Identifying Learners Robust to Low Quality Data

Andres A. Folleco, Taghi M. Khoshgoftaar, Jason Van Hulse and Amri Napolitano
Florida Atlantic University,
Boca Raton, Florida, USA
E-mail: {andres, taghi}@cse.fau.edu , {jvanhulse, amrifau}@gmail.com

Keywords: quality of data, class imbalance, random forest, robust learning

Received: October 12, 2008

Low quality or noisy data, which typically consists of erroneous values for both dependent and independent variables, has been demonstrated to have a significantly negative impact on the classification performance of most learning techniques. The impact on learner performance can be magnified when the class distribution is imbalanced or skewed. Unfortunately in real world environments, the presence of low quality imbalanced data is a common occurrence. In most scenarios, the actual quality of such datasets is unknown to the data mining practitioner. In this study, we identify learners (from a total of 11 classification algorithms) with robust performance in the presence of low quality imbalanced measurement data. Noise was injected into seven imbalanced software measurement datasets, initially relatively free of noise. Learners were evaluated using analysis of variance models based on their performance as the level of injected noise, the number of attributes with noise, and the percentage of minority instances containing noise were increased. Four performance metrics suitable for class imbalanced data were used to measure learner performance. Based on our results, we recommend using the random forest ensemble learning technique for building classification models from software measurement data, regardless of the quality and class distribution of the data.

Povzetek: Predstavljena je metoda za identificiranje robustnih klasifikatorjev pri šumnih podatkih.

1 Introduction

Not only are real-world datasets often class imbalanced, but typically their attributes (including the class) can contain erroneous values that may negatively impact learning performance [17, 29, 32]. Consequently, it is not uncommon for empirical software engineering practitioners to construct learners using suboptimal or low quality imbalanced data. Note that in this work[1], only binary classification problems were considered. In software quality classification, class imbalance occurs when the number of fault-prone (fp) modules is significantly outnumbered by the number of not fault-prone (nfp) program modules. No other related work in the software quality prediction domain was found that evaluated the robustness of learning techniques, using four performance metrics, relative to low quality imbalanced measurement data. In this study, simulated noise was injected in both the independent attributes as well as the class (i.e., labeling errors) of seven class imbalanced software engineering measurement datasets. The substantial and significant scope of our experiments make this study truly unique for the domain of empirical software engineering.

The robustness of 11 distinct algorithms trained using low quality imbalanced real-world measurement data is evaluated by analyzing four metrics related to learner performance. Noisy data was simulated by injecting artificially induced, domain realistic noise into the independent attributes and class of seven real-world class imbalanced software measurement datasets as explained in Section 4.3. Four performance metrics particularly well suited to deal with class imbalanced data were selected for this study. The area under the ROC curve (AUC), the area under the Precision-Recall curve (PRC), the Kolmogorov-Smirnov statistic (KS), and the F-measure (FM) test statistic were selected (see Section 4.2) to measure the impact on learning performance. The overall impact of noise on each learner was measured as the level of noise, the number of attributes with injected noise, and the percentage of minority instances containing noise increased across all the datasets (see Section 5). Furthermore, we illustrate the impact of the factor interaction between the overall noise levels and the percent of minority instances with noise. Conclusions are presented in Section 6.

2 Related work

Regardless of the perceived soundness of a preferred classification algorithm, learning from low quality class imbalanced data will very likely result in biased and suboptimal performance. Several studies in classification initiatives have demonstrated that the presence of noisy values

[1]This is an expanded version, by invitation, of the work accepted and presented at the 2008 IEEE International Conference on Information Reuse and Integration- IRI'08 [11]

(mainly corrupted class labels) in the training dataset will likely impact the predictive accuracy of a learning algorithm [17, 32]. Arguably, the main factors determining data quality include independent attribute noise [27, 32], dependent attribute or class noise [16, 5], and missing or omitted values [20]. The data quality characteristics considered in this study include the presence of noise in both the independent and dependent (class) attributes.

In addition, the real-world measurement datasets used in this work are inherently class imbalanced. A software measurement dataset selected for binary classification tasks (we only consider binary classification in this work) is said to be $imbalanced$ if the number of positive (fault-prone or fp) class modules is less than the number of negative (not fault-prone or nfp) class modules. Typically, minority instances make up the positive class, while the negative class is composed of the majority instances. Frequently in real-world scenarios, the practitioner is primarily interested in identifying examples from the minority group. In the context of software engineering, this is frequently seen in mission critical applications, where a premium is placed on identifying fault-prone modules during the test phase. Low quality real-world datasets containing class imbalance distributions pose a great challenge to any data mining and machine learning effort. In fact, researchers contend that data quality and class imbalance can significantly impact the reliability of machine learning models in real-world scenarios, and consequently demand empirical consideration and experimental evaluation [10, 35]. Data with such characteristics can be found in a wide variety of application domains besides software quality classification, including for example network intrusion detection [19] and fraud detection [13].

In our study, we investigate the robustness of 11 learning algorithms in the presence of low quality class imbalanced real-world software measurement datasets, initially relatively free of noise. Weiss [29] performed a preliminary study of the effects of class and attribute noise on classification with simulated datasets using the C4.5 learner. However, no real world datasets or additional learners were used. Weiss and Provost [30] investigated the effect of class distribution and training set size on classification performance. Their results imply that the natural class distribution generates higher overall accuracy. However, the overall accuracy rate is often not considered an appropriate measurement method when dealing with class imbalanced datasets. Furthermore, they also suggested that a more balanced class distribution can result in higher AUC values. Van Hulse et al. [28] conducted an investigation of the impact on classification performance from class imbalanced data injected with simulated class noise. The AUC metric was used to measure the learning performance. The independent attributes were not considered for noise injection in their study. Studies have been conducted to investigate the impact of noise using classification performance-enhancing techniques like cost sensitive learning [33, 34] with various cost ratios [9]. In general, relatively few studies have evaluated the impact of noise (in the class only) in imbalanced data. To our knowledge, no related works have investigated the impact of class and attribute noise on learners constructed from class skewed measurement datasets using several performance metrics suitable for class imbalanced distributions.

3 Learning algorithms

The open-source Java based Weka data mining and machine learning tool [31] was used to implement the learners in this study. The learners used were selected because most of them are commonly used in class imbalance scenarios and several are also used in the software engineering and software quality classification domain. The default parameter values of some of the techniques were changed when their respective classification performances improved substantially.

3.1 Random forest

The *random forest* (RF) classifier was developed by Breiman [3]. RF is a powerful, relatively new approach to data exploration, data analysis, classification, and predictive modeling. A random forest is a collection of unpruned, CART-like trees [4] following specific rules for tree growing, tree combination, self-testing, and post-processing. Trees are grown using binary partitioning in which each parent node is split into no more than two children. Each tree is grown on a different random subsample of the training data. Randomness is also injected into the tree split selection process. RF selects a relatively small subset of available attributes at random. In the Weka tool, the default for the $numFeatures$ parameter uses $\lfloor log_2 M + 1 \rfloor$ attributes selected at random for each node in the tree where M is the original number of independent attributes in the data. Attribute selection significantly speeds up the tree generation process. Once a node is split on the best splitter attribute, the process is repeated entirely on each child node. Then, a new list of predictive attributes is selected at random for each node. The trees must remain unpruned to their absolute maximum size in order to maximize the chances of including important attributes into the trees.

Bootstrapping is a process of random sampling with replacement from the training dataset. By applying bootstrapping during the tree induction process, approximately 37% of the observations in the training dataset are not used and form the out-of-bag samples. Another important parameter in the RF algorithm is the number of trees $numTrees$ in the ensemble. The default value for $numTrees$ in Weka is 10, however previous research by our group [15] found 100 to be a more appropriate value, so 100 was used instead. Combining results from multiple models (trees) generally yields better performance results than those obtained from a single model. Combining trees by averaging the votes will only be beneficial if the trees are different from each other. RF induces vastly more between-tree variation by forcing random splits on different predictive attributes. Having a diverse collection of robust trees lowers the overall error rate, avoids over-fitting training data, imbues a substantial resilience to noisy values, and therefore enhances the performance of the RF [3] ensemble classifier.

3.2 k Nearest Neighbors (Two versions)

K nearest neighbors [1] (kNN) is called IBk in the Weka implementation of an instance-based classification technique using k nearest neighbors. The class of a test case is predicted by majority voting of the k nearest neighbors. If only one nearest neighbor is selected to predict the class of a test instance, especially in the presence of outliers and/or low quality data, it may lead to increased inaccuracy. The Euclidean distance is often used as a similarity function to determine the potential candidate nearest neighbors. A possible disadvantage of IBk is that its computation time depends on the size of the number of nearest neighbors. As the number of nearest neighbors increases, so do the computational resources and time needed. In our experiments, kNN classifiers were built with changes to two parameters: The ‘distanceWeighting’ parameter was set to ‘Weight by 1/distance’ and two different ‘kNN’ classifiers were built using $k = 2$ and $k = 5$ neighbors. These were denoted ‘2NN’ and ‘5NN,’ respectively.

3.3 C4.5 Decision Tree (Two versions)

C4.5 [23] is a benchmark decision tree classification algorithm. J48 is Weka’s implementation of this algorithm. It is an inductive supervised classifier that uses decision trees to represent the underlying structure of the input data. The algorithm has four major components: the decision tree generator, the production rule generator, the decision tree interpreter, and the production rule interpreter. These modules are used for constructing and evaluating the classification tree models. The algorithm begins with an empty tree, to which is added decision and leaf nodes, starting at the root (top) node. In the next step, using one of the attributes x_j ($j = 1, \ldots, m = \#attributes$) the instances in the root node are split into two (or more) child nodes N_l and N_r. For example, if x_j is a continuous attribute and $i = 1, \ldots, n = \#instances$, we define $N_l = \{\, \mathbf{x}_i \in D \mid x_{ij} < t \,\}$, and $N_r = \{\, \mathbf{x}_i \in D \mid x_{ij} \geq t \,\}$ for some value of t from x_j. C4.5 evaluates each of the attributes x_j to determine the best split at each tree node. The splitting process is recursively applied to each of the resulting child/leaf nodes until some stopping criteria is met. After the tree is fully built, C4.5 provides the option to prune sections of the tree to avoid over-fitting. Two different versions of the C4.5 classifier were used in our experiments. The version we call C4D uses the default parameter settings in Weka to build the tree(s). The version of C4.5 we call C4N disables decision-tree pruning and enables Laplace smoothing. These settings were recommended for speed and performance by Weiss [30].

3.4 Support Vector Machine

The *support vector machine* (SVM) classifier, called SMO in Weka, can be used to solve two-class (binary) classification problems [24]. These classifiers find a maximum margin linear hyperplane within the instance space that provide the greatest separation between the two classes. Instances that are closest to the maximum margin linear hyperplane form the support vectors. Once the instances that form the support vector have been identified, the maximum margin linear hyperplane can then be constructed. We consider the following linear hyperplane separating two classes [31] from:

$$x = b + \sum \alpha_i \, y_i \, a(i) \, \cdot \, a \tag{1}$$

where i is a support vector, y_i is the class of the training instance $a(i)$, b, and α_i are numeric parameters that are adjusted based on the classification algorithm. The term $a(i) \cdot a$ represents the dot product of the test instance with one of the support vectors. Identifying a solution to the linear hyperplane by Equation 1 is the same as solving a constrained quadratic optimization problem. In this study, the SVM classifier had two changes to the default parameters: the complexity constant ‘c’ was set to 5.0 and ‘buildLogisticModels’ was set to ‘true.’ By default, a linear kernel was used.

3.5 Logistic Regression

Logistic regression (LR) is a statistical regression model [14] that can be used to estimate two-class classification problems. Using the training data instances as input, a logistic regression model is created which is used to decide the class membership of the test data instances. The logistic function used for modeling may be defined as follows:

$$f(z) = \frac{1}{1 + e^{-z}}$$

where z denotes the input instances from the training data and e denotes the base of the natural logarithm. The logistic function can take as input z any negative or positive value, while the output of the function is always in the range of zero to one. The output of the logistic regression classifier expresses the probability of an instance belonging to a certain class. An instance of a training dataset that is used as input for the logistic regression model can be:

$$z = \omega_0 + (\omega_1 \times x_1) + (\omega_2 \times x_2) + (\omega_3 \times x_3) + \ldots + (\omega_k \times x_k)$$

where ω_0 is known as the intercept, ω_1, ω_2, ω_3, …, ω_k are called the regression coefficients or model weights, x_1, x_2, x_3, …, x_k denote the corresponding instance attribute values from the training set, and k is the total number of attributes considered. Each of the weights describes the impact of the corresponding attribute value on z. Recall that z is used to determine the class membership of the instance. The weights must be adjusted in order to optimize the logistic regression model on the training data. The log-likelihood of the model is used to estimate the goodness of fit and the weights for the model are chosen to maximize this log-likelihood function. In this study, the Weka default parameter settings were used for this classifier.

3.6 Naive Bayes

Naive Bayes (NB) is a simple and fast algorithm based on the Bayesian rule of conditional probability [12]. NB assumes that attributes are independent of each other within a given class. Even though this condition may not be realistic in real-world data, NB has been known to perform well with this assumption of attribute independence. The algorithm estimates the class probabilities $P(fp|\mathbf{x})$ using the Bayes theorem by considering the following expression,

$$\begin{aligned} P(fp|\mathbf{x}) &= \frac{P(fp,\mathbf{x})}{P(\mathbf{x})} = \frac{P(\mathbf{x}|fp)P(fp)}{P(\mathbf{x})} \\ &= \frac{P(x_1|fp)...P(x_m|fp)P(fp)}{P(\mathbf{x})} \end{aligned} \quad (2)$$

where m is the number of attributes and fp is the fault-prone (or the minority/positive) class. The independence of attributes can be used to factor the class conditional probability $P(\mathbf{x}|fp)$ into $P(x_1|fp)...P(x_m|fp)$. This transformation allows for the estimation of m one-dimensional distributions $P(x_j|fp), j = 1, ..., m$, instead of estimating the joint distribution of $P(\mathbf{x}|fp)$ from the data. In our experiments, the default parameter values were used within the Weka implementation of this algorithm.

3.7 Rule-Based Classifier

RIPPER (Repeated Incremental Pruning to Produce Error Reduction - RIP) is a rule-based classifier and is named JRip [6] in Weka. This algorithm was introduced by Cohen [6] as a fast classifier of "If-Then" classification rules. Initially, the algorithm splits the training dataset into two parts. One part is used to induce rules, while the second part of the training dataset is used to validate the induced rules. If a rule's classification accuracy falls below a minimum accuracy threshold, the rule is then eliminated from the model. RIP imposes a rule induction ordering, minority class rules first, followed by the majority class. Once all of the instances in the minority class have been covered, a default rule is generated to classify the majority data. This feature reduces the description length of a rule set. The default Weka parameters for this classifier were not changed in our experiments.

3.8 Multilayer Perceptron Networks

Multilayer Perceptron (MLP) is a network of perceptrons [21]. A perceptron is the simplest neural network representing a linear hyperplane within instance space. MLPs can be used to solve complex problems. Every MLP contains an input and output layer and at least one hidden layer. A layer is an arrangement of neurons that include hidden ones which do not have any connections to external sources or environments. MLPs are typically implemented as a back-propagation neural network. In a back-propagation neural network, the error from an output neuron is fed back to the same neuron. The neuron output is the thresholded weighted sum of all its inputs from the previous layer. This process is continued iteratively until the error can be tolerated or reaches a specific threshold. MLPs map the instances in the input data onto a set of output values using three or more layers of neurons. Activation functions are used to calculate the output from the input into the neurons, which is comprised of weighted sums of the outputs from the previous layer. Two parameters from MLP were changed from their default values. The 'hiddenLayers' parameter was changed to '3' to define a network with one hidden layer containing three nodes, and the 'validationSetSize' parameter was changed to '10' to cause the classifier to leave 10% of the training data aside to be used as a validation set to determine when to stop the iterative training process.

3.9 Radial Basis Function Networks

Radial basis function networks (RBF) are another type of artificial neural network [21]. They are similar to MLPs except in the method used for processing the data within a single hidden layer. The hidden layer is of high enough dimension which provides a nonlinear transformation from the input space. The output layer in these networks provides a linear transformation from the hidden-unit space to the output space. When using RBF neurons, a category of patterns can be regarded as a Gaussian distribution of points in pattern space. The neuron fires when its input is sufficiently close to activate the Gaussian. Inputs are encoded by computing a measure of how close they are to a receptive field, e.g., distance between the input vector and the centroid of that neuron. In this study, the only parameter change for RBF was to set the parameter 'numClusters' to 10.

4 Empirical methodology

4.1 Software Measurement Datasets

The JM1, CM1, MW1, PC1, KC1, KC2, and KC3 datasets were obtained from the NASA Metrics Data Program (MDP). Learners were built using 13 basic metrics [16] as independent variables. The dependent variable was a binary module-class label, i.e., fault-prone or not fault-prone. The minority class is represented as the positive (fault-prone) class, while the majority class is represented as the negative (not fault-prone) class. All the instances in the data represent measurements taken from the software modules.

The KC1, KC2, and KC3 projects comprise a mission control system and were developed and implemented by different personnel with no overlapping software components. The KC1 system, implemented in C++, is a software component of a large ground system. The KC2 system, implemented in C++, is the science data processing component of a storage management system used for ground processing data. The KC3 system, written in Java, is software developed for collection, processing, and delivery of satellite meta-data. The PC1 system, implemented in C, is flight control software from an earth orbiting satellite. The JM1 project, implemented in C, is a real-time ground system that uses simulation to generate predictions for space missions. The MW1 project, implemented in C, is the control software of a zero gravity experiment related to combustion. The CM1 project, implemented in C, is a science instrument system used for mission measurements. The software modules fault data obtained for the software projects indicated the number of faults detected during the corresponding software development cycles.

A rule-based noise filter was applied to the CM1, MW1, PC1, KC1, KC2, and KC3 datasets to identify and remove noisy instances [18]. Table 1 provides details about the seven initial[2] datasets and their respective cleansed versions. See Van Hulse and Khoshgoftaar [26] for a detailed discussion of the cleansing process for JM1. The 'i/c' sub-headers indicate the initial and cleansed number of instances of a dataset, listed as '#initial / #cleansed'. The row labeled 'P' indicates the number of positive examples in the initial and cleansed datasets. Likewise, the row labeled 'N' indicates the number of negative examples. The 'P + N' row contains the total number of instances in the initial and cleansed datasets, respectively. The '%P' row contains the level of imbalance present in the initial and cleansed datasets. For example, PC1 initially contained 1107 instances, of which 6.9% were positive. After cleansing, 703 total instances remained of which 7.5% were positive. Only the cleansed datasets were used in this work because they were subjected to a methodical and carefully designed noise cleansing process developed by Khoshgoftaar et al. [18] (see also Van Hulse [25] for a discussion of the noise cleansing procedure). The motivation for using relatively cleansed datasets before actually injecting the noise is to ensure the reliability of the results. Adding noise to inherently low quality data can significantly bias and compromise the reliability of any results derived from using such data.

Table 2 contains the overall classification performance obtained across all eleven classifiers for each cleansed dataset. According to the AUC, PRC, KS, and FM values (described in Section 4.2), the best classification performance was obtained using the largest (and relatively cleanest) dataset, JM1. The second best performance was obtained using KC1 (second largest), and the worst performance was obtained using MW1 (the most imbalanced and nearly the smallest dataset). The average values shown in the 'Avg' row were calculated across all seven datasets.

[2]JM1 is an exception to these initial datasets because it is a subset of the much larger original JM dataset [25].

4.2 Performance Metrics

In a binary decision problem, a learner labels examples as either positive or negative. If very few examples belong to the positive class (as few as 1% or less), a learner could obtain an overall accuracy of 99% by just classifying all instances as negative. This method is useless in a domain like software quality classification because the examples of interest are typically from the positive class. Thus, performance metrics such as accuracy or the misclassification rate are inappropriate for substantially class imbalanced data.

The Receiver Operating Characteristic curve [22] (ROC) graphs true positive rates on the y-axis versus the false positive rates on the x-axis. The resulting curve illustrates the trade-off between detection and false alarm rates. Often, performance metrics consider only the default decision threshold of 0.5. ROC curves illustrate the performance across all decision thresholds. The threshold independent nature of ROC curves makes them well suited for describing the classification performance of models built on class imbalanced data. For a single numeric measure, the *area under the ROC curve* (AUC) is widely used, providing a general idea of the predictive potential of the classifier. Two classifiers can be evaluated by comparing their AUC values. Provost and Fawcett [22] give an extensive overview of ROC curves and their potential use for optimal classification.

The Precision-Recall curve [8] (PR) provides a different perspective regarding a classifier's performance on class imbalanced datasets. Precision measures that fraction of instances classified as positive that are truly positive. Recall measures the fraction of positive instances that have correct labels. The PR curve graphs recall on the x-axis and precision on the y-axis. A single numeric measure for the PR curve is the *area under the PR curve* (PRC). Often, a large change in the number of false positives can lead to a small change in the false positive rate which is used in ROC analysis. On the other hand, PR analysis typically compares false positives to true positives rather than true negatives, encapsulating the impact of the large number of negatives instances on classification performance. However, a classifier that optimizes the area under the ROC is not guaranteed to optimize the area under the PR curve [8]. The expressions for precision and recall (which is the same as the true positive rate) are as follows:

Table 1: Datasets Positive & Negative Instance Distributions

Instance	JM1 (i/c)	PC1 (i/c)	CM1 (i/c)	MW1 (i/c)	KC1 (i/c)	KC2 (i/c)	KC3 (i/c)	Total
P	470/235	76/53	48/39	31/20	325/271	106/82	43/38	1099/738
N	2393/2210	1031/650	457/277	372/291	1782/1093	414/333	415/264	6864/5118
$P+N$	2863/2445	1107/703	505/316	403/311	2107/1364	520/415	458/302	7963/5856
$\%P$	16.4/9.6	6.9/7.5	9.5/12.3	7.7/6.4	15.4/19.9	20.4/19.8	9.4/12.6	13.8/12.6

Table 2: Classification Performance by Cleansed Dataset

Data	AUC	Data	PRC	Data	KS	Data	FM
JM1	**0.9987**	**JM1**	**0.9956**	**JM1**	**0.9974**	**JM1**	**0.9972**
KC1	0.9977	KC1	0.9923	KC1	0.9763	KC1	0.9739
KC2	0.9922	KC2	0.9774	PC1	0.9607	KC2	0.9471
PC1	0.9915	PC1	0.9650	KC2	0.9532	CM1	0.9350
KC3	0.9865	CM1	0.9595	KC3	0.9521	PC1	0.9301
CM1	0.9837	KC3	0.9580	CM1	0.9487	KC3	0.9244
MW1	0.9767	MW1	0.9266	MW1	0.9428	MW1	0.9040
Avg	0.9897		0.9678		0.9616		0.9445

$$\begin{aligned} Precision &= \frac{\#TP}{\#TP + \#FP} \\ Recall &= \frac{\#TP}{\#TP + \#FN} \end{aligned} \quad (3)$$

Similarly to the PRC curve, the F-Measure (FM) is derived from $recall$ and $precision$. There is a decision threshold parameter required for this metric, and the default decision threshold of 0.5 was used in this work. Further, the expression defining the FM metric has a tunable parameter β used to indicate the relative importance of recall and precision. Typically, one can alter β to place more emphasis on either $recall$ or $precision$. In this study, β = 1.

$$FM = \frac{(1\ +\ \beta^2) \times Recall \times Precision}{\beta^2 \times Recall + Precision}$$

The Kolmogorov-Smirnov significance test [7, 13] measures the maximum difference between the empirical distribution function of the posterior probabilities $p(x)$ of instances in each class. Let $i \in \{positive \mid negative\}$ and $F_i(t) = P(p(x) \le t \mid i),\ 0 \le t \le 1$. The $F_i(t)$ can be estimated by the proportion of class $(positive \mid negative)$ instances $\le t$.

$$F_i(t) = \frac{\#class\ (i)\ instances\ with\ posterior\ probability\ \le t}{\#class\ (i)\ instances}$$

Therefore, the KS statistic is defined as follows:

$$KS = \max_{t \in [0,1]} |F_{positive}(t) - F_{negative}(t)|$$

As the separation between the two distribution functions becomes larger, the distinction between the two classes a classifier has made will also improve. The maximum possible value for the KS is one (representing perfect separation), with a minimum of zero. The KS statistic is a commonly used metric of classifier performance in the credit scoring application domain [13].

4.3 Noise Injection Procedure

This section describes the noise injection procedure employed in our empirical study. Note that attribute noise was injected into all selected instances in the derived datasets, while class noise was only injected into the training instances, so that the class labels for the test instances were left uncorrupted. In order to add attribute noise into the datasets, the level of attribute noise (L^a) and the number of significant attributes to be injected with the noise (N^a) were the parameters considered. In the case of class noise, the level of class noise (L^c) and the percentage of instances with class noise injected that were originally from the positive (fault-prone) class (L^m) were the parameters considered. In this study, the noise injection procedure consisted of both attribute and class noise and therefore all four parameters (L^a, N^a, L^c, L^m) were considered.

4.3.1 Class Noise

The class was injected with noise by swapping the respective class label of some training set instances, e.g., positive $\rightarrow$ negative or negative $\rightarrow$ positive. The number of training set instances injected with class noise was a function of two parameters, L^c and L^m. Five levels of class noise, $L^c \in \{10\%, 20\%, 30\%, 40\%, 50\%\}$, were used. The actual number of training dataset instances to be injected with class noise was calculated as $2 \times L^c \times |P|$, where $|P|$ is the number of positive instances in the dataset. For example, a dataset with 1000 instances, 10% fault-prone modules (100 positive modules), and a L^c of 30% would have a total of 60 instances (2 × 30% × 100) injected with noise.

Further, five percentages of positive instances, $L^m \in \{0\%, 25\%, 50\%, 75\%, 100\%\}$, were corrupted from the positive to the negative class. That is, five different relative proportions of positive → negative versus negative → positive were used in this study. Continuing with the previous example, suppose that the percentage of class noise corrupted from the positive class was $L^m = 75\%$. Then of the 60 noisy training set instances, 45 (or 75% of 60) of these will be from the positive class. The remaining 15 instances with injected noise will be from the negative class. Therefore, a randomly selected group of 45 instances from the 100 positive instances in the initial training dataset will have their respective class labels switched from positive to negative. In a similar fashion, a randomly selected 15 instances from the 900 negative instances in the initial training dataset, will have their class labels changed from negative to positive. Once the noise injection process is completed, the corrupted dataset of this example would have a total of 100 - 45 + 15 = 70 positive and 900 + 45 - 15 = 930 negative instances.

4.3.2 Independent Attribute Noise

Independent attributes (software metrics) were injected with noise at five levels, with $L^a \in \{10\%, 20\%, 30\%, 40\%, 50\%\}$. Noise was first injected into the most significant predictive attribute (based on the two-sample KS significance test - see Section 4.2), creating a total of 35 derived datasets (seven initial datasets and five levels of L^a). The next 35 datasets were obtained by corrupting both the most and second-most significant attributes. This procedure was repeated until the seven most significant attributes ($N^a \in \{1, 2, 3, 4, 5, 7\}$) were corrupted. The results of six attributes injected with noise were excluded from this study because of similarities to the results obtained when $N^a = 7$. A noise level of 10% implied that the values for the selected attributes were corrupted for 10% of the instances. Noise was injected by replacing the selected attribute value with a randomly selected attribute value reflecting an instance of the opposite class. For a given injected noise level, the negative and positive proportions of the instances injected with noise was approximately the same as the negative and positive proportions of the given dataset. For example, if the given dataset had a proportion of 80:20 for negative:positive instances and if 180 instances were injected with noise, then the set of instances to be corrupted with attribute noise would have contained 144 negative and 36 positive instances.

Since we are evaluating the impact of noise on classifier performance, it is sensible to inject noise into attributes that are useful for differentiating between fp and nfp instances. In the case of noise injected into a single attribute, if the chosen attribute was not useful for prediction, then the classifiers can easily circumvent the effects of attribute noise. For the purposes of this study, attribute significance was evaluated using a two-sample KS significance test. Other studies can use other attribute significance algorithms and noise injection methodologies.

4.4 Experimental Design Summary

Ten-fold cross validation was applied to build and test the learning models. Additionally, 10 independent repetitions of each experiment were performed to avoid any bias that may occur during the random selection process. The results reported in this work represent the average of these repetitions. A total of 5,544,000 learning models were built and evaluated from 11 learners × 6 (total number of significant independent attributes) × 5 (levels of attribute noise) × 7 datasets × 100 (10 runs of 10-fold CV) × 24 (levels of class noise and percentages of minority instances with noise[3]).

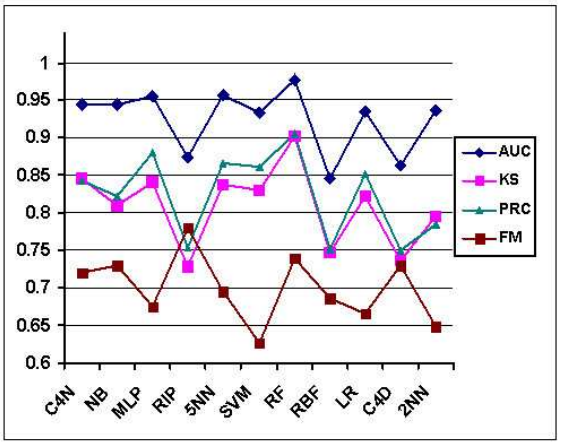

Figure 1: Overall Noise Impact on Learners

5 Learning performance

This section is organized as follows: Section 5.1 illustrates the overall impact on learning performance across all levels of noise and all datasets; Section 5.2 presents the results of the analysis of learner performance aggregated by increasing levels of injected noise; Section 5.3 presents the results of the analysis grouped by the percentage of the minority class injected with noise; Section 5.4 tabulates the impact on learner performance as the number of attributes with injected noise increases; Section 5.5 illustrates with figures the cross-effect of the increasing levels of noise and the percentage of positive instances injected with noise on learning performance; and Section 5.6 contains the analysis of

[3]The class noise case excluded (one out of 25 possible combinations) was for the 50% level of noise to be injected into 100% of the minority class. This case would cause all minority instances to be relabeled, leaving no minority class instances in the training data.

variance models (ANOVA) which statistically corroborated all the observations made from the analysis of the results.

5.1 Overall Classifier Performance

Figure 1 illustrates the impact of both class and attribute noise across all datasets on each learner. The average value obtained by the learners for each of the four performance metrics is provided. According to the AUC, KS, and PRC metrics, the learner with the best and most robust performance is RF, closely followed by 5NN and MLP. RIP, however, is the best performing learner as measured by FM, followed closely by RF, C4D, and NB. The worst performing learners are RIP, RBF, and C4D according to the AUC, KS, and PRC metrics. In contrast, the FM metric shows 2NN and SVM as the worst performing learners. Notice that SVM (in particular) and 2NN obtained above average performance according to the AUC, KS, and PRC metrics. In summary, the performances of eight of the learners (C4N, MLP, RIP, 5NN, SVM, LR, C4D, 2NN) according to the FM are significantly different from those obtained by the AUC, KS, and PRC metrics. These differences emphasize the importance of using an appropriate performance metric when evaluating model performance. The performance metric should most closely match the intended use of the learner during post-development deployment.

5.2 Impact of Increasing Levels of Noise on Learning

Table 3 presents the impact of increasing noise levels (10%, 20%, 30%, 40%, 50%) on learner performance across all datasets and all levels of L^m and N^a. More specifically, Table 3 considers the performance of the learners built from training datasets with L^a equal to L^c, and averaged over all datasets and all levels of L^m and N^a. 10% noise, for example, considers the datasets with both 10% attribute noise (L^a) and 10% class noise (L^c). Regardless of the metric used, the peformance of all learners generally decreases as the level of noise increases. At the highest level of noise (50%), some learners exhibited a slight increase in performance. This is due to the fact that the scenario with 50% class noise and 100% of the injected noise coming from the positive class could not be implemented. In this case, there would be no instances left in the minority class. This same effect can be observed in Table 4 as well.

In Table 3, the highest value in each row is bolded. The row labeled 'Avg' contains the average of the five levels of noise for each learner. The average value of the best performing learner is underlined if it is significantly better, at the 95% confidence level, than the value from the second best learner (Tables 4 and 5 also have these enhancements). The AUC, KS, and PRC metrics agree that RF is the best and most robust learner. On the other hand, RIP is the most robust learner at lower levels of noise as measured by the the FM metric, while NB is the best performing learner at higher levels of noise. In addition, RIP has the highest average FM, while RF has the second best.

5.3 Impact of Minority Class Noise on Learning

The impact of the percentage of instances from the positive class injected with class noise across all datasets and all levels of noise is presented in Table 4. This table presents the impact of minority class noise on the classification performance of each learner. Once again, according to the AUC, KS, and PRC metrics, RF is the most robust learner relative to increasing levels of L^m. Furthermore, the average performances of these three metrics (AUC, KS, PRC) showed RF as the best and most robust learner.

The FM metric, on the other hand, shows RIP as the top performing learner for L^m = 0%, 25%, and 50%. When L^m = 75% or 100%, the top performing learner was NB. However, the learner with the best averaged performance was RIP, followed by RF. Incidently, none of the learners showed an increase in FM when 100% of the positive instances had noise. The results based on the FM metric continue to be quite different from the results obtained using the AUC, KS, and PRC metrics.

5.4 Impact of the Number of Significant Attributes with Noise

Table 5 illustrates the impact of increasing the number of attributes with noise on learner performance. For each learner, the performance is averaged over all datasets and all values of L^m, L^c, and L^a. Once again according to the same three metrics (AUC, KS, and PRC), RF is the best performing learner. Relative to the FM metric, RIP is the most robust learner, while RF is the second best performing learner. Clearly, the selection of a learning technique can be dramatically influenced by the choice of performance metric used for measuring the results. It is also apparent that the learning performance of all learners drop as the number of significant attributes with injected noise increased. This is particularly noticeable when N^a was 5 or 7. For the datasets used in this study, 5 attributes represent approximately 39% of all the attributes.

5.5 Impact of Noise Levels and Percentage of Noise Injected into Minority Instances

Figures 2 to 9 illustrate with line plots the impact of the factor interaction between the overall noise levels and the percentage of noisy instances injected into the positive class L^m, across all datasets. Only the best and worst performing learners are presented for each respective performance metric due to space limitations.

Figures 2 and 3 illustrate the impact of the cross-effect between the overall noise and L^m as measured by the AUC metric. RF was the best performing learner relative to this

Table 3: Impact of Increasing Noise Levels on Learning

Met	n-%	C4N	NB	MLP	RIP	5NN	SVM	RF	RBF	LR	C4D	2NN
AUC	10	0.9728	0.9702	0.9758	0.9425	0.9808	0.9891	**0.9939**	0.9124	0.9674	0.9320	0.9708
	20	0.9651	0.9584	0.9712	0.9279	0.9742	0.9804	**0.9906**	0.8782	0.9543	0.9017	0.9581
	30	0.9465	0.9458	0.9578	0.8785	0.9645	0.9505	**0.9846**	0.8461	0.9385	0.8644	0.9414
	40	0.9209	0.9281	0.9349	0.8095	0.9478	0.8842	**0.9704**	0.8106	0.9077	0.8105	0.9201
	50	0.9178	0.9209	0.9399	0.8139	0.9210	0.8627	**0.9510**	0.7828	0.9066	0.8093	0.8899
	Avg	0.9446	0.9447	0.9559	0.8745	0.9577	0.9334	**0.9781**	0.8460	0.9349	0.8636	0.9359
KS	10	0.9175	0.8638	0.8880	0.8827	0.9100	0.9314	**0.9557**	0.8354	0.8801	0.8733	0.8778
	20	0.8994	0.8333	0.8736	0.8474	0.8844	0.9069	**0.9404**	0.7863	0.8525	0.8314	0.8389
	30	0.8423	0.8083	0.8420	0.7320	0.8495	0.8483	**0.9179**	0.7416	0.8246	0.7411	0.7965
	40	0.7789	0.7804	0.7964	0.5902	0.8018	0.7419	**0.8784**	0.6928	0.7769	0.6149	0.7546
	50	0.7911	0.7670	0.8041	0.5903	0.7477	0.7231	**0.8187**	0.6827	0.7773	0.6210	0.7102
	Avg	0.8458	0.8106	0.8408	0.7285	0.8387	0.8303	**0.9022**	0.7477	0.8223	0.7363	0.7956
PRC	10	0.9155	0.8608	0.9251	0.8754	0.9407	0.9554	**0.9728**	0.8353	0.9085	0.8655	0.9062
	20	0.8953	0.8408	0.9123	0.8496	0.9207	0.9338	**0.9584**	0.7917	0.8838	0.8291	0.8618
	30	0.8444	0.8233	0.8828	0.7618	0.8892	0.8806	**0.9316**	0.7498	0.8545	0.7585	0.8015
	40	0.7847	0.8018	0.8382	0.6412	0.8334	0.7794	**0.8779**	0.7007	0.8063	0.6507	0.7259
	50	0.7725	0.7869	**0.8403**	0.6397	0.7489	0.7601	0.7911	0.6781	0.8037	0.6499	0.6284
	Avg	0.8425	0.8227	0.8797	0.7536	0.8665	0.8618	**0.9063**	0.7511	0.8513	0.7507	0.7848
FM	10	0.8838	0.7799	0.8495	0.9051	0.8669	0.8218	**0.9208**	0.8323	0.8280	0.8918	0.8291
	20	0.8376	0.7452	0.8018	**0.8846**	0.8048	0.7139	0.8564	0.7660	0.7454	0.8506	0.7375
	30	0.7120	0.7211	0.6694	**0.7982**	0.6939	0.6088	0.7389	0.6732	0.6503	0.7240	0.6410
	40	0.5753	**0.7006**	0.5225	0.6596	0.5555	0.4938	0.5891	0.5749	0.5516	0.5806	0.5313
	50	0.5956	**0.7024**	0.5318	0.6557	0.5603	0.4938	0.5908	0.5837	0.5514	0.6014	0.5025
	Avg	0.7209	0.7298	0.6750	**0.7807**	0.6963	0.6264	0.7392	0.6860	0.6654	0.7297	0.6483

Table 4: Impact of Minority Class Noise, L^m

Met	L^m	C4N	NB	MLP	RIP	5NN	SVM	RF	RBF	LR	C4D	2NN
AUC	0%	0.9658	0.9796	0.9805	0.9448	0.9703	0.9890	**0.9909**	0.9348	0.9801	0.9442	0.9570
	25%	0.9580	0.9603	0.9718	0.9384	0.9645	0.9844	**0.9844**	0.8841	0.9602	0.9114	0.9410
	50%	0.9487	0.9395	0.9614	0.9020	0.9532	0.9486	**0.9756**	0.8215	0.9377	0.8641	0.9246
	75%	0.9360	0.9192	0.9354	0.8021	0.9418	0.8663	**0.9650**	0.7735	0.9033	0.7952	0.9171
	100%	0.9136	0.9257	0.9223	0.7777	0.9679	0.8825	**0.9805**	0.8246	0.8900	0.8014	0.9526
	Avg	0.9444	0.9449	0.9543	0.8730	0.9595	0.9342	**0.9793**	0.8477	0.9342	0.8632	0.9384
KS	0%	0.9054	0.8988	0.8970	0.8859	0.8742	0.9341	**0.9358**	0.8794	0.9113	0.8888	0.8527
	25%	0.8783	0.8366	0.8723	0.8693	0.8545	**0.9212**	0.9141	0.8059	0.8630	0.8444	0.8020
	50%	0.8607	0.7909	0.8474	0.7736	0.8207	0.8497	**0.8888**	0.7221	0.8203	0.7639	0.7723
	75%	0.8214	0.7558	0.7996	0.5731	0.8040	0.7230	**0.8648**	0.6568	0.7671	0.6037	0.7581
	100%	0.7565	0.7716	0.7746	0.5281	0.8631	0.7237	**0.9299**	0.6724	0.7428	0.5709	0.8136
	Avg	0.8445	0.8107	0.8382	0.7260	0.8433	0.8303	**0.9067**	0.7473	0.8209	0.7343	0.7997
PRC	0%	0.9065	0.8790	0.9325	0.8745	0.9164	0.9552	**0.9613**	0.8575	0.9388	0.8606	0.8547
	25%	0.8757	0.8417	0.9112	0.8618	0.8931	0.9440	**0.9277**	0.8017	0.8952	0.8348	0.7990
	50%	0.8395	0.8103	**0.8874**	0.7961	0.8507	0.8816	0.8851	0.7309	0.8494	0.7795	0.7452
	75%	0.8115	0.7848	0.8414	0.6312	0.8073	0.7636	**0.8501**	0.6722	0.7936	0.6492	0.7248
	100%	0.7808	0.8005	0.8180	0.5952	0.8944	0.7660	**0.9367**	0.6970	0.7736	0.6246	0.8430
	Avg	0.8428	0.8233	0.8781	0.7518	0.8724	0.8621	**0.9122**	0.7519	0.8501	0.7497	0.7933
FM	0%	0.8550	0.8223	0.8640	**0.9053**	0.8501	0.8881	0.8810	0.8552	0.8766	0.8952	0.7337
	25%	0.8354	0.7583	0.8229	**0.8925**	0.8344	0.7720	0.8746	0.8050	0.7954	0.8677	0.7062
	50%	0.7740	0.7133	0.7041	**0.8337**	0.7322	0.5876	0.7816	0.6960	0.6490	0.7800	0.6546
	75%	0.5940	**0.6793**	0.6185	0.6499	0.5574	0.4474	0.6051	0.5504	0.5067	0.5757	0.5871
	100%	0.5336	**0.6693**	0.4605	0.6136	0.4939	0.4229	0.5443	0.5083	0.4861	0.5120	0.5743
	Avg	0.7184	0.7285	0.6940	**0.7790**	0.6936	0.6236	0.7373	0.6830	0.6627	0.7261	0.6512

Table 5: Noise Impact by Number of Significant Attributes, N^a

Met	N^a	C4N	NB	MLP	RIP	5NN	SVM	RF	RBF	LR	C4D	2NN
AUC	1	0.9558	0.9672	0.9685	0.9029	0.9727	0.9408	**0.9840**	0.8543	0.9334	0.8926	0.9594
	2	0.9547	0.9627	0.9674	0.8949	0.9694	0.9419	**0.9844**	0.8543	0.9410	0.8867	0.9541
	3	0.9497	0.9563	0.9649	0.8824	0.9635	0.9426	**0.9816**	0.8543	0.9448	0.8717	0.9433
	4	0.9493	0.9493	0.9635	0.8795	0.9593	0.9420	**0.9823**	0.8528	0.9431	0.8707	0.9358
	5	0.9375	0.9380	0.9592	0.8682	0.9533	0.9375	**0.9748**	0.8478	0.9418	0.8515	0.9264
	7	0.9273	0.9006	0.9160	0.8341	0.9370	0.9107	**0.9682**	0.8286	0.9124	0.8218	0.9082
	Avg	0.9457	0.9456	0.9566	0.8770	0.9592	0.9359	**0.9792**	0.8487	0.9361	0.8658	0.9379
KS	1	0.8771	0.8700	0.8745	0.7863	0.8725	0.8576	**0.9251**	0.7697	0.8279	0.7887	0.8493
	2	0.8721	0.8556	0.8704	0.7680	0.8622	0.8556	**0.9254**	0.7676	0.8393	0.7784	0.8303
	3	0.8557	0.8340	0.8614	0.7426	0.8523	0.8503	**0.9140**	0.7612	0.8453	0.7483	0.8046
	4	0.8549	0.8150	0.8558	0.7383	0.8448	0.8456	**0.9133**	0.7570	0.8401	0.7469	0.7931
	5	0.8303	0.7898	0.8447	0.7192	0.8326	0.8270	**0.8917**	0.7441	0.8278	0.7174	0.7765
	7	0.7987	0.7097	0.7473	0.6511	0.7904	0.7673	**0.8648**	0.7031	0.7644	0.6672	0.7413
	Avg	0.8481	0.8124	0.8423	0.7343	0.8425	0.8339	**0.9057**	0.7504	0.8242	0.7412	0.7992
PRC	1	0.8730	0.8958	0.9105	0.8103	0.9007	0.8870	**0.9266**	0.7824	0.8619	0.8023	0.8342
	2	0.8725	0.8787	0.9067	0.7953	0.8929	0.8845	**0.9290**	0.7746	0.8710	0.7944	0.8224
	3	0.8537	0.8547	0.8982	0.7654	0.8831	0.8806	**0.9186**	0.7654	0.8751	0.7614	0.8008
	4	0.8527	0.8228	0.8946	0.7615	0.8749	0.8771	**0.9195**	0.7583	0.8689	0.7598	0.7908
	5	0.8359	0.7926	0.8855	0.7476	0.8641	0.8609	**0.9045**	0.7409	0.8581	0.7385	0.7731
	7	0.7844	0.7006	0.7928	0.6697	0.8130	0.8018	**0.8686**	0.7033	0.7850	0.6732	0.7264
	Avg	0.8454	0.8242	0.8814	0.7583	0.8715	0.8653	**0.9111**	0.7542	0.8533	0.7549	0.7913
FM	1	0.7578	0.7855	0.7055	**0.8287**	0.7289	0.6478	0.7664	0.7156	0.6891	0.7743	0.6675
	2	0.7538	0.7730	0.7032	**0.8158**	0.7241	0.6466	0.7651	0.7080	0.6912	0.7684	0.6669
	3	0.7316	0.7555	0.6962	**0.7907**	0.7147	0.6424	0.7526	0.6997	0.6880	0.7402	0.6630
	4	0.7290	0.7320	0.6920	**0.7874**	0.7057	0.6401	0.7514	0.6926	0.6857	0.7375	0.6608
	5	0.7173	0.7063	0.6847	**0.7782**	0.6986	0.6311	0.7398	0.6785	0.6706	0.7238	0.6480
	7	0.6671	0.6334	0.6041	**0.7145**	0.6396	0.5735	0.6968	0.6473	0.5959	0.6661	0.6201
	Avg	0.7261	0.7310	0.6810	**0.7859**	0.7019	0.6302	0.7454	0.6903	0.6701	0.7351	0.6544

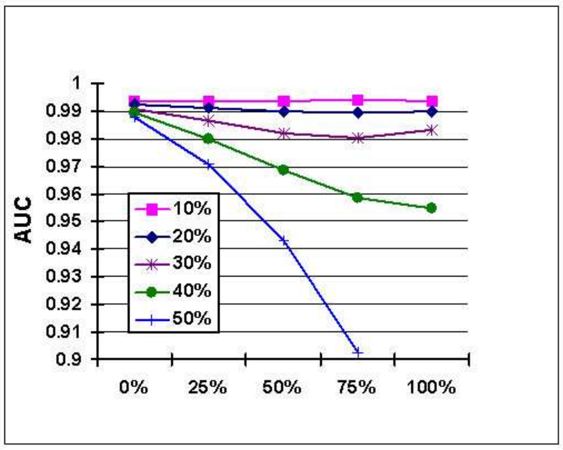

Figure 2: Cross-effect of Noise Level and L^m on RF by AUC

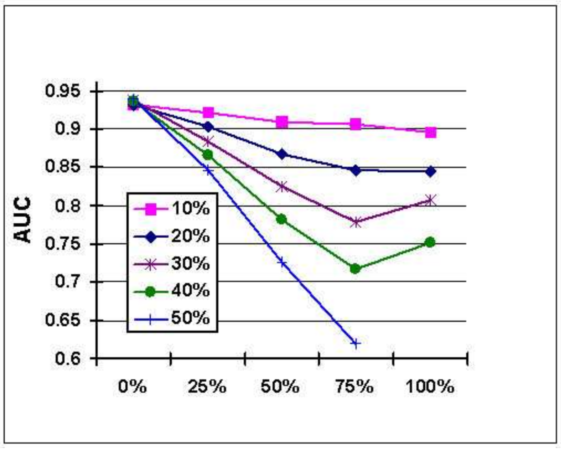

Figure 3: Cross-effect of Noise Level and L^m on RBF by AUC

noise combination. The impact on RF was only at the highest level of noise (50%) and when L^m was more than 50%. On the other hand, RBF was the learner most affected by this factor interaction. Even at the lowest level of noise of 10% and with L^m = 50% or more, the performance of RBF declined substantially.

Figures 4 and 5 illustrate the impact of the factor interaction between the overall noise and L^m as measured by the KS metric. For the RF learner, which was the best performing and most robust learner relative to the KS, significant deterioration was only seen at the 40% level of noise or higher, and when L^m was more than 50%. In contrast, RIP was the learner most affected by this interaction. The performance of RIP was affected little at the lowest level of noise of 10%. However, for all other noise levels and when L^m was 25% or more, the performance of RIP declined dramatically.

Figures 6 and 7 illustrate the impact of the factor interac-

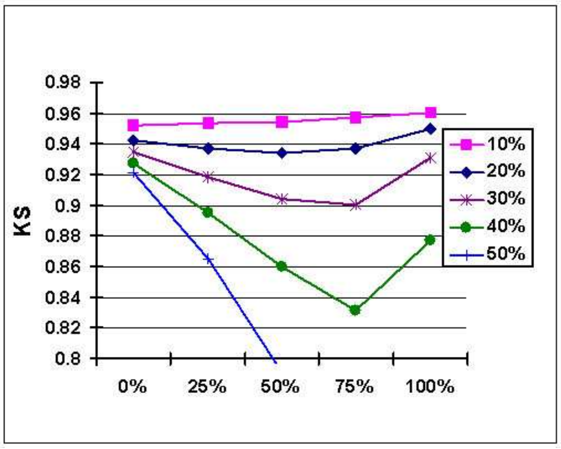

Figure 4: Cross-effect of Noise Level and L^m on RF by KS

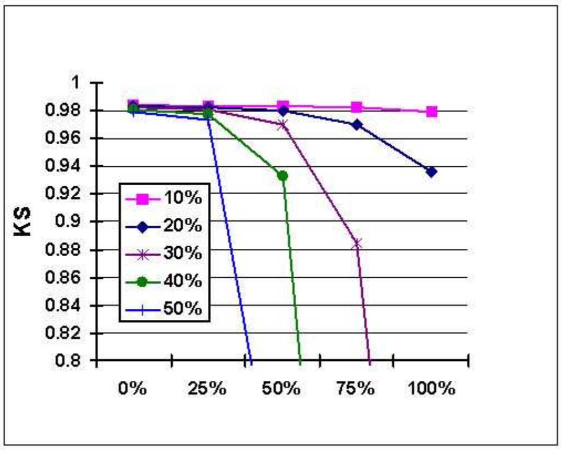

Figure 5: Cross-effect of Noise Level and L^m on RIP by KS

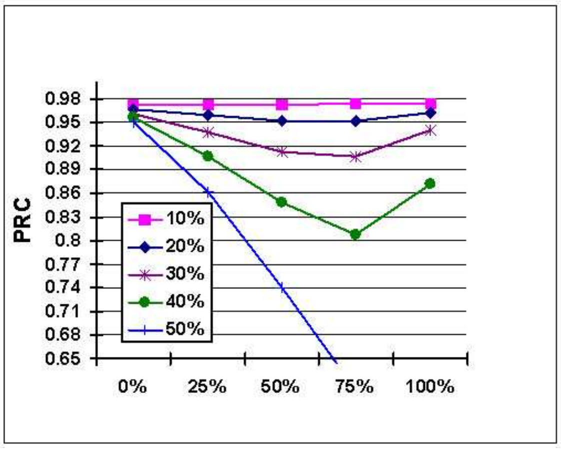

Figure 6: Cross-effect of Noise Level and L^m on RF by PRC

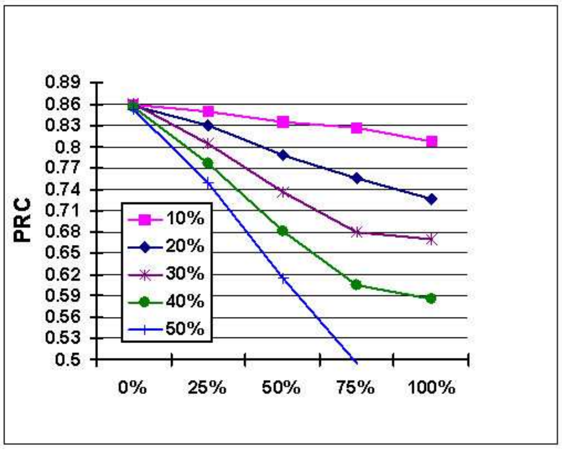

Figure 7: Cross-effect of Noise Level and L^m on RBF by PRC

tion between the overall noise and L^m as measured by the PRC metric. The best performing and most robust learner to this type of noise was RF. On the other hand, RBF was the learner most affected by this factor interaction. Even at the lowest level of noise of 10% and with L^m = 50% or more, RBF's performance declined very significantly.

Figures 8 and 9 illustrate the impact of the cross-effect between the overall noise and L^m as measured by the FM metric. The best performing and most robust learner was RIP. Nevertheless, the performance of RIP was noticeably affected when the noise level was 30% or more and L^m was 50% or more. On the other hand, SVM was the learner most affected by this factor interaction. Even at the lowest level of noise and when L^m was 25% or more, SVM's performance declined very significantly.

These figures clearly illustrate the very significant deterioration of learning performance when both the levels of noise and the percent of positive instances with noise increased. In real-life scenarios, this observation would imply that noise, and, in particular, the amount of noise in the minority class specifically, are critically important factors determining the ultimate reliability and value of any classification undertaking.

5.6 Analysis of Variance (ANOVA)

The results presented in this study are also tested and validated for statistical significance at the α = 5% level using five factor analysis of variance [2] models. An ANOVA model can be used to test the hypothesis that the classification performances of each level of the main factors are equal against the alternative hypothesis that at least one is different. Note that in this study, only the main factor representing the classification techniques (learners) and the corresponding cross-effects with the other factors were investigated in detail. The five factor models [2] used in this work can be represented as follows:

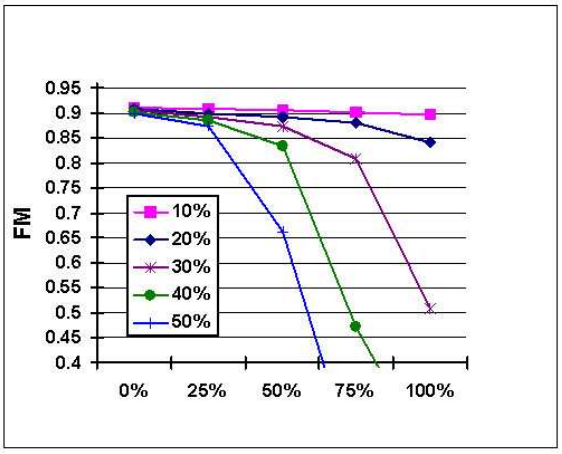

Figure 8: Cross-effect of Noise Level and L^m on RIP by FM

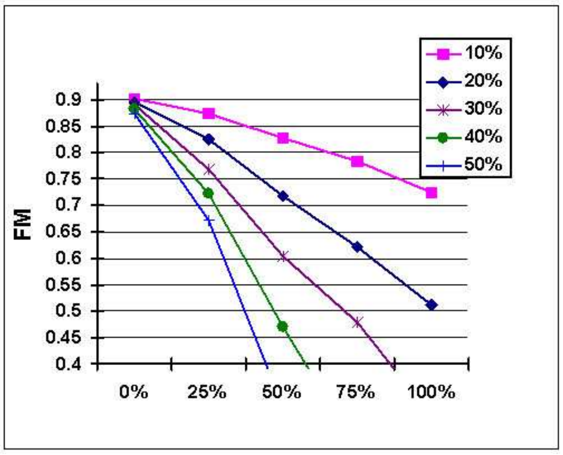

Figure 9: Cross-effect of Noise Level and L^m on SVM by FM

$$\psi_{(jklmin)} = \mu + \mathcal{C}_j + L^c_k + L^m_l + L^a_m + N^a_i + \varepsilon_{(jklmin)}$$

where the parameters of the model are defined as:

- $\psi_{(jklmin)}$ is the response variable (AUC, KS, PRC, or FM) for the n^{th} observation of the j^{th} level of $\mathcal{C}$, k^{th} level of L^c, l^{th} level of L^m, m^{th} level of L^a, and i^{th} level of N^a.

- μ is the overall mean effect on the observations of each response variable.

- $\mathcal{C}_j$ (resp., L^c_k, L^m_l, L^a_m, N^a_i) is the mean performance of level j (resp., k, l, m, i) for factor $\mathcal{C}$ (resp., L^c, L^m, L^a, N^a).

- $\varepsilon_{(jklmin)}$ is the random error.

The main factor (the learner) is tested to see if the average performance of the 11 levels (classifiers/learners) of $\mathcal{C}$ and the corresponding interactions with the five levels of L^c, the five levels of L^m, the five levels of L^a, and the six groups of N^a are equal, respectively. Only the most significant two or three-way interactions were reported in this article. An example of a two-way factor interaction can be $\mathcal{C} \times L^m_{(jl)}$, which represents the interaction of the learners and the L^m factor. To determine which response variable (AUC, KS, PRC, or FM) values are significantly different, a pair-wise comparison of each response variable with the null hypothesis that they are equal (i.e., not significantly different) can be used. In this study, we apply the Tukey's Honestly Significant Difference (HSD) test to identify which levels of the main factor are significantly different [2].

The ANOVA models were built using the AUC, KS, PRC, and FM metrics as the respective response variables. The Fisher's distribution values (F-values) of the factor and the most significant multi-way interactions were tabulated with their corresponding p-values in Table 6. This table shows the significance of the experimental factor and its interactions by the respective p-values. The main factor and the two or three-way interaction terms are statistically significant at $\alpha = 5\%$ with p-values $<< 0.001$, which are denoted < 0.0001. We report the most significant two or three-way factor interactions.

In Table 6, the learners (or $\mathcal{C}$ factor) had the highest F-values as measured by all four metrics. This agrees with our previous observation made in the analysis that the choice of learner, particularly in the presence of low quality class imbalanced data, can be critical to classification performance. The most significant multi-way factor interaction was statistically determined to be the learners and the number of instances injected with noise from the minority class, $\mathcal{C} \times L^m$. This observation indicates that on average, the performance of the classification techniques investigated in this study were most affected when the positive instances contained noise. This is true regardless of the performance metric used in this work. The second most significant factor interaction was with the levels of class noise, $\mathcal{C} \times L^c$. This indicates that on average the level of class noise can also significantly impact the performance of the learning techniques. Furthermore, the three-way factor interaction of the learners, the percentage of positive instances with noise, and the level of class noise ($\mathcal{C} \times L^c \times L^m$) also shows a significant impact on learning performance. Once again, all four performance metrics concurred on this result.

On the other hand, and according to all four metrics, the interaction with the level of attribute noise ($\mathcal{C} \times L^a$) was relatively less significant compared to the other two-way cross-effect presented in Table 6. This also corroborates previous observations made elsewhere by our group that attribute noise is not nearly as significant as class noise to classification performance.

Table 7 provides the mean values of each metric as well

Table 6: ANOVA Models

		AUC		KS		PRC		FM	
Factor	DoF	F-val	p-val	F-val	p-val	F-val	p-val	F-val	p-val
$\mathcal{C}$	10	33528.0	<0.0001	18082.2	<0.0001	16223.8	<0.0001	13660.7	<0.0001
$\mathcal{C} \times N^a$	50	201.6	<0.0001	198.8	<0.0001	204.6	<0.0001	142.4	<0.0001
$\mathcal{C} \times L^a$	40	39.4	<0.0001	57.6	<0.0001	108.1	<0.0001	76.3	<0.0001
$\mathcal{C} \times L^c$	40	1191.3	<0.0001	1438.3	<0.0001	1027.2	<0.0001	1582.9	<0.0001
$\mathcal{C} \times L^m$	40	2967.4	<0.0001	3830.0	<0.0001	1838.0	<0.0001	3300.6	<0.0001
$\mathcal{C} \times N^a \times L^a$	200	9.3	<0.0001	8.2	<0.0001	9.0	<0.0001	8.2	<0.0001
$\mathcal{C} \times N^a \times L^c$	200	5.1	<0.0001	5.7	<0.0001	4.7	<0.0001	2.6	<0.0001
$\mathcal{C} \times L^m \times N^a$	200	10.5	<0.0001	16.4	<0.0001	11.5	<0.0001	4.1	<0.0001
$\mathcal{C} \times L^c \times L^a$	160	1.1	0.3	2.3	<0.0001	1.2	0.0420	0.6	1.0
$\mathcal{C} \times L^m \times L^a$	160	2.4	<0.0001	4.3	<0.0001	1.4	0.0003	1.8	<0.0001
$\mathcal{C} \times L^c \times L^m$	160	587.2	<0.0001	831.2	<0.0001	440.3	<0.0001	837.1	<0.0001

Table 7: ANOVA Factor: $\mathcal{C}$

AUC			KS			PRC			FM		
$\mathcal{C}$	Mean	HSD	$\mathcal{C}$	Mean	HSD	$\mathcal{C}$	Mean	HSD	$\mathcal{C}$	Mean	HSD
RF	0.9792	A	RF	0.9057	A	RF	0.9111	A	RIP	0.7859	A
5NN	0.9591	B	C4N	0.8481	B	MLP	0.8814	B	RF	0.7454	B
MLP	0.9566	C	5NN	0.8425	C	5NN	0.8715	C	C4D	0.7351	C
C4N	0.9457	D	MLP	0.8423	C	SVM	0.8653	D	NB	0.7310	D
NB	0.9456	D	SVM	0.8339	D	LR	0.8533	E	C4N	0.7261	E
2NN	0.9379	E	LR	0.8242	E	C4N	0.8454	F	5NN	0.7019	F
LR	0.9361	F	NB	0.8124	F	NB	0.8242	G	RBF	0.6903	G
SVM	0.9359	F	2NN	0.7992	G	2NN	0.7913	H	MLP	0.6810	H
RIP	0.8770	G	RBF	0.7504	H	RIP	0.7583	I	LR	0.6701	I
C4D	0.8658	H	C4D	0.7411	I	C4D	0.7549	J	2NN	0.6544	J
RBF	0.8487	I	RIP	0.7343	J	RBF	0.7542	J	SVM	0.6302	K

as the significant HSD grouping levels for each learner. Note that if two or more instances of the factor have the same block letter, then their performances are not significantly different. Table 7 shows the overall impact of noise across all levels of noise and all datasets on the performance of each learner $\mathcal{C}$. According to the AUC, KS, and PRC metrics, RF performs significantly better than all of the other learners. The second best learner varies by metric (5NN for AUC, C4N for KS, and MLP for PRC). In general, RF, 5NN, and MLP perform very well as measured by AUC, KS, and PRC, while C4D, RIP, and RBF are the most affected by noise. The FM metric, however, shows RIP as the most robust learner in group 'A', followed by RF in group 'B'. The two learners most impacted by noise were 2NN and SVM, according to the FM metric. Interestingly, two of the worst performing learners (C4D and RIP) according to the AUC, KS, and PRC are among the top three performing techniques according to FM.

The best performing and most robust classifier regardless of the quality of the data was RF, according to the AUC, KS, and PRC metrics. According to the FM metric, RF was the second best performing learner and was uniquely placed in group 'B', while RIP was the best performing learner. It is noteworthy to emphasize the unmatched robustness of RF in the presence of any type and level of noise injected in these experiments. To summarize, the ANOVA analysis presented in this section has corroborated our previous observations and conclusions regarding the robustness of the RF learner in the presence of low quality and class imbalanced data. Further, the percent of instances injected with noise from the minority class and the level of class noise present in the data, respectively, had the most profound effect on learning. The results from the FM metric are substantially different from the other three metrics (AUC, KS, PRC) and thus reiterate the importance of determining the appropriate metric for measuring learner performance.

6 Conclusion

The objective of this study was to investigate the robustness of a variety of common-used learning algorithms relative to low quality, class imbalanced measurement data. Real-world software measurement data typically contains an imbalanced class distribution, and if erroneous attribute values are also present, the impact on learning would be more significant. Our classification study using 11 different learning techniques and low quality, class imbalanced data can be most helpful to practitioners in many application domains. In order to conduct this investigation, a comprehensive suite of experiments was designed and implemented with the use of seven real world measurement datasets, initially relatively free of noise. A novel noise injection procedure was designed and applied using several domain realistic noise parameters. The results were measured using four distinct performance metrics appropriate for imbalanced data. The level of comprehensiveness achieved in this study can be easily seen by the mere fact that over 5.5 million classification models were built and evaluated during our experimentation.

In general, the results unequivocally demonstrated that the quality of the measurement data (both attribute and class noise) can impact classification performance significantly. All conclusions and observations made in the analysis of this study were statistically verified by constructing analysis of variance (ANOVA) models. The learning technique with the best and most consistent performance in all experiments was the random forest ensemble classification technique. Three (AUC, KS, PRC) of the four performance metrics used in this study concurred on the determination of RF as the best and most robust learner. According to the FM metric, RF was second only to the RIP rule-based learner. On the other hand, those learners most impacted by low quality class imbalanced data were C4D, RIP, and RBF, according to the AUC, KS, and PRC metrics. Curiously, C4D and RIP learners were amoung the top three best performing techniques as measured by FM. The worst performing learners according to the FM metric were SVM and 2NN. These results are also unusual because both learners, SVM (in particular) and 2NN, are known to generate acceptable classification performances even in the presence of low quality class imbalanced data, as reported in this study with the results from the AUC, KS, and PRC metrics.

Regardless of the quality of the measurement data and whether or not the data has significant class imbalance, we are very confident in recommending the random forest ensemble classifier for learning initiatives. To our knowledge, no other related classification study has identified a learning technique significantly robust with consistently excellent performance in the presence of low quality, class imbalanced measurement data using four distinct performance metrics. Future work will consider performance enhancing techniques such as cost-sensitive learning and boosting and will also include additional datasets and learning techniques. Additional noise injection methodologies can also be considered in future work.

References

[1] D. W. Aha. *Lazy learning*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[2] M. L. Berenson, D. M. Levine, and M. Goldstein. *Intermediate Statistical Methods and Applications: A Computer Package Approach*. Prentice-Hall, Inc., 1983.

[3] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[4] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Chapman and Hall/CRC Press, Boca Raton, FL, 1984.

[5] C. E. Brodley and M. A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.

[6] W. W. Cohen. Fast effective rule induction. In *Proc. 12th International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[7] W. Conover. *Practical Nonparametric Statistics*. John Wiley and Sons, NY, 1971.

[8] J. Davis and K. Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of 23rd International Conference on Maachine Learning*, Pittsburgh, PA, 2006.

[9] C. Elkan. The foundations of cost-sensitive learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 239–246, 2001.

[10] Y. Feng, Z. Wu, and Z. Zhou. Enhancing reliability throughout knowledge discovery process. In *6th IEEE International Conference on Data Mining - Reliability Issues in Knowledge Discovery Workshop (RIKD06)*, pages 754–758, 2006.

[11] A. Folleco, T. M. Khoshgoftaar, J. Van Hulse, and L. Bullard. Identifying learners robust to low quality data. In *Proceedings of the IEEE International Conference on Information Reuse and Integration-IRI'08*, pages 190–195, July 2008.

[12] E. Frank, L. Trigg, G. Holmes, and I. Witten. Naive bayes for regression. *Machine Learning*, pages 1–20, 2000.

[13] D. J. Hand. Good practice in retail credit scorecard assessment. *Journal of the Operational Research Society*, 56:1109–1117, 2005.

[14] D. Hosmer and S. Lemeshow. *Applied Logistic Regression*. John Wiley Sons, Inc, 2nd edition, 2000.

[15] T. M. Khoshgoftaar, M. Golawala, and J. Van Hulse. An empirical study of learning from imbalanced data using random forest. In *Proceedings of 19th IEEE International Conference on Tools with Artificial Intelligence*, pages 310–317, Patras, Greece, October 2007.

[16] T. M. Khoshgoftaar, V. Joshi, and N. Seliya. Detecting noisy instances with the ensemble filter: A study in software quality estimation. *International Journal of Software Engineering and Knowledge Engineering*, 16(1):1–24, 2006.

[17] T. M. Khoshgoftaar and N. Seliya. The necessity of assuring quality in software measurement data. In *Proceedings of 10th International Software Metrics*

Symposium, pages 119–130, Chicago, IL, September 2004. IEEE Computer Society.

[18] T. M. Khoshgoftaar, N. Seliya, and K. Gao. Detecting noisy instances with the rule-based classification model. *Intelligent Data Analysis:An International Journal*, 9(4):347–364, 2005.

[19] W. Lee, S. J. Stolfo, and K. W. Mok. A data mining framework for building intrusion detection models. *Proceedings of IEEE Symposium on Security and Privacy*, pages 120–132, May 1999.

[20] R. J. A. Little and D. B. Rubin. *Statistical Analysis with Missing Data*. John Wiley and Sons, Hoboken, NJ, 2nd edition, 2002.

[21] J. Moody and C. J. Darken. Fast learning in networks of locally tuned processing units. *Neural Computation*, 1(2):281–294, 1989.

[22] F. Provost and T. Fawcett. Robust classification for imprecise environments. *Machine Learning*, 42:203–231, 2001.

[23] J. R. Quinlan. *C4.5: Programs For Machine Learning*. Morgan Kaufmann, San Mateo, California, 1993.

[24] B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors. *Advances in Kernel Methods: Support Vector Learning*. MIT Press, Cambridge, Massachusetts, 1999.

[25] J. Van Hulse. Data quality in data mining and machine learning. *Ph.D. Dissertation, Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL USA*, May 2007. Advised by T. M. Khoshgoftaar.

[26] J. Van Hulse and T. M. Khoshgoftaar. Class noise detection using frequent itemsets. *Intelligent Data Analysis: An International Journal*, 10(6):487–507, 2006.

[27] J. Van Hulse, T. M. Khoshgoftaar, and H. Huang. The pairwise attribute noise detection algorithm. *Knowledge and Information Systems Journal, Special Issue on Mining Low Quality Data*, 11(2):171–190, 2007.

[28] J. Van Hulse, T. M. Khoshgoftaar, and A. Napolitano. Skewed class distributions and mislabeled examples. In *Seventh IEEE International Conference on Data Mining - Workshops (ICDMW'07)*, pages 477–482, October 2007.

[29] G. Weiss. Learning with rare cases and small disjuncts. In *Proceedings of the 12th International Conference on Machine Learning*, pages 558–565. Morgan-Kaufmann, 1995.

[30] G. M. Weiss and F. Provost. Learning when training data are costly: the effect of class distribution on tree induction. *Journal of Artificial Intelligence Research*, 19:315–354, 2003.

[31] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, California, 2nd edition, 2005.

[32] X. Zhu and X. Wu. Class noise vs attribute noise: A quantitative study of their impacts. *Artificial Intelligence Review*, 22(3-4):177–210, November 2004.

[33] X. Zhu and X. Wu. Cost-guided class noise handling for effective cost-sensitive learning. In *4th IEEE International Conference on Data Mining (ICDM 2004)*, pages 297–304, November 2004.

[34] X. Zhu, X. Wu, T. M. Khoshgoftaar, and Y. Shi. An empirical study of the noise impact on cost-sensitive learning. In *International Joint Conference on Artificial Intelligence*, pages 1168–1174, 2007.

[35] L. Zhuang and H. Dai. Reducing performance bias for unbalanced text mining. In *6th IEEE International Conference on Data Mining - Reliability Issues in Knowledge Discovery Workshop (RIKD06)*, pages 770–774, 2006.