International Journal of Reliability, Quality and Safety Engineering
Vol. 17, No. 5 (2010) 425–464
© World Scientific Publishing Company
DOI: 10.1142/S0218539310003883

World Scientific
www.worldscientific.com

# WRAPPER-BASED FEATURE RANKING TECHNIQUES FOR DETERMINING RELEVANCE OF SOFTWARE ENGINEERING METRICS

WILKER ALTIDOR* and TAGHI M. KHOSHGOFTAAR†

*Department of Computer Science and Engineering*
*Florida Atlantic University, 777 Glades Road*
*Boca Raton, Florida 33431, USA*
**wilker.altidor@gmail.com*
†*taghi@cse.fau.edu*

KEHAN GAO

*Department of Mathematics and Computer Science*
*Eastern Connecticut State University, 83 Windham Street*
*Willimantic, Connecticut 06226, USA*
*gaok@easternct.edu*

Received 23 March 2010
Revised 17 July 2010

Classification, an important data mining function that assigns class label to items in a collection, is of practical applications in various domains. In software engineering, for instance, a common classification problem is to determine the quality of a software item. In such a problem, software metrics represent the independent features while the fault proneness represents the class label. With many classification problems, one must often deal with the presence of irrelevant features in the feature space. That, coupled with class imbalance, renders the task of discriminating one class from another rather difficult. In this study, we empirically evaluate our proposed wrapper-based feature ranking where nine performance metrics aided by a particular learner and a methodology are considered. We examine five learners and take three different approaches, each in conjunction with one of three different methodologies: 3-fold Cross-Validation, 3-fold Cross-Validation Risk Impact, and a combination of the two. In this study, we consider two sets of software engineering datasets. To evaluate the classifier performance after feature selection has been applied, we use Area Under Receiver Operating Characteristic curve as the performance evaluator. We investigate the performance of feature selection as we vary the three factors that form the foundation of the wrapper-based feature ranking. We show that the performance is conditioned by not only the choice of methodology but also the learner. We also evaluate the effect of sampling on wrapper-based feature ranking. Finally, we provide guidance as to which software metrics are relevant in software defect prediction problems and how the number of software metrics can be selected when using wrapper-based feature ranking.

*Keywords*: Feature selection; wrapper-based feature ranking; ranker aid; performance measures; sampling techniques; software metrics.

## 1. Introduction

The basic ingredients for software quality control and management are the software metrics collected at various states of a software development life cycle. It is well accepted among software practitioners, researchers, and stakeholders that software quality improvement must begin with targeted effort on improving the software development and maintenance process. For this reason, many software organizations have assembled very large repositories of data, collected electronically, pertaining to their software releases at different states of development. Data collected are often associated with defects found during pre-release as well as post-release. Given that data collection has become common practice in the domain of software development engineering, the problem for most software development organizations is not the availability of data nor the ability to access them, but rather the mining of the data to effectively discover the relevancy of the data and ultimately potential patterns that, when applied to new releases, can enhance their quality. By using data mining techniques to mine their software repositories, organizations can build predictive models to ensure the quality of future software products or releases. They are able to detect early in the development process fault prone areas upon which they can focus their fault detection effort. However, the software metrics often collected with good reasons may not all be relevant for predicting the fault proneness of software components, modules, or releases. Moreover, these same data may not be immune to the high dimensionality or class imbalance problem often encountered in data from other application domains such as text categorization and gene expression classification. This thus creates the need for the use of feature selection, which helps separate relevant software metrics from irrelevant or redundant ones, thereby selecting the set of software metrics that are best predictors of fault proneness for new components, modules, or releases.

In this empirical study, we present our proposed feature ranking called wrapper-based feature ranking, also referred to as classifier-aided (or learner-aided) feature ranking. Our wrapper-based approach is unique in that it combines aspects of wrapper techniques (i.e., using a learner to evaluate a subset of attributes) and individual feature evaluation (i.e., ranking methods such as filtering). To evaluate feature relevance, we use nine performance metrics commonly used as classifier performance evaluators. These are: Overall Accuracy, F-Measure, Geometric Mean, Arithmetic Mean, Area under the Receiver Operating Characteristic (ROC) Curve or AUC, Area under the Precision-Recall Curve or PRC, Best F-Measure, Best Geometric Mean, and Best Arithmetic Mean. We apply these metrics, which form the basis of our wrapper-based feature ranking, to software measurement data. We show differences among the ranking techniques through the lenses of AUC. We consider three approaches: one in which we use a 3-fold Cross-Validation, another, a 3-fold Cross-Validation Risk Impact method, yet another where we combine the first two methods, which we denote COMBO. For each approach, five well studied learners are used, and we show which approach leads to better performance results

for which of the five learners in terms of AUC. In addition, we examine the effect of sampling by applying seven different sampling techniques prior to wrapper-based feature ranking. We demonstrate the effectiveness and viability of our proposed wrapper-based feature ranking with two sets of experiments. The first set is with a Large Legacy Telecommunications Software (LLTS) system. The second set is with the Eclipse Defect Counts and Complexity Metrics datasets. For both sets of experiments, we report on which choices of classifier, performance metric, and methodology lead to the highest classification performance. One significant result of this empirical study is the robustness of the wrapper-based ranking techniques when Naïve Bayes is the learner for both ranking and classification. Another significant result is the validation of our new method for determining which software metrics are most useful for defect prediction. This method for capturing software metrics with most defect predictive power represents a breakthrough not only for software engineering but also for other domains in which feature selection is a pre-requisite activity.

The remainder of this paper is organized as follows. Section 2 provides some background on feature selection, including how it is used in data mining. Section 3 provides the setup for the experiments. Sections 4 and 5 present the experimental results with LLTS and Eclipse respectively. In Sec. 6, we present our overall analysis. Finally, our conclusion and future work are discussed in Sec. 7.

## 2. Related Work

Feature selection has been applied in many application domains for many years. In the domain of software engineering, we count, for instance, the work of Livshits *et al.*[1] in which they presented DynaMine, "a tool for discovering usage patterns and detecting their violations in large software systems". In their study they proposed the use of data mining techniques to discover patterns from software revision histories. Shirabad *et al.*[2] also showed how data mining can be used to discover patterns from software updates records. These patterns could be used to identify files that potentially could require changes due to a change in another source file. The work of Dick *et al.*[3] focused on the use of fuzzy clustering, an unsupervised learning technique, for the analysis of software metric databases. Catal *et al.*,[4] in their software fault prediction study, considered the effect of feature selection techniques along with dataset size and metric sets on the software fault prediction model's performance.

Feature selection techniques may be divided into four categories: First, there are *Filters*, which evaluate feature relevance by examining the intrinsic characteristics of the data without the use of a classifier,[5] and can be subdivided into three types: Ranking,[6,7] Subset evaluation,[8,9] and a new framework of feature selection in which redundancy analysis is decoupled from relevance analysis.[10] Second, *Wrappers* evaluate feature relevance by applying a predetermined classifier on subsets of attributes. The same classifier or inductive algorithm is used to both select

the relevant features and execute the mining process.[11] They are considered to be better at producing feature subsets, but are computationally expensive. Third, a *Hybrid* approach, which is a combination of filter and wrapper methods,[12,13] eliminates unwanted features by using a ranking technique and resolves any multicollinearity problem by using a wrapper method on the subset of relevant features. Finally, the *Embedded* methods are the algorithms that perform feature selection and induction simultaneously.[5,14] Because they incorporate feature selection as part of the training process they may be more efficient given that data splitting into training and validation set is not required and retraining every feature subset investigated from scratch is avoided.[6]

Chen *et al.*[15] reported that the use of one type of feature selection, namely wrappers, to COCOMO's software cost estimates, drastically improved the model's predictive power. Rodriguez *et al.*[16] showed that the use of two commonly known feature selection types, filters and wrappers, on software engineering datasets resulted in better or equal estimates. They only considered two of each type. They used consistency and correlation measures for the filter model while they used a nearest neighbor (IB1) and a decision tree (C4.5) classifiers for the wrapper model.

Our proposed feature ranking, namely wrapper-based feature ranking, is empirically investigated in this paper. To our knowledge, related work has seldom been found in the domain of software engineering and data mining applications. This new type of feature ranking technique, unlike other ones published in literature, make use of a learner in conjunction with a performance metric to determine ranking. The work of Ruiz *et al.*[17] on gene expression microarray data is the closest to our research. In their study, they presented a gene selection method based on the hybrid model of feature selection. With their novel hybrid approach to feature selection, they used either a wrapper-based or filter-based model as evaluation measure criteria to rank the genes. Their primary focus was not on these models, but rather on the overall hybrid approach and its efficiency as compared with a number of different filter and wrapper based feature subset selection methods. The ranking function was just an enabler, and its use was to advance a different research goal.

## 3. Experimental Design

### 3.1. *Methodologies*

In this study we consider three different methodologies along with five commonly known learners and nine performance metrics that together constitute our wrapper-based ranking techniques. For the first methodology, we use a 3-fold cross-validation (CV), where the set of $N$ instances in the training dataset is partitioned into 3 equal sets of size $\frac{N}{3}$. For each fold (partition), a classifier is trained on the other 2 folds (partitions), then tested on the remaining fold. This whole procedure is performed a total of 3 times using a different holdout partition each time, and the results on each partition are averaged out. The training dataset for this procedure is a new data structure, where each instance of the newly created structure contains only

two features: one independent feature and the dependent feature. For a training dataset with $M$ independent features, $M$ data structures are obtained. A set of nine performance metrics are obtained for each separate data structure and for each of the five learners, indicating the performance scores associated with that particular independent feature. Performance scores are obtained in the same manner for all features in the dataset, one at a time. After the nine performance metrics have been obtained for all five learners and for every feature, features are ranked based on a particular performance metric value for each learner. Consider the scenario shown in Table 1, an example of the wrapper-based feature ranking with the CV methodology. The first line shows the independent features of some dataset example. A performance score is obtained for each independent feature in the list. The last line shows the features ranked according to their scores, which are based on some performance metric.

The second methodology employs a 3-fold cross-validation risk impact (CV-R). CV-R is similar to CV. However, with CV-R each instance of the training dataset contains $M-1$ independent features plus the dependent feature. That is, for each of the five learners, we first build a model with all independent features and the dependent feature, obtaining all performance measures. For each independent feature, we transform the original training dataset by removing that feature, then we build a model with one-less independent features and the class feature to get the associated performance measure. The risk impact for the $i$th feature is calculated as follows: $\text{Impact}^i = PM_M - PM^i_{M-1}$, where $PM$ denotes one of the nine performance metrics and $0 \leq i \leq M-1$ indicates the $i$th feature. $PM_M$ represents PM obtained by $M$ features while $PM^i_{M-1}$ represents PM obtained by $M-1$ features, and the excluded one is the $i$th feature. Each feature is ranked based on its risk impact, with the highest rank being the one with the highest value of impact. CV-R requires the construction of $M+1$ models, one of which uses all $M$ independent features, while the remaining $M$ models use $M-1$ features. Consider the scenario shown in Table 2, an example of the wrapper-based feature ranking with the CV-R methodology. The first line shows the independent features of some dataset example. A performance

Table 1. Ex. of wrapper-based ranking with CV.

| Features | $f_0$ | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ | $f_7$ |
|---|---|---|---|---|---|---|---|---|
| | Evaluation | | Evaluated | | Score | | | |
| | 1 | | $f_0$ | | 0.87 | | | |
| | 2 | | $f_1$ | | 0.83 | | | |
| | 3 | | $f_2$ | | 0.89 | | | |
| | 4 | | $f_3$ | | 0.93 | | | |
| | 5 | | $f_4$ | | 0.90 | | | |
| | 6 | | $f_5$ | | 0.78 | | | |
| | 7 | | $f_6$ | | 0.80 | | | |
| | 8 | | $f_7$ | | 0.88 | | | |
| Rank | $f_3$ | $f_4$ | $f_2$ | $f_7$ | $f_0$ | $f_1$ | $f_6$ | $f_5$ |

Table 2. Ex. of wrapper-based ranking with CV-R.

| Features | $f_0$ | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ | $f_7$ |
|---|---|---|---|---|---|---|---|---|
| Evaluation | Evaluated | | | | Score | | Impact | |
| 1 | $f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7$ | | | | 0.93 | | — | |
| 2 | $f_1, f_2, f_3, f_4, f_5, f_6, f_7$ | | | | 0.87 | | 0.06 | |
| 3 | $f_0, f_2, f_3, f_4, f_5, f_6, f_7$ | | | | 0.85 | | 0.08 | |
| 4 | $f_0, f_1, f_3, f_4, f_5, f_6, f_7$ | | | | 0.83 | | 0.10 | |
| 5 | $f_0, f_1, f_2, f_4, f_5, f_6, f_7$ | | | | 0.90 | | 0.02 | |
| 6 | $f_0, f_1, f_2, f_3, f_5, f_6, f_7$ | | | | 0.81 | | 0.12 | |
| 7 | $f_0, f_1, f_2, f_3, f_4, f_6, f_7$ | | | | 0.80 | | 0.13 | |
| 8 | $f_0, f_1, f_2, f_3, f_4, f_5, f_7$ | | | | 0.88 | | 0.05 | |
| 9 | $f_0, f_1, f_2, f_3, f_4, f_5, f_6$ | | | | 0.89 | | 0.04 | |
| Rank | $f_5$ | $f_4$ | $f_2$ | $f_1$ | $f_0$ | $f_6$ | $f_7$ | $f_3$ |

score is obtained for each independent feature in the list along with an impact score. The performance score is based on some performance metric. The last line shows the features ranked according to their impact score. As we can see, this approach is computationally expensive, especially for datasets with many features/attributes.

Our third methodology combines characteristics of both CV and CV-R to form what we call COMBO. We proceed in the same manner as in CV by obtaining the performance scores for all independent features in the training dataset. This time, for each performance metric and learner, we select twice as many top features as we plan to use in the final model. We then use CV-R on the selected features. We first obtain all performance measures by building a model with the selected independent features and the class feature for each of the five learners. For each of the selected independent features, we build a model with one-less independent features and the class feature to get the associated performance measure. Similar to the second approach, we calculate the risk impact as follows: $\text{Impact}^i = PM_S - PM^i_{S-1}$, where $0 \leq i \leq S-1$ indicates the $i$th selected feature.

For all three approaches (CV, CV-R, and COMBO), we select the top ranked features for each performance metric, and we build models using the selected features for each learner on the training datasets. Finally, to assess the effectiveness of wrapper-based feature ranking, the top ranked features chosen by each ranking technique are evaluated with five classifiers on test datasets. As evaluation metric, we use Area Under the ROC Curve (AUC).

### 3.2. *Feature selection threshold*

This wrapper-based or classifier-aided feature ranking technique that we proposed in Ref. 18, combines the use of a learner in a wrapper method and the individual feature evaluation of a ranking method. For feature selection, different approaches may be adopted. Naturally, the size of the attribute subsets is determined by the feature selection approach chosen. In this study, the threshold for retaining the top ranked features is obtained by $\lceil \log_2 M \rceil$, where $M$ is the number of independent

features. This criterion for determining feature subset size is simple yet provides a non-biased approach with respect to the number of selected features. It is chosen based on the recommendation of a software engineering expert with more than 20 years of experience.

### 3.3. *Training and test datasets*

#### 3.3.1. *LLTS*

The first set of datasets used in this study consists of four software engineering datasets which are originated from a very large legacy telecommunications software system denoted as LLTS. Their characteristics are shown in Table 3. Each dataset represents a separate release, depicting a new version of the overall system. The releases are referred to as SP1, SP2, SP3, and SP4. In the case of SP2-SP4, each release is built upon the previous release. The dataset for each release has the same number of attributes, namely 42 independent attributes and 1 dependent attribute. All the 42 independent attributes are of numeric data type. They represent software metrics, which include product metrics (24), process metrics (14), and execution metrics (4). The dependent attribute determines the class. An instance can be classified as either fault-prone or not fault-prone. The fault-prone is the positive (or minority) class while the not fault-prone is the negative (or majority) class. For each of the releases, a fault-prone module has one or more software faults while a not fault-prone module has no uncovered fault. While each dataset contains the same number of features, each has a different number of instances. The four datasets contain 3649, 3981, 3541, and 3978 instances, respectively. More details on these datasets can be found in Ref. 19.

We use the instances of the first release (SP1) as training dataset for all the LLTS experiments in this study while those of the next three releases (SP2-SP4) as independent test datasets. This ensures a more realistic estimation of the wrapper-based ranking and the classification performance of the learners with subsets of the features.

#### 3.3.2. *Eclipse*

In our experiments we also use publicly available data, namely the Eclipse defect counts and complexity metrics dataset obtained from the PROMISE data repository.[20] In particular, we use the metrics and defects data at the software

Table 3. LLTS datasets characteristics.

| Dataset | #Attributes | #Instances | #Pos | %Pos | #Neg | %Neg |
|---|---|---|---|---|---|---|
| SP1 | 43 | 3649 | 229 | 6.28 | 3420 | 93.72 |
| SP2 | 43 | 3981 | 189 | 4.75 | 3792 | 95.25 |
| SP3 | 43 | 3541 | 47 | 1.33 | 3494 | 98.67 |
| SP4 | 43 | 3978 | 92 | 2.31 | 3886 | 97.69 |

packages level. The original data for Eclipse packages consists of three releases denoted 2.0, 2.1, and 3.0 respectively. Each release as reported by Ref. 21 contains the following information: the name of the package for which the metrics are collected (*name*), the number of defects reported six months prior to release (*pre-release defects*), the number of defects reported six months after release (*post-release defects*), a set of complexity metrics computed for classes or methods and aggregated in terms of average, maximum, and total (*complexity metrics*), the abstract syntax tree of the package consisting of the node size, type, and frequency (*structure of abstract syntax tree*(*s*)).

For our study we transform the original data by: (1) removing all non-numeric attributes, including the package names, and (2) converting the post-release defects attribute to a binary class attribute with defective (def) being the minority class and non-defective (nodef), the majority class. Membership in each class is determined by a post-release defects threshold $t$, which separates defective from non-defective packages by classifying packages with $t$ or more post-release defects as defective and the remaining as non-defective. In our study, we use $t \in \{10, 5, 3\}$ for releases 2.0 and 3.0 while we use $t \in \{5, 4, 2\}$ for release 2.1. A different set of thresholds is chosen for release 2.1 to keep similar class distributions as those of the other two releases. This results into three groups. Each group contains three datasets, one for each release with similar class distribution. All datasets contain 209 attributes (208 independent attributes and 1 dependent attribute). While the attributes are common, the number of instances vary per release. Releases 2.0, 2.1, and 3.0 contain 377, 434 and 661 instances respectively. Table 4 shows the characteristics of the datasets after transformation according to each group.

Table 4. Eclipse datasets characteristics.

| Grouping | Dataset | t | #Attributes | #Instances | #Pos | %Pos | #Neg | %Neg |
|---|---|---|---|---|---|---|---|---|
| Eclipse-1 | Eclipse-metrics-packages-2.0 | 10 | 209 | 377 | 23 | 6.10 | 354 | 93.90 |
| | Eclipse-metrics-packages-2.1 | 5 | 209 | 434 | 34 | 7.83 | 400 | 92.17 |
| | Eclipse-metrics-packages-3.0 | 10 | 209 | 661 | 41 | 6.20 | 620 | 93.80 |
| Eclipse-2 | Eclipse-metrics-packages-2.0 | 5 | 209 | 377 | 52 | 13.79 | 325 | 86.21 |
| | Eclipse-metrics-packages-2.1 | 4 | 209 | 434 | 50 | 11.52 | 384 | 88.48 |
| | Eclipse-metrics-packages-3.0 | 5 | 209 | 661 | 98 | 14.83 | 563 | 85.17 |
| Eclipse-3 | Eclipse-metrics-packages-2.0 | 3 | 209 | 377 | 101 | 26.79 | 276 | 73.21 |
| | Eclipse-metrics-packages-2.1 | 2 | 209 | 434 | 125 | 28.80 | 309 | 71.20 |
| | Eclipse-metrics-packages-3.0 | 3 | 209 | 661 | 157 | 23.75 | 504 | 76.25 |

Thus, for the Eclipse experiments, there are three training and six test datasets. Release 2.0 of each group is used as training dataset while the next two releases (2.1 and 3.0) as test datasets. Regarding the class distribution of the transformed versions of the original datasets, we hold no particular assumption as to how it may affect learning. This is rather intended to provide a true reflection of imbalanced nature often present in real world data.

### 3.4. *Classifiers/ranker aids*

In this study, we use five different machine learning algorithms or methods. These algorithms, commonly used in data mining, serve as both aids to the ranking process and inductive learners for the classification performance process. When they are used in the ranking process, they are referred to as *ranker aids*, and when they are used in classification performance process, they are regarded as *classifiers*. All five algorithms are readily available in the WEKA data mining tool,[22] which is used in this study.

(1) Naïve Bayes (NB): Is a classifier based on Bayes' rule of conditional probability. It assumes conditional independence among the predictor features (independent attributes). While this assumption is highly unlikely in real world data, research has shown that NB often performs well on datasets with highly correlated attributes.[23]
(2) Multilayer Perceptron (MLP): Is a type of neural network that uses backpropagation to classify instances. It contains three layers: an input layer, a hidden layer, and an output layer.[22]
(3) k-Nearest Neighbor (kNN): Is a typical case-based learning algorithm, which uses k nearest neighbors from a library of all the instances of the training dataset and classifies each new instance into the majority class of the k closest neighbors. In our experiments, k is set to 5, so any reference to kNN is equivalent to 5NN.
(4) Support Vector Machines (SVM): Builds a linear discriminant function using a small number of critical boundary samples from each class while ensuring a maximum possible separation.[24]
(5) Logistic Regression (LR): Is a statistical regression model for categorical prediction; it predicts the probability of occurrence of an event by fitting data to a logistic curve.

### 3.5. *Ranking performance measures*

In this study of wrapper-based feature ranking of software engineering datasets, we adopt nine performance measures to compute features' relevance scores. These ranking performance measures are defined as follows:

(1) Overall Accuracy (OA): Is a single-value measure, ranging from 0 to 1, that is obtained by: $OA = \frac{TP+TN}{N}$, where $TP$ and $TN$ denote true positives (number

of positive cases correctly classified as belonging to the positive class) and true negatives (number of negative cases correctly classified as belonging to the negative class) respectively, and $N$ is the total number of instances in the dataset.

(2) F-Measure (FM): Is a single measure that combines both precision and recall. In other words, it is the harmonic mean of precision and recall, and it is calculated as follows[22]: $FM = \frac{2RP}{R+P}$, where $R$ and $P$ are Recall ($\frac{TP}{TP+FN}$) and Precision ($\frac{TP}{TP+FP}$), respectively. $FP$ and $FN$ denote false positives (number of negative cases misclassified as belonging to the positive class) and false negatives (number of positive cases misclassified as belonging to the negative class) respectively. $FM$ uses a decision threshold of 0.5.

(3) Geometric Mean (GM): Is a single-value performance measure obtained by calculating the square root of the product of the true positive rate (percentage of positive cases correctly classified as belonging to the positive class) and the true negative rate (percentage of negative cases correctly classified as belonging to the negative class). $GM = \sqrt{TPR \times TNR}$, where $TPR = \frac{TP}{TP+FN}$ and $TNR = \frac{TN}{TN+FP}$. $GM$ uses a decision threshold of 0.5.

(4) Arithmetic Mean (AM): Is a single-value measure that is obtained by calculating the arithmetic mean of the true positive rate and the true negative rate: $AM = \frac{(TPR+TNR)}{2}$. $AM$ uses a decision threshold of 0.5.

(5) Area Under ROC (AUC): The area under the receiver operating characteristic curve is a single-value measure based on statistical decision theory and was developed for the analysis of electronic signal detection. It is the result of plotting FPR ($\frac{FP}{FP+TN}$) against TPR.[25]

(6) Area Under PRC (PRC): The area under the precision-recall characteristic curve is a single-value measure depicting the tradeoff between precision and recall. It is the result of plotting TPR against precision. Its value ranges from 0 to 1, with 1 denoting a perfect classifier.

(7) Best F-Measure (BFM): Is obtained by getting the largest value of FM when varying the decision threshold value between 0 and 1.

(8) Best Geometric Mean (BGM): Is a single value obtained by getting the maximum geometric mean value when varying the decision threshold between 0 and 1.

(9) Best Arithmetic Mean (BAM): Is obtained by getting the largest value of AM when varying the decision threshold value between 0 and 1.

### 3.6. *Classification performance measure*

The effectiveness of each ranking and the associated feature selection is assessed by evaluating the classification performance of the models subsequently trained and tested with that particular feature selection. In all our experiments we use AUC as the classification performance measure. Our choice of AUC is based on one of

its characteristics, namely its invariance to *a priori* class probability distributions. Moreover, it has been proven to be statistically consistent.[26] It is an excellent measure for determining a classifier's ability to separate the minority class from the majority class. It does not emphasize one class over the other as may be the case in some other performance measures, so it is not biased against the positive class. Given the imbalanced nature of our datasets, AUC is a very appropriate measure for comparing the classification performance of the learners in this study.

### 3.7. *Sampling techniques*

Sampling is a common and simple practice for handling datasets with imbalanced class distribution. With sampling the dataset is artificially altered to rebalance the class distribution. In all the experiments for this study, we assess whether sampling can improve wrapper-based ranking such that models built with feature selection can be systematically improved in terms of their predictive power. Besides non sampling, we consider seven sampling techniques: two versions of random oversampling (ROS35 and ROS50), two versions of random undersampling (RUS35 and RUS50), two versions of Synthetic Minority Oversampling Technique (SMOTE) denoted as S35 and S50, and the weighted version of Wilsons Editing (WW).

Random oversampling (ROS) increases the number of minority instances in a dataset by randomly duplicating minority ones until a desired class distribution is achieved.[27] After application of the random oversampling techniques, the number of positive instances is increased such that 35% and 50% of instances of the resulted dataset are now members of the positive class for ROS35 and ROS50 respectively.

SMOTE is a type of oversampling technique in which positive instances are added not by random but via extrapolation among original positive instances; that is, new instances are produced based on the k-nearest neighbors.[28] After application of SMOTE, the resulted datasets (S35 and S50) contain 35% and 50% of the instances of the positive class respectively.

Random undersampling (RUS) decreases the number of majority instances in a dataset by randomly deleting majority instances until a desired class distribution is achieve.[27] After application of the random undersampling techniques, the number of negative instances is reduced such that 35% and 50% of the instances in the resulted dataset belong to the positive class for RUS35 and RUS50 respectively.

In WW, each instance is classified using kNN with the remaining instances, and misclassified examples of the negative class are removed. Note that the distance measure in kNN is modified to take in consideration the class distribution (i.e., a weighted distance measure is obtained by multiplying the distance measure with a class distribution weighting factor).

We only consider the application of the sampling techniques prior to performing wrapper-based feature ranking. That is, if sampling is used as a preprocessing technique, it must precede the ranking process. Also, after feature selection, we

have two training data sources from which we can build classification models. We take advantage of both sources, non-sampled and sampled training datasets and evaluate their effect on the classification performance.

## 4. Experimental Results: LLTS

For the LLTS experiments, the models are built using the first software release, SP1. Next, the models are evaluated using the subsequent releases, SP2–SP4. Before any comparison, we set the classification performance benchmarks by obtaining the classification performance results of all five learners based on all attributes in terms of AUC. Table 5 shows the average AUC values of the models evaluated with SP2–SP4, and they are depicted according to the classifier and sampling technique (including No Sampling denoted by NS). For example, the top left value (0.8057) indicates the average AUC performance value over all three testing datasets when the classifier is NB and sampling is none. Likewise, the far right value at the bottom (0.8289) represents the average AUC performance value over all three testing datasets when the training data is sampled with WW prior to applying feature selection and building the model using LR. We can observe that the performances of SVM and LR improve with data sampling.

### 4.1. *LLTS: No sampling and feature selection*

We seek to understand the impact of feature selection using wrapper-based feature ranking techniques on the models' predictive powers. We determine the performance classification as the choice of ranker aid and classifier varies. With no sampling of the first release, SP1, we apply the wrapper-based ranking techniques and build corresponding models with feature selection. For feature selection, the subsets are defined by the top six ranked features obtained from the wrapper-based feature ranking of the original feature space. Note that the size of the feature subset is obtained from $\lceil \log_2 42 \rceil$ where 42 represents the number of independent features in the original LLTS datasets. These models are evaluated on the next three datasets SP2-SP4. When the ranker aid and classifier are common, the average performance is obtained over all three classification performance results. When they are different, the average of twelve classification performance results is obtained. The results for both cases are analyzed on all classifiers, but only the results on NB are shown

Table 5. LLTS — Average classification performance with all attributes in terms of AUC for all sampling.

| Learner | NS | RUS35 | RUS50 | ROS35 | ROS50 | S35 | S50 | WW |
|---|---|---|---|---|---|---|---|---|
| NB | 0.8057 | 0.7848 | 0.7965 | 0.8057 | 0.8057 | 0.8036 | 0.8037 | 0.8061 |
| MLP | 0.8315 | 0.8279 | 0.8243 | 0.8281 | 0.8248 | 0.7953 | 0.8294 | 0.7924 |
| kNN | 0.7886 | 0.7876 | 0.7849 | 0.78 | 0.7801 | 0.7608 | 0.7495 | 0.792 |
| SVM | 0.6653 | 0.8319 | 0.8318 | 0.8286 | 0.8302 | 0.8267 | 0.8221 | 0.8105 |
| LR | 0.8174 | 0.8222 | 0.822 | 0.8308 | 0.8322 | 0.8213 | 0.8244 | 0.8289 |

| | | Common Learner for Ranker Aid and Classifier | | | Different Learner for Ranker Aid and Classifier | | |
|---|---|---|---|---|---|---|---|
| Classifier | PM | CV | CV-R | COMBO | CV | CV-R | COMBO |
| NB | OA | 0.6772 | 0.7671 | 0.7198 | 0.7774 | 0.7633 | 0.7409 |
| | FM | 0.8254 | 0.8096 | 0.8048 | 0.7821 | 0.7670 | 0.7824 |
| | GM | 0.7833 | 0.8096 | 0.8239 | 0.7821 | 0.7670 | 0.7824 |
| | AM | 0.7833 | 0.8096 | 0.8048 | 0.7811 | 0.7670 | 0.7754 |
| | AUC | 0.8169 | 0.8181 | 0.8171 | 0.8156 | 0.7953 | 0.8050 |
| | PRC | 0.8256 | 0.8167 | 0.8267 | 0.8188 | 0.7651 | 0.8187 |
| | BFM | 0.8246 | 0.8324 | 0.8256 | 0.8176 | 0.7721 | 0.7983 |
| | BGM | 0.8046 | 0.8104 | 0.8165 | 0.8121 | 0.8045 | 0.7973 |
| | BAM | 0.8153 | 0.8231 | 0.8242 | 0.8142 | 0.8036 | 0.8023 |

Fig. 1. AUC for classification with feature selection and no sampling.

in Fig. 1. From all the results, we can make the following observations:

- With NB as the ranker aid and classifier, the classification performance in terms of AUC is independent of the methodologies (CV, CV-R, or COMBO) and is generally enhanced with feature selection for most of the ranking techniques while the opposite is true with either MLP or kNN.
- When the ranker aid and classifier are both either SVM or LR, the impact of feature selection is associated with the methodology used in the ranking process. With CV, the classification performance with feature selection, for the most part, is maintained if not enhanced while the reverse is true with CV-R.
- The use of kNN as both the ranker aid and the classifier is rather ineffectual, given the poor performance of all classification with feature selection compared to classification with all features, independent of the ranking technique used to determine the top ranked features.
- With different algorithms for ranker aid and classifier, models constructed with either MLP or kNN and feature selection are more likely to show a decline in their average classification performance.

### 4.2. *LLTS: Sampling effect*

The purpose of the second set of experiments on LLTS is to investigate the effect of sampling on wrapper-based ranking and subsequently on the classification with feature selection from the produced ranking. For this new set of experiments, we consider seven sampling techniques as described in Sec. 3.7.

Prior to feature selection, we preprocess the first software release (SP1) dataset by applying the seven sampling techniques, giving new structures that we denote *SP1-ST*, where *ST* represents a sampling technique. Next, we apply the wrapper-based feature ranking techniques on *SP1-ST* datasets. Using the three

methodologies (CV, CV-R, and COMBO) as before, we obtain associated feature rankings, from which we select the top six features (according to each set of rankings). That is, 135 rankings for each sampled dataset are generated, and the top six ranked software metrics of each ranking are selected to construct a pair of corresponding models. We proceed by building models in two manners. One model is trained from the non-sampled dataset while the other is built using the sampled dataset.

As an illustration, when we apply ROS35 to SP1, the resulted dataset, which is denoted as *SP1-ROS35*, is run through the wrapper-based feature ranking techniques. That leads to 135 rankings, one for each wrapper-based feature ranking technique. These rankings are denoted as $SP1\text{-}ROS35_i$, where $i = 1, 2, 3, \ldots, 135$. For each ranking, there is a sampled structure with the top six ranked features denoted by $SP1\text{-}ROS35_i^S$ as well as a non-sampled structure with the same top six ranked features denoted by $SP1\text{-}ROS35_i^{NS}$. All 270 models are evaluated with SP2, SP3, and SP4. This way, we are able to look at the effect of these various sampling techniques and determine whether there is any difference in the models' performance when the training dataset is based upon the sampled data on one hand and the original data on the other hand.

The experiments results show the classification performance of all the models in terms of AUC, with sampling prior to feature selection and same algorithm for ranker aid and classifier as well as when the ranker aid and classifier are different. They also depict the results when the training datasets are from the original as well as when they are from the sampled data structure. Furthermore, they show little or no difference between building the models with the sampled training datasets and those with the non-sampled training datasets. Due to space limitations, we only show Fig. 2, depicting the results of the experiments with WW and NB. However, the following observations are based on all the results:

- When the selected features are obtained from the use of the wrapper-based feature ranking techniques after either ROS35 or ROS50 has been applied to

| | | WW | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Common Learner for Ranker Aid and Classifier | | | | | | Different Learner for Ranker Aid and Classifier | | | | | |
| | | Non-Sampled | | | Sampled | | | Non-Sampled | | | Sampled | | |
| Classifier | PM | CV | CV-R | COMBO | CV | CV-R | COMBO | CV | CV-R | COMBO | CV | CV-R | COMBO |
| NB | OA | 0.6772 | 0.7732 | 0.6567 | 0.6774 | 0.7745 | 0.6565 | 0.7752 | 0.7772 | 0.7959 | 0.7755 | 0.7774 | 0.7959 |
| | FM | 0.8290 | 0.8029 | 0.8288 | 0.8285 | 0.8027 | 0.8282 | 0.7830 | 0.8026 | 0.7840 | 0.7832 | 0.8027 | 0.7847 |
| | GM | 0.7840 | 0.8029 | 0.8288 | 0.7840 | 0.8027 | 0.8282 | 0.7830 | 0.8026 | 0.7858 | 0.7832 | 0.8027 | 0.7865 |
| | AM | 0.8290 | 0.8029 | 0.8288 | 0.8285 | 0.8027 | 0.8282 | 0.7830 | 0.8026 | 0.7841 | 0.7832 | 0.8027 | 0.7847 |
| | AUC | 0.8167 | 0.8113 | 0.8169 | 0.8161 | 0.8119 | 0.8163 | 0.8072 | 0.8150 | 0.8108 | 0.8071 | 0.8154 | 0.8113 |
| | PRC | 0.8256 | 0.8204 | 0.8253 | 0.8249 | 0.8210 | 0.8247 | 0.8124 | 0.7689 | 0.8114 | 0.8118 | 0.7696 | 0.8114 |
| | BFM | 0.8251 | 0.8282 | 0.8185 | 0.8248 | 0.8287 | 0.8176 | 0.8127 | 0.7865 | 0.8094 | 0.8122 | 0.7877 | 0.8092 |
| | BGM | 0.8046 | 0.7985 | 0.8149 | 0.8044 | 0.8001 | 0.8150 | 0.8071 | 0.8075 | 0.7977 | 0.8075 | 0.8077 | 0.7980 |
| | BAM | 0.8046 | 0.8136 | 0.8223 | 0.8044 | 0.8148 | 0.8219 | 0.8195 | 0.8065 | 0.8042 | 0.8194 | 0.8067 | 0.8046 |

Fig. 2. Average AUC with feature selection and WW sampling prior to ranking.

the data, and both classifier and ranker aid are either NB or MLP, there is an improvement in the classification performance with feature selection. In contrast, feature selection does not benefit from oversampling when kNN is the common learner. Also, feature selection benefits tremendously from oversampling when both the ranker aid and classifier are either SVM or LR.

- With ROS sampling prior to feature selection and different algorithms for ranker aid and classifier, NB is the only learner for which feature selection benefits from both non-sampled and sampled training datasets.
- SMOTE has a negative effect on the feature selection classification performance with kNN being the common learner and a positive effect when either MLP or LR is the common learner. Also, with SMOTE the difference between common and different learner case is negligible.
- Overall, with NB as the classifier, wrapper-based feature ranking benefits from RUS whether the ranker aid is same or different and whether we use the sampled or non-sampled training dataset.
- With WW and common algorithm case, the classification performance with feature selection on both learners NB and MLP is enhanced over the benchmark with all features. Once again, the results show inadequate performance when feature selection is combined with kNN. With WW and different algoritm case, there is no significant improvement with feature selection, except when the classifier is MLP.

## 4.3. *LLTS: ANalysis Of VAriance (ANOVA)*

### 4.3.1. *ANOVA test and multiple comparisons for LLTS non-sampled datasets*

A four-way ANOVA test is performed for the prediction results obtained from LLTS non-sampled datasets. The four main effects include Factor RA (Ranker Aid), in which five learners are considered, Factor PM (Performance Metric), in which nine different performance metrics are included, Factor TM (Training Method), which include three approaches, and Factor LE (Learner), in which five classifiers are considered. Although the five classifiers in the Learner factor are the same as those in the RA factor, they play different roles during the data mining process. The interaction effects of two factors are also considered in the ANOVA test.

An n-way ANOVA test can be used to determine whether the means in a set of data differ when grouped by multiple factors. If they do differ, one can determine which factors or combinations of factors are associated with the difference. The ANOVA model can be used to test the hypothesis that the AUC for the main factors RA, PM, TM and LE and/or for their paired interactions are equal against the alternative hypothesis that at least one mean is different. If the alternative hypothesis is accepted, multiple comparisons can be used to determine which of the means are significantly different from the others. In this study, we perform the multiple-comparison tests using Tukey's Honestly Significant Difference criterion. All tests utilize a significance level $\alpha = 0.05$.

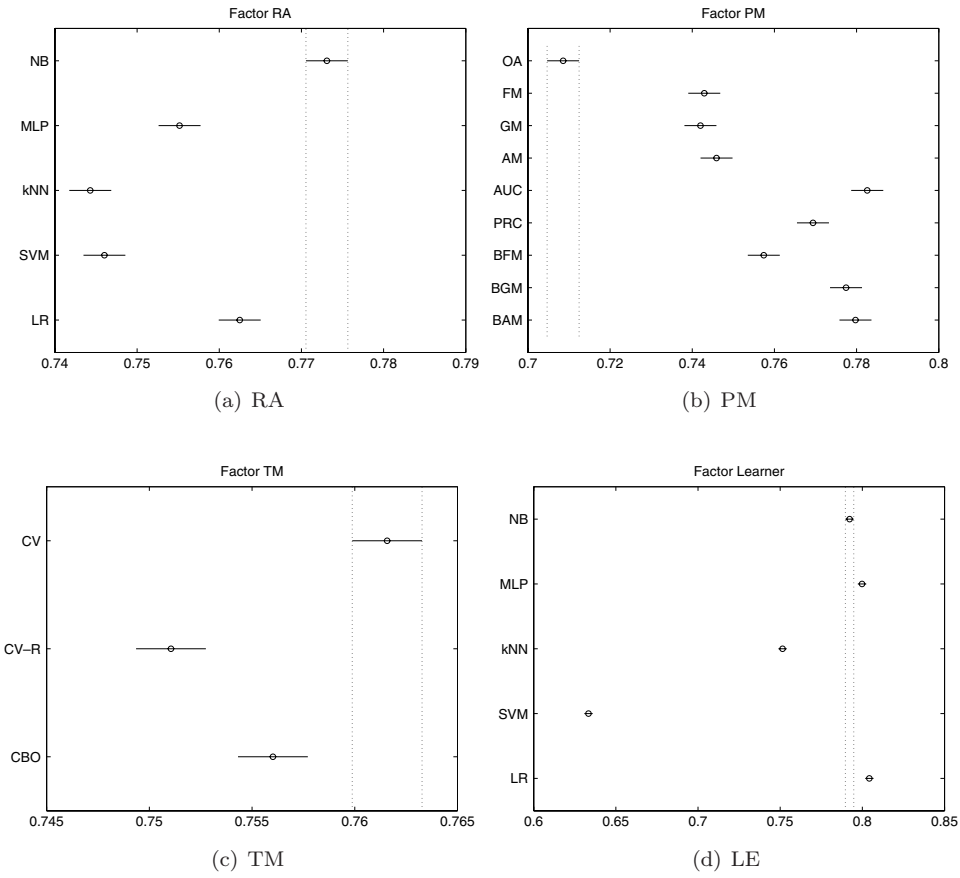
Table 6. ANOVA for LLTS non-sampled datasets.

| Source | Sum sq. | d.f. | Mean sq. | F | *p*-value |
|---|---|---|---|---|---|
| PM | 1.0414 | 8 | 0.1302 | 43.02 | 0.000 |
| RA | 0.2314 | 4 | 0.0578 | 19.12 | 0.000 |
| TM | 0.0374 | 2 | 0.0187 | 6.18 | 0.002 |
| LE | 8.3495 | 4 | 2.0874 | 689.86 | 0.000 |
| PM×RA | 0.5752 | 32 | 0.0180 | 5.94 | 0.000 |
| PM×TM | 0.1422 | 16 | 0.0089 | 2.94 | 0.000 |
| PM×LE | 0.3392 | 32 | 0.0106 | 3.50 | 0.000 |
| RA×TM | 0.2263 | 8 | 0.0283 | 9.35 | 0.000 |
| RA×LE | 0.7106 | 16 | 0.0444 | 14.68 | 0.000 |
| TM×LE | 0.0535 | 8 | 0.0067 | 2.21 | 0.024 |
| Error | 5.7309 | 1894 | 0.0030 | | |
| Total | 17.4375 | 2024 | | | |

The four-way ANOVA results for LLTS are presented in Table 6. From this table, we can see that the *p*-values (last column of the table) for the main factors RA, PM, TM and LE and their paired interaction terms are less than a typical cutoff 0.05. This indicates that the classification performances in terms of AUC, are not the same for all groups in each of these factors or terms. In other words, the classification performances are significantly different from each other for at least a pair of groups in the corresponding factors or terms.

We further conduct multiple comparisons for the main factors and some of their interactions that are the main concerns for this paper. The objective of multiple comparisons is to identify the means that significantly differ from others. Our analysis is on both the main factors and some relevant interactions; however, only the test results for the main factors are shown in Fig. 3. Each group mean is represented in each graph by a symbol (∘) and 95% confidence interval around the symbol. Two means are significantly different if their intervals are disjoint, and are not significantly different if their intervals overlap. The results show the following points.

- For Factor RA (Fig. 3(a)), NB as ranker aid presents superior performance while kNN demonstrates inferior performance.
- For Factor PM (Fig. 3(b)), OA demonstrates poorest performance, while AUC, PRC, BFM, BGM and BAM demonstrate superior performance to FM, GM, and AM with the decision threshold 0.5.
- For Factor TM (Fig. 3(c)), the CV and COMBO methods presented significantly better performance than the CV-R approach. In addition, CV is preferred over COMBO, if computational complexity is a factor.
- For Factor LE (Fig. 3(d)), SVM demonstrates worst performance among the five learners while NB, MLP, and LR perform best and kNN shows moderate performance.
- For Interaction PM×RA, 45 groups, obtained when the nine performance metrics are used in conjunction with five ranker aids, are presented. The performance of




(a) RA (b) PM

(c) TM (d) LE

Fig. 3. Multiple comparisons for LLTS non-sampled data — main factors.

each ranker is determined by the two factors (RA and PM) and/or their interaction term. Given the $p$-value in the ANOVA test for this pair-wise interaction term is less than 0.05, changing the value of one factor will significantly impact the mean values of the other factor, or vice versa. Generally, when the rankers are created with the AUC, PRC, BFM, BGM and BAM performance metrics aided by the NB learner, they present better performance than the rankers formed by different performance metrics and/or different learners.

- For Interaction RA×LE, 25 groups (five ranker aids by five classifiers) are formed. The interaction effect significantly influences the classification results. Another interesting question is — "when ranker aid and classifier shares the same learning algorithm, is there any improvement for the performance compared to the case where different learning algorithms are used in the two processes?" The observation demonstrates that on average no significant difference is found between the performances from the two cases.

4.3.2. *ANOVA test and multiple comparisons for LLTS sampled datasets*

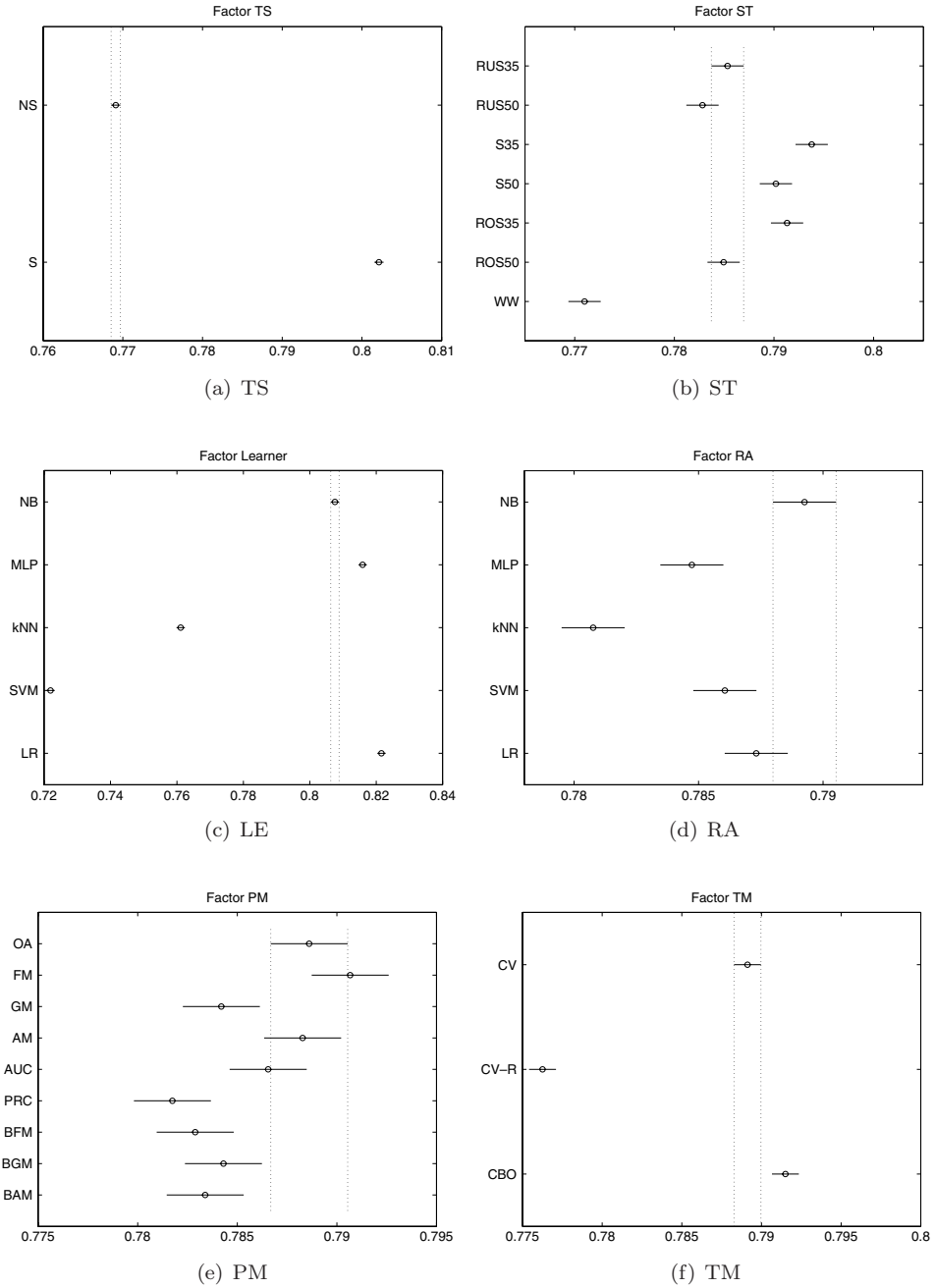
We perform a six-way ANOVA test for the performance results obtained from LLTS sampled datasets. The six main effects include the same four factors as presented in the non-sampled case. Besides, two more factors are also taken into account. One is Factor TS (Training data Source), in which two different scenarios are considered, and the other one is Factor ST (Sampling Technique), in which seven data sampling methods are included. The pairwise interaction effects are also considered in the ANOVA test. The ANOVA results are presented in Table 7 for LLTS datasets. From the table, we can see that the $p$-values for all main factors TS, ST, RA, PM, TM and LE, and all pairwise interactions are less than the cutoff 0.05.

We further conduct multiple comparisons for all main factors and some of their interactions that are the main concerns for this paper. We add four more multiple comparisons, two for main factors: TS and ST, and two for interactions: PM×TS and ST×TS, compared to the multiple comparisons worked on the non-sampled case. The results and findings are summarized as follows.

- For Factor TS (Fig. 4(a)), the classification models built based on the training dataset that is extracted from the sampled version (denoted S in the figure) of the original dataset demonstrate significantly better performance than those built based on the training dataset that is extracted from the non-sampled version (denoted NS in the figure).

Table 7. ANOVA for sampled LLTS datasets.

| Source | Sum sq. | d.f. | Mean sq. | F | $p$-value |
|---|---|---|---|---|---|
| TS | 7.7294 | 1 | 7.7294 | 3158.39 | 0 |
| ST | 1.3869 | 6 | 0.2312 | 94.45 | 0 |
| PM | 0.2318 | 8 | 0.0290 | 11.84 | 0 |
| RA | 0.2306 | 4 | 0.0576 | 23.55 | 0 |
| TM | 1.2708 | 2 | 0.6354 | 259.64 | 0 |
| LE | 41.6490 | 4 | 10.4122 | 4254.63 | 0 |
| TS×ST | 0.6503 | 6 | 0.1084 | 44.29 | 0 |
| TS×PM | 0.0390 | 8 | 0.0049 | 1.99 | 0.043 |
| TS×RA | 0.0860 | 4 | 0.0215 | 8.78 | 0 |
| TS×TM | 0.1963 | 2 | 0.0981 | 40.10 | 0 |
| TS×LE | 32.1185 | 4 | 8.0296 | 3281.06 | 0 |
| ST×PM | 0.8523 | 48 | 0.0178 | 7.26 | 0 |
| ST×RA | 0.9829 | 24 | 0.0410 | 16.73 | 0 |
| ST×TM | 0.1378 | 12 | 0.0115 | 4.69 | 0 |
| ST×LE | 1.9407 | 24 | 0.0809 | 33.04 | 0 |
| PM×RA | 0.8588 | 32 | 0.0268 | 10.97 | 0 |
| PM×TM | 0.1968 | 16 | 0.0123 | 5.03 | 0 |
| PM×LE | 0.1953 | 32 | 0.0061 | 2.49 | 0 |
| RA×TM | 0.1760 | 8 | 0.0220 | 8.99 | 0 |
| RA×LE | 0.1899 | 16 | 0.0119 | 4.85 | 0 |
| TM×LE | 0.5703 | 8 | 0.0713 | 29.13 | 0 |
| Error | 68.7194 | 28080 | 0.0024 | | |
| Total | 160.4086 | 28349 | | | |




(a) TS (b) ST

(c) LE (d) RA

(e) PM (f) TM

Fig. 4. Multiple comparisons for sampled data — main factors.

- For Factor ST (Fig. 4(b)), the SMOTE data sampling technique demonstrated superior performance to the other sampling techniques in most cases, and Wilson's method presents poorest performance among the seven techniques. Also, for a given sampling method, the ratio of 65-to-35 between the negative and positive instances in the post sampling dataset demonstrates better or similar performance to the case in which 50-to-50 is used.
- For Factor RA (Fig. 4(d)), the NB learner still presents superior performance, and kNN shows the worst performance. This is similar to the non-sampled case. However, when comparing the Factor RA in the sampled and non-sampled cases, one can find the classification results are different in terms of AUC. For the sampled case, the values of AUC fall into the range of 0.78 to 0.79 compared to the AUC range of 0.74 to 0.78 for the non-sampled case. This demonstrates that the sampled data influences the ranker aid's performance.
- For Factor PM (Fig. 4(e)), OA does not demonstrate poorest performance like in the non-sampled case. Also, AUC, PRC, BFM, BGM and BAM do not perform superiorly to FM, GM, and AM. This can be explained by the fact that traditional classification algorithms always use the performance metrics such as FM, GM, and AM with the decision threshold 0.5. This threshold may be appropriate when distributions of positive and negative examples in a given dataset are equally balanced; however, it results in a large number of mis-classifications from positive to negative class when a training dataset is imbalanced. Because sampling balances the class distribution, the performance of classification models built with the sampled training dataset will not deteriorate when using performance metrics with the decision threshold 0.5 such as OA, FM, GM, and AM during the modeling process.
- For Factor TM (Fig. 4(f)), we reach a similar conclusion as we did for the non-sampled case. That is, the CV and COMBO methods perform significantly better than the CV-R method. For the sampled case, though, the COMBO method is better than the CV one.
- For Factor LE (Fig. 4(c)), we make the same observations as we did for the non-sampled case. NB, MLP and LR demonstrate superior performance among the five learners. SVM performs worst, and kNN is moderate.
- For Interaction TS×ST, 14 groups are formed by seven data sampling techniques, each using two different training data sources, sampled and non-sampled versions of the original dataset. The results demonstrate that the sampled version perform significantly better than the non-sampled version for all the sampling techniques over the LLTS datasets.
- For Interaction TS×PM, which consists of 18 groups formed by nine performance metrics, each involved in two different training data sources, sampled and non-sampled versions of the original dataset. The results demonstrate that performance of each group is determined by the two main factors (PM and TS) making up the groups. In addition, for the two training data sources, the sampled version

performed significantly better than the non-sampled version for all performance metrics over the LLTS datasets.

- For Interaction RA×PM, 45 groups (rankers) are obtained. The performance of each ranker is determined by the two factors (RA and PM) and their interaction. Generally, the NB-aided rankers presented better performance than the rankers formed by the performance metrics aided by different learners.
- For Interaction RA×LE, 25 groups are formed. The interaction effect still significantly influences the classification results. The same question as raised in the non-sampled case is — "when ranker aid and learner shared the same learning algorithm, is there any improvement for the performance compared to the case where the different learning algorithms are used in the two processes?". This demonstrates that on average no significant difference is found between the performances from the two cases. However, it can be clearly observed that the performance of classification models is more influenced by the selected classifier than by the ranker aid.

## 5. Experimental Results: Eclipse

For Eclipse, the models are constructed and tested within the same minority class distribution group (see Sec. 3.3.2). For instance, we construct a model using release 2.0 from the first distribution group, and we evaluate the models on both releases 2.1 and 3.0 within the same distribution group. We denote the distribution groups as Eclipse-1, Eclipse-2, and Eclipse-3 respectively (see Table 4).

Researchers exploring feature selection techniques usually aim at finding subsets of the attributes that retain if not enhance models' predictability power. As a baseline for comparison, we build different models with all the attributes of the training datasets and evaluate them on the corresponding test datasets. Table 8 depicts the performance results in terms of AUC for all attributes selection; it shows the average AUC values according to the classifier and sampling technique (including No Sampling denoted by NS). These values are obtained for NB, MLP, kNN, SVM, and LR respectively.

### 5.1. *Eclipse: No sampling and feature selection*

All experiments with the Eclipse datasets use three training datasets to build the models, and two test datasets for each model are used for model evaluation. Now, to build the models, we need to select a subset of the features. In the experiments with Eclipse, the size of the feature subset is obtained from $\lceil \log_2 208 \rceil$ where 208 represent the number of independent features in the original Eclipse datasets. Thus, the top eight features are selected from each ranking. Similar to LLTS, the experiments are conducted with generally two perspectives in mind, no-sampling and sampling.

Here we address the no-sampling case, and we aim at characterizing the impact of feature selection using wrapper-based feature ranking techniques on the models' predictive powers. With no sampling of the training datasets, 2.0, we apply the

Table 8. Eclipse — Average classification performance with all attributes in terms of AUC for all sampling.

| Learner | Group | NS | RUS35 | RUS50 | ROS35 | ROS50 | S35 | S50 | WW |
|---|---|---|---|---|---|---|---|---|---|
| NB | Eclipse-1 | 0.8494 | 0.8322 | 0.8127 | 0.8492 | 0.8495 | 0.8470 | 0.8661 | 0.8494 |
| | Eclipse-2 | 0.8450 | 0.8398 | 0.8326 | 0.8434 | 0.8444 | 0.8474 | 0.8427 | 0.8443 |
| | Eclipse-3 | 0.7700 | 0.7675 | 0.7717 | 0.7665 | 0.7681 | 0.7708 | 0.7640 | 0.7732 |
| MLP | Eclipse-1 | 0.8369 | 0.8136 | 0.8117 | 0.8459 | 0.8879 | 0.8745 | 0.8821 | 0.8682 |
| | Eclipse-2 | 0.8450 | 0.8314 | 0.8345 | 0.8329 | 0.8369 | 0.8164 | 0.8272 | 0.8467 |
| | Eclipse-3 | 0.8090 | 0.8047 | 0.7820 | 0.8061 | 0.7926 | 0.8082 | 0.7905 | 0.8026 |
| kNN | Eclipse-1 | 0.8233 | 0.8341 | 0.8578 | 0.8159 | 0.8163 | 0.8388 | 0.8260 | 0.8233 |
| | Eclipse-2 | 0.8165 | 0.8041 | 0.8118 | 0.8153 | 0.8067 | 0.8291 | 0.8239 | 0.8153 |
| | Eclipse-3 | 0.8180 | 0.8148 | 0.8031 | 0.8063 | 0.8113 | 0.8158 | 0.8080 | 0.8141 |
| SVM | Eclipse-1 | 0.8240 | 0.8055 | 0.7711 | 0.7702 | 0.7701 | 0.7692 | 0.7702 | 0.8238 |
| | Eclipse-2 | 0.8203 | 0.8147 | 0.7959 | 0.8080 | 0.8033 | 0.8113 | 0.8092 | 0.8250 |
| | Eclipse-3 | 0.7906 | 0.8197 | 0.7889 | 0.7786 | 0.7991 | 0.7906 | 0.8000 | 0.8152 |
| LR | Eclipse-1 | 0.6943 | 0.7042 | 0.6329 | 0.6076 | 0.6502 | 0.6068 | 0.6068 | 0.6943 |
| | Eclipse-2 | 0.7138 | 0.6526 | 0.5872 | 0.6743 | 0.6564 | 0.6824 | 0.6824 | 0.6645 |
| | Eclipse-3 | 0.6590 | 0.6811 | 0.6873 | 0.6535 | 0.6666 | 0.6623 | 0.6623 | 0.6954 |

wrapper-based ranking techniques and build corresponding models with feature selection. These models are evaluated using the test datasets, releases 2.1 and 3.0 (again within the same class distribution).

Considering the average performance results in terms of AUC, we can observe that the classification performance is generally enhanced with feature selection, and this is so whether we use common learner or different learners for ranker aid and classifier. For the first class distribution group, feature selection from wrapper-based ranking is enhanced about 90% and 84% of the cases for same and different learner respectively. For the second group, about 77% of the time, feature selection is enhanced for both cases. Finally, for the third group, enhancement is shown about 83% and 78% of the time, respectively. These results show the power of our proposed wrapper-based feature ranking. Strongly relevant features are ranked first, allowing a subset with less than 5% of the features to retain the models's predictive power. This implies that a large portion of the feature space is irrelevant while proving the soundness of our method for defining feature relevancy.

### 5.2. *Eclipse: Sampling effect*

Continuing our experiments with the Eclipse datasets, we now investigate whether sampling has any effect on wrapper-based ranking and subsequently on the classification with feature selection from the produced ranking. We choose the same seven sampling techniques as described in Sec. 3.7.

For each class distribution grouping, we apply all the sampling techniques on the training dataset (Package Release 2.0). This preprocessing step results in 21 new data structures. For each of these new data structures, we obtain 135 rankings

by applying the wrapper-based feature ranking techniques (each ranking technique represents a combination of a ranker aid, a Performance Metric, and a Methodology). Thus, we obtain a total of 2835 rankings.

Similar to the non-sampling case, the top eight features are selected from each ranking. Once all the feature subsets are identified, we construct two sets of models. On one hand, we build the models with feature selection from the non-sampled dataset. On the other hand, we use the sampled dataset as our training dataset for building the models with feature selection. In each case, the models are evaluated with the corresponding test datasets (Releases 2.1 and 3.0). The classification performance results in terms of AUC are computed. Before comparing the classification performance of the models — those with feature selection and sampling applied prior to building the models and those with no feature selection and data sampling — we aggregate the evaluation results into two categories. First, we average the AUC values when a common learner is used for both the ranker aid and the classifier. Second, the average is taken on the AUC values when the ranker aid is different from the classifier.

Table 9 shows the percentage of cases where models built with feature selection are same as or better than those without feature selection, in terms of their classification performance (AUC). We report the results with respect to the sampling technique (indicated in the first column), whether the ranker aid and classifier use same or different learner, and whether the sampled or non-sampled data structure is used to train the models. For instance, for ROS35, we observe that when the learner is common for ranker aid and classifier, 80% and 78% of the time, feature selection outperforms no feature selection for sampled and non-sampled training data respectively. For the same sample technique but different learner for ranker aid and classifier, the two values are 83% and 81% respectively. As we can see, the patterns are the same whether a common or a different learner than the ranker aid is used for classification.

When we consider sampling prior to applying the wrapper-based feature ranking techniques and when we use the same learner as both the ranker aid and the

Table 9. Eclipse — Percent of feature selection cases that outperform no feature selection.

| | Common learner Keep sampling | | Different learner Keep sampling | |
|---|---|---|---|---|
| ST | No (%) | Yes (%) | No (%) | Yes (%) |
| ROS35 | 80 | 78 | 83 | 81 |
| ROS50 | 85 | 85 | 83 | 83 |
| S35 | 90 | 90 | 86 | 88 |
| S50 | 84 | 85 | 86 | 86 |
| RUS35 | 86 | 83 | 85 | 81 |
| RUS50 | 79 | 70 | 78 | 71 |
| WW | 83 | 84 | 85 | 86 |

classifier, the experimental results show that generally sampling improves the average classification performances whether the training source is the sampled or non-sampled data. The enhanced classification performance due to sampling is observed whether we use common learner or different learners for ranker aid and classifier.

### 5.3. *Eclipse: ANalysis Of VAriance (ANOVA)*

#### 5.3.1. *ANOVA test and multiple comparisons for eclipse non-sampled datasets*

Similar to LLTS ANOVA test and multiple comparisons for Non-sampled datasets, a four-way ANOVA test is performed for the prediction results obtained from Eclipse non-sampled datasets. We consider the same four main effects and the same interaction effects of two factors. Also for the Eclipse tests, we performed the multiple comparison tests using Tukey's Honestly Significant Difference criterion, and all tests utilize a significance level $\alpha = 0.05$.

The Eclipse ANOVA results are presented in Table 10, where three subtables are included, each representing the four-way ANOVA results for each group of Eclipse datasets. We can observe that the $p$-values (last column of the tables) for the main factors RA, PM, TM and LE and their paired interaction terms are less than a typical cutoff 0.05 in most cases, especially for Eclipse-1 and Eclipse-2. This indicates that the classification performances are significantly different from each other for at least a pair of groups in the corresponding factors or terms. The factors or terms with $p$-values greater than 0.05 (highlighted with **boldface** in the tables) indicate that their group means are the same or at least not significantly different from each other.

As in the LLTS case, only the test results for the multiple comparisons for the main factors (See Figs. 5 to 8) are shown here even though some interactions that are the main concerns for this paper are included in the discussion below. The figures shown include three subfigures each, representing the results from the three groups of Eclipse datasets, respectively. The results show the following points.

- For Factor RA (Fig. 5), no consistent conclusion can be drawn over all three groups of Eclipse datasets. However, the NB and SVM learners generally present superior performance while kNN and LR generally demonstrate inferior performance.
- For Factor PM (Fig. 6), OA consistently demonstrates poorest performance, while AUC, PRC, BFM, BGM and BAM generally demonstrate superior performance to FM, GM, and AM with the decision threshold 0.5.
- For Factor TM (Fig. 7), the CV and COMBO methods present significantly better performance than the CV-R approach. In addition, CV is recommended compared to COMBO, if computational complexity is considered as a factor. This conclusion is true for all three groups of datasets.

Table 10. ANOVA for Eclipse non-sampled datasets.

(a) Eclipse-1

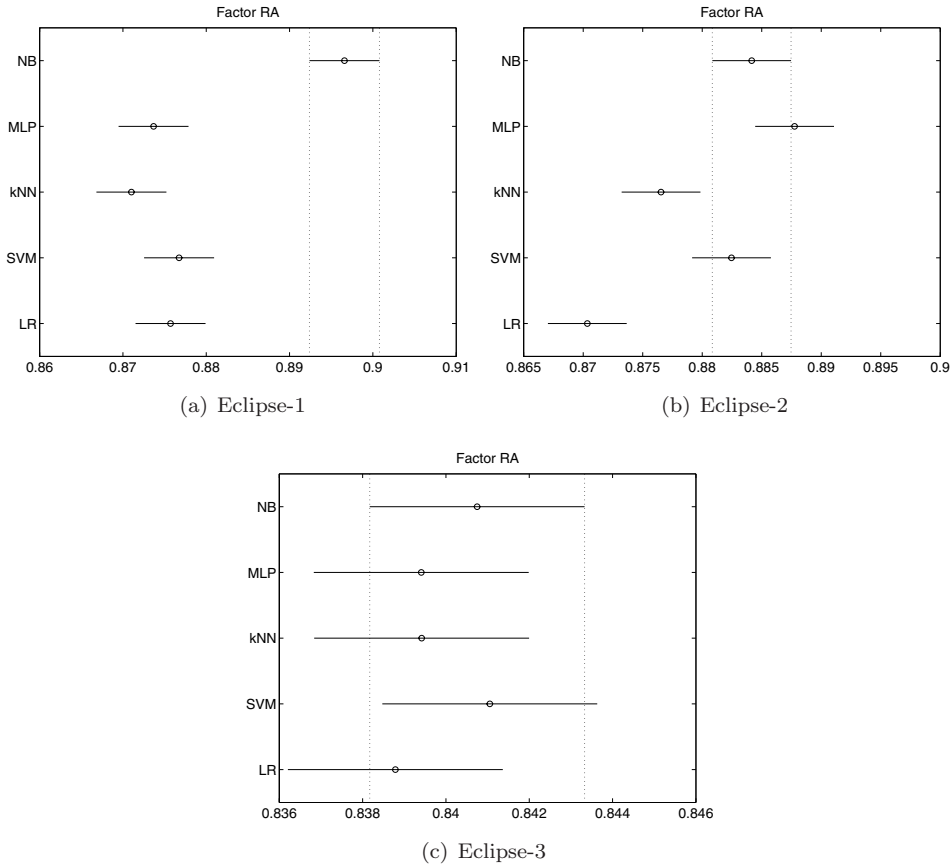
| Source | Sum sq. | d.f. | Mean sq. | F | $p$-value |
|---|---|---|---|---|---|
| PM | 0.1325 | 8 | 0.0166 | 13.82 | 0.000 |
| RA | 0.1126 | 4 | 0.0281 | 23.48 | 0.000 |
| TM | 0.2479 | 2 | 0.1240 | 103.39 | 0.000 |
| LE | 0.6035 | 4 | 0.1509 | 125.86 | 0.000 |
| PM×RA | 0.2098 | 32 | 0.0066 | 5.47 | 0.000 |
| PM×TM | 0.0620 | 16 | 0.0039 | 3.23 | 0.000 |
| PM×LE | 0.0315 | 32 | 0.0010 | 0.82 | **0.748** |
| RA×TM | 0.0507 | 8 | 0.0063 | 5.29 | 0.000 |
| RA×LE | 0.0316 | 16 | 0.0020 | 1.65 | **0.051** |
| TM×LE | 0.0761 | 8 | 0.0095 | 7.93 | 0.000 |
| Error | 1.4614 | 1219 | 0.0012 | | |
| Total | 3.0196 | 1349 | | | |

(b) Eclipse-2

| Source | Sum sq. | d.f. | Mean sq. | F | $p$-value |
|---|---|---|---|---|---|
| PM | 0.0372 | 8 | 0.0046 | 5.78 | 0.000 |
| RA | 0.0509 | 4 | 0.0127 | 15.82 | 0.000 |
| TM | 0.9107 | 2 | 0.4553 | 566.41 | 0.000 |
| LE | 0.9544 | 4 | 0.2386 | 296.80 | 0.000 |
| PM×RA | 0.1028 | 32 | 0.0032 | 4.00 | 0.000 |
| PM×TM | 0.0630 | 16 | 0.0039 | 4.90 | 0.000 |
| PM×LE | 0.0373 | 32 | 0.0012 | 1.45 | **0.051** |
| RA×TM | 0.0929 | 8 | 0.0116 | 14.44 | 0.000 |
| RA×LE | 0.0384 | 16 | 0.0024 | 2.98 | 0.000 |
| TM×LE | 0.1624 | 8 | 0.0203 | 25.25 | 0.000 |
| Error | 0.9800 | 1219 | 0.0008 | | |
| Total | 3.4298 | 1349 | | | |

(c) Eclipse-3

| Source | Sum sq. | d.f. | Mean sq. | F | $p$-value |
|---|---|---|---|---|---|
| PM | 0.0037 | 8 | 0.0005 | 0.97 | **0.455** |
| RA | 0.0010 | 4 | 0.0003 | 0.53 | **0.714** |
| TM | 0.1117 | 2 | 0.0558 | 116.25 | 0.000 |
| LE | 2.3784 | 4 | 0.5946 | 1238.04 | 0.000 |
| PM×RA | 0.0222 | 32 | 0.0007 | 1.45 | **0.052** |
| PM×TM | 0.0139 | 16 | 0.0009 | 1.81 | 0.026 |
| PM×LE | 0.0163 | 32 | 0.0005 | 1.06 | **0.376** |
| RA×TM | 0.0112 | 8 | 0.0014 | 2.91 | 0.003 |
| RA×LE | 0.0430 | 16 | 0.0027 | 5.60 | 0.000 |
| TM×LE | 0.3061 | 8 | 0.0383 | 79.67 | 0.000 |
| Error | 0.5855 | 1219 | 0.0005 | | |
| Total | 3.4930 | 1349 | | | |

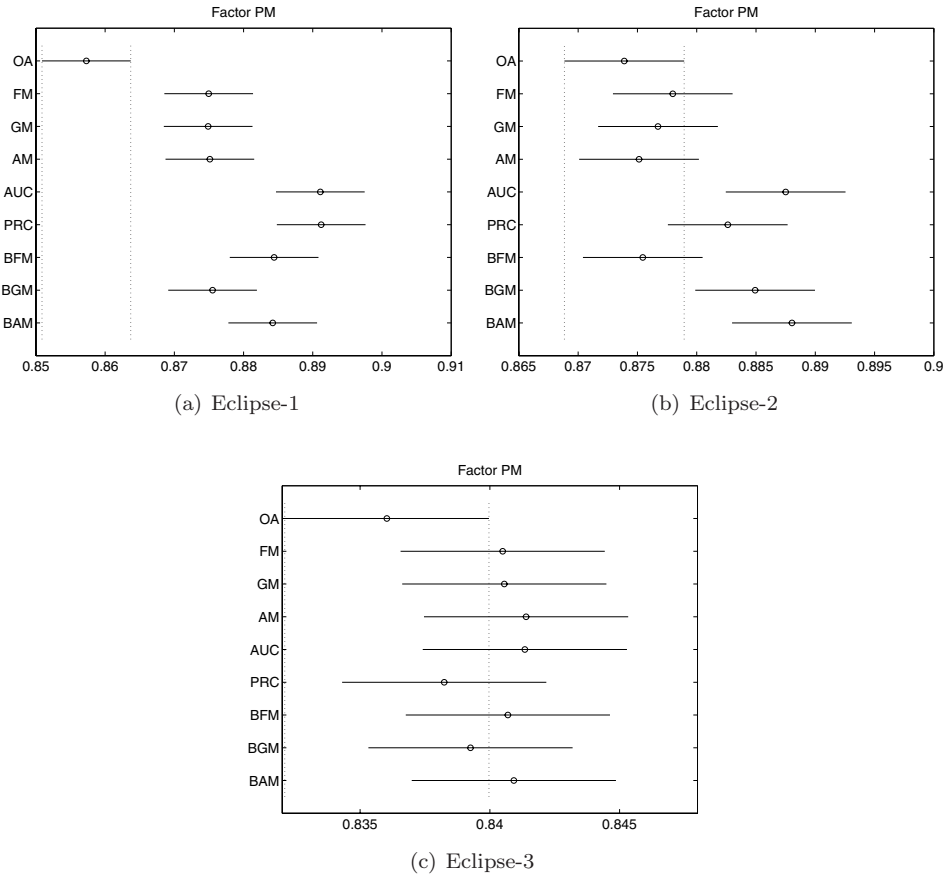



(a) Eclipse-1 (b) Eclipse-2

(c) Eclipse-3

Fig. 5. Multiple comparisons for Eclipse non-sampled data — factor RA.

- For Factor LE (Fig. 8), NB demonstrates worst performance among the five learners, and SVM performed best. Among the three moderate learners, MLP and LR presented better performance than kNN.
- For Interaction PM×RA, which includes 45 groups (rankers), OA on average is significantly worse than other performance metrics for Eclipse-1, but this may not be true for a particular ranker aid. For instance, when using SVM as a ranker aid, at least three performance metrics (FM, GM, and AM) are not as good as the OA performance metric. Generally, when the rankers are created with the AUC, PRC, BFM, BGM and BAM performance metrics aided by the NB, MLP or SVM learner, they show better performance than the rankers formed by different performance metrics and/or different learners.
- For Interaction RA×LE, 25 groups are formed. The $p$-value of the pairwise interaction is greater than 0.05 for Eclipse-1, implying that the performance of each group is determined by the two main effects (RA and LE) but not by their interaction. However, For Eclipse-2 and Eclipse-3, the interaction RA×LE also




(a) Eclipse-1 (b) Eclipse-2

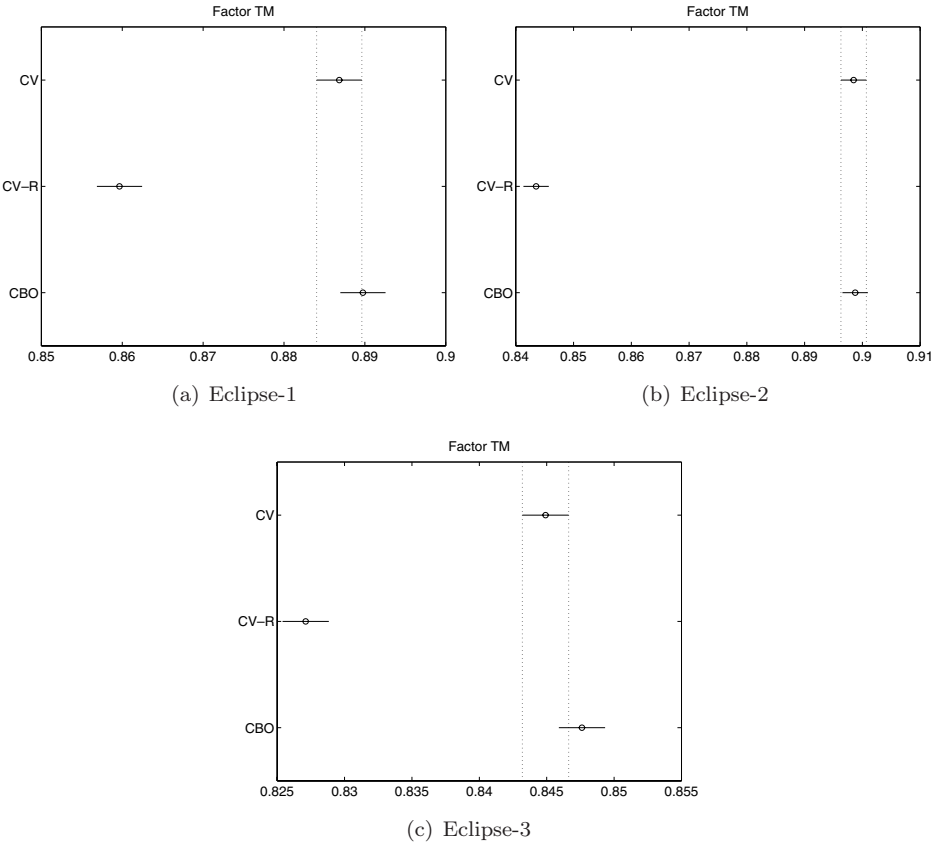
(c) Eclipse-3

Fig. 6. Multiple comparisons for Eclipse non-sampled data — factor PM.

significantly influences the performance of each group. Moreover, there is no strong evidence of performance improvements when ranker aid and classifier share the same learning algorithm compared to the different learning algorithms case. This conclusion is similar to the first case study.

### 5.3.2. *ANOVA test and multiple comparisons for Eclipse sampled datasets*

A six-way ANOVA test is performed for the prediction results obtained from Eclipse sampled datasets. The six main effects include the same four factors as presented in the non-sampled case. Besides, two more factors are also taken into account. One is Factor TS (Training data Source ), in which two different scenarios are considered, and the other one is Factor ST (Sampling Technique), in which seven data sampling methods are included. The pairwise interaction effects are also considered in the ANOVA test.




(a) Eclipse-1 (b) Eclipse-2

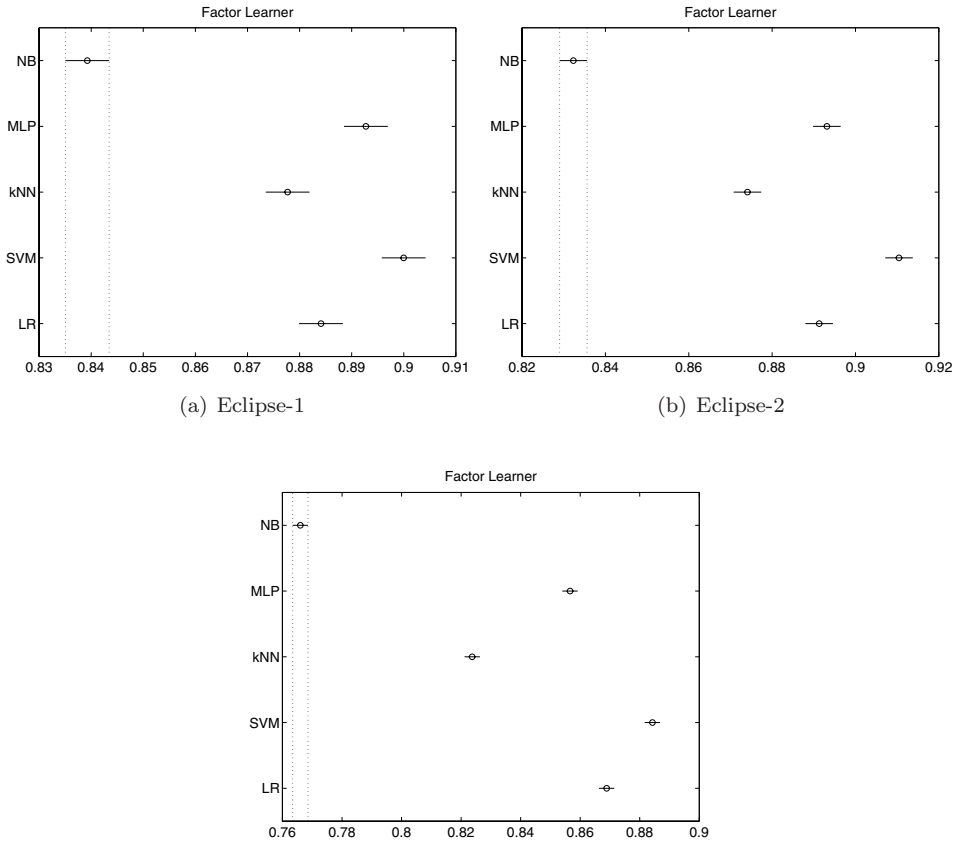
(c) Eclipse-3

Fig. 7. Multiple comparisons for Eclipse Non-sampled Data — Factor TM.

The ANOVA results are presented in Table 11 for Eclipse datasets. From the tables, we can see that the $p$-values for all main factors TS, ST, RA, PM, TM and LE, and most pairwise interactions are less than the cutoff 0.05. The three exceptions ($p > 0.05$) in each group of Eclipse datasets came from interactions TS×PM, TS×RA, and TS×TM (except for TS×TM in Eclipse-1). Thus, the performance of the classification is greatly influenced by all the factors and their pairwise interactions except for TS×PM, TS×RA, and TS×TM.

We further conduct multiple comparisons for all main factors and some of their interactions that are the main concerns for this paper. As we did for the LLTS experiments, we add four more multiple comparisons, two for main factors: TS and ST, and two for interactions: PM×TS and ST×TS, compared to the multiple comparisons worked on the non-sampled case. The results and findings are summarized as follows.

- For Factor TS (Fig. 9), the classification models built based on the training dataset that is extracted from the sampled version of the original dataset




(a) Eclipse-1 (b) Eclipse-2

(c) Eclipse-3

Fig. 8. Multiple comparisons for Eclipse non-sampled data — factor LE.

demonstrate significantly better performance than those built based on the training dataset that is extracted from the non-sampled version. This is true for Eclipse-2 and Eclipse-3. For Eclipse-1, the contrary results are obtained.

- For Factor ST (Fig. 10), the SMOTE data sampling technique demonstrate superior performance to the other sampling techniques in most cases. Also, Wilson's method presents good performance for the Eclipse datasets — this is different from the conclusion obtained from the LLTS experiments.
- For Factor RA, no consistent conclusion can be drawn over all the Eclipse datasets. This is similar to the non-sampled case (Fig. 5). Generally, the NB learner presents superior performance.
- For Factor PM, OA did not demonstrate poorest performance like the non-sampled case. Also, AUC, PRC, BFM, BGM and BAM did not always perform superiorly to FM, GM, and AM with the decision threshold 0.5. This is an observation similar to that for LLTS.

Table 11. ANOVA for sampled Eclipse datasets.

(a) Eclipse-1

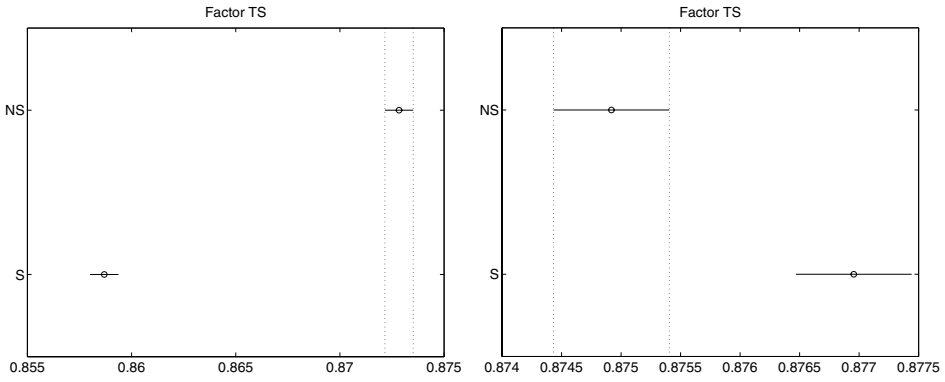
| Source | Sum sq. | d.f. | Mean sq. | F | $p$-value |
|---|---|---|---|---|---|
| TS | 0.9458 | 1 | 0.9458 | 416.78 | 0 |
| ST | 2.2951 | 6 | 0.3825 | 168.57 | 0 |
| PM | 0.2875 | 8 | 0.0359 | 15.84 | 0 |
| RA | 0.4754 | 4 | 0.1188 | 52.37 | 0 |
| TM | 15.4024 | 2 | 7.7012 | 3393.83 | 0 |
| LE | 5.0930 | 4 | 1.2732 | 561.10 | 0 |
| TS×ST | 0.9449 | 6 | 0.1575 | 69.40 | 0 |
| TS×PM | 0.0093 | 8 | 0.0012 | 0.51 | 0.847 |
| TS×RA | 0.0069 | 4 | 0.0017 | 0.76 | 0.553 |
| TS×TM | 0.0249 | 2 | 0.0125 | 5.49 | 0.004 |
| TS×LE | 0.5908 | 4 | 0.1477 | 65.09 | 0 |
| ST×PM | 0.8741 | 48 | 0.0182 | 8.03 | 0 |
| ST×RA | 1.5863 | 24 | 0.0661 | 29.13 | 0 |
| ST×TM | 1.3782 | 12 | 0.1149 | 50.61 | 0 |
| ST×LE | 0.6621 | 24 | 0.0276 | 12.16 | 0 |
| PM×RA | 0.4519 | 32 | 0.0141 | 6.22 | 0 |
| PM×TM | 0.3718 | 16 | 0.0232 | 10.24 | 0 |
| PM×LE | 0.1597 | 32 | 0.0050 | 2.20 | 0 |
| RA×TM | 2.1095 | 8 | 0.2637 | 116.20 | 0 |
| RA×LE | 0.5561 | 16 | 0.0348 | 15.32 | 0 |
| TM×LE | 0.2259 | 8 | 0.0282 | 12.44 | 0 |
| Error | 42.2748 | 18630 | 0.0023 | | |
| Total | 76.7262 | 18899 | | | |

(b) Eclipse-2

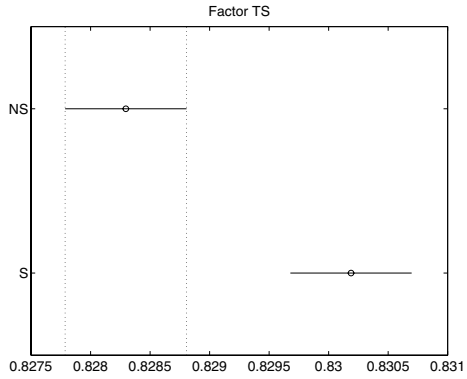
| Source | Sum sq. | d.f. | Mean sq. | F | $p$-value |
|---|---|---|---|---|---|
| TS | 0.0196 | 1 | 0.0196 | 16.84 | 0 |
| ST | 0.3234 | 6 | 0.0539 | 46.35 | 0 |
| PM | 0.1128 | 8 | 0.0141 | 12.13 | 0 |
| RA | 0.6201 | 4 | 0.1550 | 133.34 | 0 |
| TM | 17.7914 | 2 | 8.8957 | 7651.13 | 0 |
| LE | 8.2233 | 4 | 2.0558 | 1768.19 | 0 |
| TS×ST | 0.0563 | 6 | 0.0094 | 8.07 | 0 |
| TS×PM | 0.0028 | 8 | 0.0003 | 0.30 | 0.966 |
| TS×RA | 0.0044 | 4 | 0.0011 | 0.94 | 0.439 |
| TS×TM | 0.0001 | 2 | 0.0000 | 0.03 | 0.975 |
| TS×LE | 0.1060 | 4 | 0.0265 | 22.79 | 0 |
| ST×PM | 0.4190 | 48 | 0.0087 | 7.51 | 0 |
| ST×RA | 1.6234 | 24 | 0.0676 | 58.18 | 0 |
| ST×TM | 0.6933 | 12 | 0.0578 | 49.69 | 0 |
| ST×LE | 0.2336 | 24 | 0.0097 | 8.37 | 0 |
| PM×RA | 0.2765 | 32 | 0.0086 | 7.43 | 0 |
| PM×TM | 0.1902 | 16 | 0.0119 | 10.22 | 0 |
| PM×LE | 0.1420 | 32 | 0.0044 | 3.82 | 0 |
| RA×TM | 1.5731 | 8 | 0.1966 | 169.13 | 0 |
| RA×LE | 0.3444 | 16 | 0.0215 | 18.51 | 0 |
| TM×LE | 1.0153 | 8 | 0.1269 | 109.16 | 0 |
| Error | 21.6604 | 18630 | 0.0012 | | |
| Total | 55.4311 | 18899 | | | |

(c) Eclipse-3

| Source | Sum sq. | d.f. | Mean sq. | F | $p$-value |
|---|---|---|---|---|---|
| TS | 0.0169 | 1 | 0.0169 | 13.24 | 0 |
| ST | 1.2153 | 6 | 0.2026 | 158.61 | 0 |
| PM | 0.4910 | 8 | 0.0614 | 48.06 | 0 |
| RA | 0.8193 | 4 | 0.2048 | 160.39 | 0 |
| TM | 6.1892 | 2 | 3.0946 | 2423.25 | 0 |
| LE | 25.1360 | 4 | 6.2840 | 4920.76 | 0 |
| TS×ST | 0.0722 | 6 | 0.0120 | 9.43 | 0 |
| TS×PM | 0.0017 | 8 | 0.0002 | 0.17 | 0.995 |
| TS×RA | 0.0035 | 4 | 0.0009 | 0.69 | 0.599 |
| TS×TM | 0.0071 | 2 | 0.0036 | 2.80 | 0.061 |
| TS×LE | 0.1878 | 4 | 0.0470 | 36.76 | 0 |
| ST×PM | 0.9292 | 48 | 0.0194 | 15.16 | 0 |
| ST×RA | 1.6779 | 24 | 0.0699 | 54.75 | 0 |
| ST×TM | 0.2616 | 12 | 0.0218 | 17.07 | 0 |
| ST×LE | 0.2216 | 24 | 0.0092 | 7.23 | 0 |
| PM×RA | 0.7420 | 32 | 0.0232 | 18.16 | 0 |
| PM×TM | 0.3571 | 16 | 0.0223 | 17.48 | 0 |
| PM×LE | 0.2409 | 32 | 0.0075 | 5.90 | 0 |
| RA×TM | 1.4179 | 8 | 0.1772 | 138.79 | 0 |
| RA×LE | 0.5834 | 16 | 0.0365 | 28.55 | 0 |
| TM×LE | 2.4196 | 8 | 0.3025 | 236.83 | 0 |
| Error | 23.7912 | 18630 | 0.0013 | | |
| Total | 66.7826 | 18899 | | | |



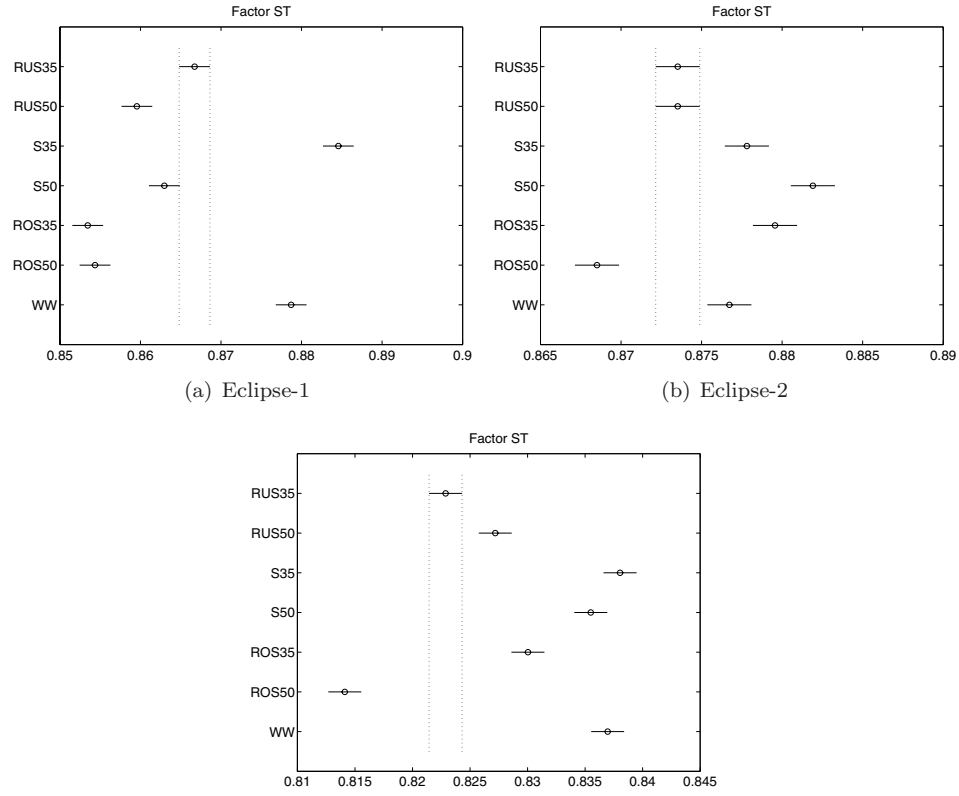

(a) Eclipse-1 (b) Eclipse-2




(c) Eclipse-3

Fig. 9. Multiple comparisons for sampled data — factor TS.

- For Factor TM, we obtain the same conclusions as we did for the non-sampled case. That is, the CV and COMBO methods perform significantly better than the CV-R method. However, CV is recommended compared to COMBO due to its less computational complexity.
- For Factor LE, we have the same conclusions as we did for the non-sampled case. For all three groups of Eclipse datasets, SVM demonstrates the best performance among the five learners, and NB performs worst. Of the three moderate learners, MLP and LR outperformed kNN.
- For Interaction ST×TS, the results demonstrate that the sampled version performed better or significantly better than the non-sampled version for five out of seven sampling techniques over the Eclipse-2 and Eclipse-3 datasets. The opposite conclusion was obtained for the Eclipse-1 datasets. That is, the non-sampled version performed better than or equal to the sampled version for all the sampling techniques.




(a) Eclipse-1 (b) Eclipse-2

(c) Eclipse-3

Fig. 10. Multiple comparisons for sampled data — factor ST.

- For Interaction PM×TS, the results demonstrate that performance of each group is determined by the two main factors (PM and TS) making up the groups, not their interaction. In addition, for the two training data sources, the non-sampled version performed significantly better than the sampled version for all performance metrics over the Eclipse-1 datasets. For Eclipse-2 and Eclipse-3, the sampled version performs better than the non-sampled version for all the performance metrics but not by much.
- For Interaction RA×PM, the performance of each ranker is determined by the two factors (RA and PM) and their interaction. Generally, the NB-aided rankers present better performance than the rankers formed by the performance metrics aided by different learners.
- For Interaction RA×LE, the classification performances are determined by the two main effects (RA and LE) and their interaction. In other words, a change in RA values greatly affects the mean values of the Learner groups. This is similar to the non-sampled case. Furthermore, there is no evidence that the common learning algorithm case is always better than the different learning algorithm case.

## 6. Analysis

At the foundation of our proposed wrapper-based feature ranking techniques are the ranker aid, the performance metric and the methodology. The performance of these techniques naturally depend on the three components. Our experiments show variation in the experimental results between LLTS and Eclipse. With Eclipse, performance enhancement is observed most of the time on tested models after they are built with feature selection obtained from wrapper-based feature ranking. This is probably due to the large feature space of the original dataset that includes 209 attributes, resulting in a considerable number of redundant/irrelevant attributes that may impair the performance of the classification models. While the experiments show data dependencies with the techniques, they also depict the stability of NB as a ranker aid. Several classification performance values are obtained in terms of AUC, and feature selection with NB for the most part enhances the classification performances. Conversely, given the poor performance of feature selection with kNN compared to classification with all attributes, the use of kNN as either the ranker aid or classifier is found to be ineffective.

Based on the experimental results, the CV and COMBO methods show significantly better performance results than the CV-R approach. Furthermore, CV is the preferred methodology for the wrapper-based feature ranking given its low computational complexity compared to the COMBO approach. In general, the approach with the CV methodology leads to models built with feature selection that maintains if not enhances the model's classification performance. For these reasons, CV is our recommended approach.

### 6.1. *"Best" attributes*

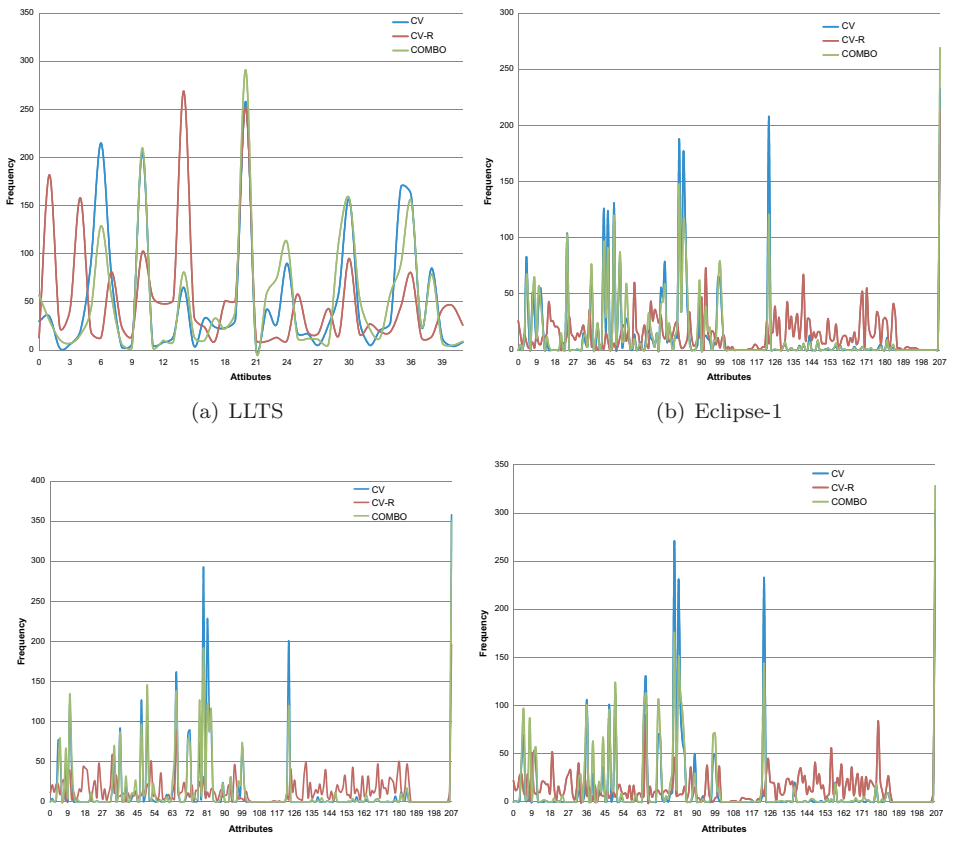
The ability to predict early on in the software product development process any software item that is judged fault-prone is of critical importance toward achieving software quality improvement. Unless one knows which item is defective and needs corrective measures, one cannot achieve improvement. Software metrics, which are an integral part of the state-of-the-practice in software engineering, are intended to help practitioners to evaluate, control and improve both software products and processes. Clearly, the collection of software metrics are important. Equally important if not more is their relevance.

Our experiments provide a new insight to how we may answer the question: which software metrics are most useful for defect prediction? For software practitioners, knowledge of which attributes are best in terms of their predictive power, is fundamentally critical. Equipped with this knowledge, they can simplify their repositories, rendering obsolete, metrics that are less informative and focusing on the collection of metrics deemed relevant to the class.

With the numerous rankings obtained through the experiments conducted in this study, we observe a wide variability among the rankings produced by the different wrapper-based feature ranking techniques. Despise the ranking variability,

we observe that some features or attributes are more frequent than others with respect to membership in the selected features. That is, for each attribute in the feature space, we record a count of its presence in the selected attributes. It is assumed that the best attributes appear most in the selected subsets. Thus, the high frequency of an attribute in the selected subsets presupposes that it is most useful in predicting the class label.

Figure 11(a) shows the frequency of all LLTS attributes. Given the generally poor performance of the CV-R methodology, we consider only CV and COMBO. For each methodology (CV and COMBO), we aggregate the total number of appearances of each attribute in the top six attributes. We rank all attributes starting with the one having the highest number of appearances to the one with the lowest. In our analysis of these two rankings, we observe that both share the same five of the top six attributes. Thus, we deduce that only these five are most useful, and they are: Number of distinct include files (FILINCUQ), Number of different designers




(a) LLTS (b) Eclipse-1

(c) Eclipse-2 (d) Eclipse-3

Fig. 11. Eclipse — Attributes frequency.

making changes (UNQ_DES), Total number of changes to the code for any reason (TOT_UPD), Maximum span of variables (VARSPNMX), and Base 2 logarithm of the number of independent paths (LGPATH).

Indeed, a large number of distinct include files in a software system can be problematic. One type of problems that may result from a large number of include files is the multiple inclusion problem. That is, one include file includes another, which includes the original include file. The hierarchies of these include files can become very complex, making it hard for software programmers to understand their inter-dependencies. Understandably, this characteristic of a software system is an excellent indicator of its quality as demonstrated by our method on the LLTS system. Likewise the number of different designers making changes is of high relevance. As practitioners, we know software is bound to change. In general, there are limitless possible causes of changes in software. Examples include but not limited to feature enhancements and additions, bug fixes, etc. As software evolves, not only does the number of changes increase, but so does the number of different programmers changing the software. However, the more individuals there are making changes; the less likely that the design principles will be retained. This without a doubt will affect the software quality. Our method rightly identifies the total number of changes introduced in a software system throughout its life cycle and the number of different designers making these changes as very relevant software metrics for predicting quality. Our method also identifies as relevant software metrics the maximum span of variables and the base 2 logarithm of the number of independent paths. Undoubtedly, the longer the amount of time variables are retained, the more prone they are for misuse, which potentially affect the operation of the system. Finally, practitioners can have an indication of the complexity of a software system by examining the number of independent paths.

In a similar fashion, we can separate the "worst" features. A feature is considered among the worst if it is among the least frequent of top feature membership. These features are less useful in predicting the class label. For LLTS, the "worst" features are: Number of knots (NNT) (Note that a "knot" in a control flow graph is where arcs cross due to a violation of structured programming principles), and Execution time of an average transaction on a system serving businesses (BUSCPU).

Figures 11(b)–11(d) show the frequency of all Eclipse attributes according to each group. Like with LLTS, we consider only CV and COMBO to determine the most relevant attributes. We analyze each attribute of Eclipse in terms of their appearances in the top selected features. Considering all three groupings of Eclipse, we note four features that are common to the top eight of all three groupings, so we deem them of high importance (relevant). These are: Number of defects reported in the last six months before release (PRE), Number of nodes in the abstract syntax trees (SUM), Total count of the qualified name nodes (QualifiedName), and Total count of the simple name nodes (SimpleName).

The results show the pre-release defects attribute having a relatively high frequency. This validates the high correlation between the number of pre-release and post-release defects reported by Zimmermann *et al.*[21] A defective instance is more likely to have had a large number of pre-release defects. In other words, a software item that is deemed of poor quality prior to release (due to a large number of observed pre-release defects) is more likely to remain poor after release. Also, the abstract syntax tree, which depicts a detailed representation of the source code, is a very reliable and convenient way to analyze the static structures of a software program. However, not all information in the tree, according to our results, is relevant for predicting the quality of the software. We can observe only three pieces of information from the tree that are relevant. The size of the tree, expressed in the number of nodes, provides a good indication of fault-proneness. Another relevant piece of information is the frequency of qualified name node types. A qualified name represents the explicit reference to namespaces to which belongs the corresponding object, function or variable. It distinguishes between multiple software items with the same name belonging to different namespaces or packages, thereby preventing name conflict/collision. The third relevant piece of information is the frequency of the software items without the namespace or package, referred to as simple name. The simple name, in some contexts, may be obscure. Therefore, ambiguity is more likely to result from a higher frequency of simple name. Conversely, the more names that are fully qualified, the less ambiguity there is. The results also show the "worst" features of Eclipse, those that lack predictive power, and they are the normalized abstract syntax tree nodes. These attributes have no relevant information for determining the fault-proneness of a software item.

To validate our findings, we build models with only the "best" attributes, and we evaluate them on the corresponding test datasets. Table 12 shows the results with the "best" LLTS attributes (FILINCUQ, UNQ DES, TOT UPD, VARSPNMX, and LGPATH). Table 13 shows the results with the "best" Eclipse attributes (PRE, SUM, QualifiedName, and SimpleName) for Eclipse 1, Eclipse 2, and Eclipse 3. Where the performance of the "best" attributes is better than that of all attributes, the AUC value is marked in **bold** font. The results demonstrate that for LLTS datasets, in 29 out of 40 cases, the "best" attributes outperform all attributes while for the Eclipse-1, 2, and 3 datasets, the numbers are 37/40, 39/40, and 34/40 respectively.

Table 12. LLTS — "Best" attributes classification performance.

| Learner | NS | RUS35 | RUS50 | ROS35 | ROS50 | S35 | S50 | WW |
|---|---|---|---|---|---|---|---|---|
| NB | **0.8256** | **0.8237** | **0.8246** | **0.8255** | **0.8255** | **0.8272** | **0.8270** | **0.8241** |
| MLP | **0.8372** | **0.8348** | **0.8321** | **0.8342** | **0.8329** | **0.8293** | 0.8288 | **0.8369** |
| kNN | 0.7654 | 0.7701 | 0.7878 | 0.7583 | 0.7582 | 0.7450 | 0.7333 | 0.7683 |
| SVM | 0.5717 | **0.8369** | **0.8369** | **0.8382** | **0.8368** | **0.8370** | **0.8359** | 0.8010 |
| LR | **0.8381** | **0.8370** | **0.8360** | **0.8375** | **0.8372** | **0.8364** | **0.8360** | **0.8373** |

Table 13. Eclipse — "Best" attributes classification performance.

| Grouping | Learner | NS | RUS35 | RUS50 | ROS35 | ROS50 | S35 | S50 | WW |
|---|---|---|---|---|---|---|---|---|---|
| Eclipse-1 | NB | **0.8726** | **0.8394** | **0.8322** | **0.8735** | **0.8733** | **0.8866** | **0.8874** | **0.8868** |
| | MLP | **0.9387** | **0.9399** | **0.9377** | **0.8963** | 0.8824 | 0.8253 | 0.8521 | **0.9380** |
| | kNN | **0.9182** | **0.8692** | **0.8964** | **0.9136** | **0.9136** | **0.8983** | **0.8934** | **0.9195** |
| | SVM | **0.9348** | **0.9370** | **0.9373** | **0.9398** | **0.9424** | **0.9402** | **0.9417** | **0.9375** |
| | LR | **0.9273** | **0.8031** | **0.8074** | **0.9158** | **0.9082** | **0.9000** | **0.8975** | **0.9319** |
| Eclipse-2 | NB | 0.8449 | **0.8550** | **0.8718** | **0.8464** | **0.8459** | **0.8540** | **0.8543** | **0.8717** |
| | MLP | **0.9230** | **0.9331** | **0.9330** | **0.9303** | **0.9323** | **0.9314** | **0.9332** | **0.9321** |
| | kNN | **0.9102** | **0.9160** | **0.9257** | **0.9046** | **0.8932** | **0.9078** | **0.9074** | **0.9127** |
| | SVM | **0.9273** | **0.9294** | **0.9298** | **0.9309** | **0.9307** | **0.9285** | **0.9308** | **0.9292** |
| | LR | **0.9210** | **0.9209** | **0.9099** | **0.9219** | **0.9220** | **0.9180** | **0.9188** | **0.9274** |
| Eclipse-3 | NB | 0.7490 | 0.7565 | **0.7726** | 0.7535 | 0.7529 | 0.7548 | 0.7633 | **0.7877** |
| | MLP | **0.8859** | **0.8960** | **0.8879** | **0.8780** | **0.9007** | **0.8814** | **0.9006** | **0.8845** |
| | kNN | **0.8398** | **0.8314** | **0.8472** | **0.8262** | **0.8144** | **0.8366** | **0.8250** | **0.8587** |
| | SVM | **0.8945** | **0.8975** | **0.8784** | **0.8905** | **0.8805** | **0.8897** | **0.8790** | **0.8876** |
| | LR | **0.8786** | **0.8692** | **0.8809** | **0.8779** | **0.8791** | **0.8702** | **0.8704** | **0.8820** |

## 7. Conclusion

In this paper, we empirically study our proposed feature ranking method called wrapper-based feature ranking. The core of this novel method comprises of a performance metric, a ranker aid and a methodology. We consider nine performance metrics, five different learners and three methodologies (CV, CV-R, and COMBO) in our evaluation. We build software defect classification models by applying wrapper-based feature ranking on one software release, and we validate the models on subsequent releases totally disjoint from the training dataset. The wrapper-based feature ranking techniques are evaluated on two sets of software engineering data. Our findings show that the performance of our wrapper-based feature ranking techniques is dependent on the combination of the ranker aid, the performance metric and the methodology. They also show that naïve Bayes is very stable as a ranker aid for wrapper-based feature ranking, given that only minor changes are observed with the ranking techniques. We validate the excellent performance of SVM with feature selection when subsets are obtained from wrapper-based ranking techniques, but only with the approach consisting of CV. We also show comparable performance of logistic regression in conjunction with CV. Our findings not only provide guidance as to which learner is more stable as a ranker aid but also which approach leads to better performance results for which of the five learners in terms of AUC. The CV methodology is more likely to lead to better classification performance in terms of AUC. We also observe an improvement in the classification performance with feature selection when sampling is applied to the training data prior to ranking.

Finally, our work provides guidance to software practitioners as to how they can determine which software metrics that are most useful for defect prediction. This is a breakthrough not only for software engineering but also for other domains in which feature selection is a pre-requisite activity. Our guidance extends to how

researchers and practitioners can determine the threshold for retaining the top ranked attributes. Future work can consider various performance metrics (other than AUC) to assess the performance of these classifier-aided ranking techniques when building the different quality prediction models.

## Acknowledgments

We are grateful to Professor Hoang Pham, EIC of the International Journal of Reliability, Quality, and Safety Engineering, for his comments and suggestions.

## References

1. B. Livshits and T. Zimmermann, Dynamine: Finding common error patterns by mining software revision histories, *SIGSOFT Softw. Eng. Notes* **30**(5) (2005) 296–305.
2. J. S. Shirabad, T. Lethbridge and S. Matwin, Mining the maintenance history of a legacy software system, in *Software Maintenance, ICSM 2003, Proceedings. International Conference on* (2003), pp. 95–104.
3. S. Dick, A. Meeks, M. Last, H. Bunke and A. Kandel, Data mining in software metrics databases, *Fuzzy Sets and Systems* **145**(1) (2004) 81–110. (computational Intelligence in Software Engineering. [Online]. Available: http://www.sciencedirect.com/science/article/B6V05-49YH3XJ-H/2/b8903e62a58f1097510da041f97e5d1c).
4. C. Catal and B. Diri, Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem, *Inf. Sci.* **179**(8) (2009) 1040–1058.
5. Y. Saeys, I. N. Inza and P. Larrañaga, A review of feature selection techniques in bioinformatics, *Bioinformatics* **23**(19) (2007) 2507–2517.
6. I. Guyon and A. Elisseeff, An introduction to variable and feature selection, *J. Mach. Learn. Res.* **3** (2003) 1157–1182.
7. T. M. Khoshgoftaar and K. Gao, A novel software metric selection technique using the area under ROC curves, in *Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering*, San Francisco, CA (July 1–3, 2010), pp. 203–208.
8. M. A. Hall, Correlation-based feature selection for discrete and numeric class machine learning, in *Proc. 17th International Conf. on Machine Learning*. Morgan Kaufmann, San Francisco, CA (2000), pp. 359–366. [Online]. Available: citeseer.ist.psu.edu/hall00correlationbased.html.
9. H. Wang, T. M. Khoshgoftaar, K. Gao and N. Seliya, Mining data from multiple software development projects, in *Proceedings of the 3rd IEEE International Workshop Mining Multiple Information Sources*, Miami, FL (December 6, 2009), pp. 551–557.
10. L. Yu and H. Liu, Feature selection for high-dimensional data: A fast correlation-based filter solution, in *Proceedings of the Twentieth International Conference on Machine Learning*. AAAI Press, (2003), pp. 856–863.
11. G. H. John, R. Kohavi and K. Pfleger, Irrelevant features and the subset selection problem, in *International Conference on Machine Learning*, (1994), pp.121–129. (Journal version in AIJ, available at http://citeseer.nj.nec.com/13663.html. [Online]. Available: citeseer.ist.psu.edu/john94irrelevant.html.)
12. H. Guo and Y. L. Murphey, Automatic feature selection — A hybrid statistical approach, *Pattern Recognition, International Conference on*, Vol. 2, (2000), p. 2382.

13. S. Das, Filters, wrappers and a boosting-based hybrid for feature selection, in ICML '01, *Proceedings of the Eighteenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. (2001), pp. 74–81.
14. A. L. Blum and P. Langley, Selection of relevant features and examples in machine learning, *Artif. Intell.* **97**(1–2) (1997) 245–271.
15. Z. Chen, T. Menzies, D. Port and B. Boehm, Feature subset selection can improve software cost estimation accuracy, in *PROMISE '05: Proceedings of the 2005 Workshop on Predictor Models in Software Engineering*, New York, NY, USA: ACM (2005), pp. 1–6.
16. D. Rodriguez, R. Ruiz, J. Cuadrado-Gallego, J. Aguilar-Ruiz and M. Garre, Attribute selection in software engineering datasets for detecting fault modules, in *EUROMICRO '07: Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications*, Washington, DC, USA: IEEE Computer Society (2007), pp. 418–423.
17. R. Ruiz, J. C. Riquelme and J. S. Aguilar-Ruiz, Incremental wrapper-based gene selection from microarray data for cancer classification, *Pattern Recogn.* **39**(12) (2006) 2383–2392.
18. W. Altidor, T. Khoshgoftaar and A. Napolitano, Wrapper-based feature ranking for software engineering metrics, in *Proceedings of the IEEE International Conference on Machine Learning and Applications — ICMLA 2009*, Miami, FL, USA (2009), pp. 241–246.
19. K. Gao, T. Khoshgoftaar and A. Napolitano, Exploring software quality classification with a wrapper-based feature ranking technique, in *Proceedings of the 21st IEEE International Conference on Tools with Artificial Intelligence*, Las Vegas, Nevada, USA (2009), pp. 67–74.
20. G. Boetticher, T. Menzies and T. Ostrand, Promise repository of empirical software engineering data, http://promisedata.org/, 2007.
21. T. Zimmermann, R. Premraj and A. Zeller, Predicting defects for eclipse, in *ICSEW '07, Proceedings of the 29th International Conference on Software Engineering Workshops*, Washington, DC, USA: IEEE Computer Society (2007), p. 76.
22. I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd edn. Morgan Kaufmann (2005). [Online]. Available: /bib/private/witten/Data Mining Practical Machine Learning Tools and Techniques 2d edu — Morgan Kaufmann.pdf.
23. P. Domingos and M. J. Pazzani, On the optimality of the simple bayesian classifier under zero-one loss, *Machine Learning* **29**(2–3) (1997) 103–130. [Online]. Available: citeseer.ist.psu.edu/domingos97optimality.html.
24. H. Liu, J. Li and L. Wong, A comparative study on feature selection and classification methods using gene expression profiles and proteomic patterns, *Genome Informatics* **13** (2002) 51–60.
25. T. Fawcett, ROC graphs: Notes and practical considerations for data mining researchers, HPL-2003-4, Intelligent Enterprise Technologies Lab, Palo Alto, CA (2003).
26. C. L. Jin, C. X. Ling, J. Huang and H. Zhang, Auc: A statistically consistent and more discriminating measure than accuracy, in *Proceedings of 18th International Conference on Artificial Intelligence (IJCAI-2003)*, pp. 329–341.
27. Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto and K.-I. Matsumoto, The effects of over and under sampling on fault-prone module detection, in *ESEM '07: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, Washington, DC, USA: IEEE Computer Society, (2007) pp. 196–204.

28. N. V. Chawla, K. W. Bowyer and P. W. Kegelmeyer, Smote: Synthetic minority oversampling technique, *Journal of Artificial Intelligence Research* **16** (2002) 321–357. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary? doi=10.1.1.18.5547.

## About the Authors

Wilker Altidor is a PhD student in the Department of Computer and Electrical Engineering and Computer Science at Florida Atlantic University. His research interests include software engineering and data mining and machine learning. Altidor received both BS and MS degrees in computer science from Florida Atlantic University.

Taghi M. Khoshgoftaar is a professor of the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University and the Director of the Empirical Software Engineering Laboratory and Data Mining and Machine Learning Laboratory. His research interests are in software engineering, software metrics, software reliability and quality engineering, computational intelligence, computer performance evaluation, data mining, machine learning, and statistical modeling. He has published more than 450 refereed papers in these areas. He is a member of the IEEE, IEEE Computer Society, and IEEE Reliability Society. He was the program chair and general chair of the IEEE International Conference on Tools with Artificial Intelligence in 2004 and 2005 respectively and was the Program chair and General Chair of the International Conference on Software Engineering and Knowledge Engineering in 2008 and 2009 respectively. He is the program chair of the IEEE International Conference on Machine Learning and Applications (2010). He has served on technical program committees of various international conferences, symposia, and workshops. Also, he has served as North American Editor of the Software Quality Journal, and was on the editorial boards of the journals Multimedia Tools and Applications and Empirical Software Engineering and is on the editorial boards of the journals Software Quality, Software Engineering and Knowledge Engineering, Fuzzy Systems, Knowledge and Information Systems and Social Network Analysis and Mining.

Kehan Gao received the Ph.D. degree in Computer Engineering from Florida Atlantic University, Boca Raton, FL, USA, in 2003. She is currently an Associate Professor in the Department of Mathematics and Computer Science at Eastern Connecticut State University. Her research interests include software engineering, software metrics, software reliability and quality engineering, computer performance modeling, computational intelligence, and data mining. She is a member of the IEEE Computer Society and the Association for Computing Machinery.

Copyright of International Journal of Reliability, Quality & Safety Engineering is the property of World Scientific Publishing Company and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.

www.manaraa.com