Automated Classification of Software Issue Reports Using Machine Learning Techniques: An Empirical Study

SANYAM, University of Calgary, Canada

NEHA SINGH, University of Calgary, Canada

TARUNEESH SACHDEVA, University of Calgary, Canada

SUMMARY OF CONTRIBUTIONS

Data Collection

- Code for fetching the summaries from the given ID's is done by Neha. Further, additional approximately 1000 datasets are fetched by writing the code in python.
- The additional datasets fetched are then manually classified by all the three members. Each member labelled around 340 datasets.
- Sanyam labelled the reports from HTTP client, Taruneesh labelled the reports from jackrabbit, and Neha focussed on Lucene datasets.

Coding

- Neha focussed on fetching the data and then collating all the dataset. She further made sure that the overall data is not imbalanced. The code for the same in the data bricks sheet can found under Part1 – Generation of dataset.
- Taruneesh took care of pre-processing the data which includes tokenization, stemming, and removal of stop words. The code for the same can be found in data bricks sheet under Part 2 – Pre-processing of data.
- Sanyam mainly focussed on the generation of the Document Term Matrix of training and test data set, followed by the normalization and vectorization. Code for the same can be found under Part 3- Splitting the data into train and test sets.
- Each team member then focussed on one machine learning model. Taruneesh focussed on Naïve Bayes model, Neha focussed on Linear SVC, and Sanyam focussed on Random Forest Classifier. The code for all the models can be found in the data bricks sheet under the respective model names.

Write Up

- Sanyam focused on the front page, abstract, and first two parts of results and introduction
- Neha focused on the part from 3.3 to 3.6 and again fetched and collated all the data for the misclassified labels and presented in the report
- Taruneesh focused on writing the discussions, conclusions, and references
- Further for the discussion part, each team member also added additional points corresponding to their respective machine learning models
- Dividing the misclassified summaries into the categories and finding the reasons for misclassification is teamwork

Link to Databricks (Code)
https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/1070300161508024/4295511380001275/3447793166406281/latest.html

1. ABSTRACT

1.1. Context

Software developers, testers and customers routinely submit issue reports to software issue trackers to record the problems they face in using a software. The issues are then directed to appropriate experts for analysis and fixing. However, submitters often misclassify an improvement request as a bug and vice versa. This costs valuable developer time. Hence automated classification of the submitted reports would be of great practical utility.

1.2. Objective

In this project we have tried to replicate the paper [2] and thus tried to get better results. On the similar lines as specified in the research paper, we have analysed how machine learning techniques can be used to perform the task of predicting the submitted issue reports into the right category. For, simplifying purpose, mainly two types of categories are taken into consideration. One is BUG which includes all the issues related to the BUG and the other is NUG, which includes all the reports which are not BUG.

1.3. Method

Method adopted to train the machine learning models is on similar lines with the research paper [2]. In the research paper, several machine learning models are trained on the approximate 6000 datasets taken from three open-source projects i.e., HTTP Client[3], Lucene[4], and Jackrabbit[5]. We took the previous dataset and then further added 1000 new datasets to it. 1000 new datasets are first manually classified into the BUG and the NUG categories. We then trained the three machine learning models on the new dataset, namely Naive Bayes (NB), Linear Support Vector Machine (LSVM) with various kernels and Random Forest (RF) Classifier. The performance of these models is evaluated with different hyper-parameters in terms of F-measure, average accuracy, average precision, and recall. The performance is also compared with the performance of the models in given in the research paper[2].

1.4. Results

Our results show that, on the new data set Linear Support Vector Machine (LSVM) performs best irrespective of any hyper-parameters or normalization of the features. However, Naïve Bayes Classifier gives the best accuracy of 0.82 and F1 score of 0.79, when trained on new data set with non-normalized features. Further, all the models seem to behave the same for the old and the new dataset. F1 scores and accuracy are almost same for all the models, irrespective of the data. RFC and LSVM seems not to be affected by the normalization. However, the scores of the Naïve Bayes model significantly fall if the model is trained on the normalized features. Lastly, all the models seem to behave at par with the models given in the research paper[2].

1.5. Conclusion

Through this work, we have replicated the model given in the research paper[2] and are able to train the models at par with the models given in the research paper. We further conclude that machine learning models can be used for classifying the labels into the BUG and the NUG categories.

2. INTRODUCTION

A software evolves continuously over its lifetime. As it is developed and maintained, bugs are filed, assigned to developers and fixed. Bugs can be filed by developers themselves, testers or customers, or, in other words by any user of the software. For open-source projects, defect tracking tools like Jira are commonly used for storing bug reports and tracking them till closure. Proprietary software also uses same or similar tools. However, along with bugs (meant for corrective maintenance of the software), it is common to file change requests that ask for adaptation of the software to new platforms (adaptive maintenance) or to incorporate new features (perfective maintenance). Given this diversity, the person filing the issue may not always make a fine-grained distinction between the different kinds of reports and instead file them as bugs only. In fact, research shows misclassifications are common.

A genuine bug should receive higher priority than other requests; therefore, a misclassification may lead to incorrect focus on the issue from the developers. Manual classification by application engineers or developers after the issue has been filed requires a thorough review of the problem and consumes considerable time [6]. We feel it is better if automated classification can be done to categorize the issues into BUG and NUG. Such a classifier could be integrated with the tracker and monitor every issue being filed. If it thinks the issue is wrongly classified by the submitter, it may prompt the user with a suggestion. Our belief is strengthened from the success of machine learning tools in various applications like spam detection, text classification and image recognition. Here a classifier is first trained with a set of labelled instances and then asked to recognize the class of a new instance.

In an elaborate study involving more than 7000 issues spanning 5 projects, researchers found that 33.8% of all reports are misclassified [6]. The consequence could be costly: developers must spend their precious time to investigate the reports and relabel them correctly. Hence it is worthwhile to explore if this classification can be done automatically.

In this project, we study how supervised learning techniques [7] can be used to automatically classify an issue report as referring to a bug or to something that is not a bug. The key intuition is certain terms that describe errors or failures in the software are more common in descriptions that truly report a bug. Thus, a classifier could be trained with a set of issue reports, each of which contains some description of the issue and is already labelled with the correct category. It can then proceed to classify an unlabelled report that it has not seen at training time.

Figure 1 Top 100 discriminant terms in the summaries of BUG and NUG categories.

In this paper, we report our experiences with application of machine learning algorithms to classify issue reports from three sources into bug and non-bug categories in a completely automated way. The issue reports selected belong to three open-source projects HttpClient4, Lucene5 and Jackrabbit6. We use the following classification algorithms: naive Bayes (NB), linear support vector machine (SVM) with various kernels, and random forest (RF) classifier. We analyse the effect of different parameters on the classifiers' predictive power. We measure classification efficiency in terms of F-measure, average accuracy, and weighted average F-measure.

While training the models, the data frames mainly include columns: ID, CLASSIFICATION, SUMMARY. SUMMARY is the summary of the issue, CLASSIFICATION is the classification type, and ID is unique number of the issue. Description is ignored and is not used in the training of the models, as descriptions are not much useful and does not help in improving the performance of the models.

<u>Basic overflow of how work</u>:

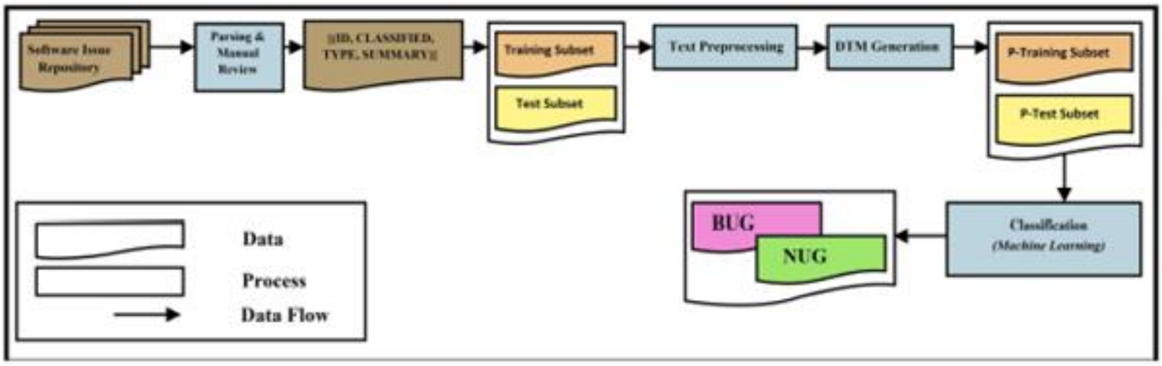


Figure 2: Overflow of how data is processed and ML model is trained

3. RESULTS

3.1. How was the new data labelled/collected?

- Approximately 6000 reports/summaries from the old dataset given in [1] are taken
- Additional 1000 new datasets are then fetched from the same sources from which old dataset was taken
- The code for fetching the new dataset is written in python and is attached along, where we extract the summaries using asyncio and web scrapping based on the unique issue ID's of the summaries
- Additional datasets are then manually classified into BUG and NUG
- Each member has labelled approximately 340 datasets
- To maintain the balance of the classes, we then additionally added the summaries that are BUG
- All the old and the new dataset is then collated
- The final data frame is in the 'df_issue_report' in data bricks sheet

Below is the image which gives a glimpse of the data frame:

ID	CLASSIFIED	SUMMARY
LUCENE-2719	NUG	Re-add SorterTem...
LUCENE-754	BUG	FieldCache keeps...
LUCENE-1487	NUG	FieldCacheTermsF...
LUCENE-2216	BUG	OpenBitSet#hashC...
LUCENE-1061	NUG	Adding a factory...
LUCENE-456	BUG	Duplicate hits a...
LUCENE-2243	NUG	FastVectorHighli...
LUCENE-2458	NUG	queryparser make...
LUCENE-2670	NUG	allow automatont...
LUCENE-3183	BUG	TestIndexWriter ...

Figure 3 Sample of combined dataset data frame.

Agreement Analysis and Statistics

Following is the agreement analysis between the team members:

- Taruneesh and Sanyam agreed on 40 labels out of 50, therefore the percentage agreement was 80%.
- Similarly, Sanyam and Neha agreed on 42 labels out of 50, therefore the percentage agreement was 84%.
- Neha and Taruneesh had an agreement on 44 out of 50 labels. Therefore, the percentage agreement was 88%. Agreement

The labels on which the team members had disagreements were resolved manually by checking the descriptions of the corresponding issue ids. Further, if the confusion persists then the summary is labelled with that class for which half of the members agreed. Sometimes, the keywords given in figure 1 were quite helpful in labelling the classes.

Following are the some of the examples for which there were disagreements and were resolved:

SUMMARY	LABEL
Queries should reject invalid nodeLocalName parameters	NUG
decoded PATH of cookie value in CookieOrigin	BUG
DiskDV probably shouldnt use BlockPackedReader for SortedDV doc-to-ord	BUG

Quality of the data

We took several factors into account for ensuring the quality of the data. First and foremost was the duplicacy in the data. We made sure that no summary is getting duplicated in the dataset. Next step was to ensure that classification we are doing is correct. So, we performed the agreement analysis quite stringently to make sure that data is being classified in right categories. Further, we added additional datasets which belonged to BUG category to make sure that final dataset is balanced.

Overall, the quality of the dataset was not an issue here because we just fetched the additional summaries from the same sources, Thus, all the data is already in sink. There are number of common keywords between the old and the new datasets. This is verified after generating the document term matrix for both the old and the new data. The difference in the number of terms in the document matrix of the old dataset is approximately 300. Thus, there are only 300 new words in the new data sets in comparison to the old datasets.

3.2. How does the newly added data compare with the original data?

As per the original issue report mentioned in the research paper, we had 3651 NUGS and 1940 BUGS.

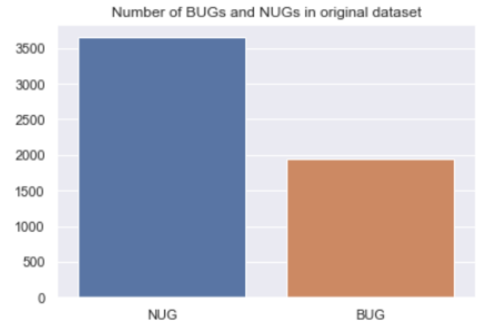
CLASSIFIED	count
NUG	3651
BUG	1940

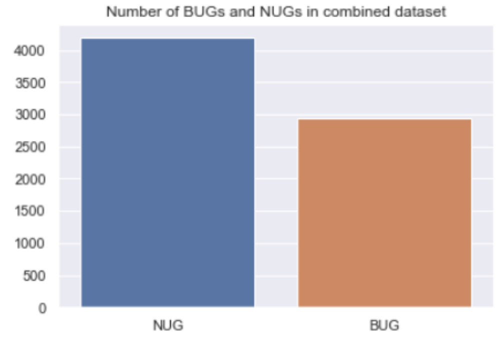
Figure 4 Count of BUGs and NUGs in original dataset

After we added the manually classified data, the overall count resulted in 4188 NUGS and 2931 BUGS.

CLASSIFIED	count
NUG	4188
BUG	2931

Figure 5 Count of BUGs and NUGs in final dataset





- Above bar plot shows the comparison of count of BUGs and NUGs in original dataset and combined dataset which consists of manually classified additional dataset.
- To conclude, we added 537 NUGS and 991 BUGS.
- Due to imbalance in the original issue report dataset where the count of NUGS in the research paper[1] was twice as much as BUGS, we added more manually classified BUGS as compared to NUGS to balance our dataset.
- After Manual Classification, the percentage of NUGS was 58.8% as compared to 41.2% for BUGS.

Following are the few documents from the new data:

9	JCR-4009	CSRF in Jackrabbit-Webdav (CVE-2016-6801)	NUG
10	JCR-4010	Release Jackrabbit 2.12.4	NUG
11	JCR-4011	Query - Jackrabbit Support Needed - on reindexing	BUG
12	JCR-4012	Include initial cost in stats for observation processing	NUG
13	JCR-4013	Calculate eventConsumerTimeRatio for entire time series	NUG

Following are the few documents from the old data:

410	JCR-1495	NamespaceAdder.addNamespace throws ClassCastException
411	JCR-2287	Mandatory jcr:activities node missing after upgrade
412	JCR-2048	Workspace is shut down while creating initial index
413	JCR-1359	Adding nodes from concurrently running sessions cause exceptions
414	JCR-2672	Cache also failed principal lookups
415	JCR-2327	java.util.UUID.fromString() too slow

We can observe from the above datasets that the new dataset is in sync with the original dataset and thus there is not much difference between the two. In addition, based on the terms in the document term matrix, we can conclude that the two datasets are similar. The issue of imbalanced dataset was handled by adding new dataset accordingly.

Below screenshots show the number of terms in the document term matrix of the old and the new dataset:

CLASSIFIED_LABEL	WORDS_TF	SUMMARY	WORDS_TF_norm
0	(2425,[4,312,424,...	Add Payload retr...	(2425,[4,312,424,...
1	(2425,[9,68,167,2...	RussianAnalyzer's...	(2425,[9,68,167,2...
1	(2425,[3,6,43,285...	GData TestDateFo...	(2425,[3,6,43,285...
0	(2425,[336,468,57...	Flexibility to t...	(2425,[336,468,57...
1	(2425,[13,27,29,5...	document with no...	(2425,[13,27,29,5...
0	(2425,[27,29,50,9...	LogByteSizeMerge...	(2425,[27,29,50,9...
1	(2425,[29,50,61,1...	Document with no...	(2425,[29,50,61,1...

Figure 6: 2425 represents the number of terms in DTM of new dataset

CLASSIFIED_LABEL	WORDS_TF	SUMMARY	WORDS_TF_norm
0	(2067,[4,355,437,...	Add Payload retr...	(2067,[4,355,437,...
0	(2067,[83,382],[1...	Nightly Builds	(2067,[83,382],[0...
1	(2067,[9,62,117,2...	RussianAnalyzer's...	(2067,[9,62,117,2...
1	(2067,[3,7,41,303...	GData TestDateFo...	(2067,[3,7,41,303...

Figure 7: 2067 represents the number of terms in DTM of old dataset

3.3. How was the data pre-processed?

We implemented a function preProcessData() for :

- The removal of punctuation,
- The removal of spaces, integral values, html tags
- Conversion to lower case and any further noise.

```
from pyspark.sql.functions import regexp_replace,col, trim, upper, lower

# function for pre-processing the data

def preProcessData(df, colName):

   # removing new line character

  df_line = df.withColumn('SUMMARY_PROCESSED', regexp_replace(col(colName), "[\\r\\n]", " "))

   # removing white

  df_line = df_line.withColumn("SUMMARY_PROCESSED",regexp_replace("SUMMARY_PROCESSED", "\\[.*\\]", " "))

  # converting the data to lower case

  df_lower = df_line.withColumn("SUMMARY_PROCESSED", lower(col("SUMMARY_PROCESSED")))

  # removing double quotes

  df_quotes = df_lower.withColumn("SUMMARY_PROCESSED", regexp_replace("SUMMARY_PROCESSED", '"', ""))

  # removing punctuation marks

  df_punc = df_quotes.withColumn("SUMMARY_PROCESSED", regexp_replace("SUMMARY_PROCESSED", '[.,/#!$%^&*;:{}=_\'~()?|]', " "))
```

```
# removing hyphens

df_hyphen = df_punc.withColumn("SUMMARY_PROCESSED", regexp_replace(col("SUMMARY_PROCESSED"), "-", " "))

# removing integral values

df_int = df_hyphen.withColumn("SUMMARY_PROCESSED", regexp_replace(col("SUMMARY_PROCESSED"), "\\d+", " "))

 # removing ids

df_id = df_int.withColumn("SUMMARY_PROCESSED", regexp_replace(col("SUMMARY_PROCESSED"), "\[[a-z A-Z]+\\s\]", ""))

 #removing asf jira

df_asf = df_id.withColumn("SUMMARY_PROCESSED", regexp_replace(col("SUMMARY_PROCESSED"), "asf jira", ""))

# removing white

df_space = df_asf.withColumn("SUMMARY_PROCESSED", regexp_replace(col("SUMMARY_PROCESSED"), "\\s+", " "))


return df_space
```

Further to pre-processing, we used NLTK Porter Stemmer and Tokenizer to prepare the data.

- Tokenization of Data

As part of tokenization, we break up a sequence of strings into pieces such as words, keywords, phrases, symbols and other elements called tokens. Tokens can be individual words, phrases or even whole sentences. In the process of tokenization, some characters like punctuation marks are discarded.

```
# tokenize_stop function tokenizes the column

from pyspark.ml.feature import Tokenizer

def tokenize_stop(df, colName):
  df_tokenizer = Tokenizer(inputCol = colName, outputCol = 'WORDS')
  df_tokenized = df_tokenizer.transform(df)
  return df_tokenized
```

Figure 8 Method to tokenize the summary

- Removal of stopwords

To highlight, our list of stopwords is very limited and comprises mainly articles, conjunctions, prepositions and pronouns. In particular, we do not remove stopwords denoting negatives like ‘not’ and modal auxiliaries like ‘should’ that influence the meaning of the reports significantly. For example, a bug report is more likely to have negatives like ‘not’ while an improvement request can specify a desirable feature with ‘should’. Similarly, temporal connectives like “before", “after", “when", etc. usually describe test scenarios used to report issues. Hence, they are also not eliminated. The resultant corpus is then stemmed and converted to a document term matrix (DTM).

```
list_stop_words = ['a', 'an', 'the','i', 'me', 'my', 'myself',
                   'we', 'our', 'ours', 'ourselves', 'you', "you're",
                   "you've", "you'll", "you'd", 'your', 'yours',
                   'yourself', 'yourselves', 'he', 'him', 'his', 'himself',
                   'she', "she's", 'her', 'hers', 'herself', 'it', "it's",
                   'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves',
                   'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
                   'these', 'those', 'of', 'at', 'by', 'for', 'with', 'in',
                   'out', 'on', 'off', 'and', 'but', 'or', 'yet', 'so', 'although',
                   'though']

# Function to write the stop word

from pyspark.sql.types import ArrayType, FloatType, StringType

def remove_stops(row):
    meaningful_words = [w for w in row if not w in list_stop_words]
    return meaningful_words

udf_remove_stops = udf(remove_stops, ArrayType(StringType()))
```

Figure 9 Method to implement removal of stop word from tokenised summary

- Stemming

Stemming is the process of removing a part of a word, or *reducing a word to its stem* or root. Here we used nltk.stem.porter to stem the summary list after stop words removal. The column WORD_STEM comtains stemmed data.

ID	CLASSIFIED	SUMMARY
LUCENE-2719	NUG	Re-add SorterTem...
LUCENE-754	BUG	FieldCache keeps...
LUCENE-1487	NUG	FieldCacheTermsF...
LUCENE-2216	BUG	OpenBitSet#hashC...
LUCENE-1061	NUG	Adding a factory...
LUCENE-456	BUG	Duplicate hits a...
LUCENE-2243	NUG	FastVectorHighli...
LUCENE-2458	NUG	queryparser make...
LUCENE-2670	NUG	allow automatont...

Figure 10 Sample data frame before stemming

```
# stemming of words
from pyspark.sql.functions import col, size
from nltk.stem.porter import *
from pyspark.sql.types import *

# Create user defined function for stemming with return type Array<String>
stemmer_udf = udf(lambda x: stem(x), ArrayType(StringType()))
# Instantiate stemmer object
stemmer = PorterStemmer()

# Create stemmer python function
def stem(in_vec):
    out_vec = []
    for t in in_vec:
        t_stem = stemmer.stem(t)
        if len(t_stem) > 2:
            out_vec.append(t_stem)
    return out_vec
```

Figure 11 Stemming method

```
+-----------+--------------------+--------------------+----------+
|         ID|             SUMMARY|          WORDS_STEM|CLASSIFIED|
+-----------+--------------------+--------------------+----------+
|LUCENE-2719| Re-add SorterTem...|[add, sortertempl...|       NUG|
| LUCENE-754| FieldCache keeps...|[fieldcach, keep,...|       BUG|
|LUCENE-1487| FieldCacheTermsF...|[fieldcachetermsf...|       NUG|
|LUCENE-2216| OpenBitSet#hashC...|[openbitset, hash...|       BUG|
|LUCENE-1061| Adding a factory...|[factori, querypa...|       NUG|
| LUCENE-456| Duplicate hits a...|[duplic, hit, mis...|       BUG|
|LUCENE-2243| FastVectorHighli...|[fastvectorhighli...|       NUG|
|LUCENE-2458| queryparser make...|[querypars, make,...|       NUG|
|LUCENE-2670| allow automatont...|[allow, automaton...|       NUG|
|LUCENE-3183| TestIndexWriter ...|[testindexwrit, f...|       BUG|
+-----------+--------------------+--------------------+----------+
```

Figure 12 Sample data frame after stemming

After stemming, we generate Document Term Matrix for Training and Test dataset.

Generation of DTM: Instead of using the training and test subsets directly, we generate a DTM for each subset. First, the training subset is pre-processed and converted to a DTM (or p-training subset). The most frequent terms, counting to no more than MAX TERMS IN DTM, are retrained in the DTM. Then the test subset is converted to a DTM (or p-test subset) comprised of only those terms that are already present in the training DTM.

We call function vectorizing_data() which uses CountVectorizer to vectorize the corpus. Then we called get_dtm_stopwords() which gives us the list of words which are present in test dataset but not in training dataset. Then we implement udf_remove_stops_dtm() which removes the list of words we extracted from above method from the WORD_STEM in test dataset. Then we call vectorizer on the remaining terms in test data. Corresponding code can be found under heading 'DTM generation of the training dataset' in data bricks sheet.

3.4. How do the models perform on the original data vs the new + original data?

To measure the performance of the classifiers, we used the measures of precision, recall, F-measure and accuracy. The four important quantities defined with respect to a class of interest are:

- 1. True Positive (TP): Number of reports correctly labelled to a class.
- 2. True Negative (TN): Number of reports correctly rejected from a class.
- 3. False Positive (FP): Number of reports incorrectly labelled to a class.
- 4. False Negative (FN): Number of reports incorrectly rejected from a class.

```
def getValues(df):
  TP = df.where("CLASSIFIED_LABEL == prediction and CLASSIFIED_LABEL = 1").count()
  TN = df.where("CLASSIFIED_LABEL == prediction and CLASSIFIED_LABEL = 0").count()
  FP = df.where("CLASSIFIED_LABEL != prediction and CLASSIFIED_LABEL = 0").count()
  FN = df.where("CLASSIFIED_LABEL != prediction and CLASSIFIED_LABEL = 1").count()
  return TP, TN, FP, FN
```

Figure 13 Method to calculate values for TP, TN, FP, FN

Where **Precision**: It is the ratio of the number of true positives to the total number of reports labelled by the classifier as belonging to the positive class.

```
def precision(tp, fp):
  denom = tp+ fp
  if denom != 0:
    return tp/denom
  else:
    return 0
```

Figure 14 Method to calculate Precision.

Recall: It is the ratio of the number of true positives to the total number of reports that actually belong to the positive class.

```
def recall(tp, fn):
  denom = tp+fn
  if denom != 0:
    return tp/denom
  else:
    return 0
```

Figure 15 Method to calculate Recall

F-Measure or F1-score: It is the harmonic mean of precision and recall.

```
def f1_score(precision, recall):
  denom = precision + recall
  if denom == 0:
    return 0
  else:
    return (2*precision*recall)/denom
```

Figure 16Method to calculate F1 score

Accuracy: It measures how correctly the classifier labelled the records.

```
def accuracy(tp, fp, tn, fn):
  r = tp + tn
  wrong = fp+fn
  total  = r+ wrong
  if total != 0:
    return r/total
  else:
    return 0
```

Figure 17Method to calculate Accuracy

We use k-fold cross-validation to measure the predictive power of the classifiers.

- From the results, we observe that Linear SVC performs best out of the three models and the factor of normalization doesn't affect the scores in the model.
- Naive Bayes performs better without normalization on any dataset.
- Random Forest Classifier performs same with or without normalization.
- Overall, adding the new dataset did not make much difference.

Comparison on model performance between old dataset vs old+new dataset

- All the models performed at par with the results given in the research paper [2].
- The highest f-measures values mentioned by the papers were in range of 0.6 to 0.83, but the indicator they have used was weighted F-measure, which generally gives higher value than the normal F-measure.
- Best F-measure score are 0.79 and 0.73 for Naïve Bayes and Linear SVC model. For, all cases Linear SVC performs better than Naïve Bayes.
- Below are four tables justifying the results.

The following table has the scores for old dataset with normalization.

Model(Classifiers)	Naive Bayes	Linear SVM	Random Forest
Metrics			
Accuracy	0.70	0.80	0.79
Average Precision	0.90	0.70	0.80
F1-Score	0.29	0.71	0.62
Recall	0.17	0.72	0.51

The following table has the scores for old dataset without normalization.

Model(Classifiers)	Naive Bayes	Linear SVM	Random Forest
Metrics			
Accuracy	0.85	0.80	0.78
Average Precision	0.76	0.75	0.78
F1-Score	0.78	0.73	0.59
Recall	0.79	0.71	0.47

The following table has the scores for old + new dataset with normalization.

Model(Classifiers)	Naive Bayes	Linear SVM	Random Forest
Metrics			
Accuracy	0.73	0.77	0.76
Average Precision	0.88	0.73	0.82
F1-Score	0.56	0.72	0.67
Recall	0.41	0.72	0.56

The following table has the scores for old + new dataset without normalization.

Model(Classifiers)	Naive Bayes	Linear SVM	Random Forest
Metrics			
Accuracy	0.82	0.78	0.76
Average Precision	0.80	0.75	0.81
F1-Score	0.79	0.73	0.65
Recall	0.77	0.71	0.55

3.5. How does the performance of the models change based on the choice of hyper parameters?

Naive Bayes

Naive bayesian algorithm supports 2 hyper-parameters:

- smoothing parameter: (alpha-float, default=1.0) This parameter is used to smooth the categorical data.
- modelType: (default is "multinomial"), multinomial is implemented for continuous features, Bernoulli is implemented for discrete features.

We evaluate the prediction quality against different parameter values to find the best parameter values. Below table shows the metric scores for NB with default and different hyper-parameters:

Hyper-parameters	Precision	Recall	F1 score	Accuracy
Default	0.88	0.41	0.56	0.73
modelType = "multinomial", smoothing = 0.0	0.81	0.61	0.7	0.78

From the above table, we can observe that the performance of Naïve Bayes classifier improved when smoothening value was set to 0 instead of default 1.

Random Forest

- n_estimators: number of trees in your forest (100)
- max_depth: maximum depth of your tree (None) - recommendation, change this parameter to be an actual number because this parameter could cause overfitting from learning your traning data too well
- min_samples_split: minimum samples required to split your node (2)

- min_samples_leaf: mimimum number of samples to be at your leaf node (1)
- max_features: number of features used for the best split ("auto")
- boostrap: if you want to use boostrapped samples (True)
- n_jobs: number of jobs in parallel run (None) - for using all processors, put -1
- random_state: for reproducibility in controlling randomness of samples (None)
- verbose: text output of model in process (None)
- class_weight: balancing weights of features, n_samples / (n_classes * np.bincount(y)) (None) - recommendation, use 'balanced' for labels that are unbalanced

We used numTrees, maxDepth and maxBins to tune the hyper-parameters for RF classifier. Below table shows the metric scores with default and tuned hyper-parameters.

Hyper-parameters	Precision	Recall	F1 score	Accuracy
Default	0.84	0.06	0.12	0.6
numTrees= 60, maxDepth=30, maxBins=40	0.81	0.55	0.66	0.76

As per above table, we can observer that the performance of Random forest classifier showed significant improvement when the model was trained with tuned hyper parameters. RFC scores change significantly after changing the hyper-parameters to (numTrees= 60, maxDepth=30, maxBins=40)

Linear Support Vector Machine

maxIter: Param (parent='undefined', name='maxIter', doc='max number of iterations (>= 0).')

regParam: Param (parent='undefined', name='regParam', doc='regularization parameter (>= 0).')

A support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks. A higher regParam value increases the regularization where as a lower value of regParam reduces the regularisation strength. Below tables shows metrics scores for SVM with default and different hyper-parameters:

Hyper-parameters	Precision	Recall	F1 score	Accuracy
Default	0.73	0.72	0.72	0.77
regParam: [10, 1, 0, 0.1, 0.01], maxIter: [10, 20, 30]	0.73	0.72	0.72	0.77

From above table we can observer that, hyper-parameter tuning did not have any significant impact of SVM classifiers performance. Next we discuss the impact of normalization on the performance of classifiers.

1. With Normalization

- When the hyper-parameters (numTrees= 60, maxDepth=30, maxBins=40) are given, Random Forest Classifier performs better as compared to without any hyper-parameters.
- The precision and recall increase significantly which concludes that the model behaves better after hyper-parameter tuning.

2. Without Normalization

- When the same hyper-parameters are given, we observe that all the metrics increase significantly which concludes that the model performs well after tuning.

The Below two tables justify our results:

Model(Classifiers)	Random Forest (With Hyper-parameters)
Metric	
Accuracy	0.76
Average Precision	0.82
F1 Score	0.67
Recall	0.56

Model(Classifiers)	Random Forest (Without Hyper-parameters)
Metric	
Accuracy	0.61
Average Precision	0.91
F1 Score	0.17
Recall	0.09

3.6. How are the misclassifications of the best performing model distributed?

- Naïve bayes gave the best accuracy of 0.82 and best F1 score of 0.79, when model is trained on the features which are not normalized.
- 100 documents are fetched from the databricks for which the above mentioned model is predicting the wrong class.
- For all the fetched documents, the probable reasons are identified because of which the model might be predicting the wrong class.
- The reasons are then grouped into broader categories.

Following tables show the categories, similar summaries, and their corresponding reasons of misclassification:

SUMMARY	Reason for misclassification	Similar summaries	Category(if any)
Kerberos Authentication Scheme	Summaries with keyword AUTHENTICATION have been predicted as BUG	setAuthPreemptive restricted to BASIC AuthScheme Windows specific implementation of the Digest auth scheme SPNEGO authentication scheme	AUTH/ AUTHENTICATION
Authorization credentials should be sent pre-emptively	Summaries with keyword AUTHENTICATION or AUTH have been predicted as BUG	Kerberos Authentication Scheme setAuthPreemptive restricted to BASIC AuthScheme Windows specific implementation of the Digest auth scheme	AUTH/ AUTHENTICATION

setAuthPreemptive restricted to BASIC AuthScheme	Summaries with keyword AUTHENTICATION or AUTH have been predicted as BUG	Kerberos Authentication Scheme setAuthPreemptive restricted to BASIC AuthScheme Windows specific implementation of the Digest auth scheme SPNEGO authentication scheme	AUTH/ AUTHENTICATION
Document the problem with MS impl of digest authentication with older JREs and stale connection check	Summaries with keyword AUTHENTICATION have been predicted as BUG	Kerberos Authentication Scheme setAuthPreemptive restricted to BASIC AuthScheme Windows specific implementation of the Digest auth scheme SPNEGO authentication scheme	AUTH/ AUTHENTICATION
Windows specific implementation of the Digest auth scheme	Summaries with keyword AUTHENTICATION or AUTH have been predicted as BUG	Kerberos Authentication Scheme setAuthPreemptive restricted to BASIC AuthScheme SPNEGO authentication scheme	AUTH/ AUTHENTICATION
SPNEGO authentication scheme	Summaries with keyword AUTHENTICATION have been predicted as BUG	Kerberos Authentication Scheme setAuthPreemptive restricted to BASIC AuthScheme Windows specific implementation of the Digest auth scheme	AUTH/ AUTHENTICATION

SUMMARY	**Reason for misclassification**	**Similar summaries**	**Category(if any)**
Sun hotspot compiler bug in 1.6.0_04/05 affects Lucene	Normally the summaries with keyword 'BUG' are 'BUG'	fix analyzer bugs found by MockTokenizer Bug with textfilters and classloaders	BUG
FieldCacheImpl's getCacheEntries() is buggy as it uses WeakHashMap incorrectly and leads to ConcurrentModExceptions	Normally the summaries with keyword 'BUG' are 'BUG'	fix analyzer bugs found by MockTokenizer Bug with textfilters and classloaders	BUG
bug form doesn't list latest version	Normally the summaries with keyword 'BUG' are 'BUG'	fix analyzer bugs found by MockTokenizer Bug with textfilters and classloaders	BUG
Add workaround for ICU bug in combination with Java7 to LuceneTestCase	Normally the summaries with keyword 'BUG' are 'BUG'	fix analyzer bugs found by MockTokenizer Bug with textfilters and classloaders	BUG
possible SynonymFilter bug: hudson fail	Normally the summaries with keyword 'BUG' are 'BUG'	fix analyzer bugs found by MockTokenizer Bug with textfilters and classloaders	BUG
Incorrect debug message in HttpMethodBase	Normally the summaries with keyword 'BUG' are 'BUG'	fix analyzer bugs found by MockTokenizer Bug with textfilters and classloaders	BUG

SUMMARY	Reason for misclassification	Similar summaries	Category(if any)
UUIDDocId cache does not work properly because of weakReferences in combination with new instance for combined indexreader	Cache related queries are both NUG and BUG. But they are being identified as BUG due to 'cache' and 'not' keyword. If 'NOT' keyword is not there then model is identifying them as NUG.	PersistentVersionManager contains grow-only cache - NUG	CACHE/NOT
Evict fixed NodePropBundle from cache	Cache related queries are both NUG and BUG. But they are being identified as BUG due to 'cache' and 'not' keyword. If 'NOT' keyword is not there then model is identifying them as NUG.	PersistentVersionManager contains grow-only cache - NUG	CACHE/NOT
Bundle cache is not cleared when *BundlePersistenceManager is closed	Cache related queries are both NUG and BUG. But they are being identified as BUG due to 'cache' and 'not' keyword. If 'NOT' keyword is not there then model is identifying them as NUG.	PersistentVersionManager contains grow-only cache - NUG	CACHE/NOT
Compressed entities are not being cached properly	Cache related queries are both NUG and BUG. But they are being identified as BUG due to 'cache' and 'not' keyword. If 'NOT' keyword is not there then model is identifying them as NUG.	PersistentVersionManager contains grow-only cache - NUG	CACHE/NOT
client cache does not respect 'Cache-Control: no-store' on requests	Cache related queries are both NUG and BUG. But they are being identified as BUG due to 'cache' and 'not' keyword. If 'NOT' keyword is not there then model is identifying them as NUG.	PersistentVersionManager contains grow-only cache - NUG	CACHE/NOT
cache module does not recognize equivalent URIs	Cache related queries are both NUG and BUG. But they are being identified as BUG due to 'cache' and 'not' keyword. If 'NOT' keyword is not there then model is identifying them as NUG.	PersistentVersionManager contains grow-only cache - NUG	CACHE/NOT
cache should not generate stale responses to requests explicitly requesting first-hand or fresh ones	Cache related queries are both NUG and BUG. But they are being identified as BUG due to 'cache' and 'not' keyword. If 'NOT' keyword is not there then model is identifying them as NUG.	PersistentVersionManager contains grow-only cache - NUG	CACHE/NOT

SUMMARY	Reason for misclassification	Similar summaries	Category(if any)
Enabling wire logging changes isEof/isStale behavior	Summaries with keyword CHANGES have been predicted as NUG	Ordering of methods in PostMethod changes behaviour	CHANGES
Ordering of methods in PostMethod changes behaviour	Summaries with keyword CHANGES have been predicted as NUG	Enabling wire logging changes isEof/isStale behavior	CHANGES
Changes from Session.move() to a top-level node aren't seen in a second session	Summaries with keyword CHANGES have been predicted as NUG		CHANGES

SUMMARY	Reason for misclassification	Similar summaries	Category(if any)
Do not consume the remaining response content if the connection is to be closed	summaries with keyword CLOSE have been identified as BUG	CompressingStoredFieldsReader should close the index file as soon as it has been read Add ProxBooleanTermQuery, like BooleanQuery but boosting when term occur "close" together (in proximity) in each document	CLOSE
CompressingStoredFieldsReader should close the index file as soon as it has been read	summaries with keyword CLOSE have been identified as BUG	Do not consume the remaining response content if the connection is to be closed Add ProxBooleanTermQuery, like BooleanQuery but boosting when term occur "close" together (in proximity) in each document	CLOSE
Add ProxBooleanTermQuery, like BooleanQuery but boosting when term occur "close" together (in proximity) in each document	summaries with keyword CLOSE have been identified as BUG	Do not consume the remaining response content if the connection is to be closed CompressingStoredFieldsReader should close the index file as soon as it has been read	CLOSE

SUMMARY	Reason for misclassification	Similar summaries	Category(if any)
IndexWriter does not properly account for the RAM consumed by pending deletes	Normally the summaries with keyword 'Deletes' are identified as 'BUG'	Buffered deletes under count RAM I/O exceptions can cause loss of buffered deletes Benchmark deletes.alg fails	DELETE
Large fetch sizes have potentially deleterious effects on VM memory requirements when using Oracle	Normally the summaries with keyword 'Deletes' are identified as 'BUG'	Buffered deletes under count RAM I/O exceptions can cause loss of buffered deletes Benchmark deletes.alg fails	DELETE

SUMMARY	Reason for misclassification	Similar summaries	Category(if any)
Update Tika dependency to 1.22	Summaries with DEPENDENCY as word, are mostly NUGs in actual data. Hence, model is learning the same.	Update derby dependency to 10.14.2.0 get rid of JSR 305 dependency webapp: update Tomcat dependency to 8.5.32 Update httpcore dependency to 4.4.10	DEPENDENCY
Update httpcore dependency to 4.4.12	Summaries with DEPENDENCY as word, are mostly NUGs in actual data. Hence, model is learning the same.	Update derby dependency to 10.14.2.0 get rid of JSR 305 dependency webapp: update Tomcat dependency to 8.5.32 Update httpcore dependency to 4.4.10	DEPENDENCY
update war-plugin dependency to 3.3.1	Summaries with DEPENDENCY as word, are mostly NUGs in actual data. Hence, model is learning the same.	Update derby dependency to 10.14.2.0 get rid of JSR 305 dependency webapp: update Tomcat dependency to 8.5.32	DEPENDENCY

		Update httpcore dependency to 4.4.10	

SUMMARY	Reason for misclassification	Similar summaries	Category(if any)
repository.xml: throw an exception on error	Mostly summaries with exception as a word are BUGs.	[PATCH] Fix possible Null Ptr exception in ConnectionFactory Lucene Query Exception: 'attempt to access a deleted document' NTCollectionConverterImpl throws a null pointer exception on update	EXCEPTION
ManageableCollectionUtil should not throw "unsupported" JcrMapping exception	Mostly summaries with exception as a word are BUGs.	[PATCH] Fix possible Null Ptr exception in ConnectionFactory Lucene Query Exception: 'attempt to access a deleted document' NTCollectionConverterImpl throws a null pointer exception on update	EXCEPTION
Change the error message in the exception at URIUtils#rewriteURI	Mostly summaries with exception as a word are BUGs.	[PATCH] Fix possible Null Ptr exception in ConnectionFactory Lucene Query Exception: 'attempt to access a deleted document' NTCollectionConverterImpl throws a null pointer exception on update	EXCEPTION
davex: preserve cause in exceptions and log affected URI	Mostly summaries with exception as a word are BUGs.	[PATCH] Fix possible Null Ptr exception in ConnectionFactory Lucene Query Exception: 'attempt to access a deleted document' NTCollectionConverterImpl throws a null pointer exception on update	EXCEPTION

SUMMARY	Reason for misclassification	Similar summaries	Category(if any)
contrib-spatial java.lang.UnsupportedOperationException on QueryWrapperFilter.getDocIdSet	Model is unable to identify a specific type of exception	Contrib RMI: NotSerializableException	EXCEPTIONTYPE
Contrib RMI: NotSerializableException	Model is unable to identify a specific type of exception	contrib-spatial java.lang.UnsupportedOperationException on QueryWrapperFilter.getDocIdSet	EXCEPTIONTYPE
IndexFormatTooNewException using Lucene 4.4 after optimize	Model is unable to identify a specific type of exception		EXCEPTIONTYPE
Log exception in AbstractDataStore.getReferenceFromIdentifier()	Model is unable to identify a specific type of exception		EXCEPTIONTYPE

SUMMARY	Reason for misclassification	Similar summaries	Category(if any)
TestCollectionUtil fails on IBM JRE	Normally the summaries with keyword 'FAIL' are 'BUG'	TestIndexWriter.testCommitThreadSafety fails on realtime_search branch new QueryParser fails to set AUTO REWRITE for multi-term queries	FAIL
UAX29URLEmailTokenizer fails to recognize emails as such when the mailto: scheme is prepended	Normally the summaries with keyword 'FAIL' are 'BUG'	TestIndexWriter.testCommitThreadSafety fails on realtime_search branch new QueryParser fails to set AUTO REWRITE for multi-term queries	FAIL
Unit tests TestBackwardsCompatibility and TestIndexFileDeleter might fail depending on JVM	Normally the summaries with keyword 'FAIL' are 'BUG'	TestIndexWriter.testCommitThreadSafety fails on realtime_search branch new QueryParser fails to set AUTO REWRITE for multi-term queries	FAIL
ReorderReferenceableSNSTest failure	Normally the summaries with keyword 'FAIL' are 'BUG'	TestIndexWriter.testCommitThreadSafety fails on realtime_search branch new QueryParser fails to set AUTO REWRITE for multi-term queries	FAIL
Build fails on system without X	Normally the summaries with keyword 'FAIL' are 'BUG'	TestIndexWriter.testCommitThreadSafety fails on realtime_search branch new QueryParser fails to set AUTO REWRITE for multi-term queries	FAIL
Re-index fails on corrupt bundle	Normally the summaries with keyword 'FAIL' are 'BUG'	TestIndexWriter.testCommitThreadSafety fails on realtime_search branch new QueryParser fails to set AUTO REWRITE for multi-term queries	FAIL
AbstractQueryTest.evaluateResultOrder() should fail if workspace does not contain enough content	Normally the summaries with keyword 'FAIL' are 'BUG'	TestIndexWriter.testCommitThreadSafety fails on realtime_search branch new QueryParser fails to set AUTO REWRITE for multi-term queries	FAIL
Provide fail-over for multi-home remote servers (if one server in a farm goes down)	Normally the summaries with keyword 'FAIL' are 'BUG'	TestIndexWriter.testCommitThreadSafety fails on realtime_search branch new QueryParser fails to set AUTO REWRITE for multi-term queries	FAIL

SUMMARY	Reason for misclassification	Similar summaries	Category(if any)
IndexWriter does not do the right thing when a Thread is interrupt()'d	Normally the summaries with keyword 'interrupt 'or 'not' identifying are 'BUG'	MultiThreadedHttpConnectionManager does not properly respond to thread interrupts Can not interrupt a request with thread.interrupt (hystrix) Interrupt flag is not preserved where InterruptedException is caught	INTERRUPT/NOT

SUMMARY	Reason for misclassification	Similar summaries	Category(if any)
Query parser builds invalid parse tree	Summaries with keyword INVALID along with combination of other words have been idntified as NUG	Invalid project url in pom.xml	INVALID
Invalid project url in pom.xml	Summaries with keyword INVALID along with combination of other words have been idntified as NUG	Query parser builds invalid parse tree	INVALID

SUMMARY	Reason for misclassification	Similar summaries	Category(if any)
Enable the use of NoMergePolicy and NoMergeScheduler by Benchmark	No negative keyword such as should not, and only positive keywords such as enable and policy names. Summaries with only merge and words with positive conjuctions are classified as NUG, thus model is learning the same. However, summaries which have merge along with the negative keywords are classified as BUG. Thus, model is predicting the same way	Performance improvement for SegmentMerger.mergeNorms() - NUG IndexWriter.setInfoStream should percolate down to mergePolicy & mergeScheduler - NUG PostingsConsumer#merge does not call finishDoc - BUG MockRandomMergePolicy optimizes segments not in the Set passed in - BUG merge LuceneTestCase and LuceneTestCaseJ4 - NUG	MERGE/NOT
Merging implemented by codecs must catch aborted merges	No negative keyword such as should not, and only positive keywords such as enable and policy names. Summaries with only merge and words with positive conjuctions are classified as NUG, thus model is learning the same. However, summaries which have merge along with the negative keywords are classified as BUG. Thus, model is predicting the same way	Performance improvement for SegmentMerger.mergeNorms() - NUG IndexWriter.setInfoStream should percolate down to mergePolicy & mergeScheduler - NUG PostingsConsumer#merge does not call finishDoc - BUG MockRandomMergePolicy optimizes segments not in the Set passed in - BUG merge LuceneTestCase and LuceneTestCaseJ4 - NUG	MERGE/NOT

Some small fixes after the flex merge...	No negative keyword such as should not, and only positive keywords such as enable and policy names. Summaries with only merge and words with positive conjuctions are classified as NUG, thus model is learning the same. However, summaries which have merge along with the negative keywords are classified as BUG. Thus, model is predicting the same way	Performance improvement for SegmentMerger.mergeNorms() - NUG IndexWriter.setInfoStream should percolate down to mergePolicy & mergeScheduler - NUG PostingsConsumer#merge does not call finishDoc - BUG MockRandomMergePolicy optimizes segments not in the Set passed in - BUG merge LuceneTestCase and LuceneTestCaseJ4 - NUG	MERGE/NOT
CMS merge throttling is not aggressive enough	No negative keyword such as should not, and only positive keywords such as enable and policy names. Summaries with only merge and words with positive conjuctions are classified as NUG, thus model is learning the same. However, summaries which have merge along with the negative keywords are classified as BUG. Thus, model is predicting the same way	Performance improvement for SegmentMerger.mergeNorms() - NUG IndexWriter.setInfoStream should percolate down to mergePolicy & mergeScheduler - NUG PostingsConsumer#merge does not call finishDoc - BUG MockRandomMergePolicy optimizes segments not in the Set passed in - BUG merge LuceneTestCase and LuceneTestCaseJ4 - NUG	MERGE/NOT
Forced merges	No negative keyword such as should not, and only positive keywords such as enable and policy names. Summaries with only merge and words with positive conjuctions are classified as NUG, thus model is learning the same. However, summaries which have merge along with the negative keywords are classified as BUG. Thus, model is predicting the same way	Performance improvement for SegmentMerger.mergeNorms() - NUG IndexWriter.setInfoStream should percolate down to mergePolicy & mergeScheduler - NUG PostingsConsumer#merge does not call finishDoc - BUG MockRandomMergePolicy optimizes segments not in the Set passed in - BUG merge LuceneTestCase and LuceneTestCaseJ4 - NUG	MERGE/NOT

SUMMARY	**Reason for misclassification**	**Similar summaries**	**Category(if any)**
Fix StandardAnalyzer to not mis-identify HOST as ACRONYM by default	Normally the summaries with keyword 'mis-identify' or not identifying are 'BUG'	InetAddressUtils.isIPv6Address is not identifying my IPv6 Address EnwikiContentSource does not properly identify the name/id of the Wikipedia article	MISS
Missing possibility to supply custom FieldParser when sorting search results	Normally the summaries with keyword 'mis' or 'missing' or not identifying are 'BUG'	BundleDbPersistenceManager consistencyFix doesn't fix	MISS

		missing non system childnode entries of the root node	
when checking tvx/fdx size mismatch, also include whether the file exists	Normally the summaries with keyword 'mis' or 'missing' or 'mismatch' or not identifying are 'BUG'	HttpMethodBase: Port mismatch in URL for redirect to absolute location CircularRedirectException is falsely thrown on URI case mismatch	MISS
Log path of missing node when re-indexing fails	Normally the summaries with keyword 'mis' or 'missing' or 'mismatch' or fail or 'not identifying' are 'BUG'	HttpClient does not retry authentication when multiple challenges are present if the primary one fails	MISS
Jcr-Server: Avoid xml parsing if request body is missing	Normally the summaries with keyword 'mis' or 'missing' or not identifying are 'BUG'	BundleDbPersistenceManager consistencyFix doesn't fix missing non system childnode entries of the root node	MISS
SUMMARY	**Reason for misclassification**	**Similar summaries**	**Category(if any)**
TestIndexModifier.testIndexWithThreads is not valid?	Summaries which include 'NOT' and with the rest of the words which does not occur much frequently in the other documents, are being marked as BUG by the model.	DefaultRedirectHandler not resolving relative location URI wrt the request URI The PostMethod did not bring back response headers from proxy servers	NOT
maxDoc should be explicitly stored in the index, not derived from file length	Summaries which include 'NOT' and with the rest of the words which does not occur much frequently in the other documents, are being marked as BUG by the model.	DefaultRedirectHandler not resolving relative location URI wrt the request URI The PostMethod did not bring back response headers from proxy servers	NOT
ServerQuery does not use RemoteAdapterFactory for creating ServerQueryResult	Summaries which include 'NOT' and with the rest of the words which does not occur much frequently in the other documents, are being marked as BUG by the model.	DefaultRedirectHandler not resolving relative location URI wrt the request URI The PostMethod did not bring back response headers from proxy servers	NOT
Jackrabbit does not allow concurrent reads to the data store if copyWhenReading=false	Summaries which include 'NOT' and with the rest of the words which does not occur much frequently in the other documents, are being marked as BUG by the model.	DefaultRedirectHandler not resolving relative location URI wrt the request URI The PostMethod did not bring back response headers from proxy servers	NOT
Node.setPrimaryNodeType should only redefine child-definitions that are not covered by the new effective nt	Summaries which include 'NOT' and with the rest of the words which does not occur much frequently in the other documents, are being marked as BUG by the model.	DefaultRedirectHandler not resolving relative location URI wrt the request URI The PostMethod did not bring back response headers from proxy servers	NOT
ItemSaveOperation should not swallow stacktrace	Summaries which include 'NOT' and with the rest of the words which does not occur much frequently in the other documents, are being marked as BUG by the model.	DefaultRedirectHandler not resolving relative location URI wrt the request URI The PostMethod did not bring back response headers from proxy servers	NOT
DefaultHttpRequestRetryHandler is not handling PUT as an idempotent method for retries, though RFC2616	Summaries which include 'NOT' and with the rest of the words which does not occur much frequently in the other documents, are being marked as BUG by the model.	DefaultRedirectHandler not resolving relative location URI wrt the request URI The PostMethod did not bring back response headers from proxy servers	NOT

section 9.1.2 clearly defines it to be one.			
MultipartPostMethod does not check for error messages	Summaries which include 'NOT' and with the rest of the words which does not occur much frequently in the other documents, are being marked as BUG by the model.	DefaultRedirectHandler not resolving relative location URI wrt the request URI The PostMethod did not bring back response headers from proxy servers	NOT
Fix junitcompat tests (so that they're not triggered when previous errors occur)	Summaries which include 'NOT' and with the rest of the words which does not occur much frequently in the other documents, are being marked as BUG by the model.	DefaultRedirectHandler not resolving relative location URI wrt the request URI The PostMethod did not bring back response headers from proxy servers	NOT

SUMMARY	**Reason for misclassification**	**Similar summaries**	**Category(if any)**
Return null for optional configuration elements	Summaries with keyword NULL have been predicted as BUG.	DISI.iterator() should never return null. Handle Returning Null consistantly	NULL
Handle Returning Null consistantly	Summaries with keyword NULL have been predicted as BUG.	Return null for optional configuration elements DISI.iterator() should never return null.	NULL
DISI.iterator() should never return null.	Summaries with keyword NULL have been predicted as BUG.	Return null for optional configuration elements Handle Returning Null consistantly	NULL

SUMMARY	**Reason for misclassification**	**Similar summaries**	**Category(if any)**
NumericRangeQuery errors with endpoints near long min and max values	Summaries including NUMERICRANGEQUERY are BUG as well as NUG. It depends upon the rest of the keywords. Here contonation of rest of the words seem positive. Thus, model is predicting them as BUG.	Support open-ended NumericRangeQuery in XmlQueryParser - NUG NumericUtils.floatToSortableIn t/ doubleToSortableLong does not sort certain NaN ranges correctly and NumericRangeQuery produces wrong results for NaNs with half-open ranges - BUG	NUMERICRANGEQUERY /NOT
NumericRangeQuery. NumericRangeTermsEnum sometimes seeks backwards	Summaries including NUMERICRANGEQUERY are BUG as well as NUG. It depends upon the rest of the keywords. Here contonation of rest of the words seem positive. Thus, model is predicting them as BUG.	Support open-ended NumericRangeQuery in XmlQueryParser - NUG NumericUtils.floatToSortableIn t/ doubleToSortableLong does not sort certain NaN ranges correctly and NumericRangeQuery produces wrong results for NaNs with half-open ranges - BUG	NUMERICRANGEQUERY /NOT

SUMMARY	Reason for misclassification	Similar summaries	Category(if any)
IndexSearcher fails to pass docBase to Collector when using ExecutorService	Summaries with keyword PASS or PASSING have been predicted as NUG.	pass computed args to surefire/failsafe invocations ISO8601: add convenience methods that do not require passing a Calendar, also support short format without ms information	PASS/PASSING
pass computed args to surefire/failsafe invocations	Summaries with keyword PASS or PASSING have been predicted as NUG.	IndexSearcher fails to pass docBase to Collector when using ExecutorService ISO8601: add convenience methods that do not require passing a Calendar, also support short format without ms information	PASS/PASSING
ISO8601: add convenience methods that do not require passing a Calendar, also support short format without ms information	Summaries with keyword PASS or PASSING have been predicted as NUG.	IndexSearcher fails to pass docBase to Collector when using ExecutorService pass computed args to surefire/failsafe invocations	PASS/PASSING

SUMMARY	Reason for misclassification	Similar summaries	Category(if any)
Behavior on hard power shutdown	Normally the summaries with keyword 'Shutdown' are 'BUG'	TransientRepository does not shutdown if first login fails Jackrabbit logs a NullPointerException on shutdown if the version manager wasn't initialized	SHUT DOWN

SUMMARY	Reason for misclassification	Similar summaries	Category(if any)
MoreLikeThis ignores custom similarity	Normally the summaries with keyword 'Similarity' are 'NUG'	Similarity can only be set per index, but I may want to adjust scoring behaviour at a field level Deprecate SimilarityDelegator and Similarity.lengthNorm New tool for reseting the (length)norm of fields after changing Similarity Let users set Similarity for MoreLikeThis	SIMILARITY
MultiPhraseQuery sums its own idf instead of Similarity.	Normally the summaries with keyword 'Similarity' are 'NUG'	Similarity can only be set per index, but I may want to adjust scoring behaviour at a field level Deprecate SimilarityDelegator and Similarity.lengthNorm New tool for reseting the (length)norm of fields after changing Similarity Let users set Similarity for MoreLikeThis	SIMILARITY

SUMMARY	Reason for misclassification	Similar summaries	Category(if any)
TCK: NamespaceRegistryTest#testUnregisterNamespaceExceptions doesn't fail if expected exception isn't thrown	Model is unable to learn on the word TCK. TCK belong to the test cases and are NUG. However, model is predicting it as BUG because of the other keywords.		TCK
TCK: DocumentViewImportTest does not call refresh after direct-to-workspace import	Model is unable to learn on the word TCK. TCK belong to the test cases and are NUG. However, model is predicting it as BUG because of the other keywords.		TCK
TCK: PredicatesTest does not respect testroot configuration property	Model is unable to learn on the word TCK. TCK belong to the test cases and are NUG. However, model is predicting it as BUG because of the other keywords.		TCK
TCK: NodeReadMethodsTest.testGetName fails with NPE if 'testroot' has no child node	Model is unable to learn on the word TCK. TCK belong to the test cases and are NUG. However, model is predicting it as BUG because of the other keywords.		TCK

SUMMARY	Reason for misclassification	Similar summaries	Category(if any)
TestParallelTermEnum fails with Sep codec	Summaries with 'test' along with some other keywords such as 'fail' are NUG in actual dataset. That's why model must be predicting this wrong	contrib/benchmark unit tests TestThreadSafety.testLazyLoadThreadSafety test failure Improve contrib/benchmark for testing near-real-time search performance	TEST/FAIL
Random Test Failure org.apache.lucene. TestExternalCodecs. testPerFieldCodec (from TestExternalCodecs)	Summaries with 'test' along with some other keywords such as 'fail' are NUG in actual dataset. That's why model must be predicting this wrong	contrib/benchmark unit tests TestThreadSafety.testLazyLoadThreadSafety test failure Improve contrib/benchmark for testing near-real-time search performance	TEST/FAIL
Several test cases fail when declaring nt:base / nt:hierarchy node types as 'abstract'	Summaries with 'test' along with some other keywords such as 'fail' are NUG in actual dataset. That's why model must be predicting this wrong	contrib/benchmark unit tests TestThreadSafety.testLazyLoadThreadSafety test failure Improve contrib/benchmark for testing near-real-time search performance	TEST/FAIL

Test case failure for testConnTimeout	Summaries with 'test' along with some other keywords such as 'fail' are NUG in actual dataset. That's why model must be predicting this wrong	contrib/benchmark unit tests TestThreadSafety.testLazyLoadThreadSafety test failure Improve contrib/benchmark for testing near-real-time search performance	TEST/FAIL
Fail smoketester if there is LUCENE_XXXX or SOLR_XXXX in Changes.txt	Summaries with 'test' along with some other keywords such as 'fail' are NUG in actual dataset. That's why model must be predicting this wrong	contrib/benchmark unit tests TestThreadSafety.testLazyLoadThreadSafety test failure Improve contrib/benchmark for testing near-real-time search performance	TEST/FAIL
FieldCache related test failure	Summaries with 'test' along with some other keywords such as 'fail' are NUG in actual dataset. That's why model must be predicting this wrong	contrib/benchmark unit tests TestThreadSafety.testLazyLoadThreadSafety test failure Improve contrib/benchmark for testing near-real-time search performance	TEST/FAIL
jackrabbit-data: occasional test failures in TestLocalCache.testAutoPurge	Summaries with 'test' along with some other keywords such as 'fail' are NUG in actual dataset. That's why model must be predicting this wrong	contrib/benchmark unit tests TestThreadSafety.testLazyLoadThreadSafety test failure Improve contrib/benchmark for testing near-real-time search performance	TEST/FAIL
Improve o.a.j.jcr2dav.RepositoryStubImpl to test with custom servlet path mapping	Summaries with 'test' along with some other keywords such as 'fail', 'improve' are NUG in actual dataset. That's why model must be predicting this wrong	contrib/benchmark unit tests TestThreadSafety.testLazyLoadThreadSafety test failure Improve contrib/benchmark for testing near-real-time search performance	TEST/FAIL
EOL Jackrabbit 2.18	Summaries with 'test' along with some other keywords such as 'fail' are NUG in actual dataset. That's why model must be predicting this wrong	contrib/benchmark unit tests TestThreadSafety.testLazyLoadThreadSafety test failure Improve contrib/benchmark for testing near-real-time search performance	TEST/FAIL

SUMMARY	Reason for misclassification	Similar summaries	Category(if any)
Update monitor is not released	Summries with keyword UPDATE have been predicted as NUG	Update Apache Lucene Update Jackrabbit trunk to Oak 1.18.0 Simplify IndexWriter.commitMergedDeletesAndUpdates update Apache parent pom to version 21	UPDATE
Simplify IndexWriter. commitMergedDeletesAndUpdates	Summries with keyword UPDATE have been predicted as NUG	Update Apache Lucene Update Jackrabbit trunk to Oak 1.18.0 update Apache parent pom to version 21 Update monitor is not released	UPDATE
update Apache parent pom to version 21	Summries with keyword UPDATE have been predicted as NUG	Update Apache Lucene Update Jackrabbit trunk to Oak 1.18.0 Simplify IndexWriter.commitMergedDeletesAndUpdates Update monitor is not released	UPDATE
Update Jackrabbit trunk to Oak 1.18.0	Summaries with keyword UPDATE has been predicted as NUG.	Update Apache Lucene Simplify IndexWriter.commitMergedDeletesAndUpdates update Apache parent pom to version 21 Update monitor is not released	UPDATE
Update Apache Lucene	Summaries with keyword UPDATE has been predicted as NUG.	Update Jackrabbit trunk to Oak 1.18.0 Simplify IndexWriter.commitMergedDeletesAndUpdates update Apache parent pom to version 21 Update monitor is not released	UPDATE
Wrong trailing index calculation in PatternReplaceCharFilter	Summaries with keyword WRONG have been predicted as BUG	Wrong schemaObjectPrefix parameter in default repository.xml Wrong method signatures in AbstractHttpClient	WRONG
Wrong schemaObjectPrefix parameter in default repository.xml	Summaries with keyword WRONG have been predicted as BUG	Wrong trailing index calculation in PatternReplaceCharFilter Wrong method signatures in AbstractHttpClient	WRONG
Wrong method signatures in AbstractHttpClient	Summaries with keyword WRONG have been predicted as BUG	Wrong trailing index calculation in PatternReplaceCharFilter Wrong schemaObjectPrefix parameter in default repository.xml	WRONG
ResidualProperties Converter uses wrong AtomicType Converter on update	Summries with keyword UPDATE have been predicted as NUG		UPDATE

4. DISCUSSION

a) Naive Bayes Algorithm Real World Application (Thought and written by Taruneesh)

Let's say we have data on 1000 pieces of fruit. The fruit being a Banana, Orange or some other fruit and imagine we know 3 features of each fruit, whether it's long or not, sweet or not and yellow or not, as displayed in the table below:

Fruit	Long	Sweet	Yellow	Total
Banana	400	350	450	500
Orange	0	150	300	300
Other	100	150	50	200
Total	500	650	800	1000

So, from the table what do we already know?

- 50% of the fruits are bananas
- 30% are oranges
- 20% are other fruits

Based on our training set we can also say the following:

- From 500 bananas 400 (0.8) are Long, 350 (0.7) are Sweet and 450 (0.9) are Yellow
- Out of 300 oranges, 0 are Long, 150 (0.5) are Sweet and 300 (1) are Yellow
- From the remaining 200 fruits, 100 (0.5) are Long, 150 (0.75) are Sweet and 50 (0.25) are Yellow.

Which should provide enough evidence to predict the class of another fruit as it's introduced.

So let's say we're given the features of a piece of fruit and we need to predict the class. If we're told that the additional fruit is Long, Sweet and Yellow, we can classify it using the following formula and subbing in the values for each outcome, whether it's a Banana, an Orange or Other Fruit. The one with the highest probability (score) being the winner.

P(c/x) = (P(x/c) * P(c)) / P(x)

From this formula, we calculate the Probabilities for Banana, Orange and Other fruits.

In this case, based on the higher score (0.252 for banana) we can assume this Long, Sweet and Yellow fruit is in fact, a Banana.

Real time Prediction: Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.

Multi class Prediction: This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.

Text classification/ Spam Filtering/ Sentiment Analysis: Naïve Bayes classifiers mostly used in Text Classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam Filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments).

b) Random Forest Algorithm Real World Application (Thought and written by Sanyam)

Random Forest algorithm is a supervised classification algorithm. It uses a tree-like graph to show the possible consequences. If you input a training dataset with targets and features into the decision tree, it will formulate some set of rules. These rules are then used to perform predictions.

There are several applications where a RF analysis can be applied. Following are some of the examples:

- Banking Sector: The banking sector consists of most users. There are many loyal customers and fraud customers. With the RF algorithm we can easily determine whether the customer is a loyal or fraud. A system uses a set of a random algorithm which identifies the fraud transactions by a series of the pattern.
- Medicines: Medicines needs a complex combination of specific chemicals. Thus, to identify the great combination in the medicines, Random forest can be used. With the help of machine learning algorithm, it has become easier to detect and predict the drug sensitivity of a medicine. Also, it helps to identify the patient's disease by analysing the patient's medical record.
- Stock Market: RF algorithm also plays role in the stock market analysis. The behaviour of the stock market can be analysed with the RF algorithm and thus expected loss or profit can be predicted for a particular stock.
- E-Commerce: When you will find it difficult to recommend or suggest what type of products your customer should see. This is where you can use a random forest algorithm. Using a machine learning system, you can suggest the products which will be more likely for a customer. Using a certain pattern and following the product's interest of a customer, you can suggest similar products to your customers.

c) Linear SVM Algorithm Real World Application (thought and written by Neha)

The linear SVM is a standard method for large-scale classification tasks. The linear SVMs algorithm outputs an SVM model. A support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data points of any class (so-called functional margin).

Following are some of the real-world examples where a LSVM analysis can be applied:

1. Face Detection

- SVM classify parts of the image as a face and non-face and create a square boundary around the face.
- It classifies the parts of the image as face and non-face. It contains training data of n x n pixels with a two-class face (+1) and non-face (-1)
- Then it extracts features from each pixel as face or non-face. Creates a square boundary around faces on the basis of pixel brightness and classifies each image by using the same process.

2. Classification of images

Use of SVMs provides better search accuracy for image classification. It provides better accuracy in comparison to the traditional query-based searching techniques.

3. Text and hypertext categorization

SVMs allow Text and hypertext categorization for both inductive and transudative models. They use training data to classify documents into different categories. It categorizes on the basis of the score generated and then compares with the threshold value. It Uses training data to classify documents into different categories such as news articles, e-mails, and web pages Examples:

- Classification of news articles into "business" and "Movies"
- Classification of web pages into personal home pages and others

For each document, calculate a score and compare it with a predefined threshold value. When the score of a document surpasses threshold value, then the document is classified into a definite category. If it does not surpass threshold value then consider it as a general document. Classify new instances by computing score for each document and comparing it with the learned threshold.

4. Handwriting Recognition

SVMs are also used to recognize handwritten characters used widely.

5. Protein Fold and Remote Homology Detection

Protein remote homology detection is a key problem in computational biology. Supervised learning algorithms on SVMs are one of the most effective methods for remote homology detection. The performance of these methods depends on how the protein sequences modelled. The method used to compute the kernel function between them.

5. CONCLUSION

- We were able to replicate the trained models given in the research paper[2].
- We used machine learning techniques to automatically classify issue reports into bug and non-bug categories.
- We took 3 datasets from [1], collated them and further added 1000 new issue reports which were manually classified.
- Our results indicate that best F-measure score are 0.79 and 0.73 for Naïve Bayes and Linear SVC model respectively.
- For, all cases Linear SVM performs better than Naïve Bayes. This is interesting given that we perform minimal pre-processing on the data and run our experiments and collected issue reports from three large open-source projects such as HTTPCLIENT, LUCENE and JACKRABBIT.
- Our results suggest it might be useful to classify issue reports using simple classifiers before further analysis by developers.
- Other research directions include exploring whether more attributes from the re- ports lead to better classification, tuning the classifiers to a greater degree, expanding the datasets and classifiers, and determining how to select a training subset in practice when faced with a very large repository of issue reports.

REFERENCES

[1] https://www.st.cs.uni-saarland.de/softevo/bugclassify/

[2] Automated Classification of Software Issue Reports Using Machine Learning Techniques: An Empirical Study

[3] http://hc.apache.org/httpclient-3.x/

[4] https://lucene.apache.org/

[5] http://jackrabbit.apache.org/jcr/index.html

[6] Herzig K, Just S, Zeller A (2013) It's not a bug, it's a feature: how misclassification impacts bug prediction. In: Proceedings of the 2013 International Conference on Software Engineering (ICSE'13), IEEE, pp 392–401

[7] Sebastiani F (2002) Machine learning in automated text categorization. ACM Computing Surveys 34(1):1–47

[8] https://medium.com/@Synced/how-random-forest-algorithm-works-in-machine-learning-3c0fe15b6674

[9] https://towardsdatascience.com/all-about-naive-bayes-8e13cef044cf

[10] https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76

[11] https://www.stata.com/meeting/canada18/slides/canada18_Zou.pdf

[12] https://data-flair.training/blogs/applications-of-svm/