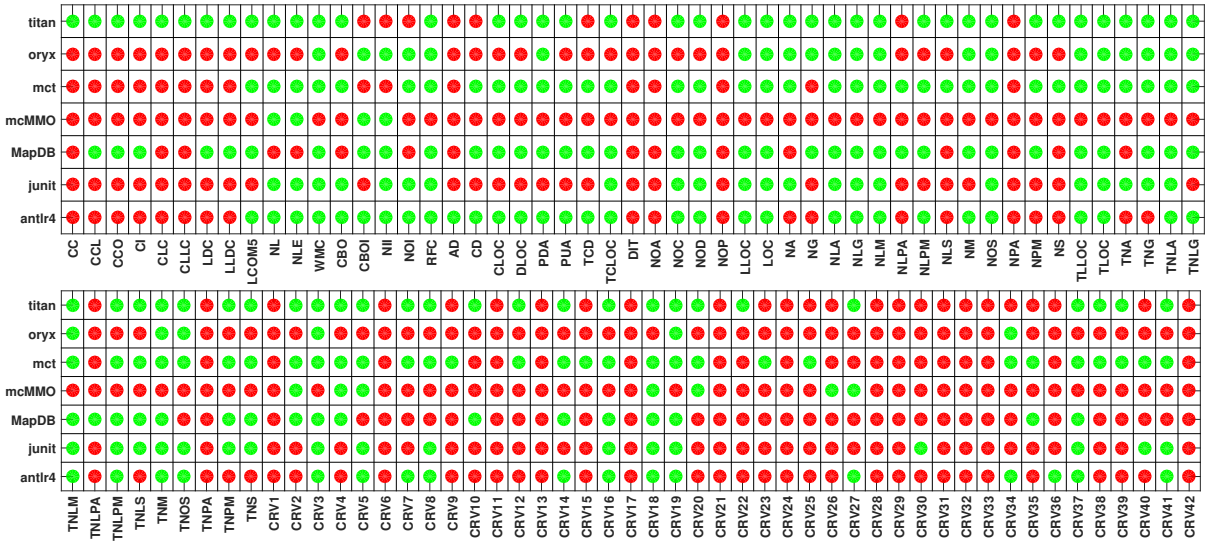


Fig. 2: List of 102 source code metrics and 7 dataset. Each cell represents outcome of hypothesis testing. Figure relates to RQ1. A red dot denotes that the null hypothesis is accepted and a green dot denotes that the null hypothesis is rejected.

0.05. If the p-value is less than 0.05, then we reject the null hypothesis. We reject the null hypothesis at a p-value of less than 0.05 as our inference is that the difference between the mean of the two groups is not because of chance and instead the difference in the mean is statistically significant. Figure 2 displays an illustration which reveals the cases for which the null hypothesis is accepted and rejected. A red dot denotes that the null hypothesis is accepted and a green dot denotes that the null hypothesis is rejected. Figure 2 reveals several metrics for which the number of green dots are ≥ 5. Our analysis reveals that there are 31 metrics having green dots ≥ 5. There are 16 metrics having green dots ≥ 6. We set the cutoff (a heuristic) as number of green dots ≥ 5 and use 31 metrics as features for the learning algorithm. We discard the remaining 72 metrics as noninformative or irrelevant based on the statistical significance test results. Following is the answer to RQ1.

Answer RQ1: *Statistical significance test reveals that out of the* 102 *source code metrics,* 31 *metrics affects the need for refactoring and there is a relation between* 31 *metrics and refactoring opportunities at class-level.*

RQ2: *What is the correlation between the set of source code metrics identified as features and predictors affecting the need for refactoring?*

We compute the association between 31 metrics consisting of the subset of the original 102 variables using the Pearson's correlations coefficient (r). The 31 metrics are the subset of the original 102 metrics which are found to affect the target class based on the statistical significance test performed in RQ 1. The Pearson's coefficient of correlation denoted by r measures the strength and direction of the linear relationship between two variables. We apply a correlations coefficient analysis to investigate if a dimension reduction technique can be useful to further reduce the 31 metrics to a smaller set by only using a smaller number of uncorrelated variables. Figure 3 displays our experimental results on correlation analysis between the 31 features in the form of a 31 by 31 matrix. Figure 3 shows the results for the antlr4 project. However, the results in Figure 3 are representative and there are slight differences across the projects. In Figure 3, a black filled circle represents an r value between 0.7 and 1.0 indicating a strong positive relationship respectively between the two variables. A white circle denotes an r value between 0.3 and 0.7 indicating a weak positive relationship respectively. A blank cell represents no linear relationships between the two variables. We do not find any negative correlation between any combinations of the two variables and hence there are is no symbol used for showing a negative correlation.

Figure 3 reveals that there are several variables which are strong positive and weak positive correlated while there are several variables which are not correlated. For example, based on Figure 3, we infer that there is a strong positive linear relationship between RFC and 14 other variables WMC, LLOC, LOC, NLM, NOS, TLLOC, TLOC, TNLM, TNLPM, TNOS, CRV5, CRV16, CRV18 and CRV19. On the other hand, we observe a weak linear relationship between NL and PDA as well as NLE and TNLA. We observe that metrics such as WMC, RFC, LLOC, LOC, NLM, NOS, TLLOC, TLOC, TNLM, TNLPM, TNOS, CRV3, CRV5 and CRV19 have several black circles which means that they are strongly correlated with several other metrics. This result

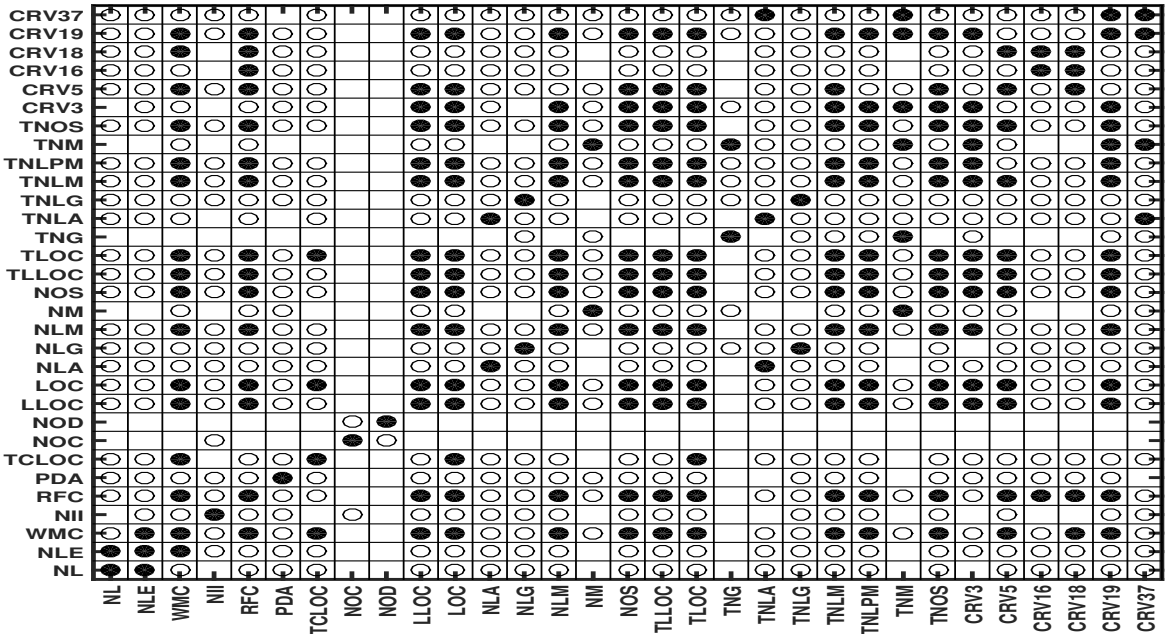


Fig. 3: Pearson's Correlation Coefficient between 31 Class Level Source Code Metrics for the antlr4 Project

shows that there is a scope of dimensionality reduction to a much smaller set by using uncorrelated variables as features. Figure 3 shows that metrics such as NL, NLE, NII, PDA, TCLOC, NLG, TNLG and CR16 are weakly correlated with several metrics. Metrics such as NOC and NOD are highly uncorrelated.

Answer RQ2: *There exist a strong correlation between several metrics. Weak correlation is also present.* 5 *out of* 31 *metrics are relatively uncorrelated. Results indicate opportunity for dimensionality reduction by identifying orthogonal features or principal components.*

RQ 3: *What is the impact of PCA based feature extraction technique on the classifier performance?*

RQ 4: *What is the impact of SMOTE synthetic minority over-sampling technique for handling imbalanced data technique on the LS-SVM classifier performance?*

RQ 5: *What is the relative performance of the three different types of LS-SVM kernels?*

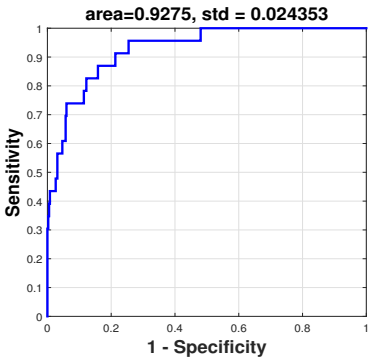
Feature Extraction using PCA: We apply Principal Component Analysis (PCA) based feature extraction technique in which new features are created by forming a combination of the existing features. The new features formed are transformed features rather than a subset of existing features. As shows in Table II, we extract six features which is much smaller in number than the original number of features. The six features labeled as PC1 to PC6 in Table II are for the antlr4 project and are a linear combination of the original 31 metrics. The six principal components are uncorrelated. However, unlike the original features, it is difficult to interpret the principal components as they are formed in a new coordinate system. Hence there is a tradeoff between speed (model training and testing) as well as performance accuracy on one hand and interpretability on the other hand. Using PCA, we reduce the irrelevant, noninformative and redundant features and reduce the dimensionality of the feature space from more than 30 (correlated features from the original 102 features) to 6. PCA is a reduction technique and hence improves the model training and testing speed. The Interpretation of the principal components is not direct and the user needs to interpret the components based on finding which predictors or input variables are most strongly correlated with each component. Table II reveals the correlations between the 6 principal components and the original metrics in the form of weights of variables in a linear equation. Table II also shows the eigenvalues of the six principal components and the proportion of variance in terms of percentage variance explained by them. For example, the eigenvalue for PC1 is 9.016 and the % variance is 29.084. The first principal component $PC1$ has the largest % variance of 29.084.

TABLE II: 6 Principal Components as Linear Combination of the 31 Metrics

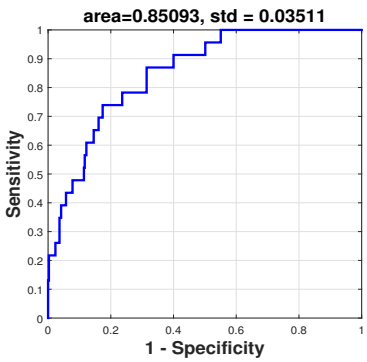
	antlr4 Project					
Metrics	**PC1**	**PC2**	**PC3**	**PC4**	**PC5**	**PC6**
NL	.079	.791	.195	.227	-.054	.015
NLE	.111	.789	.224	.272	-.039	.032
WMC	.514	.700	.351	.208	.027	.085
NII	.209	.159	.469	.147	.019	.571
RFC	.649	.555	.398	.040	.069	.072
PDA	.327	.327	.659	.002	-.091	.007
TCLOC	.293	.664	.232	-.054	.120	.037
NOC	.072	.039	.075	.046	.028	.931
NOD	-.025	.003	-.043	-.009	.009	.841
LLOC	.794	.541	.069	.091	.059	.078
LOC	.728	.613	.140	.090	.085	.077
NIA	.194	.353	.233	.834	.082	.055
NLG	.148	.287	.848	.158	.177	.061
NLM	.815	.272	.449	.058	-.025	.050
NM	.797	.032	.084	-.158	.311	-.068
NOS	.669	.658	.108	.173	.064	.097
TLLOC	.771	.524	.023	.248	.143	.085
TLOC	.711	.590	.092	.229	.156	.082
TNG	.163	.018	.237	.065	.929	.015
TNLA	.220	.320	.094	.876	.167	.050
TNLG	.176	.260	.781	.190	.402	.096
TNLM	.806	.285	.383	.239	.125	.092
TNLPM	.829	.162	.409	.214	.099	.029
TNM	.603	.040	.000	.116	.755	-.010
TNOS	.649	.627	.053	.309	.123	.094
CRV3	.823	.194	-.015	.174	.377	.032
CRV5	.522	.729	.141	.099	.055	.032
CRV16	.203	.775	.071	.161	.173	.036
CRV18	.210	.791	.222	.164	.059	-.003
CRV19	.785	.210	.258	.370	.248	.086
CRV37	.351	.291	.083	.516	.639	.102
Eigenvalues	9.016	7.220	3.252	2.622	2.540	2.016
% variance	29.084	23.290	10.491	8.457	8.193	6.503
Cumulative % var.	29.084	52.375	62.866	71.323	79.516	86.019

ROC Curve Analysis: We measure the performance of the classifier using Area Under the ROC Curve (AUC). Figure 4 and 5 shows several ROC (Receiver Operating Characteristics) Curve which is a graphical plot to show the discriminatory ability of the classifier. The ROC curve is created by varying the discriminatory threshold of the classifier and plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold values [18]. The x-axis in all the ROC curves in Figure 4 and 5 represents the false positive rate and is computed as 1 - specificity [18]. The y-axis represents the sensitivity. Figure 4 and 5 displays the ROC curves for the antlr4 project. Figure 4 shows six ROC curves (All metrics, PCA and 3 types of LSSVM kernel functions) for without SMOTE dataset and similarly 5 displays six ROC curves for with SMOTE dataset.

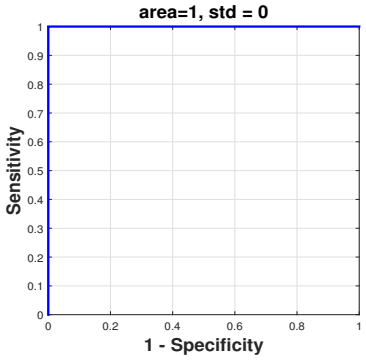
Our objective of creating the ROC curves is to analyze the performance of the classifier through visualization and also compare the performance of the classifiers through visual analytics in addition to computing performance metrics like AUC, precision, recall and f-measure. ROC graphs are particularly useful and applied in case of imbalanced dataset. This is because an ROC curve plots false positive rate against true positive rate and hence insensitive to class distribution changes. The ROC graphs in Figure 4 and 5 reveals the relative tradeoffs between the true positives and



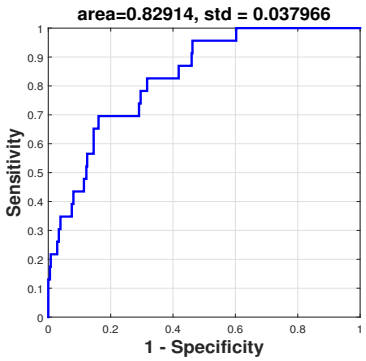
(4.1) ALL+ RBF-Kernel



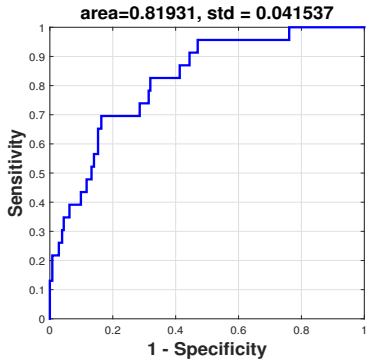
(4.2) ALL+ Linear-Kernel



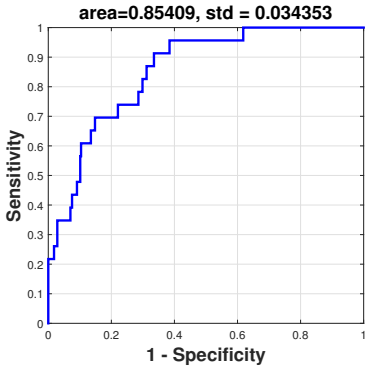
(4.3) ALL+ Polynomial-Kernel



(4.4) PCA+ RBF-Kernel

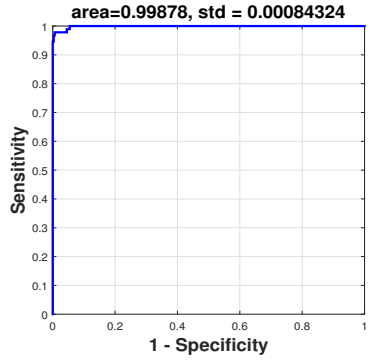


(4.5) PCA+ Linear-Kernel

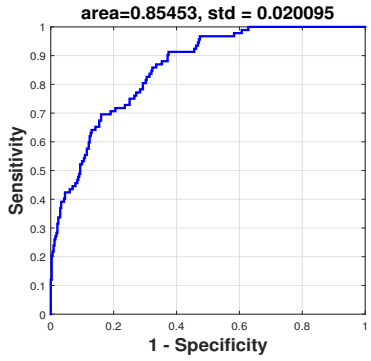


(4.6) PCA+ Polynomial-Kernel

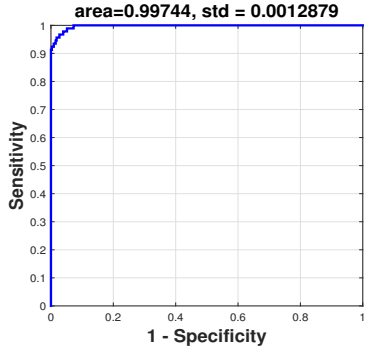
Fig. 4: Comparison of Various ROC Curves and AUC Values (Without SMOTE)



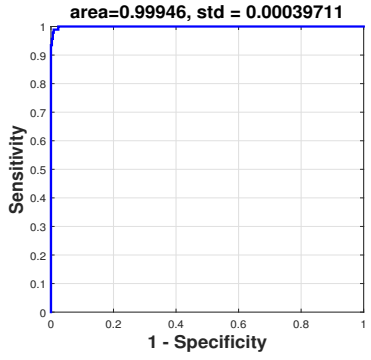
(5.1) ALL+ RBF-Kernel



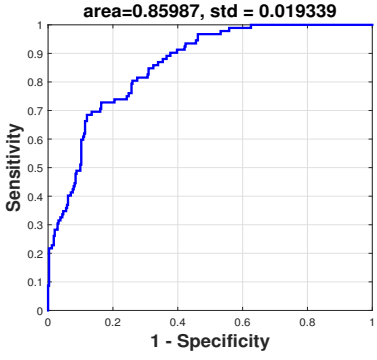
(5.2) ALL+ Linear-Kernel



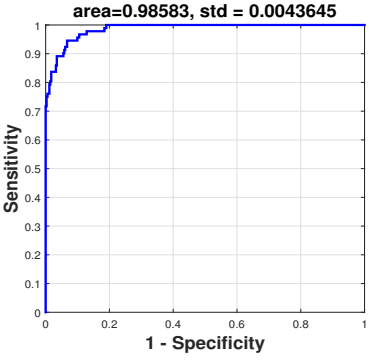
(5.3) ALL+ Polynomial-Kernel



(5.4) PCA+ RBF-Kernel



(5.5) PCA+ Linear-Kernel



(5.6) PCA+ Polynomial-Kernel

Fig. 5: Comparison of Various ROC Curves and AUC Values (With SMOTE)

false positives. The ROC curves in Figure 5 shows that the classifier using SMOTE has a better discriminatory power than the classifiers developed without SMOTE (refer to Figure 4) as the ROC curves in Figure 5 are passing more through the upper left corner (which is the 100% sensitivity and 100% specificity pair) than the ROC curves in Figure 4. Figure 4 reveals that the polynomial kernel performs better than the linear and RBF kernel as the curves are more closer to the upper left corner demonstrating a higher overall accuracy of the predictive model.

Descriptive Statistics and Box-Plot: Table III and Figure 6 shows the descriptive statistics and the box-plot diagram of the AUC values for 7 possible types of population of classifiers (1) Without SMOTE (2) With SMOTE (3) All Metrics (4) PCA Metrics (5) Linear Kernel (6) Polynomial Kernel (7) RBF Kernel. For each of the linear, polynomial and RBF kernel we generate 28 data points consisting of 28 AUC values. For with and without SMOTE, we generate 42 AUC values and similarly for all metrics and PCA metrics we generate 42 AUC values. Table III and Figure 6 reveals that with-SMOTE performs better than without-SMOTE as the mean, median, Q1 and Q3 AUC values of with-SMOTE is more than the corresponding value for without-SMOTE. The mean and median AUC values for all metrics is 0.96 and 0.99 whereas the mean and median AUC values for PCA metrics is 0.91 and 0.93 respectively. The difference between the mean and median values as well as Q1 and Q3 values shows that that all metrics performs better than PCA metrics. The PCA based transformation results in decreasing the number of dimensions or number of features but does not result in a better AUC value.

From Table III and Figure 6, we infer that the LS-LSM RBF kernel variant outperforms linear and polynomial kernel. The mean, median, Q1 and Q3 AUC values for RBF kernel is more than the corresponding values for polynomial and linear kernel. The mean, median, Q1 and Q3 AUC values for linear kernel is more than the corresponding values for polynomial kernel. The box-plots in Figure 6 shows the variability and degree of dispersion in the data. Using the standard deviation values and the box plot in Figure 6, we can compare the

TABLE III: Descriptive Statistics of the Performance in terms of AUC

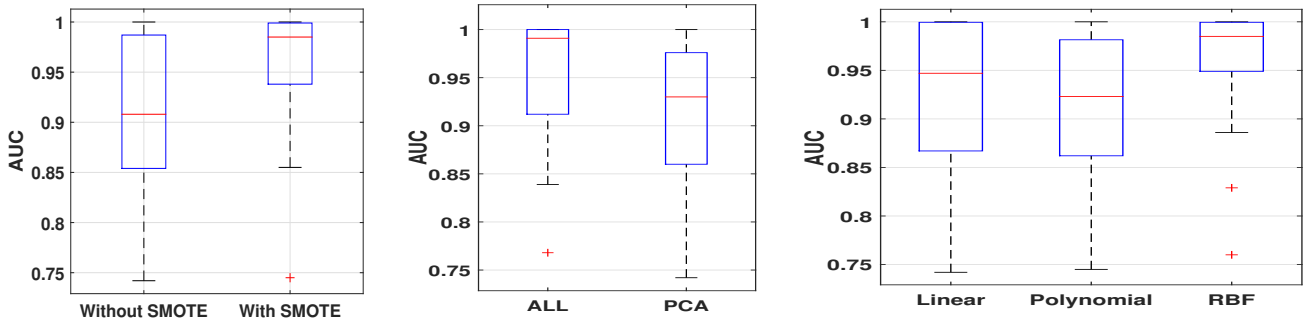
	AUC						
	Min	**Max**	**Mean**	**Median**	**Std Dev**	**Q1**	**Q3**
Without SMOTE	0.74	1.00	0.91	0.91	0.07	0.85	0.99
With SMOTE	0.75	1.00	0.96	0.99	0.06	0.94	1.00
ALL	0.77	1.00	0.96	0.99	0.06	0.91	1.00
PCA	0.74	1.00	0.91	0.93	0.07	0.86	0.98
Linear	0.74	1.00	0.93	0.95	0.08	0.87	1.00
Polynomial	0.75	1.00	0.92	0.92	0.07	0.86	0.98
RBF	0.76	1.00	0.96	0.99	0.06	0.95	1.00

TABLE IV: AUC Values for All Combinations of Classifiers Obtained as Means of 10-Fold Cross-Validation

Data Set	**Metrics**	**Without SMOTE**			**With SMOTE**		
		Linear	**Polynomial**	**RBF**	**Linear**	**Polynomial**	**RBF**
DS1	ALL	0.851	1.000	0.927	0.999	0.855	0.997
DS1	PCA	0.819	0.854	0.829	0.999	0.860	0.986
DS2	ALL	0.892	0.985	0.987	0.999	0.960	0.999
DS2	PCA	0.872	0.944	0.952	0.994	0.899	0.949
DS3	ALL	0.894	0.996	0.960	1.000	0.969	0.992
DS3	PCA	0.837	0.922	0.887	0.938	0.912	0.976
DS4	ALL	0.862	1.000	1.000	1.000	0.997	1.000
DS4	PCA	0.956	1.000	0.988	1.000	0.978	1.000
DS5	ALL	0.907	0.909	1.000	1.000	0.967	0.999
DS5	PCA	0.892	0.942	0.949	0.989	0.873	0.961
DS6	ALL	0.878	0.768	1.000	1.000	0.924	1.000
DS6	PCA	0.742	0.855	0.760	1.000	0.745	0.953
DS7	ALL	0.839	0.990	1.000	1.000	0.912	0.984
DS7	PCA	0.843	0.847	0.886	0.974	0.864	0.892

distributions between the seven different sets or population of the AUC values. The span between the first and third quartile is more for without-SMOTE than with-SMOTE. Similarly from Figure 6, we infer that the span between first and third quartile for PCA metrics is more than all metrics and for linear is more than polynomial and RBF.

Accuracy, F-Measure, Precision and Recall: Table V and IV shows detailed analysis for all the experiments and different combination of classifiers. Table V displays the performance results in terms of accuracy, f-measure, precision and recall. The DS1 to DS7 dataset mentioned in Table V and IV refers to the seven projects: antlr4, junit, MapDB, mcMMO, mct, oryx and titan. Table IV reveals that the minimum AUC value of the RBF kernel based LS-SVM with SMOTE is 0.892. The AUC value of the RBF kernel based LS-SVM with SMOTE



(6.1) With and Without SMOTE (6.2) ALL and PCA (6.3) Kernel Methods

Fig. 6: Box-Plot Diagrams of the AUC Values for 7 Possible Types of Population of Classifiers

TABLE V: Accuracy, F-Measure, Precision and Recall for All Combination of Classifiers Obtained as Means of 10-Fold Cross Validation

Without SMOTE													
		Linear-Kernel				Polynomial-Kernel				RBF-Kernel			
Data Set	Metrics	Accuracy	F-Measure	Precision	Recall	Accuracy	F-Measure	Precision	Recall	Accuracy	F-Measure	Precision	Recall
DS1	ALL	94.90	0.97	1.00	0.95	99.30	1.00	1.00	0.99	95.80	0.98	1.00	0.96
DS1	PCA	94.90	0.97	1.00	0.95	95.30	0.98	1.00	0.95	94.90	0.97	1.00	0.95
DS2	ALL	98.60	0.99	1.00	0.99	99.40	1.00	1.00	0.99	99.80	1.00	1.00	1.00
DS2	PCA	98.60	0.99	1.00	0.99	99.20	1.00	1.00	0.99	99.10	1.00	1.00	0.99
DS3	ALL	99.00	1.00	1.00	0.99	99.80	1.00	1.00	1.00	99.50	1.00	1.00	1.00
DS3	PCA	99.00	1.00	1.00	0.99	99.50	1.00	1.00	1.00	99.00	1.00	1.00	0.99
DS4	ALL	95.50	0.98	1.00	0.96	100.00	1.00	1.00	1.00	100.00	1.00	1.00	1.00
DS4	PCA	95.50	0.98	1.00	0.96	100.00	1.00	1.00	1.00	97.80	0.99	1.00	0.98
DS5	ALL	99.30	1.00	1.00	0.99	99.30	1.00	1.00	0.99	99.90	1.00	1.00	1.00
DS5	PCA	99.30	1.00	1.00	0.99	99.40	1.00	1.00	0.99	99.30	1.00	1.00	0.99
DS6	ALL	97.20	0.99	1.00	0.97	97.00	0.98	1.00	0.97	99.40	1.00	1.00	0.99
DS6	PCA	97.00	0.98	1.00	0.97	97.40	0.99	1.00	0.97	97.00	0.98	1.00	0.97
DS7	ALL	99.00	0.99	1.00	0.99	99.60	1.00	1.00	1.00	99.70	1.00	1.00	1.00
DS7	PCA	99.00	0.99	1.00	0.99	99.10	1.00	1.00	0.99	99.00	0.99	1.00	0.99
With SMOTE													
DS1	ALL	83.50	0.90	0.98	0.84	98.10	0.99	1.00	0.98	98.80	0.99	1.00	0.99
DS1	PCA	82.30	0.90	0.98	0.83	94.70	0.97	0.98	0.95	98.80	0.99	1.00	0.99
DS2	ALL	97.10	0.98	1.00	0.97	99.10	1.00	1.00	0.99	99.40	1.00	1.00	0.99
DS2	PCA	95.20	0.98	1.00	0.95	97.00	0.98	1.00	0.97	98.90	0.99	1.00	0.99
DS3	ALL	98.60	0.99	1.00	0.99	98.80	0.99	1.00	0.99	99.80	1.00	1.00	1.00
DS3	PCA	96.20	0.98	1.00	0.96	97.90	0.99	1.00	0.98	97.90	0.99	1.00	0.98
DS4	ALL	97.80	0.99	0.99	0.99	100.00	1.00	1.00	1.00	100.00	1.00	1.00	1.00
DS4	PCA	91.50	0.95	0.99	0.92	100.00	1.00	1.00	1.00	98.90	0.99	1.00	0.99
DS5	ALL	97.80	0.99	1.00	0.98	99.40	1.00	1.00	0.99	99.80	1.00	1.00	1.00
DS5	PCA	97.10	0.99	1.00	0.97	97.90	0.99	1.00	0.98	98.70	0.99	1.00	0.99
DS6	ALL	90.50	0.95	0.99	0.91	99.80	1.00	1.00	1.00	99.80	1.00	1.00	1.00
DS6	PCA	88.50	0.94	1.00	0.89	93.70	0.97	1.00	0.94	100.00	1.00	1.00	1.00
DS7	ALL	96.70	0.98	1.00	0.97	98.10	0.99	1.00	0.98	99.10	1.00	1.00	0.99
DS7	PCA	95.90	0.98	1.00	0.96	97.20	0.99	1.00	0.97	98.50	0.99	1.00	0.98

TABLE VI: T-Test Statistical Examination Results (AUC Scores)

(a) With and Without SMOTE

	P-value			Mean Difference	
	Without SMOTE	With SMOTE		Without SMOTE	With SMOTE
Without SMOTE	1.000	0.002	Without SMOTE	0.000	-0.049
With SMOTE	0.002	1.000	With SMOTE	0.049	0.000

(b) ALL and PCA

	P-value			Mean Difference	
	ALL	PCA		ALL	PCA
ALL	1.000	0.000	ALL	0.000	0.045
PCA	0.000	1.000	PCA	-0.045	0.000

(c) Different kernels

	P-value				Mean Difference		
	Linear	Polynomial	RBF		Linear	Polynomial	RBF
Linear	1.000	0.767	0.021	Linear	0.000	0.009	-0.030
Polynomial	0.767	1.000	0.010	Polynomial	-0.009	0.000	-0.039
RBF	0.021	0.010	1.000	RBF	0.030	0.039	0.000

is always above 0.99 several times. As shown in Table IV, the AUC value is never perfect (1.0) for polynomial kernel with SMOTE and linear kernel without SMOTE. The range of AUV value for linear kernel without SMOTE is 0.742 to 0.956.

Table V shows f-measure value which is computed as the harmonic mean of both the precision and recall to provide a single measurement of the classifiers performance. Table V reveals that the RBF kernel based LS-SVM technique is able to achieve an f-measure of 1.0 for several dataset and both in case of SMOTE and without SMOTE technique. The f-measure value is never 1.0 for linear kernel with SMOTE. Table V reveals the precision and recall values also to understand the relative contribution of it towards the f-measure. For example, Table V shows a precision of 1.0 several time which means that the classifier is able detect the correct positive results 100% of the time. Similarly, we observe a high recall for several cases which indicates that the classifiers performance in terms of the percentage of positive records being labeled as positive is good.

T-Test Statistical Examination Results: Table VI displays the result of t-test to compare the means of two groups and determine if they are different from each other. The objective of performing t-test is to determine if the difference is actual or could have happened by chance. In addition to the results displayed in Table III and Figure 6, we perform t-test analysis as a form of hypothesis testing to minimize threat to conclusion validity. We use the p-value as 0.05.

From Table VI, we observe that the p-value is 0.002 corresponding to a 0.2% chance of obtaining a result like this if the H_0 was true and hence we can reject the null hypothesis. Similarly the p-value when comparing RBF kernel with linear kernel and RBF kernel with polynomial kernel is 0.021 and 0.010 respectively which shows that RBF kernel performs better than linear and polynomial kernel. However, the p-value when comparing polynomial and linear kernel is 0.767 which shows that p-value is above the 0.05 cutoff value and there is a support for the null hypothesis. The null hypothesis is not rejected in this case. The t-test shows that

PCA does not perform better than the all metrics.

Answer RQ 3: *Application of PCA results in extraction of six orthogonal features from input metrics but the statistical significance test does not show an improvement in classifier performance.*

Answer RQ 4: *Application of SMOTE technique improves performance. SMOTE technique results in high recall of the minority class as well as maintaining high precision of the majority class. Statistical significance test shows boost in classification performance in a highly imbalanced refactoring dataset.*

Answer RQ 5: *LS-LSM RBF kernel variant outperforms linear and polynomial kernel. Linear kernel outperforms polynomial kernel in terms of the difference in the mean AUC values. Statistical significance test confirms that RBF kernel actually performs better and the results are not by chance.*

VII. Threats to Validity

We report our validity analysis and provide justification on how we overcame or minimized the possible four different types of threats to validity [19][20] in our experiments. We apply statistical significance test to mitigate conclusion validity and present evidences to show that the performance improvement is actual and not by chance. However, the p-value considered is a standard value and a more stringent p-value can have an effect on the hypothesis testing outcome. We conduct experiments on dataset belonging to seven projects to investigate if our results are generalizable and thus mitigate external validity.

We downloaded a refactoring dataset from PROMISE repository which is manually validated and of high quality to ensure that there are no annotation or measurement errors. We applied 10 fold cross-validation approach and executed the experiments more than once to ensure that there are no errors while measuring and observing data. However, there are still possibilities of threats to internal validity as 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 as well as small number of examples of the minority class (refactored classes). Another threat to validity is that the source code metrics may not be the only factor leading to refactoring and there can be other factors not included as part of this study.

VIII. Conclusion

Our premise is that support for automatic identification of classes in need of refactoring can help developers in improving the structure and design of their code. We present an approach based on machine learning algorithm and source code metrics to classify classes in need of refactoring. We apply a preprocessing step to filter 31 source code metrics from an initial set of 102 metrics which are indicators of refactoring and can be used as predictors. We conduct experiments on a publicly available dataset consisting of manually annotated and validated classes. We apply PCA for feature extraction, SMOTE for handling imbalanced data and LS-SVM with three kernel variants as the machine learning algorithm. Statistical significance testing and performance evaluation measured using AUC shows that LS-SVM with RBF kernel using 31 metrics and SMOTE results in the best performance.

References

[1] M. Fowler and K. Beck, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

[2] T. Mens and T. Tourwé, "A survey of software refactoring," *IEEE Transactions on software engineering*, vol. 30, no. 2, pp. 126–139, 2004.

[3] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[4] J. A. Suykens, L. Lukas, and J. Vandewalle, "Sparse approximation using least squares support vector machines," in *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, vol. 2. IEEE, 2000, pp. 757–760.

[5] L. Zhao and J. Hayes, "Predicting classes in need of refactoring: an application of static metrics," in *2nd International PROMISE Workshop, Philadelphia, Pennsylvania USA (co-located with the IEEE Conference on Software Maintenance)*, 2006.

[6] A. Alkhalid, M. Alshayeb, S. A. Mahmoud *et al.*, "Software refactoring at the class level using clustering techniques," *Journal of Research and Practice in Information Technology*, vol. 43, no. 4, p. 285, 2011.

[7] J. Al Dallal, "Constructing models for predicting extract subclass refactoring opportunities using object-oriented quality metrics," *Information and Software Technology*, vol. 54, no. 10, pp. 1125–1141, 2012.

[8] H. Liu, Z. Niu, Z. Ma, and W. Shao, "Identification of generalization refactoring opportunities," *Automated Software Engineering*, vol. 20, no. 1, pp. 81–110, 2013.

[9] M. Fokaefs, N. Tsantalis, E. Stroulia, and A. Chatzigeorgiou, "Jdeodorant: identification and application of extract class refactorings," in *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 2011, pp. 1037–1039.

[10] N. Tsantalis and A. Chatzigeorgiou, "Identification of refactoring opportunities introducing polymorphism," *Journal of Systems and Software*, vol. 83, no. 3, pp. 391–404, 2010.

[11] T. Tourwé and T. Mens, "Identifying refactoring opportunities using logic meta programming," in *Software Maintenance and Reengineering, 2003. Proceedings. Seventh European Conference on*. IEEE, 2003, pp. 91–100.

[12] "The promise repository of empirical software engineering data," 2015.

[13] I. Kádár, P. Hegedűs, R. Ferenc, and T. Gyimóthy, "A manually validated code refactoring dataset and its assessment regarding software maintainability," in *Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering*. ACM, 2016, p. 10.

[14] I. Kádár, P. Hegedus, R. Ferenc, and T. Gyimóthy, "A code refactoring dataset and its assessment regarding software maintainability," in *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, vol. 1. IEEE, 2016, pp. 599–603.

[15] M. Kim, M. Gee, A. Loh, and N. Rachatasumrit, "Ref-finder: a refactoring reconstruction tool based on logic query templates," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 2010, pp. 371–372.

[16] J. Suykens, L. Lukas, P. Van Dooren, B. De Moor, J. Vandewalle *et al.*, "Least squares support vector machine classifiers: a large scale algorithm," in *ECCTD*, vol. 99, 1999, pp. 839–842.

[17] J. Sandberg and M. Alvesson, "Ways of constructing research questions: gap-spotting or problematization?" *Organization*, vol. 18, no. 1, pp. 23–44, 2011.

[18] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[19] R. Feldt and A. Magazinius, "Validity threats in empirical software engineering research-an initial survey." in *SEKE*, 2010, pp. 374–379.

[20] D. E. Perry, A. A. Porter, and L. G. Votta, "Empirical studies of software engineering: a roadmap," in *Proceedings of the conference on The future of Software engineering*. ACM, 2000, pp. 345–355.