# Validating Requirements Reviews by Introducing Fault-Type Level Granularity: A Machine Learning Approach

**Conference Paper** · February 2018

DOI: 10.1145/3172871.3172880

**4 authors:**

Maninder Singh
St. Cloud State University
**19** PUBLICATIONS **149** CITATIONS
SEE PROFILE

Vaibhav Anu
Montclair State University
**55** PUBLICATIONS **396** CITATIONS
SEE PROFILE

Gursimran Walia
North Dakota State University
**81** PUBLICATIONS **1,288** CITATIONS
SEE PROFILE

Anurag Goswami
Bennett University
**51** PUBLICATIONS **2,170** CITATIONS
SEE PROFILE

# Validating Requirements Reviews by Introducing Fault-Type Level Granularity: A Machine Learning Approach

Maninder Singh[1], Vaibhav Anu[2]
[1,2]Department of Computer Science
North Dakota State University
Fargo-USA
[1]maninder.singh@ndsu.edu
[2]vaibhav.anu@ndsu.edu

Gursimran S. Walia[3]
[3]Department of Computer Science
North Dakota State University
Fargo-USA
[3]Gursimran.walia@ndsu.edu

Anurag Goswami[4]
[4]Department of Computer Science
Bennett University
Noida-India
Anurag.goswami@bennett.edu.in

## ABSTRACT

Inspections are a proven approach for improving software requirements quality. Owing to the fact that inspectors report both faults and non-faults (i.e., false-positives) in their inspection reports, a major chunk of work falls on the person who is responsible for consolidating the reports received from multiple inspectors. We aim at automation of fault-consolidation step by using supervised machine learning algorithms that can effectively isolate faults from non-faults. Three different inspection studies were conducted in controlled environments to obtain real inspection data from inspectors belonging to both industry and from academic backgrounds. Next, we devised a methodology to separate faults from non-faults by first using ten individual classifiers from five different classification families to categorize different fault-types (e.g., omission, incorrectness, and inconsistencies). Based on the individual performance of classifiers for each fault-type, we created targeted ensembles that are suitable for identification of each fault-type. Our analysis showed that our selected ensemble classifiers were able to separate faults from non-faults with very high accuracy (as high as 85-89% for some fault-types), with a notable result being that in some cases, individual classifiers performed better than ensembles. In general, our approach can significantly reduce effort required to isolate faults from false-positives during the fault consolidation step of requirements inspections. Our approach also discusses the percentage possibility of correctly classifying each fault-type.

*ISEC '18,* February 9–11, 2018, Hyderabad, India
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6398-3/18/02…$15.00
https://doi.org/10.1145/3172871.3172880

## CCS Concepts

• **Software and its engineering→ Software creation and management→ Designing software→** Requirement analysis • **Software and its engineering→ Software creation and management→ Software verification and validation→** Empirical software validation**.**

## KEYWORDS

Fault types; Machine learning; Ensemble; Inspection reviews; Supervised learning;

**ACM Reference format:**

M. Singh, V. Anu, G.S. Walia and A. Goswami. 2018. In Proceedings of ACM SIGSOFT Innovations in Software Engineering Conference, Hyderabad, India, February 2018 (ISEC'18), 11 pages.

DOI: 10.1145/3172871.3172880

## 1 INTRODUCTION

In the last decade, there has been an intense research activity to automate software requirement inspections [1]. A fault is a manifestation of missing, incorrect or ambiguous information [3]. A software is desirable that is fault-free to a greater extent during early stages where the cost of finding and fixing is less [2-3]. Many companies employ inspections to remove faults during early stages [4]. Reviews that are generated as part of inspection are susceptible to misinterpretation because of inherently ambiguous nature of Natural Language (NL) documents. Apart from ambiguity, these faults can also be classified based on more refined granularity such as Omitted (O) information, Incorrect Facts (IF) w.r.t actual execution environment, Inconsistent Information (II) across various sections of requirement document, Extraneous (E) information and Miscellaneous (M) [4]. We use the term '***fault types'*** to express faults that belongs to A, O, IF, II, M and E category throughout this paper. The existing techniques that have been applied to classify various reviews (that belonged to code, design and movies etc.) consider categorization into fault vs non-fault parent categories [9-10]. These existing techniques report classification results w.r.t. number of accurately classified fault/non-faults leaving behind the actual fault granularity (O, A, IF, II and M) that are correctly classified. The main contribution of this work, which is based on fault type granularity, is that it provides answers to some important questions such as:

1) What type of faults are most likely to occur in a requirements inspection document?
2) What type of faults are most likely to be correctly classified by supervised learning classifiers (Bayesian, Decision tree, Ensemble etc.)?
3) What type of faults are most misclassified during classification?
4) Which techniques (classification, ensemble or clustering) provide most accurate results in detecting faults?

Considering the fault-type granularity aspect of inspection reviews would ensure estimation of faults types that are still not found during an inspection. Based on estimation results, the requirements author can take post-inspection decision whether or not to re-inspect the requirements document. The automation of reviews over fault type granularity provide additional decision support to inspector as well as saves time that he/she can use fixing the faults rather than re-inspecting the document.

In this paper we present our approach to analyze fault-types using classifiers that belongs to the family of Bayesian, Support Vector, Ensemble, Regression and Trees (Table 1). We trained the classifiers from each classification family to form a model and tested this model against our test set (that contained faults of the type: A, O, IF, II, M, E). Our approach analyzed reviews generated from three different inspections studies [4, 20-21] conducted at NDSU over industrial strength requirements documents. The three NL documents used in these studies are Loan Arranger System (LAS), Restaurant Interactive Menu (RIM) system, and Parking Garage Control System (PGCS). The terms non-faults and false-positives are used interchangeably in the paper.

**Paper Structure:** This paper performs an empirical evaluation on inspection reviews collected during three inspection studies. Section 2 provides background information on review-mining and requirement fault-types. Section 3 describes our proposed solution to the problems stated in introduction. Section 4 describes the experiment design for our technique. Section 5 describes data collection. Section 6 describes results and analysis. Section 7 provide some discussion on results. Section 8 discusses some threats to validity of our study. Section 9 discusses the conclusion and future work of the paper.

Table 1: Type of Classifiers Used

| Classification family | Name of classifiers |
|---|---|
| 1. Bayesian | Naïve Bayes (NB), Multinomial NB and Bernoulli NB |
| 2. Support Vector Classification (SVC) | Linear SVC and NuSVC |
| 3. Ensemble | Random Forest and Extra Tree |
| 4. Regression | Logistic Regression and Stochastic Gradient Descent (SGD) |
| 5. Trees | Decision Trees |

## 2 BACKGROUND

There has been extensive research that has been done to predict number of faults in software engineering that use machine learning (ML) approaches. Many studies include Decision trees [6, 12], Random forest [12], Bayesian classifiers [1,6, 11,13], Linear Regressions [13] and Support vector machines [7]. Many researches that carried out automation of code, design and test reviews through supervised learning have been published [5,14].

Many studies have used ensemble techniques over past few years. The idea of ensemble is to develop a strong classifier from several weak classifiers. Researchers have used voting, bagging, boosting for software fault predictions [18-19]. It has been also proved that ensemble produces more efficient results than individual classifier. Rathore et al. [15] evaluated linear and non-linear heterogeneous ensemble methods using three different base learners. The paper focused on heterogeneous ensemble method as it alleviates possibility of classification bias i.e. making same error repeatedly. Their proposed methods create an ensemble to predict faults through ensemble generation using bootstrapping and meta-training; ensemble integration is performed through integration function that combines the predictions of learning classifiers. Rathore mentioned that they used two linear combination techniques and two non-linear combination techniques to integrate prediction outcome. Msrl et al. [16] presented ensemble methods for faults prediction using Naïve Bayes, voting of features and neural networks. They concluded that ensemble provides better results than any individual classifier.

Sun et al. [17] conducted an empirical investigation to measure inspection fault detection capability of software analysis methods using voting. Sun et al. mentioned that largest number of faults were detected through voting method. Various fault detection

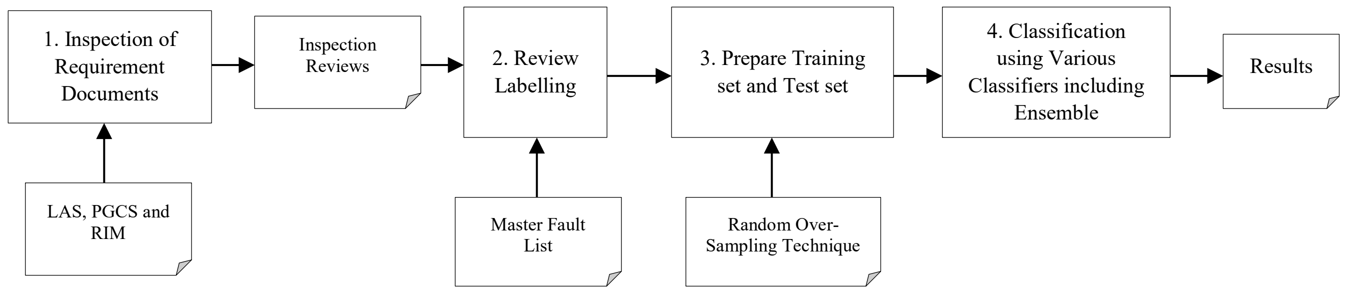

1. Inspection of Requirement Documents
Inspection Reviews
2. Review Labelling
3. Prepare Training set and Test set
4. Classification using Various Classifiers including Ensemble
Results
LAS, PGCS and RIM
Master Fault List
Random Over-Sampling Technique

Figure 1: Overall steps of experiment

methods that were analyzed in their study consisted of self-checks, voting, static analysis, code reading, Fagan inspections and testing. Their findings state that voting predicts most number of faults as well as most variant fault-types. Some inferences that were taken from these studies shows that diverse classifiers make better ensemble than individual classifiers. These evidences showed that various fault-types could be more accurately predicted if ensemble could be developed from classifiers that belonged to different families.

Some studies that are discussed above also mentioned the use of boosting techniques to learn from the results of weak classifiers (by re-adjusting weights assigned to feature sets) and adding them to final strong classifier model. Mustafa et al. [18] and Nguyen et al. [19] discuss the role of Random Over-Sampling (ROS) and Random Under-Sampling (RUS) techniques during classification of imbalanced data sets. Based on their guidelines, we performed analysis of our imbalanced data sets with ROS and RUS. We performed 10-fold cross-validation of training data to analyze the performance for sampling and boosting approaches.

One of our motivation was to perform analysis of various fault-type over ensemble technique. We have performed some analysis of various fault-types like Ambiguous (A), Omission (O), Extraneous etc. in our previous studies from human-error point of view [3, 21]. It was also a motivation to study the impact of classification approaches to automate review classification w.r.t fault-types.

## 3 PROPOSED WORK

In this study, we developed an automated approach to mine inspection reviews with the goal of categorizing between faults vs. non-faults. Next, we analyzed results based on fault types (A, M, O, II, IF and E) across various classifiers used in this study. The current work was conducted at NDSU over the inspection data that was collected in previous three inspections studies. The data in the form of reviews was generated by inspection of Loan Arranger System [21], Restaurant Interactive Menu [20] and Parking Garage Control System [4]. There were in total 77 Inspectors out of which 35 were graduate students, 27 were under-graduate students and 20 were experienced industry people. More details regarding our proposed approach is discussed in this section below. Figure 1 presents an overview of the procedure followed in the current work.

### 3.1 Model Selection

Model selection is a process to select those classifiers to train over training data set that could correctly classify an unseen review from test data set into fault/non-fault. We used supervised learning classification technique to train our models for each classifier. We started with 10 classifiers (refer Table 1) that were used in this study and these classifiers were chosen because of their relative better performance during training run. During our study run, we analyzed each classifier individually over each fault-type. Next, we combined the best classifiers to form ensembles to perform voting to obtain majority outcome; the majority outcome is the final assigned category of the test review, e.g. if out of 9 classifiers, seven (7) classified a review as fault and only two (2) as non-fault then the final outcome of that review is fault (being the majority outcome of voting). The odd number of classifiers was intentionally chosen for ensemble to avoid equal prediction conflict. More details will be provided in Section 3.2 (training and test data sets).

### 3.2 Pre-processing and Formation of Training/Test Sets

In this section, we present details regarding pre-processing of reviews to enable division of data into training and testing set.

*3.2.1 Pre-processing*: The reviews generated are in NL and contained some inconsequential words that contribute very little towards classification (like prepositions, determiners and punctuations etc.). These inconsequential words were removed from the review corpus through NLTK's *stopwords* method. Next, there were multiple reviews that contained words that appears in several inflicted forms (e.g. walk, walks, walking, walked conveys same meaning but are treated differently during text classification because of grammar rules). So, pre-processing of such words was taken care through *lemmatization* in NLTK package. Lemmatization returns the base form of the word i.e. in above example walk, walks, walking and walked are changed to walk. The inspection reviews are treated as Binary class problem (fault vs. non-fault). Once a review is correctly classified as fault with supervised learning classifiers; we further add a fault-type label to the review originally assigned to it by an inspector. Inspection reviews contains faults and non-faults in imbalance ratio i.e. number of faults are comparatively less (minority class) than number of non-faults (majority class). This inequity in ratio generates class-imbalance that results in bias towards majority class. To overcome class-imbalance, some pre-processing is required, and several solutions presented in literature (such as Sampling, Boosting and Ensemble [1, 18, 19]) were tested to counter the imbalanced reviews issue. These processes involved the use of Random Over-Sampling (ROS), Random Under-Sampling (RUS) and AdaBoost.

*3.2.2 Training model*: We split our reviews data into 70%-30% ratio for training and testing purpose. Training of reviews was performed on 70% of total reviews obtained from all three inspection studies. The inspection studies used three different requirements document (i.e. LAS, RIM and PGCS). Training data resulted in number of fault types as listed in Table 2. We selected 70% of each fault-type from each inspection study to include into training set and similarly, remaining 30% from each fault-types and from each study were included in testing data set. This distribution of faults across training and testing set was performed to ensure representation of each fault type (with similar distribution) in both training and test sets. Next, we followed 10-fold cross validation repeated 10 times (and then we took the mean of these 10 repetitions) to estimate the most effective classifiers to build training models. The class imbalance problem in training set was

Table 2: Fault distribution across fault-types and inspection documents

| Fault type | LAS (Industry 20 sub) | PGCS | | RIM (Grad 21 sub) | Total # of fault types | Train (LAS+ PGCS +RIM) | **Test (LAS+ PGCS +RIM )** |
|---|---|---|---|---|---|---|---|
| | | Grad (14 sub) | UG (27 sub) | | | | |
| A | 7 | 12 | 24 | 14 | 57 | 41 | 16 |
| O | 10 | 16 | 29 | 21 | 76 | 53 | 23 |
| IF | 3 | 11 | 30 | 43 | 87 | 60 | 27 |
| II | 17 | 20 | 40 | 32 | 109 | 77 | 32 |
| E | 0 | 6 | 10 | 6 | 22 | 15 | 7 |
| M | 2 | 2 | 2 | 3 | 9 | 5 | 4 |
| | | | | | | | |
| Total | 39 | 67 | 135 | 119 | 360 | 251 | 109 |

removed by adopting Random Over-Sampling (ROS) as shown in Figure 2. More details on cross validation score for sampling and boosting are presented in Section 3.3.

*3.2.3 Test set*: There was one test set that consisted of 457 number of reviews collected from all 3 studies (refer Figure 2). Our test set consisted of at least 30% of reviews from each inspection document. The selection of reviews in test set was random and through automation, we made sure that it contains desired (70%-30%) instances of each fault type in both training and testing (see Table 2). Moreover, the reviews in test set were not over-sampled in order to avoid duplicate misclassification error rate i.e. if a true-fault review is misclassified into non-fault then all its over-sampled instances are also misclassified into non-fault resulting in high misclassification error rate. In a similar way, if a review is correctly classified into true fault category then all its oversampled instances are also classified into true-faults and hence, may affect final results.

## 3.3 Working of Proposed Approach

The working of our approach is discussed in this section. We also discuss the approach through which we analyzed various fault-types after classification.

*3.3.1 Supervised learning classification*

In this section, we discuss various supervised learning classifiers that were used to train each classifier (see Table 1). There were 10 different classifiers from five supervised learning families (Table 1). There cross-validation scores of 10 classifiers taken for this study are shown in Table 3. The classification accuracy is analyzed for AdaBoost, Random Over-Sampling (ROS), Random Under-Sampling (RUS) and without sampling. It was seen that the best cross validation score was received for ROS. Hence, we performed ROS to address the class-imbalance problem with our data. It was also seen that AdaBoost (NuSVC) did not perform better than independent NuSVC classifier. The selection of these classifiers was based on their successful usage in previous studies to mine reviews. Each review in test set belonged to at most one of the fault type (A, O, IF, II, M and E). We tracked each review against each classifier, then counted correctly classified fault and its fault type. Next, each classification outcome was collected and analyzed to answer our research questions (discussed Section 4.1).

*3.3.2 Voting through Ensemble*

Table 3: Cross validation results for model selection

| Classifiers | Without Sampling | Ada Boost | RUS | ROS |
|---|---|---|---|---|
| Naïve Bayes (NB) | 0.773 | 0.754 | 0.657 | 0.785 |
| Multinomial NB | 0.819 | 0.763 | 0.626 | 0.796 |
| Bernoulli NB | 0.817 | 0.822 | 0.604 | 0.809 |
| Decision Tree | 0.770 | 0.826 | 0.546 | 0.837 |
| Linear SVC | 0.824 | 0.850 | 0.576 | 0.852 |
| Random Forest | 0.818 | 0.849 | 0.584 | 0.857 |
| Extra Tree | 0.835 | 0.858 | 0.649 | 0.975 |
| Logistic Regression | 0.825 | 0.775 | 0.608 | 0.835 |
| SGD | 0.783 | 0.783 | 0.561 | 0.855 |
| NuSVC | 0.669 | 0.669 | 0.520 | 0.563 |

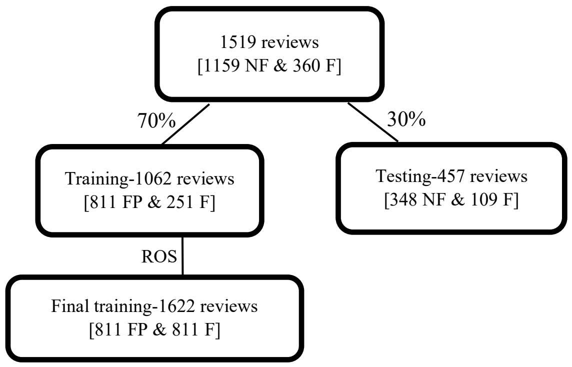


Figure 2: Training and testing set split with ROS of requirement reviews

We selected those classifiers that performed best against each fault-type and combined these classifiers to form ensemble that generate voting to assign final class to each review. We used odd number of classifiers to create ensemble and as mentioned earlier, the number of classifiers is intentionally kept odd to avoid equal prediction conflict. Our motive was to build ensemble based on most accurate classifiers for a fault-type.

## 4 EXPERIMENTAL DESIGN

In this section, we discuss research questions that are investigated in this study, experiment design (that includes experiment methodology for various dependent and independent variables used in this study) and experiment procedure (that includes inspection process along with the working of our proposed approach). We implemented our approach using Python-3.4 32-bit open source package. We additionally used Scikit learn module version-0.19.0 to implement inbuilt classification packages. The whole experiment was run on 64-bit core-i7 processor with 16 GB physical memory.

## 4.1 Experiment Methodology

***Research Questions (RQs)***: There were the following two major research questions that were investigated in this study:

*RQ1: Which supervised learning classifiers, when applied to reviews collected during requirements inspections, are able to categorize individual fault-types into faults vs non-faults with higher accuracy?*

*RQ2: Do ensembles created from the best performing individual classifiers provide improved categorization of reviews into faults vs non-faults?*

To investigate above mentioned research questions, the following dependent and independent variables were measured:

***Independent variables***: Independent variables are those which are manipulated to measure their effect on one or more dependent variables. Following independent variables were manipulated during study run:

a) *Type of Classifiers*: There were total of 10 classifiers used in this study. These classifiers were chosen based on their performance during review-classification as reported by other researchers during our literature review (section 2).

b) *Test Set*: The test split percentage was chosen to be 30% of total reviews that were generated from three different inspection studies conducted in industry and academia. The total number of reviews in test were 457 and Table 2 shows distribution of various fault-types. We made sure to include at least 30% instances of each fault-type in the test set.

c) *Training model*: The percentage split for training model was chosen to be 70% and contained equal percentage of reviews from each of the three inspection studies and each fault-type.

***Dependent Variables***: These are the variables that were observed to measure the effect of independent variables on them.

a) *Accuracy (A)*: It is the measure of true-faults and false-positives reviews correctly detected by classifiers over total number of reviews.

b) *Geometric Mean (G-mean)*: It is an evaluation metric to calculate prediction results of minority class in a given classification problem. It represents geometric mean of precision and recall (refer Table 6). It is measured as square root of the product of precision and recall. In this paper, precision has not been analyzed separately but through G-mean; because we were interested in performing analysis against various classification families to find correct faults out of total true-faults; also, precision (positive predictive value) is important metric that should not be ignored. For this purpose, G-mean was chosen to neutralize the bias if either precision or recall alone is taken as evaluation metric.

$$G\text{-}mean = \sqrt{(precision \times recall)}$$

***Participating Subjects***: Overall, there were in total 77 subjects involved in this study. Table 4 shows more details on participating subjects w.r.t under-graduate, graduate and industry inspectors. Most of these subjects had at least 2 years of experience in software engineering.

Table 4: Participating Subjects

| Inspector experience | # of inspectors |
|---|---|
| Under-Graduate (UG) students | 27 |
| Graduate (Grad) students | 35 |
| Industry personnel | 20 |

***Artifacts***: There were in total three (3) artifacts involved in this study namely LAS, PGCS and RIM (explained in Section 1 and 3). LAS and PGCS artifacts were developed by requirement experts and is used at Microsoft to train their employees over fault checklist technique to carry out inspections [4]. RIM was developed under real project conditions through interaction with real client. These three documents contained seeded faults that were available to authors in the form of master fault list. The inspectors' reviews were checked against master fault lists to label each review as fault or non-fault. Table 5 shows the details regarding these documents.

Table 5: Artifacts Used for Training and Testing Sets

| Artifact | # of seeded (known) faults | # of pages | # of inspectors in this study |
|---|---|---|---|
| LAS | 30 | 11 | 20 (industry personnel) |
| PGCS | 35 | 14 | 41 (27-UG and 14 Grad) |
| RIM | 150 | 21 | 21 (Grad Students) |

## 4.2 Experiment Procedure

The experiment procedure is explained through the following 5 steps as shown in Figure 1.

***Step 1 inspection of requirements document***: This step involved inspection process over three requirements document namely LAS, PGCS and RIM. The explanation for inspection procedure is beyond the scope of this paper but can be found in [4, 20]. The outcome of this step are inspection reviews in the form of natural language (NL) text.

***Step 2 Review Labelling***: The reviews obtained as outcome of step 1 are further labelled into fault and non-fault categories by two authors on this paper. The categorization/labelling was performed by cross-checking each review from each study (LAS, PGCS and RIM) with corresponding master fault list. This step was necessary to transform reviews into machine ready format.

***Step 3 Prepare Training and test set***: Due to class-imbalance problem, we performed Random Over-Sampling (ROS) on the data. The training and test set were split in 70%-30% ratio and it was made sure that both training and test set contains each fault-type in similar ratio (i.e. 70%-30% respectively). The training was performed over each classifier (discussed in Section 3).

***Step 4 (a) Classification Configurations***: We used 10-fold cross validation repeated 10 times during model selection (refer Table 3). We extracted features after removing stopwords along with their occurrence frequencies. NLTK (version 3.2.2), Matplotlib (2.0.0), Numpy (1.12.1) and Scikit-learn package (version 0.19.0) were used to configure the classifiers. The classifiers were built over default parameter setting except SGD (with max_iter=100 and tol=none), Random Forest (with n_estimators=100) and Ensemble (with n_estimators=100).

***Step 4 (b) Classification results***: Our proposed approach was applied over test set using individual classifiers as well as using ensemble-classifiers. The outcome of this step was classification results in the form of accuracy and G-mean scores.

***Step 5 Results***: The results were analyzed using correctly classified fault-type and number of classifiers (Ensemble) to be used against a particular fault type that could provide more concrete results.

Table 6: Data Collection & Evaluation Metrics

| Metric | **Description** |
|---|---|
| $T_r$ | Total # of reviews fed to the model |
| $C_{tf}$ | True fault count in $T_r$ |
| $C_{fp}$ | False positive count in $T_r$ |
| TP | # of reviews classified as True-faults by model |
| TN | # of reviews classified as False-positives by model |
| FP | # of non-faults reviews classified as faults by model |
| FN | # of fault reviews classified as non-fault by model |
| $F_i$ | It is an array that stores total # of each fault-types |
| $F_{acc}$ | An array that stores correctly classified # for each fault-type |
| Acc | # of correctly identified TP (or TN) over $C_{tf}$ (or $C_{fp}$) |
| Precision | TP/(TP+FP) |
| Recall | TP/(TP+FN) |
| G-mean | $\sqrt{(precision)}x(recall)$ |

## 5 DATA COLLECTION

This section discusses the variables that are used to calculate evaluation metrics (i.e. accuracy and G-mean). Table 6 describes these metrics.

## 6 RESULTS AND ANALYSIS

This section report results regarding classification accuracy of individual classifiers that are taken under consideration (Table 1). The analysis is performed around two research questions discussed in Section 4. Our analysis discusses various classifiers accuracy in predicting each fault-type as well as analysis of ensemble using most accurate classifiers to test each fault-types. The results are organized around the following two research questions:

### 6.1 RQ-1: Fault-type vs. Individual Classifiers

This research question was focused on investigating the performance of individual classifier against each fault-type described in this paper. The fault distribution around three documents is shown in Table 2. We trained 10 different classifiers (Table 1) over 70% of inspection reviews obtained from three different studies (LAS, PGCS and RIM). The evaluation accuracy of trained models was tested over remaining 30% reviews (457 reviews in total). The evaluation was performed using Accuracy percentage and G-mean (definition in Table 6). The results are shown in Figure 3, y-axis (vertical axis) represent Accuracy in percentage and x-axis (horizontal axis) shows various classifiers that are used to perform the analysis. The results for each fault-type is discussed below:

***6.1.1 Fault-type A (Ambiguity)***: From Figure 3 (1), the classifiers are arranged in descending order of their accuracy results. There

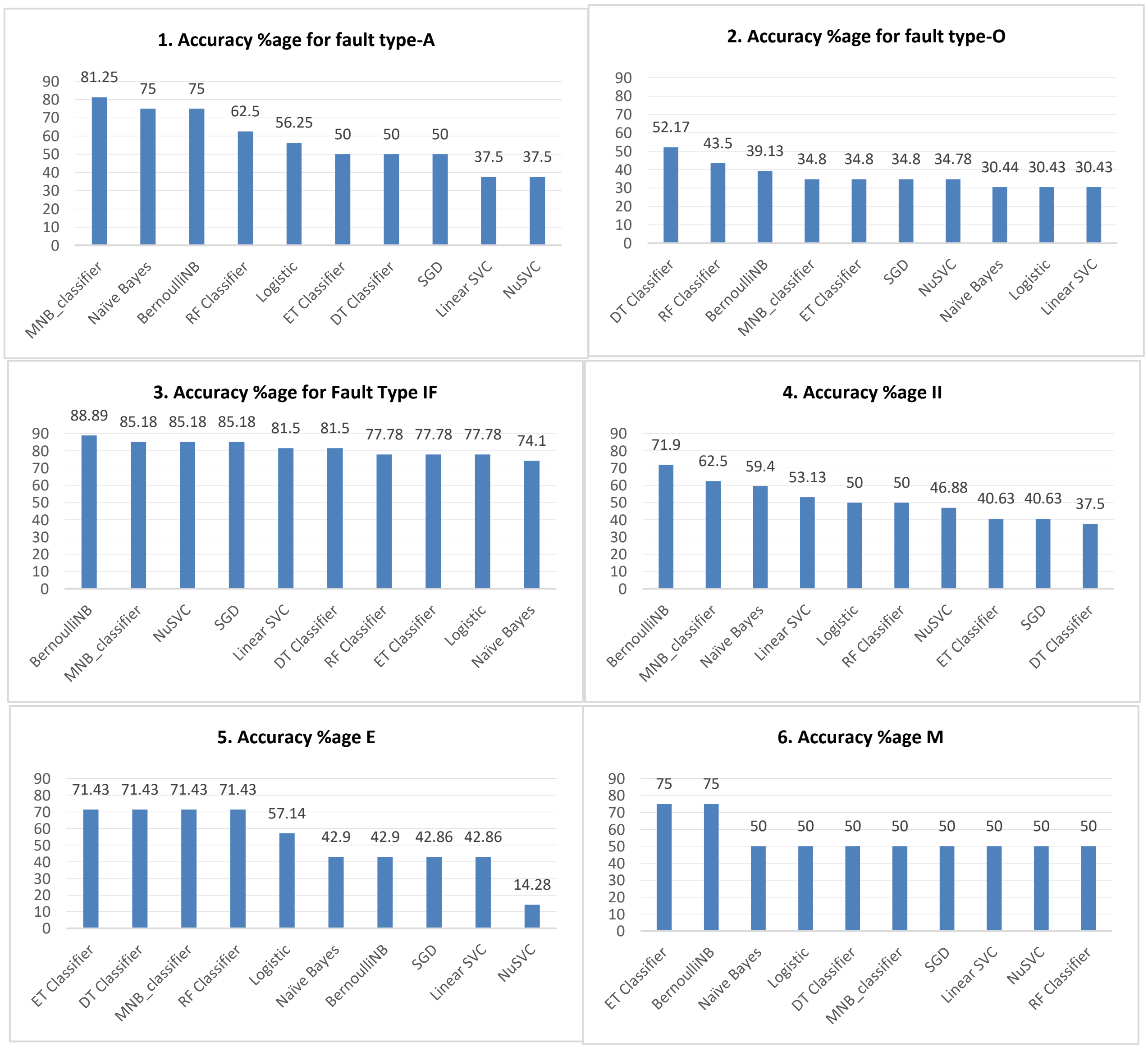


Figure 3: Results of fault type versus individual classifiers

are total 16 faults in test set that represent fault-type ambiguity (Table 2, Test column). The accuracy varied from 81.25% (13 out of 16) to 37.5% (6 out of 16) for different classifiers. It is seen that only three (out of ten) classifiers performed well compared to the other classifiers. These classifiers are Multinomial naïve Bayes (MNB), Naïve Bayes (NB) and Bernoulli Naïve Bayes (BNB). It is also observed that the best performance in predicting fault-type A (Ambiguity) was shown by Bayesian family classifiers only.

***6.1.2 Fault-type O (Omission)***: Figure 3 (2) shows omission fault-types that are 23 in total present in test set (Table 2). The accuracy for omission fault type varies from 52.17% (12 out of 23) to 30.43% (7 out of 23). The only classifier that showed highest prediction accuracy against fault-type O was shown by Decision Tree (DT) classifier that belonged to Trees family of classifiers. The fault-type O is the least accurately predicted fault-type.

***6.1.3 Fault-type IF (Incorrect Facts):*** Figure 3 (3) shows fault-type IF that are 27 in number. The classification accuracy was observed highest by Bernoulli NB classifier (90%) and went down to 74.1% (by Naïve Bayes). Overall, all classifiers performed well to predict fault-type IF as three classifiers (MNB, Nu Support Vector Classifier (NuSVC), and Stochastic Gradient Descent (SGD) showed 85.18% accuracy and five classifiers showed accuracy between 77% to 82%. So, almost all classifiers predicted IF fault with a significantly high accuracy.

***6.1.4 Fault-type II (Inconsistent Information):*** Figure 3 (4) in shows that the accuracy varies from 71.9% (highest) to 37.5% (lowest) for fault-type II. There were 32 faults in total that belonged to fault-type II. Results show that all three classifiers from Bayesian family are the top performers.

***6.1.5 Fault-type E (Extraneous):*** There were 7 total Fault-type E in test set. There are four classifiers that equally performed and ranked top with 71.43% accuracy (Figure 3 (5)). These four classifiers are Extra Tree (ET) Classifier, Decision Tree (DT) Classifier, MNB and Random Forest (RF) Classifier. Interestingly, every classifier from Trees and Ensemble family performed well against fault-type E.

***6.1.6 Fault-type M (Miscellaneous)***: Figure 3 (6), shows prediction results corresponding to fault-type miscellaneous (M). There were 4 faults in total and two classifiers Extra Tree (ET) and Bernoulli NB (BNB) were able to predict 75% (3 out of 4) accurately. Rest all classifiers were able to predict at most 50% of true faults.

Table 7: Classifiers that qualified for Ensemble

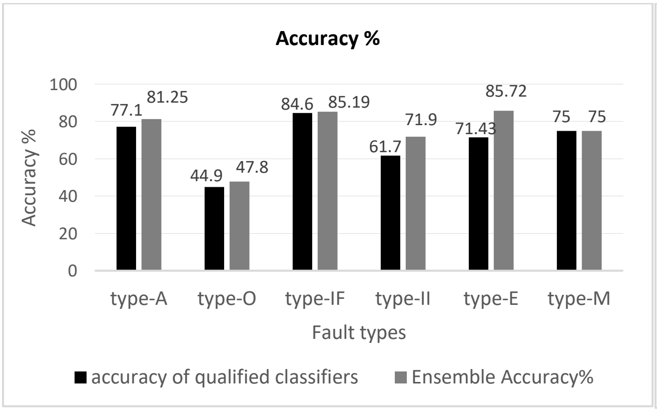
| Fault type | Mean of 10 classifiers | # and Name of classifiers that qualified for ensemble | Mean Acc. % of qualified classifiers for ensemble |
|---|---|---|---|
| Type-A | 57.5 | 3 (NB, MNB and BNB) | 77.1 |
| Type-O | 36.53 | 3 (DT, RF and BNB) | 44.9 |
| Type-IF | 81.48 | 6 (BNB, MNB, SGD, NuSVC, Linear SVC and DT) | 84.6 |
| Type-II | 51.26 | 4 (BNB, NB, MNB and Linear SVC) | 61.7 |
| Type-E | 52.87 | 4 (ET, DT, RF and MNB) | 71.43 |
| Type-M | 55 | 2 (BNB and ET) | 75 |

## 6.2 RQ-2 Fault-type vs. Ensemble

This research question is focused on investigating improvement in each fault-type when the *best-performing* individual classifiers are combined to create an ensemble that classify test review through voting. The selection of classifiers for ensemble is done on the basis of results that are shown for RQ-1 in Section 6.1. For each fault-type, the selection of classifiers for ensemble was almost different and was on the basis of merit (see Figure 3). For our binary classification problem; it is required that the outcome be classified into either fault or non-fault. Hence, to avoid equal prediction conflict using ensemble, we selected odd number of classifiers. The merit of classifiers was based on accuracy percentage (i.e. include those classifiers that are above accuracy mean of all classifiers for each fault-type); for example, fault-type A (Figure 3(1)) has a mean of 57.5% for all the classifiers (refer Table 7) and only first three classifiers are above this mean value.

In fault-type-A example, the number of classifiers happens to be odd but if it were even then this even number tie would have been resolved based on overall G-mean (Figure 5) score of classifiers (classifier with greater G-mean score is included for each even tie). G-mean is also considered to evaluate improvement (in accuracy if any) of ensemble over individual classifier (Figure 4). The following observations are noted for each ensemble that is created for each fault-type:

***6.2.1 Ensemble for fault-type A***: From Table 7, three classifiers that qualify for ensemble are NB, MNB and BNB (notably, all are from Bayesian family). As the number of classifiers that qualified



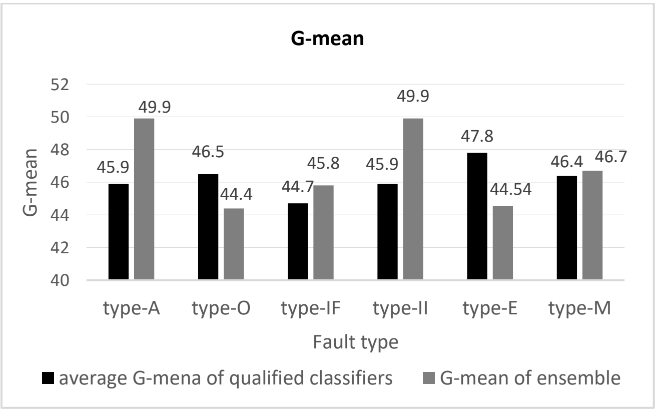


Figure 4: Performance comparison of individual classification versus ensemble

| Classifiers | MNB | NB | BNB | RF | ET | DT | Logistic | SGD | Linear SVC | NuSVC |
|---|---|---|---|---|---|---|---|---|---|---|
| G-mean | 47.7 | 43.3 | 46.7 | 47.8 | 46 | 45.1 | 43.1 | 42.4 | 41.4 | 41.8 |

Figure 5: G-mean values of individual classifiers

is odd, there does not exist equal prediction conflict, so all three of these classifiers are chosen to build ensemble for fault type-A. The improvement with ensemble was seen w.r.t. G-mean score and classification accuracy (refer Figure 4). The results in Figure 4 shows classification accuracy with targeted ensemble along with comparison of improvement in G-mean for each fault-type.

***6.2.2 Ensemble for fault-type O:*** Fault-type O ensemble was developed from three classifiers that qualified and namely DT, RF and BNB. Almost all classifiers performed poor against fault-type O. Decision Trees (DT) that performed best among all could only perform with an accuracy of around 52% (refer Figure 3(2)). The ensemble for fault type O had average accuracy of 44.9% for classifiers that qualified, and this accuracy improved to 47.8% using ensemble but G-mean for ensemble type-O did not improve (Figure 4) making this ensemble unfit for fault type-O.

***6.2.3 Ensemble for fault-type IF:*** The ensemble for fault-type IF was made from 5 out of 6 classifiers that qualified. Linear SVC was removed from inclusion into ensemble because of its lower G-mean value among 6 qualified classifiers making the total number of classifiers odd i.e. 5. There was good improvement in G-mean as well as in accuracy using ensemble. This result showed that use of ensemble for fault-type IF was helpful.

***6.2.4 Ensemble for fault-type II:*** The ensemble consisted of 3 out of 4 qualified classifiers and again for this fault-type, linear SVC was removed because of its lower G-mean value. The 3 classifiers for ensemble were again all surprisingly from Bayesian family. This ensemble for fault-type II showed good improvement; as seen in Figure 4 that average accuracy (of qualified classifiers) versus ensemble accuracy is improved significantly (62% to 72%) along with improvement in G-mean score from 45.9% to 49.9%.

***6.2.5 Ensemble for fault-type E and M:*** The ensemble for fault-type E was formed with the selection of 3 classifiers out of 4 qualified; namely DT, MNB and RF leaving ET because of its lower G-mean value among all. Although ensemble for fault type-E showed improvement in accuracy but fails to show improvement for G-mean (Figure 4). On the other hand, with fault type-M, there were 2 qualified classifiers namely ET and BNB. As the number being even, one classifier had to be removed from consideration leaving behind only one for ensemble. One classifier for ensemble classifies the same way as individual and thus there was no improvement either in accuracy or G-mean.

## 7 DISCUSSIONS OF RESULTS

The major goals of this study are to analyze the prediction accuracy of various classifiers against specific fault-types and to analyze the improvement in accuracy/G-mean through the use of ensemble. For this purpose, we used ten classifiers from 5 supervised learning classification families (refer Table 1). There were in total 6 different fault-types that were being evaluated in this study namely Ambiguity (A), Omission (O), Incorrect Fact (IF), Inconsistent Information (II), Extraneous (E) and Miscellaneous (M). There were total 109 faults (A-16, O-23, IF-27, II-32, E-7 and M-4) refer Table 2 and 348 false-positives (refer Figure 2). A discussion about the results shown in Section 6 is presented here.

### 7.1 RQ-1 Fault-type Vs. Individual Classifier

This research question was focused on determining performance for each fault-type against the ten chosen classifiers belonging to five different classification families. Our belief was that each classification family will perform differently on reviews as each family executes classification problem through different mathematical functions. For example, Bayesian family includes prior and posterior probabilities and Decision Trees (DT) uses entropy/information-gain theory to classify test data. In rest of this section, major observations are discussed initially for each fault-type followed by discussion on implication of results. So, the major observations that are collected for individual classifiers against each fault-type are listed below:

***7.1.1 Fault-type A***: Some major observations from results in Section 6 for ambiguous (A) fault-type are discussed below:

- It is seen that all classifiers from Bayesian family outperformed other classifiers used in this study and classification results of all Bayesian classifiers were very close to each other making them first preference to be used against locating fault-type A.
- Support vector classification family (Linear SVC and NuSVC) performed poorest making them unfit to use against ambiguous fault-types.

**Implications**: We observed that the reason for Bayesian classifiers to outperform other classifiers was because of description of ambiguous fault types. Bayesian classifiers works with the use of mathematical functions on probability that output most likely classification class for the review under consideration. We observed that various specific feature terms like 'what', 'when', 'meant by' etc. occurs frequently for ambiguous reviews that makes high probability for a test review to be classified as fault if it contains such features. This gives us useful insight into ways to train inspectors to write quality description of inspection review that could improve classification results.

***7.1.2 Fault–type O***: This was the least correctly-identified fault-type by any classifier used in this study. Automated classification technique could find at most 12 omission (O) faults out of twenty-three (23). Major observations for this fault-type are as follows:

- Only DT classifier could predict highest number of omission faults (12 out of 23).
- Mean accuracy of all the classifiers against omission fault-type is 36.5% (this is worst performance of classifiers among all fault-types in this study).

***Implications***: There is fair possibility that omission type faults are responsible for overall degradation of automated classification results. Omission faults (23 in count) for this study that included 109 total true-faults are responsible for decrease of more than 10% overall accuracy. We observed our test and training data set for omission type and found that the features that mostly described omission (e.g. 'not' and 'no') are usually removed from analysis using NLTK's stopwords (see Section 3). Our next enhancement of this technique is going to be careful investigation and solution to retain some valuable features while throwing away unwanted stopwords from text. These features may be considered redundant information but for our scenario these features provide much more value to fault-types.

***7.1.3 Fault-type IF***: Incorrect Facts (IF) were the only fault-types that were predicted with good accuracy by all classifiers under consideration in this study. Major observations for fault-type IF are as follows:

- Highest accuracy was 89% (24 correctly classified out of 27) while least accuracy was 74%.
- Almost every classification family performed well in predicting correct classification class for test review. The average accuracy of all classifiers fort fault-type IF is found to be 81.5%.

**Implication**: While taking a closer look at our test and training data, we found that Incorrect Facts (IF) were the easiest ones to locate because these consisted of many mismatched facts/variable-values across requirements document that were reported by almost every inspector. So, there were mostly similar description of IF fault-type present in both training and test reviews. So, all classifiers performed well during testing because they learned the same mathematical values and features during training.

***7.1.4 Fault-type II***: Inconsistent Information (II) fault-type were highest in count in true-faults (77 in training and 32 in test set; for more details refer Table 2 for fault distribution). The major observations are discussed below:

- Bayesian classifiers outperformed for fault-type II with Bernoulli NB at top with highest accuracy of 72%.
- Mean accuracy (51.3%) of fault-type II is not as high as IF (81.5%) although they report semantically similar faults.

**Implications**: Again, the behavior of classifiers was cross-checked with inspection reviews collected during study and it was found that majority of reviews that reported fault-type II contained some one-character variable names specific to each requirements document that were important to distinguish fault vs. non-fault but were removed as part of stopwords. Having them not removed from analysis makes predictions even more poor, so that is why we had no other choice but to remove them. There is strong implication to use some specific terminology while describing a fault review in natural language e.g. 'do not occur' can be written as 'missing' or one-character keywords could be changed to some more meaningful variables such that these are not removed from analysis. This is our next step to investigate and implement/develop proper taxonomy to describe fault reviews.

***7.1.5 Fault-type E and M***: These fault-types being least in number are discussed together in this sub-section. There were in total eleven (7-E and 4-M) reviews in test set that represented these fault-types. Major observations are as follows:

- The best performance for fault-type E was shown by ET, DT, MNB and RF classifiers while for fault-type M, ET and BNB performed highest.

**Implications**: Fault-type extraneous (E) was reported using few common features across training and testing set. From the results it is seen that DT, ET and RF classifiers performs well and could report significant count. Fault-type M contained very few reviews (4 in total) and it is very hard to perform analysis on few counts. So, some more concreate classification confidence is required to explain classification outcome.

## 7.2 RQ-2 Fault-type Vs. Ensemble

The purpose of this RQ was to investigate the performance of ensemble classifiers in categorizing a review into fault vs non-fault. The ensemble was created based on top individual classifiers that performed better than other classifiers for each fault-type (refer Section 6 for more details). The discussion of ensemble approach in this section is presented individually over each fault-types followed by implications of result.

***7.2.1 Ensemble for Fault-type A***: Ensemble for fault-type A consisted of three classifiers namely NB, MNB and BNB all from Bayesian family (see Figure 6). Some major observations found are discussed as follows:

a) All ensemble classifiers that qualified belonged to Bayesian family.
b) Ensemble showed improvement in G-mean.
c) The faults are categorized only if 2 out of 3 ensemble classifiers labelled it fault; making the classification confidence of 67% (2 out of 3).

**Implications**: For fault-type A, ensemble with three classifiers is better than one. The classification results (faults/non-faults) are more reliable than individual classifier. Ensemble for fault-type A also performed better in predicting other fault-types i.e. IF and II.

| Fault Type | A (16) | O (23) | IF (27) | II (32) | E (7) | M (4) |
|---|---|---|---|---|---|---|
| # of classified faults | 13 | 9 | 24 | 23 | 1 | 2 |
| Accuracy % | 81.25 | 39.13 | 88.9 | 71.9 | 14.3 | 50 |
| Ensemble g-mean | 49.9 | | | | | |

Figure 6: Ensemble for fault type A (NB + MNB + BNB)

***7.2.2 Ensemble for Fault-type O***: Ensemble for fault-type O was built over three classifiers namely DT, RF and BNB (Figure 7). There were total 23 omission faults and following are some major findings:

a) The ensemble did not perform better than individual classifiers; in fact, it performed worst of all.
b) G-mean value did not show any improvement.
c) The ensemble could not perform better for any fault-type.

**Implications**: Fault-type O are hard to locate through individual classifiers as well as through ensemble. It opens a new research investigation to develop taxonomy to describe omission faults in such a way that are effectively learned by classifiers.

| Fault Type | A (16) | O (23) | IF (27) | II (32) | E (7) | M (4) |
|---|---|---|---|---|---|---|
| # of classified faults | 9 | 11 | 20 | 13 | 4 | 2 |
| Accuracy % | 56.25 | 47.8 | 74.1 | 40.63 | 57.14 | 50 |
| Ensemble g-mean | 44.4 | | | | | |

Figure 7: Ensemble for fault type O (DT+RF+BNB)

***7.2.3 Ensemble for Fault-type IF***: Incorrect Fact (IF) type fault was most accurately and easily located in inspection reviews. There were 27 total faults of type-IF and ensemble using five classifiers was able to locate 23 faults. The following are some observations from Figure 8:

a) The ensemble predicted good number (85%) of IF faults.
b) There has been improvement in G-mean score of ensemble.

**Implications**: It has been observed that ensemble for fault-type A performs slightly better with one more accurately classified fault than ensemble for fault-type IF. The G-mean value of ensemble A is also better than ensemble IF. Lower G-mean for ensemble IF is reflected because of its under-performance over fault-type A, O and II than ensemble for fault-type A.

| Fault Type | A (16) | O (23) | IF (27) | II (32) | E (7) | M (4) |
|---|---|---|---|---|---|---|
| # of classified faults | 11 | 8 | 23 | 17 | 4 | 2 |
| Accuracy % | 68.75 | 34.8 | 85.19 | 53.125 | 57.14 | 50 |
| Ensemble g-mean | 45.8 | | | | | |

Figure 8: Ensemble for fault type IF (DT+MNB+BNB+SGD+NuSVC)

**7.2.4 Ensemble for fault-type II**: It was seen that for fault-type II, Bayesian classifiers qualified for ensemble and performed best among all other classifier (refer Figure 9). Ensemble showed improvement in G-mean while accuracy remained same as that of individual classifier (BNB). For fault-type II, Bayesian ensemble categorized reviews into fault vs. non-fault with at least 67% classification confidence.

| Fault Type | A (16) | O (23) | IF (27) | II (32) | E (7) | M (4) |
|---|---|---|---|---|---|---|
| # of classified faults | 13 | 9 | 24 | 23 | 1 | 2 |
| Accuracy % | 81.25 | 39.13 | 88.9 | 71.9 | 14.3 | 50 |
| Ensemble g-mean | 49.9 | | | | | |

Figure 9: Ensemble for fault type II (NB + MNB + BNB)

**7.2.5 Ensemble for fault-type E**: This fault-type consisted of very few count and ensemble with DT, MNB and RF could label these fault-types more accurately (85.72%). The ensemble for E also performed good over fault-type IF (74%) but was not as accurate as Bayesian family classifiers that gives accuracy of 89% for fault-type IF (see Figure 10).

| Fault Type | A (16) | O (23) | IF (27) | II (32) | E (7) | M (4) |
|---|---|---|---|---|---|---|
| # of classified faults | 9 | 10 | 20 | 12 | 6 | 2 |
| Accuracy % | 56.25 | 43.5 | 74.1 | 37.5 | 85.72 | 50 |
| Ensemble g-mean | 44.54 | | | | | |

Figure 10: Ensemble for fault type E (DT+MNB+RF)

***7.2.1 Ensemble for Fault-type M***: Ensemble for M consisted of just one classifier i.e. Bernoulli Naïve Bayes (BNB). The accuracy% of this ensemble is no different than accuracy of individual BNB classifier (see Figure 11). The only concern in using one classifier for ensemble M is; *one*, it is not ensemble because to make an ensemble there should be more than one classifiers; *two*, it does not predict with classification confidence i.e. no voting is possible. So, we believe fault-type M is better found by individual BNB classifier.

| Fault Type | A (16) | O (23) | IF (27) | II (32) | E (7) | M (4) |
|---|---|---|---|---|---|---|
| # of classified faults | 12 | 9 | 24 | 23 | 3 | 3 |
| Accuracy % | 75 | 39.13 | 88.89 | 71.9 | 42.9 | 75 |
| Ensemble g-mean | 46.7 | | | | | |

Figure 11: Ensemble for fault type M (BNB)

## 8 THREATS TO VALIDITY

There were few threats to validity which were known to us that we removed. The reviews are written in NL that inherit scope to spelling mistakes. We analyzed each fault/non-fault to correct any spelling mistakes. We split our data using 70%-30% split ratio into training and testing sets respectively but there were few cases in which only 2 reviews corresponding to a fault-type were available and we had to include 1 in training and 1 in testing (refer Table 2 for more details). Also, conclusions are hard to draw and generalize for reviews per fault-type that are very few. The reviews were split randomly into training and testing set to avoid any researcher bias. We also made sure to include consistent number of reviews from all three documents into training and test sets in this study. The 70-30 percentage split was applied to each fault-type from three documents to be included in training and testing sets such that our training model and testing sets are free from bias/inconsistency. The class-imbalance problem was tackled through Random oversampling of fault reviews. We tried to include all those classifiers that have been tested against mining of reviews.

## 9 CONCLUSION AND FUTURE SCOPE

This paper has presented and evaluated an automated method for verifying the reviews obtained during requirements inspections (i.e., separating faults from non-faults). Our methodology relies on the language used by inspectors for different fault types (e.g., Ambiguities, inconsistencies, and incorrectness). Our results showed that supervised learning classifiers through ensemble were able to successfully predict faults of different fault-types with a significantly high accuracy (up to 89% in some cases). Use of ensemble techniques was prolific for fault-type A, IF, II and E while individual classifiers performed well for fault-type O and M. Fault-type E and M are hard to report while fault-type O, E and M are hard to generalize using classification. Misclassification error rate is higher for fault-type O and M. Our results also indicate that overall, classifiers from the Bayesian classification family render more accurate predictions when identifying faults. Furthermore, our evaluations with ensemble classifiers indicated that for certain fault-types (e.g. omission (O) and Miscellaneous (M)) individual classifiers are better than using ensembles and for other faults (e.g. Ambiguity, Incorrect Facts and Inconsistent Information) ensemble yield better prediction accuracies as well as improved G-mean. Post-inspection, requirements author can decide whether to re-inspect requirements document based on number of fault-types obtained or initial estimations on still not found faults (that is our future work) from automation process.

Our future work includes performing more inspection studies in both industrial and academic settings to gather exhaustive fault-data that can be used to train the classifiers. In future, we will also evaluate the influence of parts of speech (i.e., verb, noun, adjectives) used by inspectors in their reviews on the accuracy with which classifiers can predict faults of different fault-types. It is very undesirable to lose any fault due to misclassification error, so using part of speech tags, we will evaluate reviews using ensemble to assign a ranking or priority. Doing so will ensure that none fault has been slipped.

## REFERENCES

[1] Yang, Cheng-Zen, et al. "An empirical study on improving severity prediction of defect reports using feature selection." Software Engineering Conference (APSEC), 2012 19th Asia-Pacific. Vol. 1. IEEE, 2012.

[2] J. Moeyersoms, E. Junque de Fortuny, K. Dejaeger, B. Baesens and D. Martens, "Comprehensive software fault and effor prediction: A

data mining approach," *The journal of Systems and Software,* pp. 80-90, 2015.I.S.

[3] G. S. walia and J. C. Carver, "Evaluating the Use of Requirement Error Abstraction and Classification Method for Preventing Errors during Artifact Creation: A Feasibility Study," in *International symposium on software reliability engineering*, San Jose, USA, 2010.

[4] A. Goswami and G. S. Walia, "Teaching Software Requirements Inspections to Software Engineering Students through Practical Training and Reflection," in *Proceedings of the 123rd American Society of Engineering Education Conference on Computer Education ASEE 2016*, New Orleans, Louisiana,USA, 2016.

[5] A. Bacchelli and C. Bird, "Expectations, Outcomes and Challenges of Modern Code Review," in *International Conference of Software Engineering* , San Francisco, USA, 2013

[6] A. Bosu, M. Greiler and C. Bird, "Characteristics of Useful Code Reviews: An Empirical Study at Microsoft," in *12th Working Conference on Mining Software Repositories*, Florence, Italy, 2015.

[7] C. Francis, N. Pepper, H. Strong, "Using support vector machines to detect medical fraud and abuse", *Engineering in Medicine and Biology Society EMBC 2011 Annual International Conference of the IEEE*, pp. 8291-8294, 2011

[8] K. K. Marri, "models for evaluating review effectiveness," in *3rd Annual International Software Testing Conference*, India, 2001.

[9] I. L. Margarido, J. P. Faria, R. M. Vidal and M. Vieira, "Classification of defect types in requirement specification: Literature review, proposal and assessment," in *6th Iberian Conference on Informamtion Systems and Technologies (CISTI 2011)*, Chaves, Portugal, 2011.

[10] Weiss, Gary M. "Mining with rarity: a unifying framework." ACM Sigkdd Explorations Newsletter 6.1 (2004): 7-19.

[11] N. Chen, J. Lin, S. Hoi and X. Z. Xiao, "AR-Miner: Mining Informative Reviews for Developers from Mobile App Marketplace," in *ICSE 2014: 36th International Conference on Software Engineering*, Hyderabad, India.

[12] M. Halkidi, D. Spinellis, G. Tsatsaronis, M. Vazirgiannis, Data mining in software engineering, Intelligent Data Analysis 15 (2011) 413–441.

[13] Shu-Hsien, L., Pei-Hui, C., & Pei-Yuan, H. (2012). Data mining techniques and applications – a decade review from 2000 to 2011. Expert Systems with Applications, 39(12), 11303–11311.

[14] L. V. G. Carreno and K. Winbladh, "Analysis of User Comments: An Approach for Software Requirements Evolution," in *ICSE 2013: International Conference of Software Engineering*, San Francisco, USA.

[15] Rathore, Santosh Singh, and Sandeep Kumar. "Linear and non-linear heterogeneous ensemble methods to predict the number of faults in software systems." Knowledge-Based Systems 119 (2017): 232-256.

[16] Mısırlı, Ayşe Tosun, Ayşe Başar Bener, and Burak Turhan. "An industrial case study of classifier ensembles for locating software defects." Software Quality Journal 19.3 (2011): 515-536.

[17] So, Sun Sup, et al. "An empirical evaluation of six methods to detect faults in software." Software Testing, Verification and Reliability 12.3 (2002): 155-171.

[18] Mustafa, Ghulam, et al. "Solving the class imbalance problems using RUSMultiBoost ensemble." Information Systems and Technologies (CISTI), 2015 10th Iberian Conference on. IEEE, 2015.

[19] Nguyen, Hien M., Eric W. Cooper, and Katsuari Kamei. "A comparative study on sampling techniques for handling class imbalance in streaming data." Soft Computing and Intelligent Systems (SCIS) and 13th International Symposium on Advanced Intelligent Systems (ISIS), 2012 Joint 6th International Conference on. IEEE, 2012.

[20] Anu, V., Walia, G., and Bradshaw, G. "Incorporating Human Error Education into Software Engineering Courses via Error-based Inspections", 48th ACM Technical Symposium on Computer Science Education, SIGCSE 2017. March 8 – 11, 2017, Seattle, USA.

[21] Goswami, A., Walia, G., and Singh, A. "Using Learning Styles of Software Professionals to Improve their Inspection Team Performance", Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering SEKE 2015. Pages 680-685, July 6- 8, Pittsburgh, USA.

[22] O. Abdelwahab, M. Bhagat, C.J. Lowrance and A. Elmaghraby, "Effect of training set size on SVM and Naïve Bayes for Twitter sentiment analysis, IEEE international symposium on signal processing and information technology, 2015.

[23] Phung, Son Lam, Abdesselam Bouzerdoum, and Giang Hoang Nguyen. "Learning pattern classification tasks with imbalanced data sets." (2009).

[24] Yang, Cheng-Zen, et al. "An empirical study on improving severity prediction of defect reports using feature selection." Software Engineering Conference (APSEC), 2012 19th Asia-Pacific. Vol. 1. IEEE, 2012.

[25] B. Pang, L. Lee, S. Vaithyanathan, "Thumbs up?:Sentiment Classification Using Machine Learning Techniques," in *Proceedings of the ACL conference on Empirical methods in Natural Language Processing*, vol-10, pp 79-86, 2002.

[26] J. Carver, F. Shull, and V. Basili, "Observational studies to accelerate process experience in classroom studies: An evaluation," in Proceedings - 2003 International Symposium on Empirical Software Engineering, ISESE 2003, 2003, pp. 72–79

[27] F. Shull, J. Carver, and G. H. Travassos, "An empirical methodology for introducing software processes," ACM SIGSOFT Softw. Eng. Notes, vol. 26, no. 5, p. 288, 2001.

[28] A. E. Hassan and T. Xie, "Software Intelligence: the future of mining software engineering data," in *ACM FoSER '10: Proceedings of the FSE/SDP workshop on Future of Software Engineering Research*, Santa Fe- New Mexico, USA, 2010.

[29] Y. Zhou, Y. Tong, R. Gu and H. Gall, "Combining Text Mining and Data Mining for Bug Report Classification," in *IEEE International Conference on Software Maintenance and Evolution*, Victoria, BC, Canada, 2014.

[30] G. Bavota, "Mining Unstructured Data in Software Repositories:Current and Future Trends," in *IEEE 23rd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Osaka, Japan, 2016.

[31] S. Mcintosh, Y. Kamej, B. Admas and A. E. Hassan, "The Impact of Code Review Coverage and Code Review Participation on Software Quality," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, Hyderabad, India, 2014.