How Many Software Metrics Should be Selected for Defect Prediction?

Huanjing Wang
Western Kentucky University
huanjing.wang@wku.edu

Taghi M. Khoshgoftaar
Florida Atlantic University
khoshgof@fau.edu

Naeem Seliya
University of Michigan–Dearborn
nseliya@umich.edu

Abstract

A software practitioner is interested in the solution to "for a given project, what is the minimum number of software metrics that should be considered for building an effective defect prediction model?" During the development life cycle various software metrics are collected for different reasons. In the case of a metrics-based defect prediction model, an intelligent selection of software metrics prior to building defect predictors is likely to improve model performance. This study utilizes the proposed threshold-based feature selection technique to remove irrelevant and redundant software metrics (a.k.a. features or attributes). A comparative investigation is presented for evaluating the size of the selected feature subsets. The case study is based on software measurement data obtained from a real-world project, and the defect predictors are trained using three commonly used classifiers. The empirical case study results demonstrate that an effective defect predictor can be built with only three metrics; and moreover, model performances improved when over 98.5% of the software metrics were eliminated.

Introduction

A typical software defect prediction model is trained using software metrics and fault data that have been collected from previously-developed software releases or similar projects. The model can then be applied to program modules with unknown defect data. The characteristics of software metrics (a.k.a. features or attributes) influences the performance and effectiveness of the defect prediction model. We are interested in providing a solution to the problem of how many features should be used to build a defect predictor. From a practitioner's point of view, modeling with a small set of metrics is very appealing. In a given set of software metrics, it is likely that some of them are superfluous in characterizing the project's knowledge. Some of them provide redundant knowledge, or provide no new information, or in some cases, have an adverse effect on the defect prediction model. For example, recent studies demonstrate performance improvement of defect prediction models when irrelevant and redundant features are removed before modeling (Gao, Khoshgoftaar, & Wang 2009; Wang, Khoshgoftaar, & Hulse 2010).

Feature selection is the process of choosing a subset of features. It is broadly classified as feature ranking and feature subset selection, where feature ranking sorts the attributes according to their individual predictive power, and feature subset selection finds subsets of attributes that collectively have good predictive power. Filters are feature selection algorithms in which a feature subset is selected without involving any learning algorithm. A wrapper-based feature selection technique is learner-dependent, because they use feedback from a learning algorithm to determine which feature(s) to include in building a classification model.

An exhaustive literature review is out of scope to space considerations; however some key studies are presented. Guyon and Elisseeff (Guyon & Elisseeff 2003) provide a good overview on the various aspects of the feature selection problem and outlined key approaches used for feature selection, including feature construction, feature ranking, multivariate feature selection, efficient search methods, and feature validity assessment methods. Liu and Yu (Liu & Yu 2005) provide a comprehensive survey of feature selection algorithms and presented an integrated approach to intelligent feature selection.

The application of feature selection in the software quality and reliability domains is very limited. This study will add to the respective scientific knowledge base. Chen et al. (Chen *et al.* 2005) have studied the applications of wrapper-based feature selection in the context of software cost/effort estimation. They conclude that the reduced data set improved the estimation.

In the context of software defect prediction, we conduct an empirical investigation of our proposed Threshold-Based Feature Selection technique (TBFS). It belongs to the filter-based feature ranking techniques category. Five different and effective versions of TBFS feature rankers are considered in this study. These five different versions are based on five different performance metrics (Witten & Frank 2005), including Mutual Information (MI), Kolmogorov-Smirnov (KS), Deviance (DV), Area Under the ROC (Receiver Operating Characteristic) Curve (AUC), and Area Under the Precision-Recall Curve (PRC). Different sizes of feature subsets are selected using the five TBFS techniques. Subsequently, using the smaller subsets of selected attributes,

classification models are built with the following classifiers: multilayer perceptron (MLP), k-nearest neighbors (KNN), and logistic regression (LR). Finally, the performance of a classifier is evaluated with respect to the AUC performance metric.

The empirical validation of the different models was implemented through a case study of the nine data sets from the Eclipse project. In the experiments, ten runs of five-fold cross-validation were performed. We ranked the features and selected the top features according to their respective scores. The experimental results demonstrate that three attributes are sufficient to build defect prediction models. From a software practice point of view, researchers and practitioners would like to work with a smaller set of metrics for defect prediction rather than analyze a large number of metrics. This is the first study to investigate the impact of size of feature subsets on the performance of classifiers for TBFS.

Threshold-Based Feature Ranking

Filter-based feature ranking techniques (rankers) order the features independent of any learning (classification) algorithm. The best features are then selected from the ranking list. There are various ways to rank features – we used five different threshold-based feature ranking techniques (TBFS). The procedure of TBFS is shown in Algorithm 1. First each attribute's values are normalized between 0 and 1 by mapping F^j to $\hat{F}^j$. The normalized values are treated as posterior probabilities. Each independent attribute (software predictor variable) is then paired individually with the class attribute (fault-prone or not-fault-prone label) and the reduced data set is evaluated using five different performance metrics based on a set of posterior probabilities. In standard binary classification, the predicted class is assigned using the default decision threshold of 0.5. The default decision threshold is often not optimal, especially when the relative class distribution is imbalanced. Therefore, we propose the use of performance metrics that can be calculated at various points in the distribution of $\hat{F}^j$. At each threshold position, the values above the threshold are classified as positive, and negative otherwise. We then consider swapping the positive and negative, i.e. values about the threshold are classified as negative, and positive otherwise. Whichever direction of the positive and negative labeling produces the more optimal attribute values is used.

In a binary classification problem such as fault-prone (positive) or not-fault-prone (negative), there are four possible classification rates: true positive rate (TPR), true negative rate (TNR), false positive rate (FPR), and false negative rate (FNR). These four classification rates can be calculated at each threshold $t \in [0, 1]$ relative to the normalized attribute $\hat{F}^j$. The threshold-based feature ranking technique utilizes the classification rates as described below.

- *Mutual Information (MI)* measures the mutual dependence of two random variables. High mutual information indicates a large reduction in uncertainty, and zero mutual information between two random variables means the variables are independent.

Algorithm 1: Threshold-Based Feature Selection

input :

1. Data set D with features $F^j, j = 1, \ldots, m$;
2. Each instance $x \in D$ is assigned to one of two classes $c(x) \in \{fp, nfp\}$;
3. The value of attribute F^j for instance x is denoted as $F^j(x)$;
4. Metric $\omega \in$ {MI, KS, DV, AUC, PRC};
5. A predefined threshold: number (or percentage) of the features to be selected.

output:

Selected feature subsets.

for $F^j, j = 1, \ldots, m$ **do**

  Normalize $F^j \mapsto \hat{F}^j = \frac{F^j - \min(F^j)}{\max(F^j) - \min(F^j)}$;

  Calculate metric ω using attribute $\hat{F}^j$ and class attribute at various decision threshold in the distribution of $\hat{F}^j$. The optimal ω is used, $\omega(\hat{F}^j)$.

Create feature ranking R using $\omega(\hat{F}^j)\ \forall j$.

Select features according to feature ranking R and a predefined threshold.

- *Kolmogorov-Smirnov (KS)* utilizes the Kolmogorov-Smirnov statistic to measure the maximum difference between the empirical distribution functions of the attribute (software metric) values of the instances (program modules) in each class. It is effectively the maximum difference between the curves generated by the true positive and false positive rates as the decision threshold changes between 0 and 1.
- *Deviance (DV)* is the residual sum of squares based on a threshold t. It measures the sum of the squared errors from the mean class given a partitioning of the space based on the threshold t. As deviance represents errors, the minimum value is considered optimal.
- *Area Under the ROC (Receiver Operating Characteristic) Curve* (AUC) has been widely used to measure classification model performance (Fawcett 2006). The ROC curve is used to characterize the trade-off between true positive rate and false positive rate. In this study, ROC curves are generated by varying the decision threshold t used to transform the normalized attribute values into a predicted class.
- *Area Under the Precision-Recall Curve* (PRC): is a single-value measure that originated from the area of information retrieval. The area under the PRC ranges from 0 to 1. The PRC diagram depicts the trade off between recall and precision.

The TBFS can be extended to incorporate additional metrics, such as F-measure, Odds Ratio, Gini Index, and Geometric Mean. Our preliminary study showed that MI, KS, DV, AUC, and PRC-based TBFS performed better. We implemented the TBFS technique within the WEKA framework (Witten & Frank 2005).

Classifiers

The three classifiers (Witten & Frank 2005) used in our case study are Multilayer Perceptron, k-Nearest Neighbors, and Logistic Regression. All three learners themselves do not have a built-in feature selection capability and are commonly used in the software engineering community. All

classifiers were implemented in the WEKA tool (Witten & Frank 2005). We used default parameter settings for the different learners as specified in WEKA. Parameter settings were changed only when doing so improved classifier performance significantly.

Multilayer Perceptrons (MLP) attempt to artificially mimic the functioning of a biological nervous system. In our study, the `hiddenLayers` parameter was changed to '3' to define a network with one hidden layer containing three nodes, and the `validationSetSize` parameter was changed to '10' to cause the classifier to leave 10% of the training data aside to be used as a validation set to determine when to stop the iterative training process (Witten & Frank 2005).

k **nearest neighbors** (KNN) classifiers, also known as IBK (instance-based classifier), belong to the category of lazy learners. The choice of distance metric is critical. KNN was built with changes to three parameters. The `distanceWeighting` parameter was set to 'Weight by 1/distance', the `kNN` parameter was set to '30', and the `crossValidate` parameter was set to 'true'. In addition, we modified the learner so that it chooses the `k` which produces the highest mean of the accuracies for each class (i.e., the arithmetic mean between the true positive rate and true negative rate) (Witten & Frank 2005).

Logistic regression (LR) is a statistical technique that can be used to solve binary classification problems. Based on the training data, a logistic regression model is created which is used to decide the class membership of future instances (Witten & Frank 2005).

Classifier Performance Metric

Traditional performance measures such as F-measure, overall classification accuracy, or misclassification rate are inappropriate when dealing with the classification of imbalanced data. In a domain such as software quality prediction, the number of *fp* (fault-prone) modules is much lower than the number of *nfp* (not-fault-prone) modules. Instead, we use a performance metric that considers the ability of a classifier to differentiate between the two classes: the Area Under the ROC (Receiver Operating Characteristic) curve (AUC). It has been shown that AUC has lower variance and is more reliable than other performance metrics (such as precision, recall, and F-measure) for software defect prediction (Jiang *et al.* 2009).

The AUC is a single-value measurement, whose value ranges from 0 to 1. The ROC curve is used to characterize the trade-off between hit (true positive) rate and false alarm (false positive) rate. A classifier that provides a large area under the curve is preferable over a classifier with a smaller area under the curve. Traditional performance metrics consider only the default decision threshold of 0.5. ROC curves illustrate the performance across all decision thresholds. A perfect classifier provides an AUC that equals 1. In our study, AUC is used both to select the most predictive subset of features using TBFS and to evaluate the classifiers constructed using those set of features.

Table 1: Software Data Set Characteristics

Data	#Metrics	#Modules	#fp	%fp	#nfp	%nfp
Eclipse 2.0-10	208	377	23	6%	354	94%
Eclipse 2.0-5	208	377	52	14%	325	86%
Eclipse 2.0-3	208	377	101	27%	276	73%
Eclipse 2.1-5	208	434	34	8%	400	92%
Eclipse 2.1-4	208	434	50	12%	384	88%
Eclipse 2.1-2	208	434	125	29%	309	71%
Eclipse 3.0-10	208	661	41	6%	620	94%
Eclipse 3.0-5	208	661	98	15%	563	85%
Eclipse 3.0-3	208	661	157	24%	504	76%

Software Measurement Data

The software metrics and fault data for this case study were collected from a real-world software project, i.e., the Eclipse project (Zimmermann, Premraj, & Zeller 2007). From the PROMISE data repository (Zimmermann, Premraj, & Zeller 2007), we obtained the Eclipse defect counts and complexity metrics data set. In particular, we use the metrics and defects data at the software package level. The original data for the Eclipse packages consists of three releases denoted 2.0, 2.1, and 3.0 respectively. We transform the original data by: (1) removing all nonnumeric attributes, including the package names, and (2) converting the post-release defects attribute to a binary class attribute: fault-prone (*fp*) and not-fault-prone (*nfp*). Membership in each class is determined by a post-release defects threshold t, which separates *fp* from *nfp* packages by classifying packages with t or more post-release defects as *fp* and the remaining as *nfp*. In our study, we use $t \in \{10, 5, 3\}$ for release 2.0 and 3.0 while we use $t \in \{5, 4, 2\}$ for release 2.1. These values are selected in order to have data sets with different levels of class imbalance. All nine derived data sets contain 208 independent attributes (predictor metrics). Releases 2.0, 2.1, and 3.0 contain 377, 434, and 661 program modules respectively.

The nine data sets used in this work reflect different distributions of class skew (i.e, the percentage of *fp* modules in the data set). Table 1 lists the characteristics of the nine data sets utilized in this work.

Empirical Design

When using a filter-based feature ranking technique, the number of features (size of feature subset) that will be selected is a modeling parameter that must be known to the techniques. In this study, we also investigate the impact of size of feature subset. We rank the metrics and choose the top 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, and 20 metrics according to their respective scores. Following the feature selection techniques, the three different types of classification models are constructed with data sets containing only the selected attributes. We also construct the classification models using the original data sets which include 208 metrics. The defect prediction models are evaluated in terms of the AUC performance metric. We seek to understand the impact of (1) different size of feature subset; (2) the five filter-based rankers; and (3) the three different learners on the models' predictive power. In the experiments, ten independent runs

of five-fold cross-validation were performed. In total, 81000 (9 data sets * 12 sizes * 5 rankers * 3 learners * 10 runs * 5 folds) + 1350 (9 data sets * 3 learners * 10 runs * 5 folds) = 82350 models were build in our case study.

Empirical Results

Table 2 shows the AUC results for each individual learner. Note that each value presented in the table is the average over the ten runs of five-fold cross-validation outcomes across all nine data sets. The best ranker for a given learner (classifier) is shown in **boldfaced**. Each value in the table is determined by three dimensions: (1) feature ranking techniques (MI, KS, DV, AUC, and PRC); (2) classifiers (MLP, KNN, and LR); and (3) size of feature subset (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, and 20). A total of 180 values are considered in the three tables. The results are mapped to figure 1. Table 3 shows the classification results on original data sets without using any feature selection techniques. We also summarize the average performance (last row of table 3) for each learner across nine data sets. From these tables and figures, we can observe the following:

- For the five threshold-based rankers, AUC performed best on average. Of course, once the subset of features is fixed, the classification performance is determined by the learners we select. For instance, the feature subset selected by AUC demonstrated better performance than the feature subsets selected by other rankers for all 12 cases when MLP was applied (see Table 2(a)); nine out of 12 cases when KNN was used (see Table 2(b)); and seven out of 12 cases for LR (see Table 2(c)).
- For the MLP learner, the best model is built with four features selected by the AUC ranker;
- For the KNN learner, the best model is built with seven features selected by the AUC ranker;
- For the LR learner, the best model is built with three features selected by the AUC ranker;
- Overall, the best classification model is built with three features selected by the AUC ranker using LR learner.
- The classification models built with smaller feature subsets are better than models built with complete feature set.

Analysis of Results

We performed a one-way ANalysis Of VAriance (ANOVA) F-test to statistically examine the various effects on performances of the classification models across all the data sets. The ANOVA tests were performed on the three classifiers individually. The main factor represents the 13 different sizes (12 sizes of subset and full data sets) that were studied for the attribute selection. The null hypothesis for the ANOVA test is that all the group population means are the same, while the alternate hypothesis is that at least one pair of means is different.

Table 4 shows the ANOVA results. It includes three sub-tables, each representing the result for each individual learner (MLP, KNN, and LR). All the p values are zero, indicating that for the main factor, the alternate hypothesis is

Table 2: Empirical results in terms of AUC

(a) MLP

Size	MI	KS	DV	AUC	PRC
1	0.8659	0.8627	0.8480	0.8803	0.8638
2	0.8771	0.8724	0.8686	0.8856	0.8817
3	0.8741	0.8713	0.8704	0.8860	0.8792
4	0.8742	0.8708	0.8718	**0.8862**	0.8787
5	0.8732	0.8697	0.8711	0.8853	0.8766
6	0.8711	0.8684	0.8705	0.8840	0.8738
7	0.8690	0.8679	0.8674	0.8810	0.8724
8	0.8676	0.8619	0.8635	0.8778	0.8696
9	0.8673	0.8622	0.8610	0.8766	0.8656
10	0.8665	0.8625	0.8605	0.8757	0.8644
15	0.8598	0.8534	0.8550	0.8657	0.8582
20	0.8497	0.8491	0.8496	0.8587	0.8509

(b) KNN

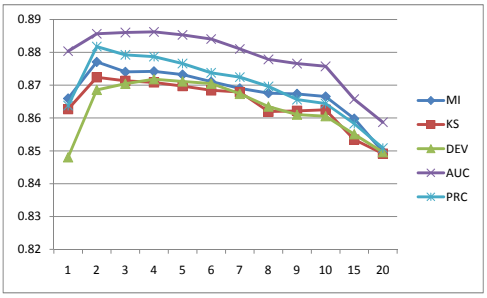
Size	MI	KS	DV	AUC	PRC
1	0.8555	0.8543	0.8406	0.8725	0.8568
2	0.8726	0.8696	0.8659	0.8832	0.8778
3	0.8745	0.8717	0.8668	0.8847	0.8792
4	0.8773	0.8734	0.8731	0.8851	0.8819
5	0.8779	0.8721	0.8772	0.8873	0.8827
6	0.8783	0.8744	0.8809	0.8881	0.8847
7	0.8796	0.8758	0.8832	**0.8885**	0.8855
8	0.8804	0.8765	0.8839	0.8876	0.8863
9	0.8811	0.8768	0.8832	0.8871	0.8866
10	0.8813	0.8766	0.8830	0.8856	0.8875
15	0.8808	0.8762	0.8815	0.8840	0.8845
20	0.8786	0.8754	0.8792	0.8809	0.8813

(c) LR

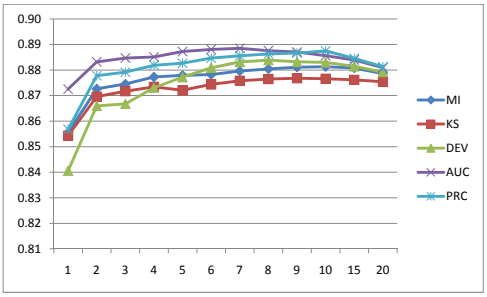
Size	MI	KS	DV	AUC	PRC
1	0.8842	0.8824	0.8734	0.8959	0.8840
2	0.8992	0.8973	0.8943	0.9057	0.9022
3	0.9019	0.8996	0.8972	**0.9077**	0.9019
4	0.9021	0.8980	0.8978	0.9067	0.9023
5	0.9023	0.8965	0.8964	0.9039	0.9000
6	0.8976	0.8934	0.8956	0.9016	0.8929
7	0.8956	0.8899	0.8925	0.8970	0.8901
8	0.8915	0.8843	0.8878	0.8911	0.8856
9	0.8881	0.8823	0.8841	0.8838	0.8785
10	0.8857	0.8804	0.8819	0.8789	0.8740
15	0.8661	0.8599	0.8656	0.8606	0.8592
20	0.8481	0.8431	0.8458	0.8414	0.8415

Table 3: Classification Results on Original Data sets

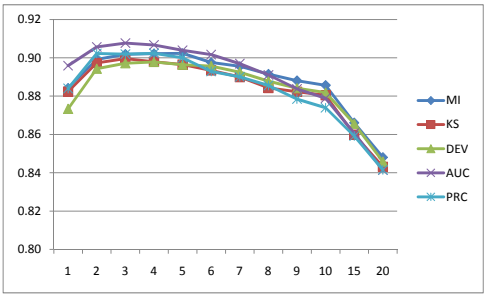
Data	MLP	KNN	LR
Eclipse 2.0-10	0.8102	0.7999	0.7390
Eclipse 2.0-5	0.8463	0.7949	0.7465
Eclipse 2.0-3	0.7858	0.7512	0.6796
Eclipse 2.1-5	0.7687	0.6382	0.6262
Eclipse 2.1-4	0.7361	0.6652	0.6531
Eclipse 2.1-2	0.7621	0.7412	0.6443
Eclipse 3.0-10	0.7697	0.7325	0.6596
Eclipse 3.0-5	0.8422	0.8114	0.7632
Eclipse 3.0-3	0.8076	0.7770	0.7452
Average	0.7921	0.7457	0.6952



(a) MLP



(b) KNN



(c) LR

Figure 1: Classification Results

Table 4: Analysis of Variance, Size

(a) MLP

Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
A	0.8317	12	0.06931	37.99	0
Error	9.9923	5477	0.00182		
Total	10.824	5489			

(b) KNN

Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
A	1.84508	12	0.15376	107.29	0
Error	7.84892	5477	0.00143		
Total	9.694	5489			

(c) LR

Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
A	4.7388	12	0.3949	246	0
Error	8.7923	5477	0.00161		
Total	13.5311	5489			

Table 5: Analysis of Variance, Learner

Source	Sum Sq.	d.f.	Mean Sq.	F	p-value
A	0.626	2	0.31301	180.1	0
Error	28.1491	16197	0.00174		
Total	28.7751	16199			

accepted, namely, at least two group means are significantly different from each other. In this study, we also performed the multiple comparison tests using Tukey's Honestly Significant Difference (HSD) criterion. All tests of statistical significance utilize a significance level $\alpha = 0.05$. Both ANOVA and multiple comparison tests were implemented in MATLAB. Multiple comparison results as shown in Figure 2 display graphs with each group mean represented by a symbol ($\circ$) and the 95% confidence interval as a line through the symbol. Two means are significantly different if their intervals are disjoint, and are not significantly different if their intervals overlap. The ANOVA and HSD results demonstrate the following points.

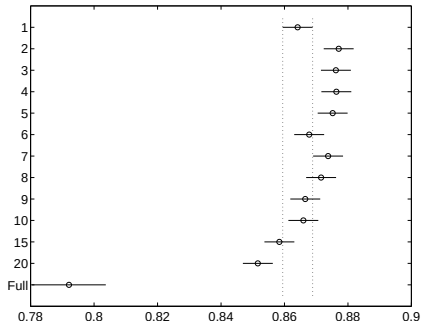
- Feature selection presents superior performance and performs significantly better than no feature selection.
- The classification models built with seven features outperformed models built with fewer features or more features when KNN is used to build classification models.
- For the MLP classifier, models built with two features outperformed other sizes;
- For for LR classifier, models built with three features performed best. However there is no significant difference the models built with two, three, four, and five features.

We also performed a one-way ANOVA test for learners regardless of rankers and sizes (see Table 5 and Figure 3). We conclude that LR perform significantly better than KNN and MLP. Overall, only three features are sufficient to build classification models, therefore this is our recommendation for this study. However, the selected classification algorithm is important in determining the size of the best features.

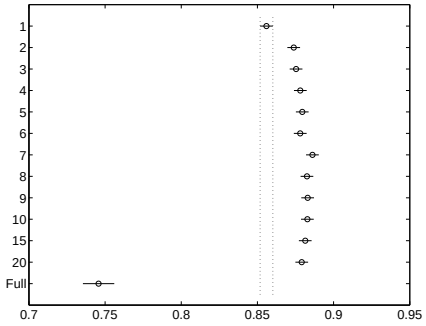
Conclusion

The paper addresses the question of "what is the minimum number of software metrics that should be used to build a software defect prediction model for a given system?" It is not uncommon to see excessive software metrics collected and stored in software project repositories. Selection of software metrics that are important for software defect prediction is critical, and useful to the practitioner. In this study, we investigated five threshold-based feature ranking techniques to select different sizes (subsets) of software metrics. The main goal of this study is to examine and compare the sizes of selected feature subsets and evaluate their effectiveness in the context of software defect prediction.

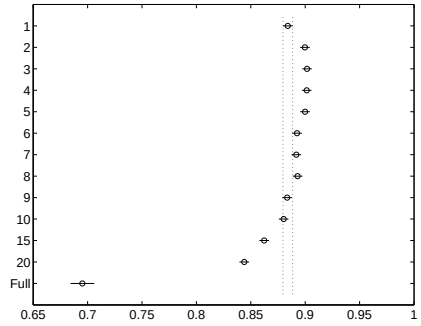
The experiments were conducted on three groups of software data sets, each group having three separate releases. We then built classification models with the selected feature subsets using three different classifiers. We also compared the models built with selected metrics and models built with



(a) MLP



(b) KNN



(c) LR

Figure 2: Multiple Comparison, Size

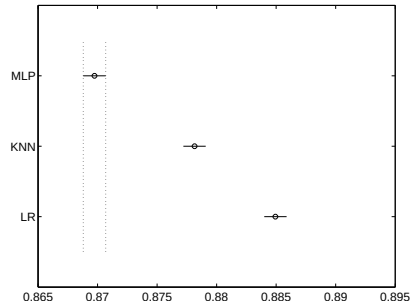


Figure 3: Multiple Comparison, Learner

the complete feature set. The experiments demonstrate that in our case study on average three software metrics are sufficient to build effective software defect prediction models. Furthermore, even after removing 98.5% of the available number of software metrics the defect prediction models performed better than when we used all the features. This is an important fact for software practitioners, since practitioners prefer using fewer software metrics for data collection, management, comprehension, and modeling. We note that, using a specific classification algorithm can yield different results.

Future work may include experiments using additional data sets from other software engineering and non-software engineering domains. Moreover, more classifiers (such as C4.5 (Quinlan 1993)) can be considered for model building process. An interesting area of research would be to determine which specific metrics are selected in the end to build the prediction model.

References

Chen, Z.; Menzies, T.; Port, D.; and Boehm, B. 2005. Finding the right data for software cost modeling. *IEEE Software* (22):38–46.

Fawcett, T. 2006. An introduction to ROC analysis. *Pattern Recognition Letters* 27(8):861–874.

Gao, K.; Khoshgoftaar, T. M.; and Wang, H. 2009. An empirical investigation of filter attribute selection techniques for software quality classification. In *Proceedings of the 10th IEEE International Conference on Information Reuse and Integration*, 272–277.

Guyon, I., and Elisseeff, A. 2003. An introduction to variable and feature selection. *Journal of Machine Learning Research* 3:1157–1182.

Jiang, Y.; Lin, J.; Cukic, B.; and Menzies, T. 2009. Variance analysis in software fault prediction models. *Software Reliability Engineering, International Symposium on* 0:99–108.

Liu, H., and Yu, L. 2005. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering* 17(4):491–502.

Quinlan, J. 1993. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

Wang, H.; Khoshgoftaar, T. M.; and Hulse, J. V. 2010. A comparative study of threshold-based feature selection techniques. In *IEEE International Conference on Granular Computing*, 499–504.

Witten, I. H., and Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2 edition.

Zimmermann, T.; Premraj, R.; and Zeller, A. 2007. Predicting defects for eclipse. In *ICSEW '07: Proceedings of the 29th International Conference on Software Engineering Workshops*, 76. Washington, DC, USA: IEEE Computer Society.