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

Lov Kumar[1] Ashish Sureka[2]

[1]Thapar University, India (lovkumar505@gmail.com)

[2]Ashoka University, India (ashish.sureka@ashoka.edu.in)

APSEC 2017 [4-8 December 2017, Nanjing, Jiangsu, China]

Table of Contents

Table of Contents

Refactoring - More Maintainable Code

Refactoring of Object-Oriented Applications

Refactoring - changing internal structure of the software system or its design to decrease the complexity of the software without changing its external behavior or functionality [8][13]

Process of refactoring involves modifying classes, methods and variables

Non-trivial for developers to identify all the code elements or segments of legacy software or in a large complex system which requires refactoring

Problem Encountered by Software Developers

Which Code Segment to Refactor?

Our Motivation: To build tool support for software developers for identifying methods and classes which requires refactoring

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

Table of Contents

Research Questions - RQ1 and RQ2

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?

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

Research Questions - RQ3, RQ4 and RQ5

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 oversampling 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?

Related Work - 1

Zhao et al. [20]

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 [20]

Alkhalid et al. [3]

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 [3].

Related Work - 2

Dallal et al. [2]

Dallal et al. investigate the application of several size, cohesion and coupling metrics for identifying need of refactoring at class-level [2]

Liu et al. [12]

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 [12].

Related Work - 3

Fokaefs et al. [7]

Fokaefs et al. - JDeodorant an Eclipse plugin capable of recognizing opportunities for extracting cohesive classes from God classes [7]

Tsantalis et al. [19]

Tsantalis et al. describe a method to identify refactoring suggestions introducing polymorphism [19].

Tourwe et al. [18]

Tourwe et al. present a method for identifying refactoring opportunities using logic meta-programming [18].

Research Contributions

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.

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.

Experimental Dataset

We use a publicly available dataset available at tera-PROMISE Repository[a] for our experiments.

We use the manually validated dataset on refactoring uploaded by Kadar et al. [10][9]. Kadar et al. create a refactoring and source code metrics dataset of two subsequent releases of seven open-source Java applications [10][9].

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

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

Experimental Dataset

The seven Java software systems are available on GitHub[a].

The refactoring extraction by Kadar et al. is done using RefFinder tool [11] and the source code metrics are computed using the SourceMeter tool[b]

The dataset is of high quality as all the false positive instances generated by the RefFinder tool are removed by manually validating the dataset.

[a] https://github.com/
[b] https://www.sourcemeter.com/

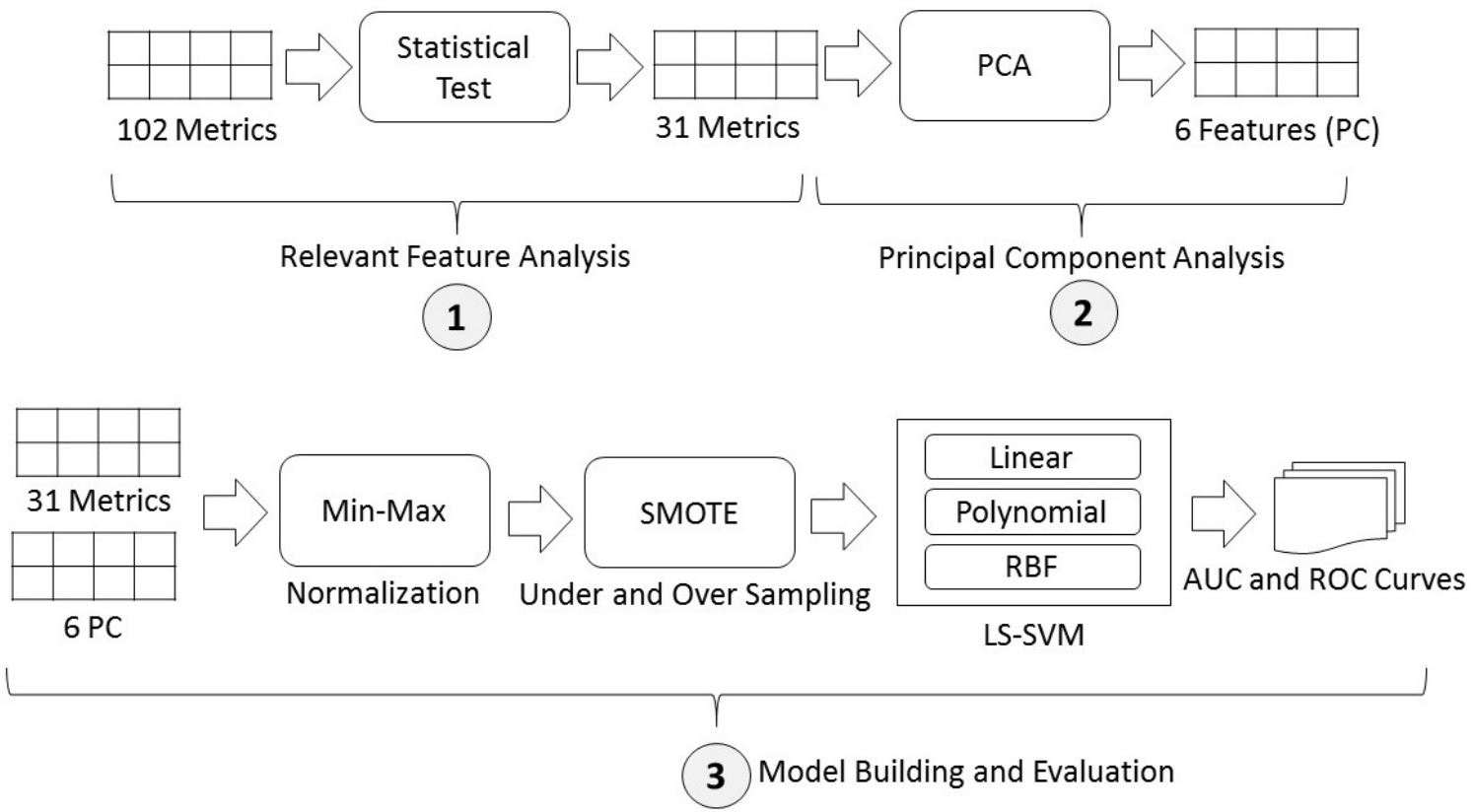
Solution Approach - PCA - SMOTE

Preprocessing step consisting of a statistical hypothesis test to identify a subset of source code metrics affecting refactoring.

Apply PCA [Principal component analysis] based feature extraction technique for dimensionality reduction

We use a well-known technique called as SMOTE [Synthetic Minority Over-sampling Technique] to handle imbalanced data.

Multi-Step Process



Solution Approach - Feature Scaling - LS-SVM

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.

Solution Approach - Kernels - AUC and ROC Curves

We examine the performance of three different types of kernels (linear, polynomial and RBF).

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.

Table of Contents

Statistical Significant Difference in Metrics

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.

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

Hypothesis Testing - RQ1

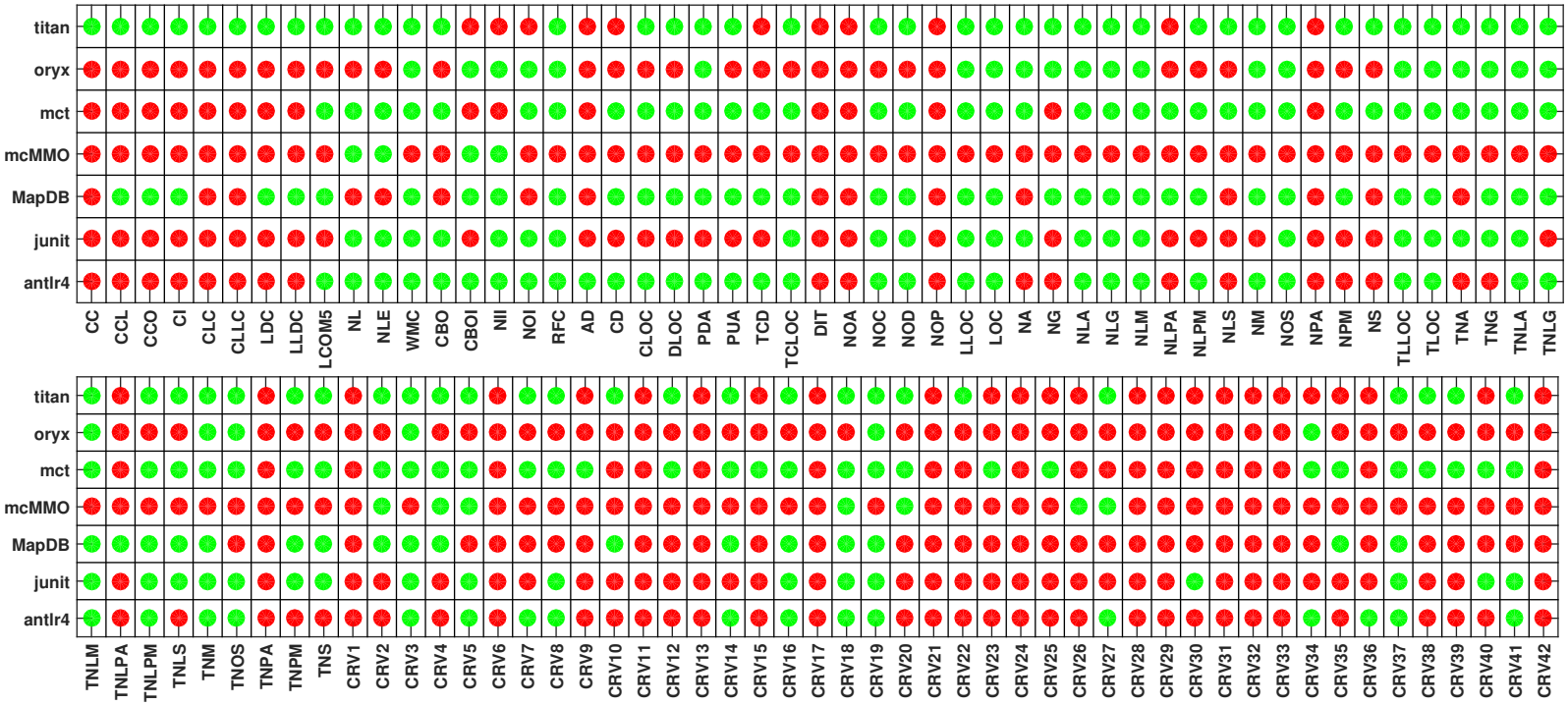


Figure: List of 102 source code metrics and 7 dataset. Red dot: null hypothesis accepted. Green dot: null hypothesis rejected

Statistical Significant Difference in Metrics

Alternate hypothesis H_a 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 apply the Wilcoxon rank-sum test to determine whether the null hypothesis should be accepted or rejected.

A red dot denotes that the null hypothesis is accepted and a green dot denotes that the null hypothesis is rejected.

Statistical Significant Difference in Metrics

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.

Answer to 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.

Table of Contents

Association between 31 Metrics

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.

Pearson's Correlation Coefficient

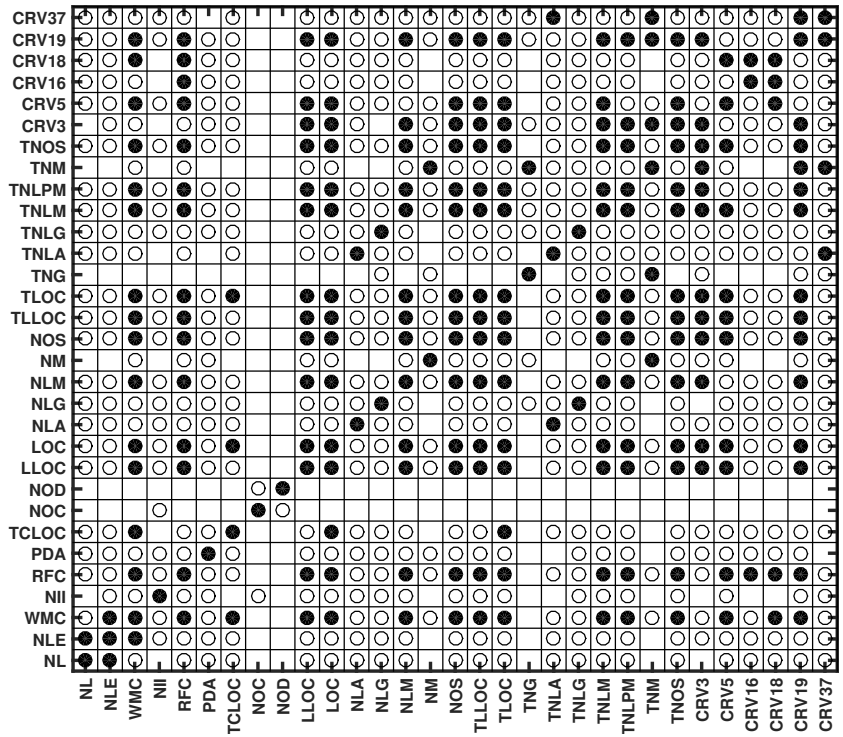


Figure: Correlation Coefficient between 31 Source Code Metrics

Insights - RQ2

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.

Weak linear relationship between NL and PDA as well as NLE and TNLA.

Metrics such as NOC and NOD are highly uncorrelated.

Answer to RQ2

There exists 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.

Table of Contents

Six Principal Components

We extract six features which is much smaller in number than the original number of features.

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.

Principal Components - Linear Combination of 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

Principal Components - Linear Combination of 31 Metrics

	antlr4 Project					
Metrics	**PC1**	**PC2**	**PC3**	**PC4**	**PC5**	**PC6**
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

Feature Extraction using PCA

Table (previous slide) reveals the correlations between the 6 principal components and the original metrics in the form of weights of variables in a linear equation

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.

Answer to RQ3

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.

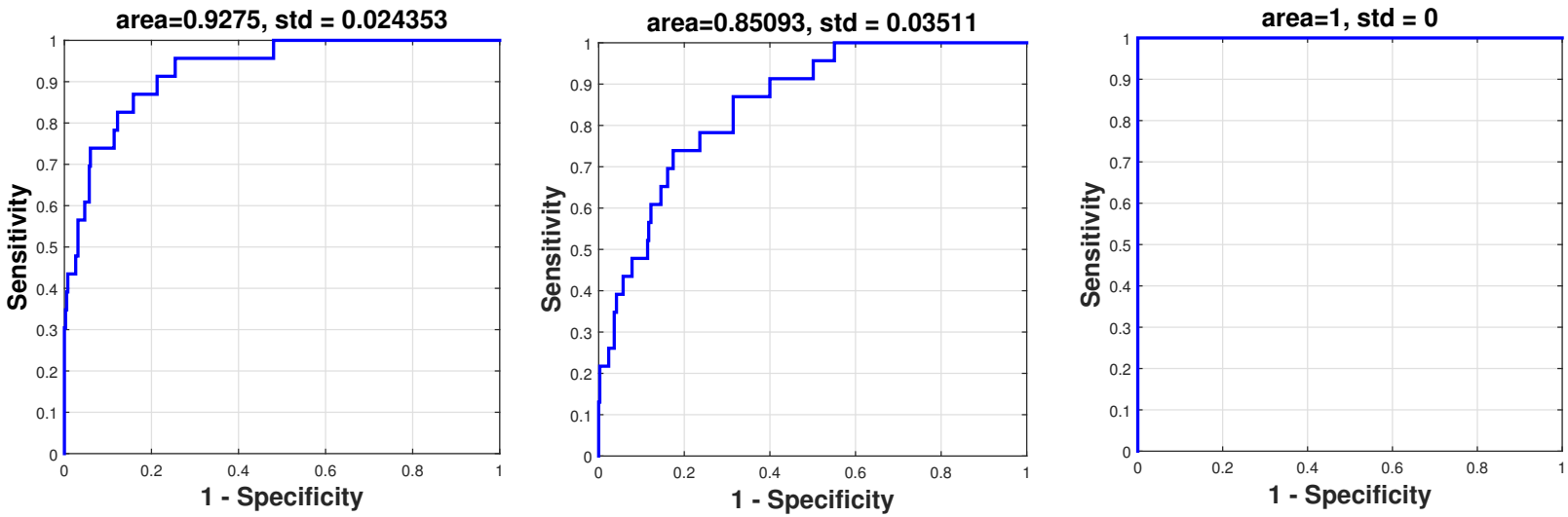
Table of Contents

ROC Curve Analysis

Figures in next slides shows several ROC (Receiver Operating Characteristics) Curve which is a graphical plot to show the discriminatory ability of the classifier.

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.

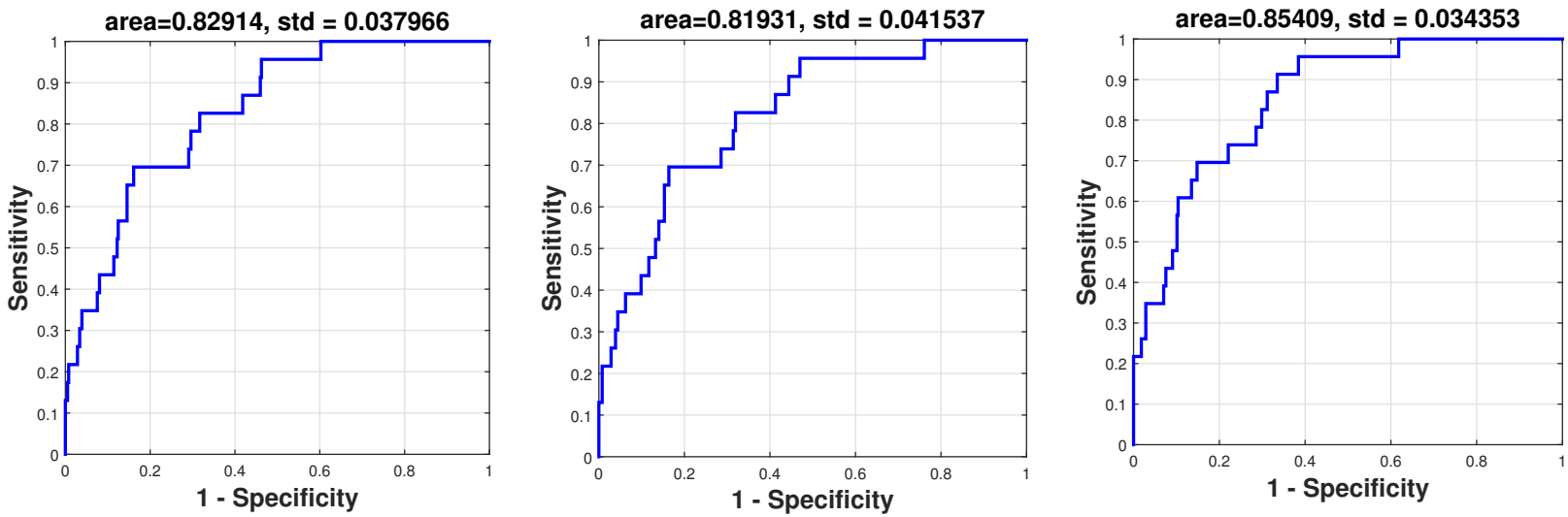
ROC Curves and AUC Values (Without SMOTE)



((a)) ALL+ RBF-Kernel ((b)) ALL+ Linear-Kernel ((c)) ALL+ Polynomial-Kernel

Figure: Comparison of Various ROC Curves and AUC Values (Without SMOTE)

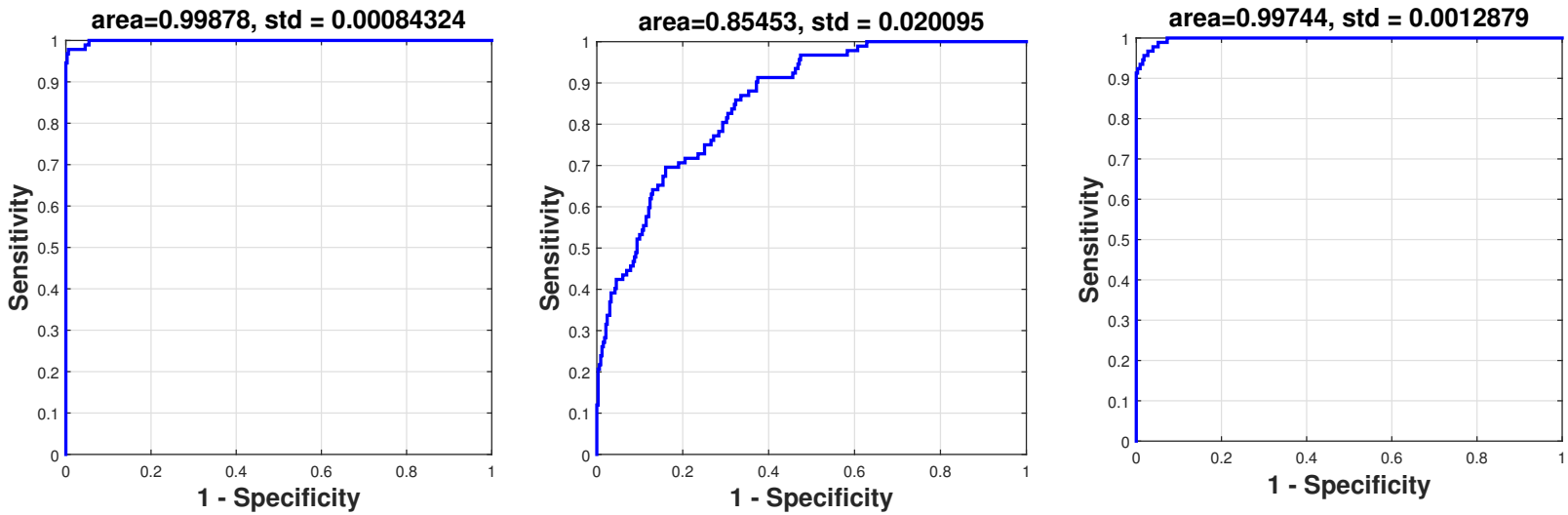
ROC Curves and AUC Values (Without SMOTE)



((a)) PCA+ RBF-Kernel ((b)) PCA+ Linear-Kernel ((c)) PCA+ Polynomial-Kernel

Figure: Comparison of Various ROC Curves and AUC Values (Without SMOTE)

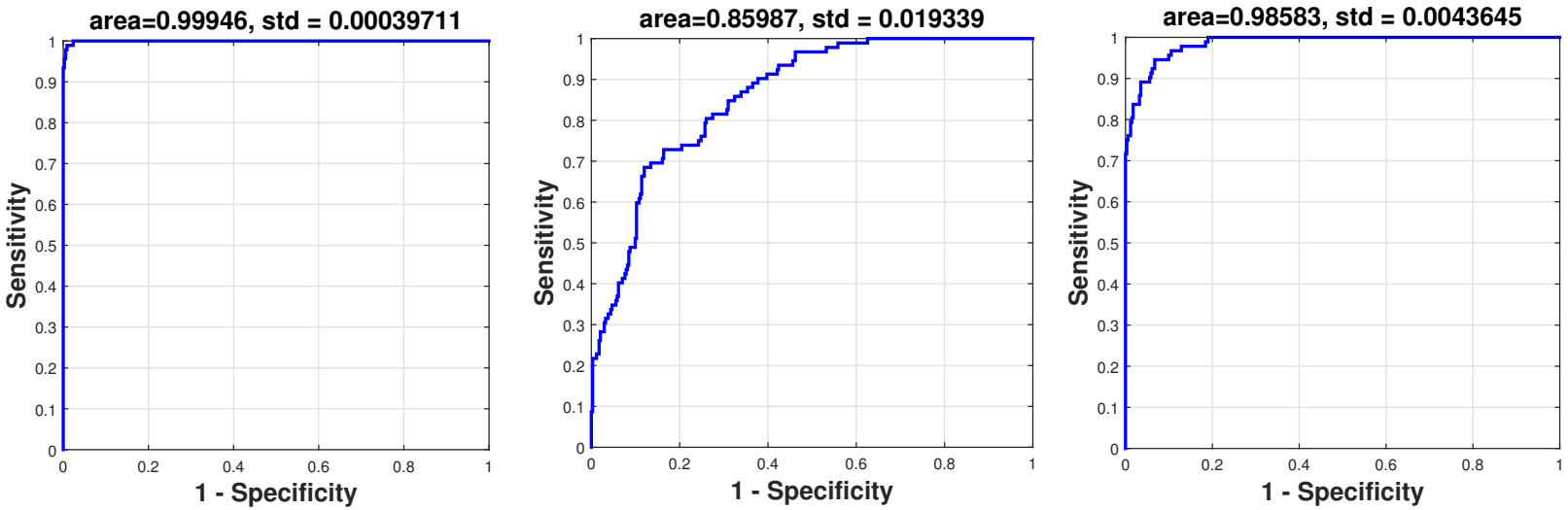
ROC Curves and AUC Values (With SMOTE)



((a)) ALL+ RBF-Kernel ((b)) ALL+ Linear-Kernel ((c)) ALL+ Polynomial-Kernel

Figure: Comparison of Various ROC Curves and AUC Values (With SMOTE)

ROC Curves and AUC Values (With SMOTE)



((a)) PCA+ RBF-Kernel ((b)) PCA+ Linear-Kernel ((c)) PCA+ Polynomial-Kernel

Figure: Comparison of Various ROC Curves and AUC Values (With SMOTE)

ROC Curve Analysis

The classifier using SMOTE has a better discriminatory power than the classifiers developed without SMOTE.

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.

Answer RQ4

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.

Table of Contents

Performance Results

The DS1 to DS7 dataset mentioned in slides refers to the seven projects: antlr4, junit, MapDB, mcMMO, mct, oryx and titan.

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 is always above 0.99 several times.

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.

Accuracy, F-Measure, Precision and Recall

		Without SMOTE							
		Linear-Kernel				Polynomial-Kernel			
Data Set	Metrics	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
DS1	PCA	94.90	0.97	1.00	0.95	95.30	0.98	1.00	0.95
DS2	ALL	98.60	0.99	1.00	0.99	99.40	1.00	1.00	0.99
DS2	PCA	98.60	0.99	1.00	0.99	99.20	1.00	1.00	0.99
DS3	ALL	99.00	1.00	1.00	0.99	99.80	1.00	1.00	1.00
DS3	PCA	99.00	1.00	1.00	0.99	99.50	1.00	1.00	1.00
DS4	ALL	95.50	0.98	1.00	0.96	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
DS5	ALL	99.30	1.00	1.00	0.99	99.30	1.00	1.00	0.99
DS5	PCA	99.30	1.00	1.00	0.99	99.40	1.00	1.00	0.99
DS6	ALL	97.20	0.99	1.00	0.97	97.00	0.98	1.00	0.97
DS6	PCA	97.00	0.98	1.00	0.97	97.40	0.99	1.00	0.97
DS7	ALL	99.00	0.99	1.00	0.99	99.60	1.00	1.00	1.00
DS7	PCA	99.00	0.99	1.00	0.99	99.10	1.00	1.00	0.99

Accuracy, F-Measure, Precision and Recall

		With SMOTE							
		Linear-Kernel				Polynomial-Kernel			
Data Set	Metrics	Accuracy	F-Measure	Precision	Recall	Accuracy	F-Measure	Precision	Recall
DS1	ALL	83.50	0.90	0.98	0.84	98.10	0.99	1.00	0.98
DS1	PCA	82.30	0.90	0.98	0.83	94.70	0.97	0.98	0.95
DS2	ALL	97.10	0.98	1.00	0.97	99.10	1.00	1.00	0.99
DS2	PCA	95.20	0.98	1.00	0.95	97.00	0.98	1.00	0.97
DS3	ALL	98.60	0.99	1.00	0.99	98.80	0.99	1.00	0.99
DS3	PCA	96.20	0.98	1.00	0.96	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
DS4	PCA	91.50	0.95	0.99	0.92	100.00	1.00	1.00	1.00
DS5	ALL	97.80	0.99	1.00	0.98	99.40	1.00	1.00	0.99
DS5	PCA	97.10	0.99	1.00	0.97	97.90	0.99	1.00	0.98
DS6	ALL	90.50	0.95	0.99	0.91	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
DS7	ALL	96.70	0.98	1.00	0.97	98.10	0.99	1.00	0.98
DS7	PCA	95.90	0.98	1.00	0.96	97.20	0.99	1.00	0.97

Performance Results

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.

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.

Descriptive Statistics and Box-Plot

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

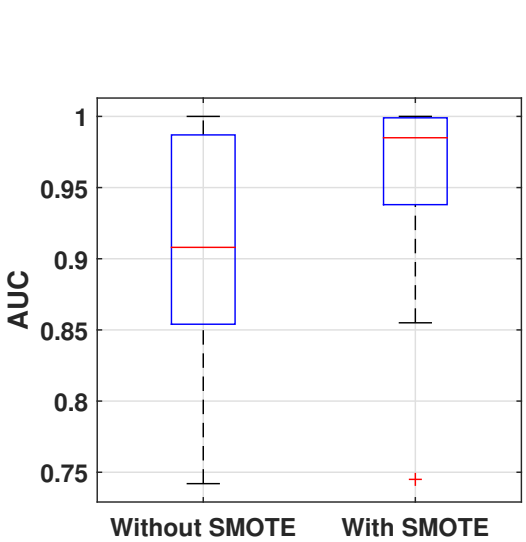
The mean and median AUC values for all metrics is 0.96 and 0.99 whereas the mean and median AUC values for PCA metrics is 0.91 and 0.93 respectively.

Performance in terms of AUC

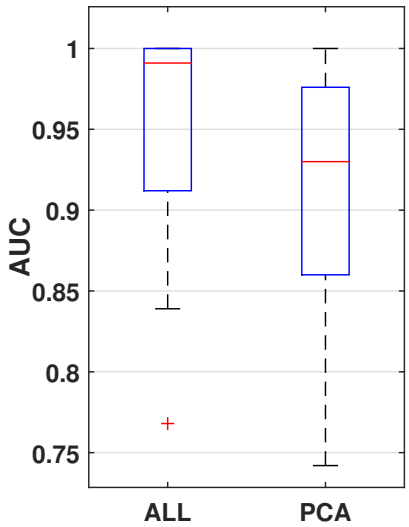
Table: Descriptive Statistics of the Performance in terms of AUC

	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

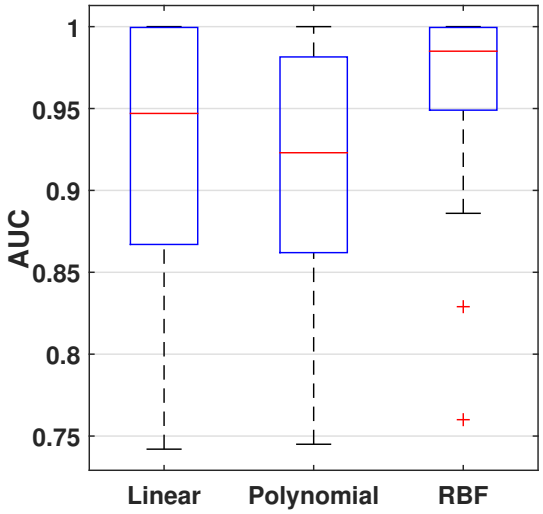
Box-Plot Diagrams of the AUC Values



((a)) With and Without SMOTE



((b)) ALL and PCA



((c)) Kernel Methods

Figure: 7 Possible Types of Population of Classifiers

Descriptive Statistics and Box-Plot

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.

AUC Values for All Combinations of Classifiers

Table: 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

p-value Results

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

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

T-Test Results (AUC Scores)

	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

((a)) With and Without SMOTE

	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

((b)) ALL and PCA

T-Test Results (AUC Scores) - 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

((c)) Different kernels

p-value Results - Polynomial and Linear Kernel

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 t-test shows that PCA does not perform better than the all metrics.

Answer RQ5

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.

Key Takeaways

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.

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 I

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

[2] Jehad Al Dallal. Constructing models for predicting extract subclass refactoring opportunities using object-oriented quality metrics. *Information and Software Technology*, 54(10):1125–1141, 2012.

[3] Abdulaziz Alkhalid, Mohammad Alshayeb, Sabri A Mahmoud, et al. Software refactoring at the class level using clustering techniques. *Journal of Research and Practice in Information Technology*, 43(4):285, 2011.

[4] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[5] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[6] Robert Feldt and Ana Magazinius. Validity threats in empirical software engineering research-an initial survey. In *SEKE*, pages 374–379, 2010.

References II

[7] Marios Fokaefs, Nikolaos Tsantalis, Eleni Stroulia, and Alexander Chatzigeorgiou. Jdeodorant: identification and application of extract class refactorings. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 1037–1039. ACM, 2011.

[8] Martin Fowler and Kent Beck. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

[9] István Kádár, Péter Hegedus, Rudolf Ferenc, and Tibor Gyimóthy. A code refactoring dataset and its assessment regarding software maintainability. In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, volume 1, pages 599–603. IEEE, 2016.

[10] István Kádár, Péter Hegedűs, Rudolf Ferenc, and Tibor 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*, page 10. ACM, 2016.

References III

[11] Miryung Kim, Matthew Gee, Alex Loh, and Napol 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*, pages 371–372. ACM, 2010.

[12] Hui Liu, Zhendong Niu, Zhiyi Ma, and Weizhong Shao. Identification of generalization refactoring opportunities. *Automated Software Engineering*, 20(1):81–110, 2013.

[13] Tom Mens and Tom Tourwé. A survey of software refactoring. *IEEE Transactions on software engineering*, 30(2):126–139, 2004.

[14] Dewayne E Perry, Adam A Porter, and Lawrence G Votta. Empirical studies of software engineering: a roadmap. In *Proceedings of the conference on The future of Software engineering*, pages 345–355. ACM, 2000.

[15] Jörgen Sandberg and Mats Alvesson. Ways of constructing research questions: gap-spotting or problematization? *Organization*, 18(1):23–44, 2011.

References IV

[16] JAK Suykens, Lukas Lukas, P Van Dooren, Bart De Moor, Joos Vandewalle, et al. Least squares support vector machine classifiers: a large scale algorithm. In *ECCTD*, volume 99, pages 839–842, 1999.

[17] Johan AK Suykens, Lukas Lukas, and Joos Vandewalle. Sparse approximation using least squares support vector machines. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, volume 2, pages 757–760. IEEE, 2000.

[18] Tom Tourwé and Tom Mens. Identifying refactoring opportunities using logic meta programming. In *Software Maintenance and Reengineering, 2003. Proceedings. Seventh European Conference on*, pages 91–100. IEEE, 2003.

[19] Nikolaos Tsantalis and Alexander Chatzigeorgiou. Identification of refactoring opportunities introducing polymorphism. *Journal of Systems and Software*, 83(3):391–404, 2010.

[20] Liming 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.