JURNAL ILMIAH KOMPUTER GRAFIS, Vol.16, No.1, JULI 2023, pp. 218 -247
p-ISSN :1979-0414(print), e-ISSN :2621-6256 (online)
DOI: https://doi.org/10.51903/pixel.v16i1.1885

Characterization Of Composition Error Summary Using Machine Learning Techniques And Natural Language Processing

Mars Caroline Wibowo[1], **Budi Raharjo**[2]
[1]University of Science and Computer Technology (STEKOM University), Semarang, 57166, Indonesia
Email: caroline@stekom.ac.id
[2]University of Science and Computer Technology (STEKOM University), Semarang, 57166, Indonesia
Email: budiraharjo@stekom.ac.id

ARTICLE INFO

Article history :

Received : 23 Mei 2023
Received in revised from : 16 Juni 2023
Accepted : 29 Juni 2023
Available online : 31 Juli 2023

ABSTRACT

As software technology becomes more complex, software maintenance costs become more expensive. In connection with this, the development of software engineering makes the software system has many Composition choices that can be adjusted to the needs of the user. Error fixing involves analyzing Error Summary and modifying code. If bug-fixing steps are made as efficiently and effectively as possible then maintenance costs can be minimal. The purpose of this research is to establish a tool of machine learning for identifying Composition Error Summary and to find out the types of special Composition choices that can be used to save costs, time, and effort. In this study, the T-test was applied to appraise the analytical implication of conduct metrics when the "F-test" was taken to the Variance's test. Classifiers used in this study are "All words" or "AW", "Highly Informative Words" or "H-IW", and "Highly Informative Words plus Bigram" or "H-WB". Identical validation and Vexed validation techniques were used to calculate the effectiveness of machine learning tools. The results of this research denote that the instrument is competent for definitive Composition Error Summary and other Composition choices for definite Error Summary. This research determines the practicality of machine learning techniques in corrective issues relevant to Error summary. The result of this study also explained that Composition/non-CompositionError Summaries have contrasting aspects that can be accomplished by machine learning devices. The advanced tool could be upgraded in some areas to create it more powerful. The array identification section of the current study has limitations, an array with different words and

Composition recognition tools tend to prefer Compositions with more words, so improvements to this could implicate consideration of the semantics of Error Summary, equivalent, and use of n-grams. Also, in using the technology of machine learning and Natural Language processing some advancements to be made to the present characterization structure so for future research it is highly recommended to clear up the first's Error Summary before operating several operations in the present study.

Keywords: *Composition Error Summary, Machine Learning, Natural Language Processing.*

INTRODUCTION

Software maintenance activities include Error Summary and Error fixing. Smith (1982) explained software maintenance costs account for more than 2/3 of the cost of software products. Up to thirty-one percent of errors related to composition occur in commercial software systems and open-source, with the majority of composition caused by errors in composition option settings (Yin et al. 2011). To know the error, developers have to notice the long Error description. Nearly of error summaries in software open source as Firefox platform projects in 2014 were three hundred words or more and these words are considered very long (Rastkar et al. 2014). To specify the relevant Composition choice in reproducing the errors summary, Developers should be aware of effective techniques for tagging the Composition Error Summary and identifying the root causes of the Composition choice. Learn Error Summary using machine learning techniques is gaining popularity and is proving to be competent. A scheme of this study aimed at enhancing Composition-aware techniques is developed, which can help developers easily deError and reproduce errors that require special Composition for exposition. With this proposed framework, Composition diagnosis can be improved. Developers can name Composition Error Summary automatically and on time, as well as be able to extrapolate bug-relevant Composition choices.

A software Error is a wrong, damaged, default, or flop in a computer program that returns incorrect results indeed to smash. As the total and difficulty of software systems go up, so do the item and kind of software error. Great means have been made to classify Error Summary, and identify and improve errors. Software Error Summary is plain text and can include error logs, tread to emulate the error, output, variant, stage, and OS information. To orbit the improvement of error summary, it can mark it as fresh, acknowledged, closed transcribed, and constant. A great error summary should be definite and contain as much information as possible, but there is no effective way to avoid long error messages with lots of irrelevant information. So holding a device that can extract the statistics may be very useful.

Related Research

Dommati et al, (2013) explained that identifying duplicate error summaries using machine learning tools can save time and effort. During a current error detail show up, the information of the original language related to the available highly identical Error Summary is given to the person who marked the Error Summary as duplicate. The study by Kim and Dongsun (2013) proposes a two-stage predictive model using information from the summary of errors as a fix solution, the research uses a bag-of-words N-LP approach to identify features, extracts word tokens from the Summary of Errors, uses machine learning tools as input in training the classifier, and determines the type of software to improve location predictions as desired.

Text mining to error summary to classify security Error Summary, in this study the machine learning model in Error Summary was tagged manually, then a trained model was used to identify faulty security Error Summary and manually tag them as security Error Summary (Gegick et al, 2010). Another study by Murphy and Rastkar (2014) classified 3 text mining named Email Classifier, Email & Meeting Classifier, and Corpus Summary Error Classifier are the most accurate. The study of the error summary is divided into several components including the product name and version number (Sureka, 2012). The results show that there is a correlation between the terms in the Error Summary and the components that could be practiced to correctly predict the Error Summary component. Matter et al, (2009) propose an approach for expert software developers in matching, assigning, and analyze vocabulary in error summaries automatically using a machine learning approach.

Brian et al, (2007) Applying machine learning techniques in definite code errors using the C4-5 decision tree, aims to identify various error conditions based on input and output information. Zimmermann et al, (2007) used logistic regression to train and classify possible file sets. The text mining approach is used to predict the severity of Error

Summary, an evaluation of the open-source software product approach shows that the use of text mining for predictions can achieve moderate to high accuracy (Lamkanf et al, 2010). Turhan et al, (2009) “compared to non-machine learning, a rule-based model that requires inspecting forty-five percent of the source code suggest a machine learning-based model, where some seventy percent of defects can identify by interested only three percent”. This advises that a machine learning (ML) approach is a more practical and efficient way to identify errors.

Compared to previous research activities, this study uses a machine learning approach to quote appropriate information from Error Summary. This research for Composition /non-Composition Error Summary and treasure trove Composition choice is identical with Error Summary, this brings into account the unique characteristics of Error Summary, namely validating the performance of the classifier using an identical approach to Vexed-verification. This study uses Natural Language Processing approach to quote Composition choices that may be related to Composition Error Summary and processes Compositions into word lists by dividing Compositions according to the tokens used to link the words. Using Compositions as documents rather than Error Summary reduces corpus size, and speeds up the process of identifying Compositions. The two-step process developed in the research is the identification of Composition Error Summary, the identity of structure choice, this will be very useful for developers in working on errors in a short time and cost-effectively.

LITERATURE REVIEW

Chowdhury, (2003) describes Natural Language Processing (N-LP) as a field of computer science research developed from artificial intelligence (AI) and computational linguistics. In this study, N-LP was applied to process summary errors including word identification, extraction, deletion of unnecessary vocabulary, tokenization, and word lemmatization (Chowdhury, 2003). To forecast the error types in the Error Summary after N-LP processing, ML operations are continued. N-LP is also used to identify Summary Errors regarding a particular Composition with word tokenization as a basic step (Grefenstette et al, 1994), stopword removal (Wilbur et al, 1992), stemming (Ahmed et al, 2004), part-of-speech tagging (Brill and Eric, 2004), lemmatization (Plisson et al, 2004), and chunking & chinking (Arellano et al, 2015). In this research, these steps are used to convert Error Summary into “word sets” in machine learning (ML) processing. Fukumizu et al, (2004) explain that dimensional reduction reduces the time and storage required. This can improve classifier performance by removing multicollinearity. For more minimize the dimensions, stemming or “lemmatization” is carried out. Because this research used N-LP, the stopwords were taken from the NL-TK corpus packet, and the lemmatization was taken from the N-LP stem. WordNet space.

In text mining, the term feature extraction has the meaning of extracting words as features from the text (Gawade and Trunal, 2016). Information acquisition is the generality level of a word in a certain class, it is used to extract high information features because every piece of information obtained in every word must be counted. (Lee et al, 2006). To assess the similarity of words in class, you can use Chi-sq. the greater rate of a word, also expected word is identical with a class, assuming that there are Summary Errors from the two classes, namely t1 and t2 (Liu et al, 1995). The chi-sq score indicates the probability that the word is identical with the t1 type, projected as:

$$Chi\_sq\_score = \frac{n_{xx} \times (n_{ii} \times n_{oo} - n_{io} \times n_{oi})^2}{(n_{ii} \times n_{io}) \times (n_{ii} \times n_{io}) \times (n_{io} \times n_{oo}) \times (n_{io} \times n_{oo})}$$

The score indicating the probability that *n* is identical with class t2 that could be determined in the same way as the formula above, igram identifies two words that may appear together. Including a bigram increases the likelihood of classifying Error Summary correctly (Tan et al, 2002). The probability of 2 words appearing jointly is projected using chi-sq. Machine learning is interdisciplinary computer science and statistics that grew out of artificial intelligence (Sebastiani & Fabrizio, 2002). This study leverages the use of ML, in general, to correct and control an ever-increasing number of errors. Machine learning tools are used to identify Summary Compositional/noncompositional Errors. In this research machine learning is used to create a classifier from tagged Error Summary to recognize new Error Summary as cohesive Compositions or unrelated Compositions.

There are three types of machine learning based on several previous researchers, the first is supervised machine learning (Kotsiantis et al, 2007), unsupervised machine learning (Gentleman & Carey, 2008) and reinforcement learning (Sutton et al, 1998). Unsupervised learning has no data input tags. The system has to find settings based on input data because this metric is unclear to evaluate classifiers. That the algorithm of ML is usable, including Naïve Bayes (Murphy & Kevin, 2006), Decision Trees (Safavian et al, 1990), and Logistic Regression (Hosmer et al, 2000). In N-LP, Max. degeneration analysis is also available and commonly used (Liu et al, 2005). Apart from these classifiers, there is also a basic classifier that can be created and compared in a simple way, namely ZeroR (Witten et al, 1990).

ZeroR and Naïve Bayes

Witten et al, (1990) explained that ZeroR only relies on predictive tags, although it does not have much predictability, ZeroR sets the predictability of the classifier as low. The classifier of "Naïve Bayes" uses a Bayesian algorithm and is statistically based (Murphy & Kevin, 2006). Witten et al, (1990) also explained, to find tags, research uses Bayes' rules to represent P(tag|feature) in the form P(tag) and P(feature|tag):

$$P(tag|features) = \frac{P(features|tag) \times P(tag)}{P(feature)}$$

By this equation, Liu et al, (2005) described that the Features are taken to input in ML classifier, text mining is in the form of a collection of words, bigrams, or even trigrams. To facilitate the work method, the separator must make naive assumptions, then the equation can be rewritten as:

$$P(tag|features) = \frac{P(tag) \times P(f1|tag) \ldots \times P(fn|tag)}{P(feature)}$$

where *f1, f2, … fn = each feature.*

Decision Tree (DT) and Logistic Regression (LR)

The decision tree has a base that corresponds to the actual tree structure where branches represent decision nodes and leaves represent assigned tags. The decision node will conclude that division to catch establish the feature value. To build a decision tree, you can start by choosing a good appearance for the decision node (Safavian et al, 1990). The fundamental approach for selecting an agreement node is to take into consideration all fit faces and view what is the most appropriate in forecasting the tags of the coaching data and formerly use those features.

One method that can be used is the entropy method, with the total of the probabilities in every tag multiplied by the log probability of the same tag. The feature that achieves the maximum entropy will be selected as the decision node (Wang et al, 1992). Max degeneration represents the top tier that can be reached from the opening disorganized input. Hosmer et al, (2000) explain that logistic regression acts as a counter to the likelihood that a tag is assigned to the Error Summary when given a set of inputs. The function used here is the Bernoulli distribution function as a probability and the sigmoid function as an input modifier. The equation will be shown as this:

$$p(y|x,w) = Ber(y|sigm(W^T x)) \ \ and \ \ sigm(\alpha) \triangleq \frac{1}{1+\exp(-\alpha)}$$

where *Ber* is the Bernoulli function, *Sigm* is the sigmoid function, *y* is the tag, *xis* the input feature and *w* is the model weight vector. In the prediction step, Error Summary is tagged with the highest probability.

Maxent (Maximum Entropy Classifier)

Mount et al, (2011) describe that "Maxent" is equivalent to logistic regression, Maxent chooses the features with the least number of unreasonable expectations, with means the Maximum Entropy (ME) as the input.

Tools of Machine Learning (TML)

There are many open-source machine-learning tools available, including Weka (Witten et al, 1999), NL-TK (Bird and Steven, 2006), and Sklearn (Pedregosa et al, 2011). Weka has a wide range of tools for running ML and N-LP. NL-TK and Sklearn are used in Python because NL-TK has many N-LP syllabus and Sklearn has many classification modules for machine learning (Bird & Steven, 2006; Pedregosa et al, 2011).

TML - Sklearn

Pedregosa et al, (2011) described the Sklearn or sci-kit-learn as a study of ML algorithms for Python programming. It contains various classification algorithms, regression, clustering, and words clarification facilities, such as the Tfidf-Vectorizer which is used to calculate the tfidf score of the words appearing in the document. TF-IDF is the *Term Frequency – Inverse Document Frequency* (Blei et al, 2003). Blei et al, (2003) define the "*Term Frequency* (tft,d)" is defined as the number of occurrences of the term, t, in document d. If t is not in d, the value of tft,d is zero. *Document Frequency* (dft) is defined as the number of documents in the corpus that contain the term t. If t is not in any documents in the corpus, dft equals zero. *Inverse Document Frequency* (idft) is used to reduce the influence of terms that appear in many documents, it is defined as:

$$idf_t = log\frac{M}{df_t}$$

where M is the amount of the sum of docs in the corpus. By this formula, a bigger value of dft produces a small idft. tfidf is used to measure the level of importance of a term in one document, so that term will have a high score for tfidf (Blei et al, 2003). Then, tfidf define as:

$$tf - idf_{t,d} = tf_{t,d} \times idf_t$$

TML - Weka

Hall et al, (2009) described that the Weka is a collection of ML tools written in Java, and includes GUI and command line operations. The GUI interface has three different applications that suit your needs and are easy to use, namely: Explorer (Kirkby et al. 2007), Experimenter (Scuse et al. 2007), and KnowledgeFlow (Bouckaert et al. 2013). Witten et al, (1999) explain that Weka can accept internal CSV file input which must later be converted into an arff file. “KnowledgeFlo”w backing the outflow of data from one element to the next. The elements are practical sections that show specific tasks. In KnowledgeFlow, components form a user-selectable palette, which can later be nested and linked together to meet a user's specific needs.

TML - N-LP Tool Kit

Bird & Steven, (2006) define that *Natural Language Processing Tool Kit (NL-TK) is* recorded in Python and intended for N-LP processing using Python. While NL-TK is used more often for N-LP, this also consists of several popular classifiers for machine learning purposes, including NaiveBayes, DecisionTree, MaxEnt, and others. Because it is a native NL-TK classifier, after applying the NL-TK natural language processing algorithm this classifier is easier to use (Bird and Steven, 2006).

Performance Evaluation Metrics for Classification

Van et al, (2001) explain that “**Accuracy** is the simplest metric used to evaluate a classifier”. **Precision** hereafter referred to as specific prognostic value is the percentage of positive data (TP) that is correctly predicted for all positive predicted data (Van et al, 2001). **Recall** also known as sensitivity, is the percentage of correct predictions of all positives (Davis et al, 2006). Thus, it can be written in mathematical form:

$$Precision = \frac{TP}{TP + FP} \times 100\%$$

Where TP is “True Positive”, FP is “False Positive”, and FN is “False Negative”. TP+FP is all data that is predicted to be positive, and TP+FN is all data that is positive (Davis et al, 2006). David et al, (2011) describe the **F-measure** as a more comprehensive measure of performance because it takes into account the effects of precision and recall and is a harmonic average of the:

$$Recall = \frac{TP}{TP + FN} \times 100\%$$

Vexed-verification

Krogh et al, 2011 explained that Vexed-verification in ML is a validation technique for assessing classifier performance. Vexed verification evaluates how the classifier results will generalize to independent data sets. In machine learning, it is customary to use 10-bend Vexed verification. Nine sections are used for training each time, with one section used for testing to be repeated 10 times. Average show metrics are projected to decide how great the classifier is. Because ML is established in data, machine learning outcomes surely need to be statistically proven for implication. In this study, 10x validation was performed more frequently. With 10x 10x Vexed verification, individual work metrics will generate 100 data points. With this amount of data, the T-test as a hypothesis test can be applied.

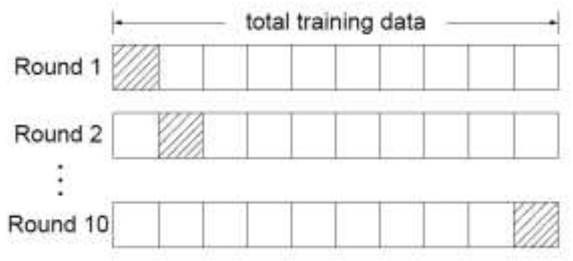


Figure 1. Sketch of 10-bend Vexed verification.

RESEARCH METHODOLOGY

Statistic test

In general, statistical tests including the T-test (Glantz et al, 2006) have two hypotheses: the null hypothesis (Anderson et al, 2000) which assumes that the two data sets are statistically equal; and the alternative hypothesis (Box et al, 1978), which assumes that the two data sets are statistically different. P-value (Moore & David, 2007) calculated in a statistical test is the probability of knowing whether one data set is significantly different from another. The common p-value is 0.05, this represents the level of confidence in ninety-five percent. Confidence intervals are interval estimates coupled with probability statements (Moore & David, 2007). “When the value is 0.05 on the results of the T-test then the p-value will be less than 0.05 which means that the two data sets are significantly different, or the alternative hypothesis is true” Glantz et al, (2006).

\

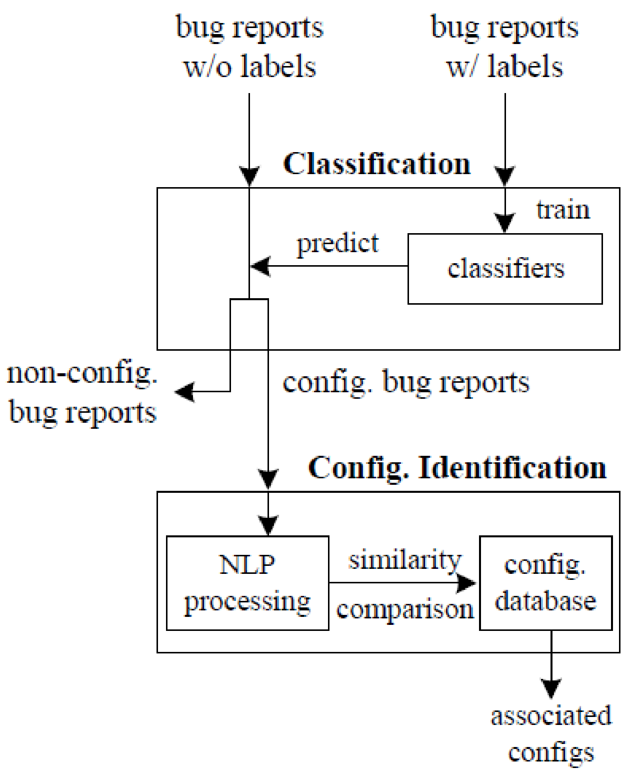


Figure 2. Mechanism flow of Error Summary method and Composition identification.

F-test and T-Test

Moore & David, (2007) explain that The F-test is used to test the population with the same variance, this is done by comparing the ratios of the two variants. The F-test in this study was used to test whether the variation in the results of performance metrics between the two classifications is the same. The basic technique uses *All Words or* AW for the method in all docs, and the other use *Highly Informative Words or* H-IW and *Highly Informative Words plus Bigram or* H-WB. The results of the F-test will determine the T-test that can be used to determine the statistical significance of metric data between AW and H-IW/H-WB (Moore & David, 2007). In this study the t-test used was a hypothesis test on the average data, where the t-test used was paired t-test and unpaired t-test (Glantz et al, 2006).

Figure 2 explains 2 main steps for the design of this study. The 1st is classification: fetching an Error Summary with an input tag, training the classifier on an Error Summary, then using this classifier to predict an untagged Error Summary. The second step is Composition Identification: receives a tagged Composition Error Summary, takes the N-LP setting to come across analogy among error report and Composition name, and displays the identical Composition tag sorted from more possible become less likely. Figure 3 is an over-specified image of the method steps. Web pages that have Error summary output even containing extraneous output that needs to be smuggled, begin the step by obtaining beneficial output from a web page given an Error Summary URL. The text file is constructed that consists of only the involved output from the web page. Mostly, only the entry text and comments from the web page are included.

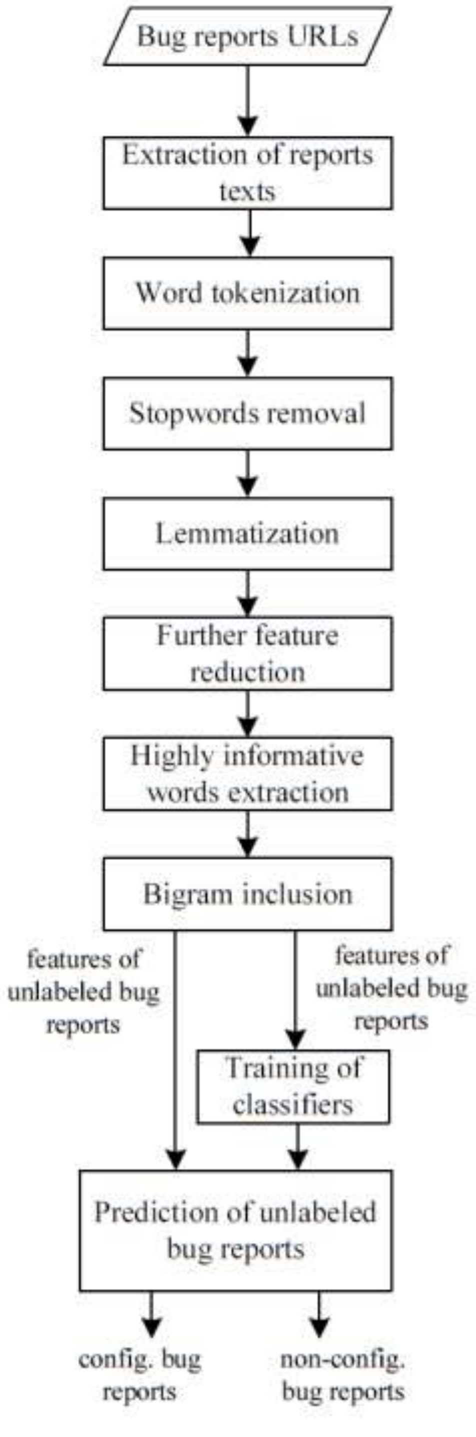

Bug reports URLs
Extraction of reports texts
Word tokenization
Stopwords removal
Lemmatization
Further feature reduction
Highly informative words extraction
Bigram inclusion
features of unlabeled bug reports
features of unlabeled bug reports
Training of classifiers
Prediction of unlabeled bug reports
config. bug reports
non-config. bug reports

Figure 3. The Error Summary method process.

This research's physiognomy is words for ML. Feature extraction includes six steps as per Figure 3, it is an N-LP procedure, and the NL-TK Python officer is taken to facilitate the task. The Error Summary text is battered to be vocabulary, then "stopwords" will be evacuated. Sometimes found the word "computer" = "com", and "configuration" as "config." etc., so vocabulary like this must be changed to its original form first. In the NL-TK package, metrics and collocations contain a chi-sq implementation. The collocations module has a BigramCollocationFinder class that can be used to find n-grams. These two classes are used to identify frequently occurring informative vocabulary and bigrams. The selected features can be arranged in the form of a dictionary, words/bigrams are used as keys and certain values are used as main values. The format used is {word: True}, where "word" is the selected word, and "True" is the value of the selected word. To evaluate the effectiveness of this study, three groups of feature words were extracted for each Error Summary database: AW, H-IW, and H-WB. Hall et al, (2009) describe that 1st phase in Weka is to convert the text file. The direction takes the CLI is java.weka.core.converters.TextDirectoryLoader works to generate the correct ARFF file. Within the directory, there should be two distinct subdirectories containing all known Summary Composition Errors, and containing all known Summary Non-Composition Errors. The name of the subdirectory must be distinguished because this name will be a guide for Weka in assigning grades to the class.

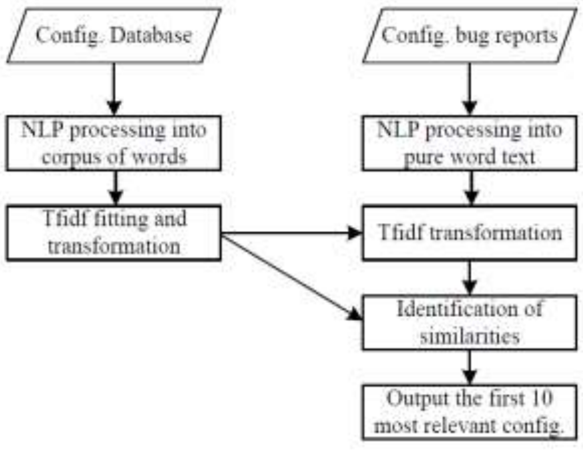
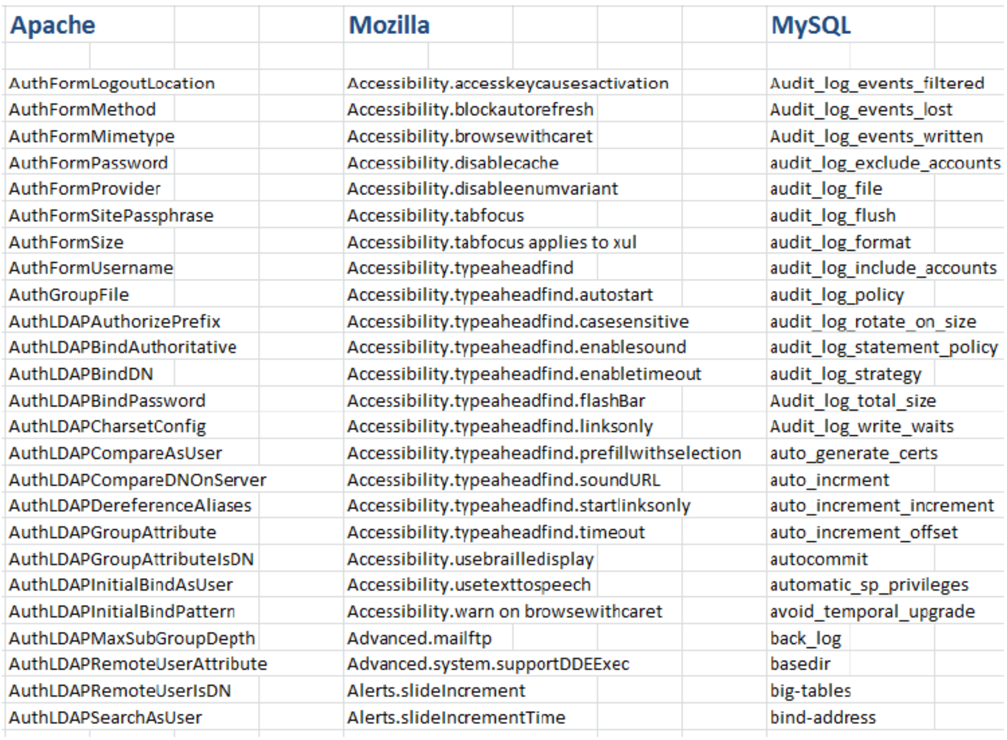

Config. Database
Config. bug reports
NLP processing into corpus of words
NLP processing into pure word text
Tfidf fitting and transformation
Tfidf transformation
Identification of similarities
Output the first 10 most relevant config.

Figure 4. Identification of the Composition Error Summary.

The created ARFF file contains only two features. To modify the text into words, the StringToWordVector filter can be used in the Weka Explorer sub-application to divide text features into word features. There are several options available to get Very Informative Words and Very Informative Words plus bigrams, in this case, the steps are similar to steps

2 to 8 in Figure 3, but there are some differences. The NL-TK classifiers used are "Naïve Bayes", Decision Trees, and MaxEnt, then for the sklearn classifiers used are Naïve Bayes, Decision Trees, Logistic Regression, and SVC or Support Vector Machine for Classification, then finally the Weka classifier which is used here is ZeroR, NaiveBayes, J48 DT, LR, and SVM. For programmatic classification in Weka it is the same as NL-TK and Sklearn. Meanwhile, in Explorer, you can only do one round of Vexed 10 bend verification which produces statistical significance analysis data with a very minimal amount.

Figure 4 shows the stream down that continues from Figure 2. There is a Composition database and Composition Error Summary. The Composition table is a take in Composition names and, Composition names may be case-sensitive periods or underscores separating the words. In Composition clarification, words are separated by underscores and periods. Occasionally when 2 words are mixed sine either there, a regular expression is used to divide them. Vocabulary is returned to its root form using lemmatization and then combined with spaces to form new strings of words, the result is, one series of words constitutes one document.



Apache	Mozilla	MySQL
AuthFormLogoutLocation	Accessibility.accesskeycausesactivation	Audit_log_events_filtered
AuthFormMethod	Accessibility.blockautorefresh	Audit_log_events_lost
AuthFormMimetype	Accessibility.browsewithcaret	Audit_log_events_written
AuthFormPassword	Accessibility.disablecache	audit_log_exclude_accounts
AuthFormProvider	Accessibility.disableenumvariant	audit_log_file
AuthFormSitePassphrase	Accessibility.tabfocus	audit_log_flush
AuthFormSize	Accessibility.tabfocus applies to xul	audit_log_format
AuthFormUsername	Accessibility.typeaheadfind	audit_log_include_accounts
AuthGroupFile	Accessibility.typeaheadfind.autostart	audit_log_policy
AuthLDAPAuthorizePrefix	Accessibility.typeaheadfind.casesensitive	audit_log_rotate_on_size
AuthLDAPBindAuthoritative	Accessibility.typeaheadfind.enablesound	audit_log_statement_policy
AuthLDAPBindDN	Accessibility.typeaheadfind.enabletimeout	audit_log_strategy
AuthLDAPBindPassword	Accessibility.typeaheadfind.flashBar	Audit_log_total_size
AuthLDAPCharsetConfig	Accessibility.typeaheadfind.linksonly	Audit_log_write_waits
AuthLDAPCompareAsUser	Accessibility.typeaheadfind.prefillwithselection	auto_generate_certs
AuthLDAPCompareDNOnServer	Accessibility.typeaheadfind.soundURL	auto_incrment
AuthLDAPDereferenceAliases	Accessibility.typeaheadfind.startlinksonly	auto_increment_increment
AuthLDAPGroupAttribute	Accessibility.typeaheadfind.timeout	auto_increment_offset
AuthLDAPGroupAttributeIsDN	Accessibility.usebrailledisplay	autocommit
AuthLDAPInitialBindAsUser	Accessibility.usetexttospeech	automatic_sp_privileges
AuthLDAPInitialBindPattern	Accessibility.warn on browsewithcaret	avoid_temporal_upgrade
AuthLDAPMaxSubGroupDepth	Advanced.mailftp	back_log
AuthLDAPRemoteUserAttribute	Advanced.system.supportDDEExec	basedir
AuthLDAPRemoteUserIsDN	Alerts.slideIncrement	big-tables
AuthLDAPSearchAsUser	Alerts.slideIncrementTime	bind-address

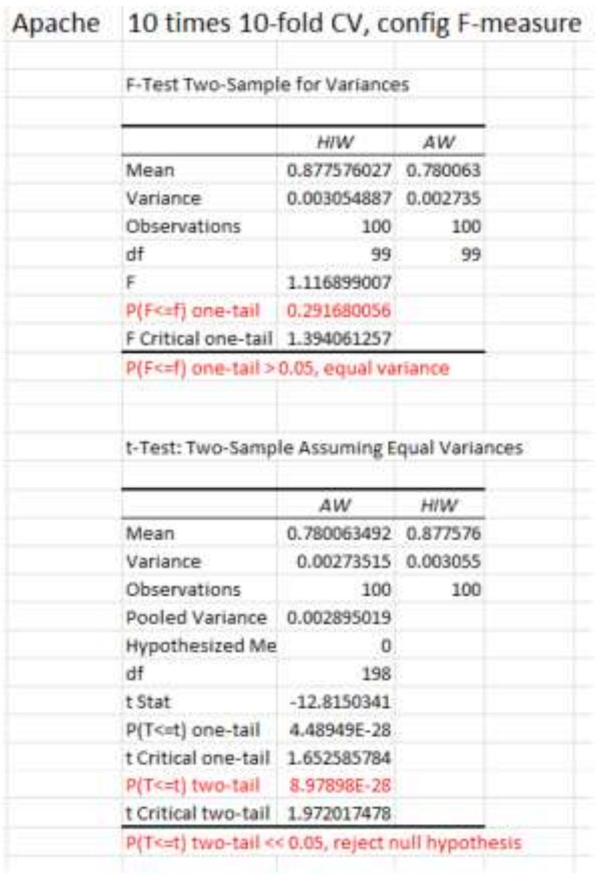
Figure 5. Some Compositions on 3 open-source.

\

The Composition Corpus processed with the Tfidf Vectorizer in Sklearn is used to convert the document set to a TF-IDF feature matrix. The Tfidf Vectorizer learns vocabulary from the TfIdfVectorizer's existing Composition corpus, unigrams and bigrams for the ngram_range parameter are included to help increase the chances of finding a match between Error Summary and Composition. Furthermore, the Tfidf Vectorizer is used to modify the Error Summary. Each Error Summary is processed individually, after the Error Summary transformation, the Tfidf Vectorizer will display the Tfidf score of each word that appears in the Composition and Error Summary. Here, to calculate the similarity score, the Naive approach is adopted. This is done by adding up the scores of all the words that appear in each Composition and Summary of Errors and displaying the Composition with the 10 highest scores.

Data Set

The success of identification, Composition and method of Error Summary in this pattern was calculated on Error Summary from open-source software projects is MySQL, Apache and Mozilla. For data diversity, this research is not limited to certain software components. The reason for choosing Error Summary from this open source project is that popular software has Error Summary publicly available, and some Mozilla Error Summary are already tagged related to some Compositions on the Mozilla website. For Composition Error Summary that do not identify Compositions then thus consistent manually. In addition, Error Summary tagging as Composition or non-Composition is also done manually. This involves using a keyword search in the Error Summary database and reading the report. For each software project, 300 Error Summary were collected, with an equal number of Composition and non-Composition Error Summary. Thus, the number of Error Summary collected for the three software projects was 900. Compiling an equal number of Composition and non-Composition Error Summary is to ensure that the classifiers trained on these Error Summary are not biased. Nevertheless, Michael et al, (2010) explained that compiling a Summary of Errors and tagging them properly would be very time-consuming.



Apache 10 times 10-fold CV, config F-measure

F-Test Two-Sample for Variances

	HIW	AW
Mean	0.877576027	0.780063
Variance	0.003054887	0.002735
Observations	100	100
df	99	99
F	1.116899007	
P(F<=f) one-tail	0.291680056	
F Critical one-tail	1.394061257	

P(F<=f) one-tail > 0.05, equal variance

t-Test: Two-Sample Assuming Equal Variances

	AW	HIW
Mean	0.780063492	0.877576
Variance	0.00273515	0.003055
Observations	100	100
Pooled Variance	0.002895019	
Hypothesized Me	0	
df	198	
t Stat	-12.8150341	
P(T<=t) one-tail	4.48949E-28	
t Critical one-tail	1.652585784	
P(T<=t) two-tail	8.97898E-28	
t Critical two-tail	1.972017478	

P(T<=t) two-tail << 0.05, reject null hypothesis

Figure 7. “F-test” of variance and average T-test of Apache F-measure Composition 10x CV 10x using NL-TK Naïve Bayes.

Vexed-verification

Vexed verification slash overfitting and enables the outcomes of a trained classifier to generalize about Summary Error predictions. All data sets were verified by Vexed 10 bends for all classifiers. To obtain sufficient statistical significance analysis data, the Vexed 10 bends verification was run ten times on every case to obtain 100 data points for each performance metric. From each type of Summary Error (Composition or non-Composition), thirty Summary Errors will be used for testing, while the other 120 will be used, so, to get one hundred data points in each performance metric, 5x validation must be run 20 times.

In this validation, the Error Summary will be randomized 20 times. After each scrambling, the Error Summary is further dividable into five categories for Composition and non-composition. Four sections are copied to the list used for testing, and one part is copied to another directory used for testing. Feature extraction for training is only performed on reports in the training directory, while for testing it is extracted from reports

\

from the test directory. After one test and training, one point of data the Error Summary is further divided into five is collected for all metrical, then the training directory is populated and then cleaned and filled again with data for the next round. After all, rounds have been completed, the Error Summary is shuffled, and the process is repeated up to 20 times. To differentiate it from the annoying 10x 10x 10 validation, hereinafter referred to as 20 multiples 5 testing. 20x5 testing walk is more passive than 10 by 10 CV. The first step that makes it slow is feature extraction. At CV 10 times, feature extraction is only done once, while at training and testing 20x5 feature extraction is done one hundred times.

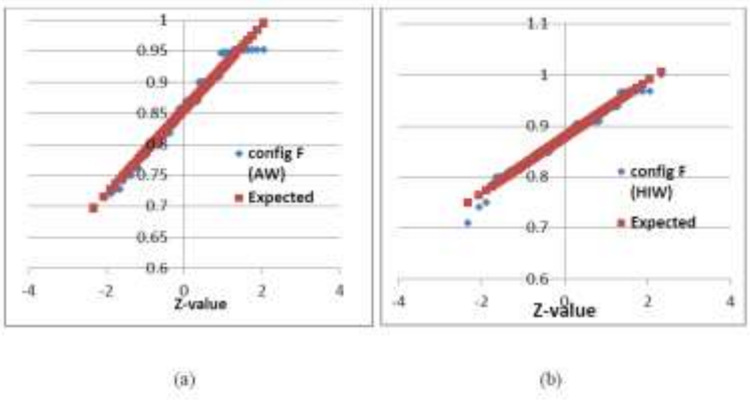


Figure 6. QQ plot of Mozilla F-measure Composition restricted using NL-TK Naïve Bayes H-IW

Statistical Significance Test

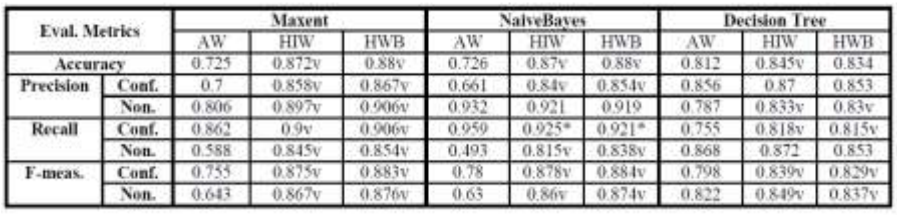
Unpaired t-tests were accomplished to calculate the analytical implication of the conduct metrics. AW method in this study was evaluation as the measured class, while H-IW and WB were the analysis groups. The purpose of this research is to appraise the strength of using Highly Informative Words and bigrams as features in classifying Error Summaries compared to using all words as features. hence, multiple test metrics will be used. first, the "F-test" was used to compare the variance of control data vs treatment data which had uneven deviation, the T-test with different variances in excel was used to test the significance. To use the F-Test, the normality of the data must be verifiable. In this study, the QQ plot (Kratz et al, 1996) was used to check the normality of the data, this can also be done in Excel. Normal rates are determined using the NORM.INV function in Excel with the CDF (Cumulative Distribution Function) parameter. Z-values are determined using the NORM.S.INV function with the CDF parameter. Finally, the data is plotted with Z-values on the x-axis, while the raw data and expected values are on the y-axis. The basic data expected values generally do not deviate much which is an indication that the data is normally distributed. Thus, using the F-test to test the variance is valid.

RESULTS AND ANALYSIS

Figure 7 shows the results of the Apache analytical connotation analysis on the “F-measure Composition”. The up table shows the “F-test” results on the F-size Composition variant. Because the “p-value” is greater than zero point zero five, the null hypothesis is considered true, this means that there is no statistical significance between the two variants. Thus, the “T-test” can be carried out and the result is a “p-value” significantly less than 0.05 so that the null hypothesis is rejected, and the mean value is significantly different. Figure 7 shows the results of the Apache analytical connotation test on the “F-measure” Composition. The table shows the “F-Test” results on the F-Size Composition variant. Because the p-value is greater than zero point zero five, the null hypothesis is considered true, which means that there is no statistical significance between the two variants. Thus, the T-test can be performed and the p-value, as a result, is necessarily less than 0.05, so the null hypothesis is rejected because the mean values differ significantly.

Tables 1 through 15 show the regular standards of the achievement metrics. One table is a result of only one open-source project from Mozilla or MySQL, it uses one classification software with 10:10x CV training or 50x2 testing. Adopting the “Weka Experimenter” format, H-IW and H-WB statistics in tables are tagged with a “*” or “v” postfix. The “*” symbol indicates that the H-IW/H-WB data is analytically worse than the AW data, while the “v” symbol indicates that the H-IW/H-WB data is analytically better than the AW data. If there is no statistical difference between AW/H-IW and H-WB the table is not tagged with any symbols.

Table 1. Apache training and testing 20 times 5 times using the NL-TK classifier



Eval. Metrics		Maxent			NaiveBayes			Decision Tree		
		AW	HIW	HWB	AW	HIW	HWB	AW	HIW	HWB
Accuracy		0.725	0.872v	0.88v	0.726	0.87v	0.88v	0.812	0.845v	0.834
Precision	Conf.	0.7	0.858v	0.867v	0.661	0.84v	0.854v	0.856	0.87	0.853
	Non.	0.806	0.897v	0.906v	0.932	0.921	0.919	0.787	0.833v	0.83v
Recall	Conf.	0.862	0.9v	0.906v	0.959	0.925*	0.921*	0.755	0.818v	0.815v
	Non.	0.588	0.845v	0.854v	0.493	0.815v	0.838v	0.868	0.872	0.853
F-meas.	Conf.	0.755	0.875v	0.883v	0.78	0.878v	0.884v	0.798	0.839v	0.829v
	Non.	0.643	0.867v	0.876v	0.63	0.86v	0.874v	0.822	0.849v	0.837v

Table 1 until table 5 shows the results of Apache. Table 1 and Table 2 are the results of the NL-TK classification. In this case, the use of H-IW and H-WB improves prediction performance. From AW to H-IW, F-measure non-Composition increases by 50%. By

\

excluding the test data on the choice of training aspects and the 50x2 test, all three classification schemes show decreasing results. Although CV10x training and 10x CV testing (10x10) and 50x2 use other docs for training and testing, more test Error Summary used on the 50x2 will stabilize the variation in results as there are too few Test Error Summary, and 120 Error summary in every kind of 50x2.

Table 2. Apache 10x CV 10x using the NL-TK classifier

Eval. Metrics		Maxent			NaiveBayes			Decision Tree		
		AW	HIW	HWB	AW	HIW	HWB	AW	HIW	HWB
Accuracy		0.679	0.857v	0.845v	0.725	0.837v	0.833v	0.808	0.848v	0.838v
Precision	Conf.	0.701	0.838v	0.806v	0.657	0.799v	0.783v	0.847	0.861v	0.848
	Non.	0.749	0.885v	0.902v	0.93	0.898*	0.916	0.783	0.842v	0.845v
Recall	Conf.	0.81	0.889v	0.912v	0.963	0.91*	0.93*	0.757	0.835v	0.828v
	Non.	0.548	0.825v	0.778v	0.488	0.763v	0.736v	0.859	0.861	0.848
F-meas.	Conf.	0.701	0.861v	0.854v	0.779	0.849v	0.849v	0.797	0.846v	0.835v
	Non.	0.567	0.852v	0.833v	0.632	0.821v	0.813v	0.817	0.85v	0.839v

In Table 3 and Table 4 it can be seen that the H-IW or H-WB classifier works greater than AW, when compared to NL-TK, the Sklearn classifier generally works superior, even when using AW, most of the F-measures are still greater than 0.8. The biggest performance increase is 22.4%.

Table 3. 10x 10x CV Apache using the Sklearn classifier

Eval. Metrics		Logistic			NaiveBayes		
		AW	HIW	HWB	AW	HIB	HWB
Accuracy		0.869	0.836*	0.913v	0.892	0.877*	0.896
Precision	Conf.	0.899	0.845*	0.946v	0.872	0.869	0.88
	Non.	0.856	0.84*	0.892v	0.925	0.895*	0.924
Recall	*Conf.*	0.838	0.834	0.88v	0.925	0.894*	0.923
	Non.	0.9	0.839*	0.947v	0.858	0.859	0.868
F-meas.	Conf.	0.862	0.835*	0.909v	0.895	0.879*	0.898
	Non.	0.873	0.836*	0.917v	0.887	0.874	0.892
Eval. Metrics		Decision Tree			SVM		
		AW	HIW	HWB	AW	HIW	HWB
Accuracy		0.801	0.853v	0.848v	0.621	0.871v	0.812v
Precision	Conf.	0.814	0.87v	0.867v	0.895	0.96v	0.974v
	Non.	0.804	0.833v	0.842v	0.575	0.817v	0.739v
Recall	Conf.	0.791	0.818v	0.831v	0.268	0.776v	0.642v
	Non.	0.811	0.872v	0.865v	0.974	0.967	0.981
F-meas.	Conf.	0.796	0.839v	0.844v	0.396	0.855v	0.766v
	Non.	0.803	0.849v	0.85v	0.722	0.884v	0.841v

Weka's classification is shown in table 5. When using "Weka", feature unlocking is performed in Weka Explorer, thus, programmatically training and testing 50x2 is not possible. Globally, the "Weka ZeroR" classifier has the worst performance of all classifiers because it only marks all Summary Errors as Composition Error Summary. DT and LR work better without using Highly Informative vocabulary and bigrams. In general, when using AW, Logistic Regression can take hours to complete whereas, when using H-IW or H-WB it only takes about 10 minutes. In the lowest case, AW to H-WB degrades by 15.8% on F-measures not configured with Logistic Regression.

Table 4. Apache training and testing 20 times 5 times using the Sklearn classifier

Eval. Metrics		Logistic			NaiveBayes		
		AW	HIW	HWB	AW	HIB	HWB
Accuracy		0.871	0.835*	0.9v	0.882	0.867*	0.84*
Precision	Conf.	0.902	0.836*	0.94v	0.853	0.863	0.798*
	Non.	0.85	0.841	0.871v	0.922	0.879*	0.905*
Recall	Conf.	0.835	0.837	0.857v	0.927	0.877*	0.917
	Non.	0.907	0.833*	0.943v	0.836	0.857v	0.763*
F-meas.	Conf.	0.865	0.834*	0.895v	0.887	0.868*	0.852*
	Non.	0.875	0.834*	0.905v	0.875	0.866	0.826*
Eval. Metrics		Decision Tree			SVM		
		AW	HIW	HWB	AW	HIW	HWB
Accuracy		0.801	0.836v	0.844v	0.615	0.854v	0.791v
Precision	Conf.	0.827	0.861v	0.866v	0.918	0.956v	0.972v
	Non.	0.786	0.82v	0.833v	0.569	0.792v	0.714v
Recall	Conf.	0.768	0.807v	0.821v	0.254	0.742v	0.601v
	Non.	0.833	0.865v	0.868v	0.977	0.965*	0.981
F-meas.	Conf.	0.793	0.83v	0.84v	0.387	0.833v	0.737v
	Non.	0.806	0.84v	0.848v	0.718	0.869v	0.826v

Table 5. CV Apache 10x 10x using the Weka classifier

Eval. Metrics		ZeroR			NaiveBayes			Decision Tree		
		AW	HIW	HWB	AW	HIB	HWB	AW	HIW	HWB
Accuracy		0.5	0.5	0.5	0.814	0.854v	0.863v	0.844	0.82*	0.822*
Precision	Conf.	0.5	0.5	0.5	0.844	0.896v	0.865v	0.856	0.855	0.849
	Non.	0.0	0.0	0.0	0.801	0.832v	0.873v	0.846	0.801*	0.811*
Recall	Conf.	1.0	1.0	1.0	0.783	0.809v	0.869v	0.838	0.779*	0.795*
	Non.	0.0	0.0	0.0	0.845	0.899v	0.857v	0.851	0.861	0.849
F-meas.	Conf.	0.667	0.667	0.667	0.807	0.845v	0.863v	0.843	0.811*	0.816*
	Non.	0.0	0.0	0.0	0.818	0.86v	0.861v	0.844	0.827*	0.826*
Eval. Metrics		Logistic Regression			Support Vector Machine					
		AW	HIW	HWB	AW	HIW	HWB			
Accuracy		0.887	0.755*	0.756*	0.886	0.898	0.891			
Precision	Conf.	0.927	0.765*	0.756*	0.908	0.945*	0.933*			
	Non.	0.863	0.759*	0.771*	0.877	0.868	0.865			
Recall	Conf.	0.845	0.749*	0.77*	0.865	0.849*	0.847*			
	Non.	0.929	0.762*	0.743*	0.908	0.947v	0.935v			
F-meas.	Conf.	0.88	0.752*	0.758*	0.882	0.891	0.885			
	Non.	0.892	0.755*	0.751*	0.889	0.903	0.896			

Table 6 until table 10 shows the classification results of Mozilla. Table 6 and table 7 is the result of NL-TK classification. Maxent has the lowest work when all words are used as features, it also needs more time for reporting errors. In the worst case, the composition of the F-measure is not zero, but when using H-IW, the F-measure boost is quite to 0.875.

Table 6. CV Mozilla 10x 10x using the NL-TK classifier

Eval. Metrics		Maxent			NaiveBayes			Decision Tree		
		AW	HIW	HWB	AW	HIW	HWB	AW	HIW	HWB
Accuracy		0.5	0.888v	0.878v	0.5	0.862v	0.862v	0.877	0.891v	0.88
Precision	Conf.	0.5	0.843v	0.841v	0.5	0.8v	0.82v	0.877	0.91v	0.881
	Non.	0.0	0.96v	0.935v	0.0	0.975v	0.935v	0.889	0.882	0.889
Recall	Conf.	1.0	0.965*	0.94*	1.0	0.98*	0.942*	0.885	0.872*	0.885
	Non.	0.0	0.811v	0.815v	0.0	0.745v	0.782v	0.87	0.909v	0.875
F-meas.	Conf.	0.667	0.898v	0.886v	0.667	0.879v	0.874v	0.878	0.887	0.88
	Non.	0.0	0.875v	0.868v	0.0	0.839v	0.847v	0.876	0.893v	0.879

Table 7. Mozilla training and testing 20 times 5 times using the NL-TK classifier

Eval. Metrics		Maxent			NaiveBayes			Decision Tree		
		AW	HIW	HWB	AW	HIW	HWB	AW	HIW	HWB
Accuracy		0.5	0.875v	0.895v	0.5	0.832v	0.873v	0.879	0.894v	0.881
Precision	Conf.	0.5	0.822v	0.852v	0.5	0.768v	0.835v	0.889	0.904v	0.884
	Non.	0.0	0.957v	0.955v	0.0	0.959v	0.929v	0.876	0.889v	0.881
Recall	Conf.	1.0	0.962*	0.96*	1.0	0.966*	0.935*	0.871	0.884v	0.878
	Non.	0.0	0.788v	0.829v	0.0	0.698v	0.811v	0.888	0.903v	0.883
F-meas.	Conf.	0.667	0.886v	0.902v	0.667	0.853v	0.881v	0.878	0.892v	0.88
	Non.	0.0	0.862v	0.886v	0.0	0.802v	0.863v	0.881	0.895v	0.881

As shown in Tables 8 and 9, when using the Sklearn classifier, the use of "H-IW" and bigram disapproved of many benefits except for "SVM". By using "H-IW" or "H-WB", you can experience the benefits of saving time, especially if there are a large total of Error Summary to be classified.

Table 8. Mozilla CV 10x 10x using the Sklearn classifier

Eval. Metrics		Logistic			NaiveBayes		
		AW	HIW	HWB	AW	HIB	HWB
Accuracy		0.94	0.822*	0.926*	0.887	0.887	0.87
Precision	Conf.	0.96	0.816*	0.927*	0.85	0.86	0.824*
	Non.	0.928	0.84*	0.932	0.943	0.93*	0.942
Recall	Conf.	0.921	0.843*	0.928	0.948	0.933*	0.95
	Non.	0.959	0.802*	0.924*	0.826	0.841v	0.791*
F-meas.	Conf.	0.938	0.826*	0.926*	0.894	0.893	0.881
	Non.	0.942	0.817*	0.926*	0.877	0.88	0.857*
		Decision Tree			SVM		
Accuracy		0.872	0.875	0.877	0.732	0.926v	0.869v
Precision	Conf.	0.886	0.869*	0.876*	0.931	0.925	0.861*
	Non.	0.866	0.892v	0.889v	0.666	0.937v	0.89v
Recall	Conf.	0.859	0.887v	0.885v	0.505	0.934v	0.888v
	Non.	0.884	0.862*	0.868*	0.96	0.918*	0.85*
F-meas.	Conf.	0.87	0.875	0.877	0.644	0.927v	0.871v
	Non.	0.872	0.83*	0.875	0.784	0.924v	0.865v

Table 9. Mozilla training and testing 20 times 5 times using the Sklearn classifier

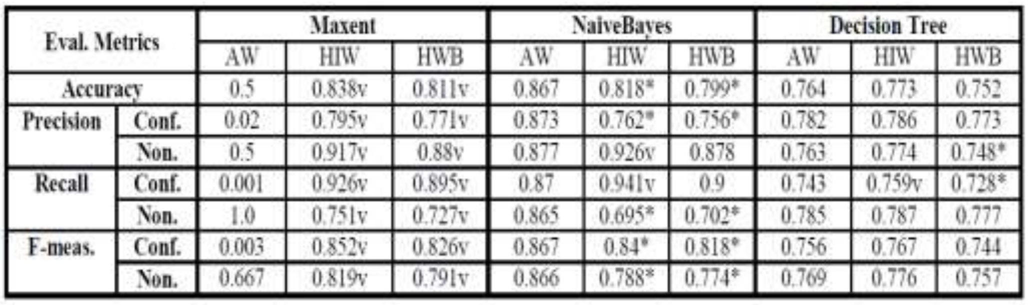
Eval. Metrics		Logistic			NaiveBayes		
		AW	HIW	HWB	AW	HIB	HWB
Accuracy		0.941	0.838*	0.926*	0.885	0.887	0.88
Precision	Conf.	0.959	0.834*	0.927*	0.843	0.85	0.84
	Non.	0.927	0.851*	0.932	0.944	0.94	0.936
Recall	Conf.	0.922	0.852*	0.928	0.95	0.945	0.943
	Non.	0.959	0.827*	0.924*	0.819	0.828	0.815
F-meas.	Conf.	0.939	0.841*	0.926*	0.892	0.894	0.887
	Non.	0.942	0.837*	0.926*	0.875	0.878	0.87
		Decision Tree			SVM		
Accuracy		0.874	0.864	0.877	0.728	0.927v	0.869v
Precision	Conf.	0.883	0.855*	0.872	0.927	0.924	0.861*
	Non.	0.871	0.879	0.889v	0.658	0.933v	0.89v
Recall	Conf.	0.865	0.879v	0.885v	0.496	0.932v	0.888v
	Non.	0.883	0.848*	0.868*	0.96	0.921*	0.85*
F-meas.	Conf.	0.872	0.865	0.877	0.642	0.927v	0.871v
	Non.	0.875	0.861*	0.875	0.78	0.926v	0.865v

Table 10. Mozilla CV 10x 10x using the Weka classifier

Eval. Metrics		ZeroR			NaiveBayes			Decision Tree		
		AW	HIW	HWB	AW	HIB	HWB	AW	HIW	HWB
Accuracy		0.5	0.5	0.5	0.809	0.832v	0.836v	0.89	0.873*	0.882
Precision	Conf.	0.5	0.5	0.5	0.876	0.89	0.885	0.906	0.874*	0.878*
	Non.	0.0	0.0	0.0	0.772	0.799v	0.807v	0.886	0.886	0.898v
Recall	Conf.	1.0	1.0	1.0	0.727	0.764v	0.778v	0.875	0.883	0.893v
	Non.	0.0	0.0	0.0	0.891	0.899	0.894	0.905	0.863*	0.87*
F-meas.	Conf.	0.667	0.667	0.667	0.789	0.817v	0.824v	0.887	0.875	0.882
	Non.	0.0	0.0	0.0	0.824	0.843v	0.845v	0.892	0.87*	0.88
Eval. Metrics		Logistic Regression			Support Vector Machine					
		AW	HIW	HWB	AW	HIW	HWB			
Accuracy		0.86	0.791*	0.822*	0.94	0.931	0.933			
Precision	Conf.	0.866	0.803*	0.842*	0.951	0.96	0.96			
	Non.	0.869	0.795*	0.817*	0.935	0.912*	0.915*			
Recall	Conf.	0.864	0.787*	0.803*	0.93	0.903*	0.907*			
	Non.	0.855	0.796*	0.841*	0.949	0.959	0.959			
F-meas.	Conf.	0.861	0.789*	0.818*	0.939	0.928	0.931			
	Non.	0.857	0.79*	0.825*	0.94	0.933	0.935			

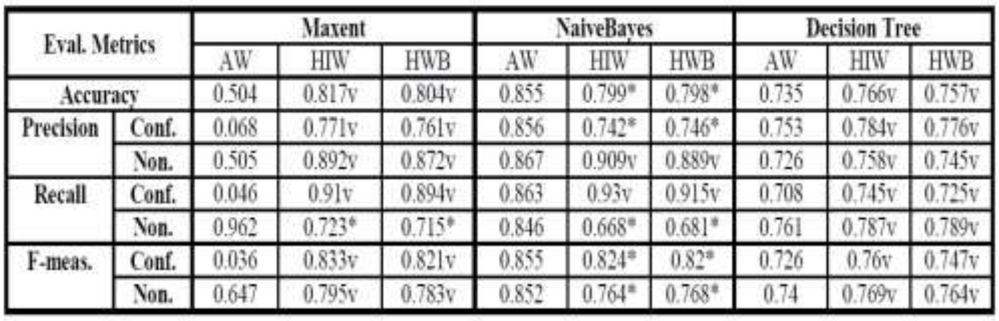
Using the Weka classifier, in general, when using AW is having many characteristics in work when compared to H-IW/H-WB, besides LR that consistently gives better performance but takes longer. Table 10 shows the output of the Mozilla Error approach report using the Weka classifier. The results Error Summary of MySQL using the NL-TK classifier is shown in tables 11 and table 12. As in the Apache and MySQL Error Summarying methods, when using “AW” or “H-IW/H-WB” as a feature, Maxent is even more sensitive. When using AW, the performance was always much worse, and vice versa, in twenty-multiple-five training and testing using H-IW/H-WB, the decision trees performed much better but were not analytically particular of the 10x10 CV.

Table 11. MySQL 10x CV 10x using the NL-TK classifier



Eval. Metrics		Maxent			NaiveBayes			Decision Tree		
		AW	HIW	HWB	AW	HIW	HWB	AW	HIW	HWB
Accuracy		0.5	0.838v	0.811v	0.867	0.818*	0.799*	0.764	0.773	0.752
Precision	Conf.	0.02	0.795v	0.771v	0.873	0.762*	0.756*	0.782	0.786	0.773
	Non.	0.5	0.917v	0.88v	0.877	0.926v	0.878	0.763	0.774	0.748*
Recall	Conf.	0.001	0.926v	0.895v	0.87	0.941v	0.9	0.743	0.759v	0.728*
	Non.	1.0	0.751v	0.727v	0.865	0.695*	0.702*	0.785	0.787	0.777
F-meas.	Conf.	0.003	0.852v	0.826v	0.867	0.84*	0.818*	0.756	0.767	0.744
	Non.	0.667	0.819v	0.791v	0.866	0.788*	0.774*	0.769	0.776	0.757

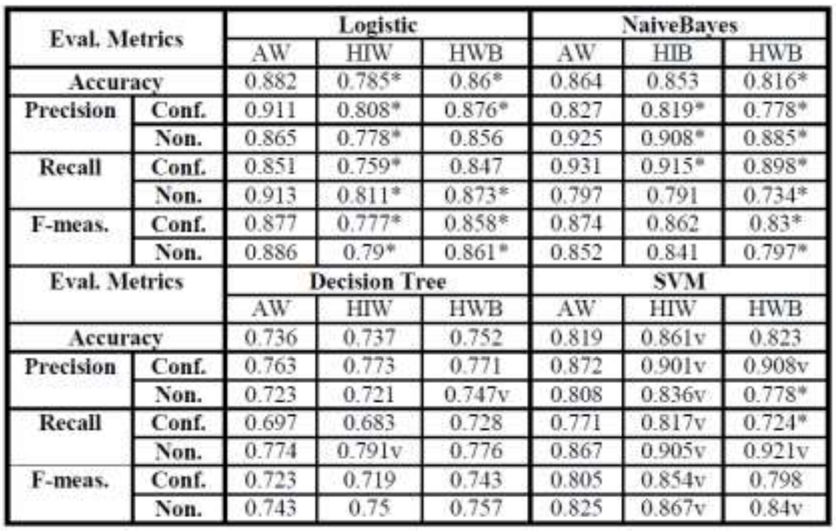
Table 12. Training and testing of MySQL 20 times 5 times using the NL-TK classifier



Eval. Metrics		Maxent			NaiveBayes			Decision Tree		
		AW	HIW	HWB	AW	HIW	HWB	AW	HIW	HWB
Accuracy		0.504	0.817v	0.804v	0.855	0.799*	0.798*	0.735	0.766v	0.757v
Precision	Conf.	0.068	0.771v	0.761v	0.856	0.742*	0.746*	0.753	0.784v	0.776v
	Non.	0.505	0.892v	0.872v	0.867	0.909v	0.889v	0.726	0.758v	0.745v
Recall	Conf.	0.046	0.91v	0.894v	0.863	0.93v	0.915v	0.708	0.745v	0.725v
	Non.	0.962	0.723*	0.715*	0.846	0.668*	0.681*	0.761	0.787v	0.789v
F-meas.	Conf.	0.036	0.833v	0.821v	0.855	0.824*	0.82*	0.726	0.76v	0.747v
	Non.	0.647	0.795v	0.783v	0.852	0.764*	0.768*	0.74	0.769v	0.764v

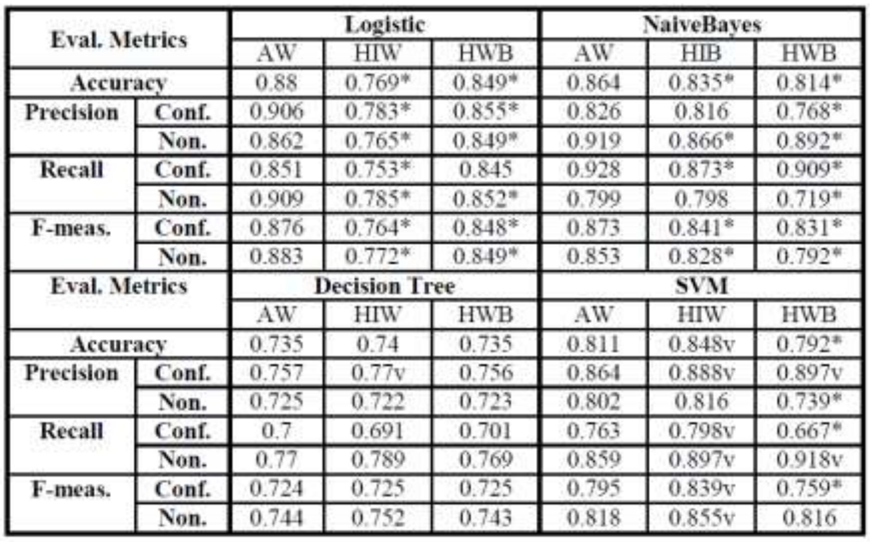
In the Sklearn classifier, when using AW SVM it is more sensitive. compared to AW, the performance was always worse, the Decision Trees here did not show a significant difference either when using AW or when using H-IW/H-WB. Here, Naïve Bayes' performance with H-IW/H-WB is known to be quite poor.

Table 13. MySQL 10x CV 10x using the Sklearn classifier



Eval. Metrics		Logistic			NaiveBayes		
		AW	HIW	HWB	AW	HIB	HWB
Accuracy		0.882	0.785*	0.86*	0.864	0.853	0.816*
Precision	Conf.	0.911	0.808*	0.876*	0.827	0.819*	0.778*
	Non.	0.865	0.778*	0.856	0.925	0.908*	0.885*
Recall	Conf.	0.851	0.759*	0.847	0.931	0.915*	0.898*
	Non.	0.913	0.811*	0.873*	0.797	0.791	0.734*
F-meas.	Conf.	0.877	0.777*	0.858*	0.874	0.862	0.83*
	Non.	0.886	0.79*	0.861*	0.852	0.841	0.797*
Eval. Metrics		Decision Tree			SVM		
		AW	HIW	HWB	AW	HIW	HWB
Accuracy		0.736	0.737	0.752	0.819	0.861v	0.823
Precision	Conf.	0.763	0.773	0.771	0.872	0.901v	0.908v
	Non.	0.723	0.721	0.747v	0.808	0.836v	0.778*
Recall	Conf.	0.697	0.683	0.728	0.771	0.817v	0.724*
	Non.	0.774	0.791v	0.776	0.867	0.905v	0.921v
F-meas.	Conf.	0.723	0.719	0.743	0.805	0.854v	0.798
	Non.	0.743	0.75	0.757	0.825	0.867v	0.84v

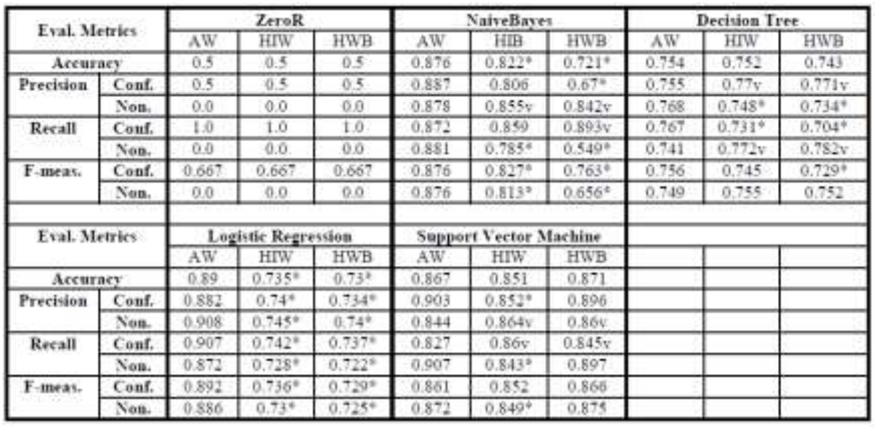
Table 14. training MySQL 20x and testing using the Sklearn classifier 5x



Eval. Metrics		Logistic			NaiveBayes		
		AW	HIW	HWB	AW	HIB	HWB
Accuracy		0.88	0.769*	0.849*	0.864	0.835*	0.814*
Precision	Conf.	0.906	0.783*	0.855*	0.826	0.816	0.768*
	Non.	0.862	0.765*	0.849*	0.919	0.866*	0.892*
Recall	Conf.	0.851	0.753*	0.845	0.928	0.873*	0.909*
	Non.	0.909	0.785*	0.852*	0.799	0.798	0.719*
F-meas.	Conf.	0.876	0.764*	0.848*	0.873	0.841*	0.831*
	Non.	0.883	0.772*	0.849*	0.853	0.828*	0.792*
Eval. Metrics		Decision Tree			SVM		
		AW	HIW	HWB	AW	HIW	HWB
Accuracy		0.735	0.74	0.735	0.811	0.848v	0.792*
Precision	Conf.	0.757	0.77v	0.756	0.864	0.888v	0.897v
	Non.	0.725	0.722	0.723	0.802	0.816	0.739*
Recall	Conf.	0.7	0.691	0.701	0.763	0.798v	0.667*
	Non.	0.77	0.789	0.769	0.859	0.897v	0.918v
F-meas.	Conf.	0.724	0.725	0.725	0.795	0.839v	0.759*
	Non.	0.744	0.752	0.743	0.818	0.855v	0.816

As shown in Table 15, in the Weka calculator, SVM can perform better with H-IW/H-WB while LR and NB perform better with AW. NL-TK classifiers are more sensitive to the use of AW/H-IW and H-WB, especially Maxent's NL-TK classifier. NL-TK classifiers can work greater when using words and bigrams with high information, besides, Error Summary of different projects have different responses to AW/H-IW and H-WB. Error Summary Classification Apache generally does better when using H-IW/H-WB than using AW. In “Weka classifier”, Logistic Regression has better performance with AW, while in Sklearn SVM it has better performance with H-IW/H-WB. This is because sometimes the Error Summary has over information due to an execution error.

Table 15. MySQL 10x CV 10x using the Weka classifier



Eval. Metrics		ZeroR			NaiveBayes			Decision Tree		
		AW	HIW	HWB	AW	HIB	HWB	AW	HIW	HWB
Accuracy		0.5	0.5	0.5	0.876	0.822*	0.721*	0.754	0.752	0.743
Precision	Conf.	0.5	0.5	0.5	0.887	0.806	0.67*	0.755	0.77v	0.771v
	Non.	0.0	0.0	0.0	0.878	0.855v	0.842v	0.768	0.748*	0.734*
Recall	Conf.	1.0	1.0	1.0	0.872	0.859	0.893v	0.767	0.731*	0.704*
	Non.	0.0	0.0	0.0	0.881	0.785*	0.549*	0.741	0.772v	0.782v
F-meas.	Conf.	0.667	0.667	0.667	0.876	0.827*	0.763*	0.756	0.745	0.729*
	Non.	0.0	0.0	0.0	0.876	0.813*	0.656*	0.749	0.755	0.752
Eval. Metrics		Logistic Regression			Support Vector Machine					
		AW	HIW	HWB	AW	HIW	HWB			
Accuracy		0.89	0.735*	0.73*	0.867	0.851	0.871			
Precision	Conf.	0.882	0.74*	0.734*	0.903	0.852*	0.896			
	Non.	0.908	0.745*	0.74*	0.844	0.864v	0.86v			
Recall	Conf.	0.907	0.742*	0.737*	0.827	0.86v	0.845v			
	Non.	0.872	0.728*	0.722*	0.907	0.843*	0.897			
F-meas.	Conf.	0.892	0.736*	0.729*	0.861	0.852	0.866			
	Non.	0.886	0.73*	0.725*	0.872	0.849*	0.875			

In NL-TK, words that are selected as features are treated the same as in the aggregated Error summary, the contention "set" and "value" do not arise in the Error Summary, these are treated as important features so that in Table 16 they do not appear at

\

all. Table 16 lists only the features sorted in *chi-sq* scores. Some of the most informative features identified by NL-TK Naive Bays for Mozilla crash reports are shown in figure 8. The ratio in the right column shows the likelihood of a feature appearing in Error vs. Summary.

Table 16. Mostly Error Summary Descriptive in Mozilla, Apache, and MySQL

Mozilla		Apache		MySQL	
crash	8375.5	configuration	542.4	option	253.4
build	1675.2	module	200.3	global	212.7
talkback	736.3	conf	196.4	configuration	208.1
reproducible	676.0	directive	175.8	cnf	171.2
identifier	553.7	enable	170.2	usr	136.2
option	442.9	src	112.3	ref	93.4
agent	376.5	xindice	99.0	connector	81.9
preference	357.3	ssl	95.0	socket	80.0
code	272.8	configure	45.8	sock	68.2

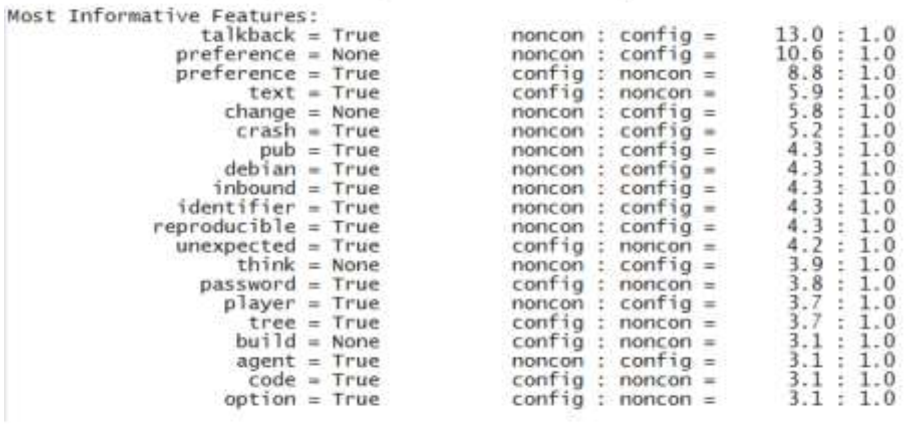

```
Most Informative Features:
               talkback = True           noncon : config =     13.0 : 1.0
             preference = None           noncon : config =     10.6 : 1.0
             preference = True           config : noncon =      8.8 : 1.0
                   text = True           config : noncon =      5.9 : 1.0
                 change = None           noncon : config =      5.8 : 1.0
                  crash = True           noncon : config =      5.2 : 1.0
                    pub = True           noncon : config =      4.3 : 1.0
                 debian = True           noncon : config =      4.3 : 1.0
                inbound = True           noncon : config =      4.3 : 1.0
             identifier = True           noncon : config =      4.3 : 1.0
           reproducible = True           noncon : config =      4.3 : 1.0
             unexpected = True           config : noncon =      4.2 : 1.0
                  think = None           noncon : config =      3.9 : 1.0
               password = True           config : noncon =      3.8 : 1.0
                 player = True           noncon : config =      3.7 : 1.0
                   tree = True           config : noncon =      3.7 : 1.0
                  build = None           config : noncon =      3.1 : 1.0
                  agent = True           noncon : config =      3.1 : 1.0
                   code = True           config : noncon =      3.1 : 1.0
                 option = True           config : noncon =      3.1 : 1.0
```

Figure 8. Some of the most descriptive aspects meaningful NL-TK NaiveBayes in a Mozilla Error Summary.

Table 17. F-measures of the average Compositions and non-compositions of the five classifiers

	Config. F-measure			Nonconfig. F-measure		
	AW	HIW	HWB	AW	HIW	HWB
ZeroR	0.667	0.667	0.667	0	0	0
Maxent	0.472	0.868	0.862	0.42	0.845	0.84
NaiveBayes	0.827	0.857	0.85	0.714	0.837	0.823
DecisionTree	0.806	0.818	0.817	0.816	0.824	0.824
Logistic	0.888	0.795	0.852	0.893	0.796	0.854
SVM	0.706	0.878	0.832	0.816	0.89	0.862

Table 17 shows the average F-measure Compositions and non-Compositions used in six classifiers. Regardless of which method is used, at “ZeroR” the performance does not change, this is the lowest classifier because its non-Composition F-measure value is 0. For the other five classifiers, it works better using H-IW/H-WB. The aspect abstraction approach was alternated against AW to H-IW with double unconfirmed F-measure values, here Maxent made the biggest improvement. The confidentiality that prerequisite the most from the use of H-IW/H-WB is SVM, which is an increase of twenty-four percent compared to the F-measure Composition. When using H-IW/H-WB Logistic Regression doesn't work properly. When using AW it takes longer to complete the Error Summary. Table 17 shows that most of the classifiers have much better performance than ZeroR. Most of the F-measure values of this classifier are greater than zero point eight, this involves the analysis of Error Summary as Composition or non-Composition is effective. Weka is the most consistent classifier, while NL-TK is the most sensitive classifier to this method. When proper methods are used, the NL-TK classifier can work very well. LR and MaxEnt are the late classifiers, especially when the method used is AW. Here, Naïve Bayes sometimes has faster performance than DT. However, in another case, the method used is often as sensitive as SVM. In Table 1-15, the difference in numbers - 0.1 is considered statistically different. When the classifier has poor performance the data points will vary widely. The most obvious difference here is the NL-TK classification results in Apache, namely AW and H-IW/H-WB.

Identify the identical Composition with the Composition Error Summary

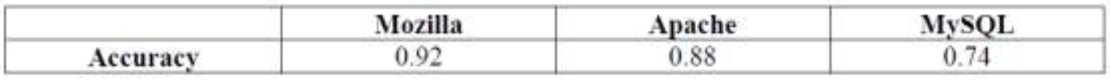
To identify the Composition associated with the Composition Error Summary, the steps in Figure 2 will be used. First, each Error Summary corpus will consist of three hundred Error Summary which will then be divided to be two parts, namely one hundred Compositions and one hundred non-Compositions used for training, and the remaining 100 will be used as no tag Error Summary for prediction. Once Error Summary is estimated, they are used to recognize connecting Compositions. For the identification of Composition options, all Composition Fault Reports used in the test will be respected. There are 50 Composition Error Summary used in testing each project. For Mozilla, such Composition Error Summary has been identified on the Mozilla Composition website. Since Error Summary can be associated with multiple Compositions, the Composition identification tool will show the first ten Compositions which will then help the developer fix the Error Summary. "TFI-DF" is used to quantify the level of importance of a term in a document. This tool uses TFI-DF to appraise the Error Summary Similarity and its Composition. Compositions are sorted from top to bottom based on similarity scores, this is defined as:

$$Similarity(d, b) = \sum_{t \in b} tf - ibf_{t,b}$$

\

Where d is the error report and b is the Composition Table 18 shows the overall results, which in finding Composition related Composition Error Summary investigated this Composition identification tool is effective, especially Mozilla, the score shown is high, namely 0.92. in MySQL, this tool has the worst performance because the MySQL Composition has an irregular number of words and is very variable.

Table 18. Accuracy of connecting Composition Error Summary with Composition



	Mozilla	Apache	MySQL
Accuracy	0.92	0.88	0.74

CONCLUSIONS

A Composition Error Summary method tool in this study was developed and Composition choice extraction was carried out in two works. First, an analysis model on tagged Error Summary to predict untagged Error Summary as structured or unstructured Error Summary is trained. 2nd step uses N-LP and the clue improvement to excerpt Composition choices from the systematic Composition Error Summary. This study uses nine hundred Error Summary from three open sources. The output display that the approach used in this study can accurately and effectively distinguish between Composition and non-Composition Error Summary. In this study ML software packages in classifying Error Summary were also compared. Of the various methods used to extract features, the NL-TK classifier is more sensitive. The sklearn classifier is heavily influenced by the method used, the SVM classification works better with “H-IW/H-WB” than with “AW”, and Weka’s classifier is not much different from H-AW or H-IW/H-WB, planning throw-back classifier works better with AW even takes a longer time.

“ZeroR” is used as a basic classifier because it has no predictive power. Maxent and SVM are very sensitive to the method used and so is Sklearn. Whereas Naïve Bayesis slightly affected by the method used, especially NL-TK Naïve Bayes, although, in broad, its work is better than “Decision Tree”. “Logistic regression” has the best performance in terms of metrics, but its performance is quite slow. Thus, it can be said that, globally, all restricted achieve better enough than the measure ZeroR. When used to conclude Error Summary in Mozilla using AW, Maxent and Naive Bayes are slightly different because their abilities are pretty bad.

Suggestions for future research

When ML techniques and N-LP present research indicates hopeful outcomes nevertheless, there are many facilities to be made to the characterization structure. In the Apache Error Summary, the use of “H-IW” or “H-WB” advance the achievement of the restricted approximately, especially the “NL-TK classifier”, while the difference in the use of AW / H-IW/H-WB in the MySQL Error Summary is not much. The Error Summary study revealed that the MySQL Error Summary contains a lot of information that is irrelevant to the type of Error Summary, significantly nevertheless, inappropriate words appear frequently, which makes all of that seem necessary, and hence they are included in the H-IW table. So, one of the future works is the Error Summary must be cleaned up first before carrying out any operation in the research.

Given that characteristics in words and phrases used in one summary have a large potential for error, using multiple crash summaries for testing would provide more information, improve classifier performance, and be able to make test outcomes more authentic, and this could be another area for future research. To make this section extra ambiguous, it is necessary to study the conduct metrics of the various classifiers as the total Error Summary for training and testing increases. For the results to be generalizable, it is necessary to vary and include more Error Summary than any other software project. The Composition identification part of this study has limitations. Composition identification tools tend to prefer Compositions with lots of words. A fix for this might involve considering reporting semantic errors, synonyms, and the use of n-grams. Finally, we are now combining separate pieces of the program into a thoroughly segmental part of the software. These sections contain a coding program to generate the plain text of the Error Summary URL, code to make the method, and code to calculate the tfidf value for the Composition Error Summary and display the best matching Composition. We recommend that you put the Weka classifier in your Python code instead of separate Java code.

REFERENCE

Anderson, David R., Kenneth P. Burnham, and William L. Thompson. (2000). Null hypothesis testing: problems, prevalence, and an alternative. The journal of wildlife management: 912-923.

Arellano, Andres, Edward Carney, and Mark A. Austin. (2015). Natural Language Processing of Textual Requirements. The Tenth International Conference on Systems (ICONS 2015), Barcelona, Spain.

Blei, David M., Andrew Y. Ng, and Michael I. Jordan. (2003). Latent dirichlet allocation." Journal of machine Learning research 3. 993-1022.

Briand, Lionel C., Yvan Labiche, and Xuetao Liu. (2007). Using machine learning to support debugging with tarantula." The 18th IEEE International Symposium on Software Reliability (ISSRE'07). IEEE.

Brill, Eric. (2000). Part-of-speech tagging. Handbook of natural language processing: 403-414.

Chowdhury, Gobinda G. (2003). Natural language processing. Annual review of information science and technology 37.1: 51-89.

Davis, Jesse, and Mark Goadrich. (2006). The relationship between Precision-Recall and ROC curves." Proceedings of the 23rd international conference on Machine learning. ACM, 2006.

Dommati, Sunil Joy, Ruchi Agrawal, and S. Sowmya Kamath. (2013). Error Classification: Feature Extraction and Comparison of Event Model using Naive Bayes Approach. arXiv preprint arXiv:1304.1677 (2013).

Fukumizu, Kenji, Francis R. Bach, and Michael I. Jordan. (2004). Dimensionality reduction for supervised learning with reproducing kernel Hilbert spaces." Journal of Machine Learning Research 5: 73-99.

Gawade, Trunal. (2016). Feature Extraction using Text mining." International Journal Of Emerging Technology and Computer Science 1.2.

Gegick, Michael, Pete Rotella, and Tao Xie. (2010). Identifying security Error Summary via text mining: An industrial case study." 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010). IEEE.

Gentleman, R., and V. J. Carey. (2008). Unsupervised machine learning." Bioconductor Case Studies. Springer New York. 137-157.

Glantz, Stanton A. (2002). Primer of biostatistics: 246.

Hall, Mark, et al. (2009). The WEKA data mining software: an update." ACM SIGKDD explorations newsletter 11.1 : 10-18.

Hosmer, David W., and Stanley Lemeshow. (2000). Introduction to the logistic regression model." Applied Logistic Regression, Second Edition: 1-30.

Jin, Dongpu, et al. (2014). Compositions everywhere: Implications for testing and debugging in practice." Companion Proceedings of the 36th International Conference on Software Engineering. ACM.

Kim, Dongsun, et al. (2013). Where should we fix this bug? a two-phase recommendation model. IEEE transactions on Software Engineering 39.11: 1597-1610.

Kirkby, Richard, Eibe Frank, and Peter Reutemann. (2007). WEKA Explorer User Guide for Version 3-5-6.

Kotsiantis, Sotiris B., I. Zaharakis, and P. Pintelas. (2007). Supervised machine learning: A review of classification techniques: 3-24.

Kratz, Marie, and Sidney I. Resnick. (1996). The QQ-estimator and heavy tails. Stochastic Models 12.4: 699-724.

Lamkanfi, Ahmed, et al. (2010). Predicting the severity of a reported bug. 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010). IEEE.

Lee, Changki, and Gary Geunbae Lee. (2006). Information gain and divergence-based feature selection for machine learning-based text categorization." Information processing & management 42.1: 155-165.

Liu, Ting, et al. (2005). Semantic role lableing system using ME classifier. Proceedings of the Ninth Conference on Computational Natural Language Learning. Association for Computational Linguistics.

Matter, Dominique, Adrian Kuhn, and Oscar Nierstrasz. (2009). Assigning Error Summary using a vocabulary-based expertise model of developers. 2009 6th IEEE International Working Conference on Mining Software Repositories. IEEE.

Michael Gegick, Pete Rotella, Tao Xie. (2010). Identifying Security Error Summary via Text mining: An Industry Case Study. InMining software repositories (MSR), 2010 7th IEEE working conference on 2010 May 2 (pp. 11-20). IEEE

Moore, David S. (2007). The basic practice of statistics. Vol. 2. New York: WH Freeman.

Murphy, Kevin P. (2006). Naive bayes classifiers. University of British Columbia.

Pedregosa, Fabian, et al. (2011). Scikit-learn: Machine learning in Python." Journal of Machine Learning Research 12.Oct: 2825-2830.

Powers, David Martin. (2011). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation.

Rastkar, Sarah, Gail C. Murphy, and Gabriel Murray. (2014). Automatic summarization of Error Summary. IEEE Transactions on Software Engineering40.4: 366-380.

Scuse, David, and Peter Reutemann. (2007). Weka experimenter tutorial for version 3-5-5." University of Waikato.

Sebastiani, Fabrizio. (2002). Machine learning in automated text categorization." ACM computing surveys (CSUR) 34.1: 1-47.

\

Smith, B. (1982). An approach to graphs of linear forms." Referencia de un trabajo no publicado), sin publicar.

Sureka, Ashish. (2012). Learning to classify Error Summary into components." International Conference on Modelling Techniques and Tools for Computer Performance Evaluation. Springer Berlin Heidelberg.

Sutton, Richard S., and Andrew G. (1998). Barto. Reinforcement learning: An introduction. Vol. 1. No. 1. Cambridge: MIT press.

Turhan, Burak, Gozde Kocak, and Ayse Bener. (2009). Data mining source code for locating software errors: A case study in telecommunication industry." Expert Systems with Applications 36.6: 9986-9990.

Van Halteren, Hans, Jakub Zavrel, and Walter Daelemans. (2001). Improving accuracy in word class tagging through the combination of machine learning systems." Computational linguistics 27.2: 199-229.

Wang, Fu, Jiazheng Xu, and Zhide Liang.)1992). Maximum Entropy Method. Textures and Microstructures 19: 55-58.

Witten, Ian H., et al. (1999). Weka: Practical machine learning tools and techniques with Java implementations.

Yin, Zuoning, et al. (2011). An empirical study on Composition errors in commercial and open source systems. Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. ACM.

Zimmermann, Thomas, Rahul Premraj, and Andreas Zeller. (2007). Predicting defects for eclipse." Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on. IEEE.