Background

Software bug repositories are valuable assets for software maintenance activity. Users reports bugs to these repositories. Users of these repositories are usually non-technical and cannot assign correct class to these bugs and sometimes they report some bugs which are not actually bug. For these QA managers need to classify the reports. But these kind of activities are very tedious and time consuming. Due to these automatic bug classification is very important in software industry.

Bug Classification

A software bug means an error in a system for which it shows unexpected behaviour. Bug classification means classify the software bugs into different categories. We can classify as bug, non-bug or different types of bugs like logical bug, UI bug, security bug etc. There are different software bug repositories where the bug are reported. The QA manager then classify the bugs and assign those to corresponding developers.

Bug Repository

To report the bugs there exist several open source bug tracking system also known as bug repository like as Bugzilla, Jira, Mozilla, Eclipse etc. Bugs are reported in these repositories. To report a bug we need to provide some information like title, description, severity, priority etc. and a unique identification number is assigned to this report.

Stop Words Elimination and Stemming

For text classification stop words elimination is very important. Stop words are that words which has no significant meaning but it increase complexity in classification. There is no universal stop words list it depends on application. Stemming is the process by which we convert a word to its original form like converting the words argue, argued, argues, arguing and argus to its root form argu.

Machine Learning Techniques

A classifier is a function that assign a label from finite set of classes to observations. There are three types of machine learning techniques are available for classification-

- Unsupervised learning
- Supervised learning

- Reinforcement learning

Unsupervised learning like clustering classify the data based on some fitness or cost function like similarity or distance. Unsupervised learnings are most popular because there is no need to label the data, but it is very hard in some context to interpret the resulting classification and linking the classification with characteristics of data.

There are several clustering techniques like -

- K-Means
- Hierarchical Clustering
- Gaussian Mixture Model
- Hidden Markob Model
- Self-Organizing Map

Supervised learning means deducing a function from labelled training data. Each training data is consists of input vector and supervisory signal. The supervised learning algorithms analyse the training data and deduce a function which is used to know the label of new data. This function is built by maximizing the gain or minimizing the cost. Some supervised learning algorithms are –

- Naïve Bayes
- Decision Tree
- Support Vector Machine
- Linear Regression etc.

Cosine Similarity

Cosine similarity is a measure of similarity between two vectors of an inner product space that measure the cosine of the angle between them. The cosine of zero degree is one and it is less than one for any other angle.

$$\mathrm{COS}\Theta = a.b/\sqrt{a^2}.\sqrt{b^2}$$

Steps of calculating cosine similarity between two texts:

Step1: Identify all distinct words in two texts

Step 2: Identify all frequency of occurrences of these words in both text and treat it as vector

Step 3: Apply cosine function

Suppose we have two texts

Text1: Julie loves me more than linda loves me

Text2: Jana likes me more than Julie loves me

For the given texts, we can use the following vector representation

Distinct words	Frequency in text1	Frequency in text2
Julie	1	1
Loves	2	1
Me	2	2
More	1	1
Than	1	1
Linda	1	0
Jana	0	1
Likes	0	1

Vector A = [1, 2, 2, 1, 1, 1, 0, 0]

Vector B = [1, 1, 2, 1, 1, 0, 1, 1]

So, a.b = 9 and √(a^2) = 12, √(b^2) = 10

Cosine similarity = 0.822

A Similarity Histogram based Sentence Clustering

This clustering approach is an incremental dynamic method of building the sentence clusters. In incremental clustering approaches, data objects are processed one at a time and data objects are incrementally assigned to their respective clusters while they progress. The main concept for the similarity histogram-based clustering method is to keep each cluster as coherent as possible and a degree of coherency in a cluster at any time is monitored with a Cluster Similarity Histogram. Cluster similarity histogram is a concise representation of the set of pair-wise sentence similarities distribution in a cluster. A histogram consists of a number of bins that correspond to fixed similarity value intervals. Each bin height represents the count of pair-wise sentence similarities in the corresponding interval.

A perfect cluster would have a histogram, where all pair-wise similarities are of maximum value and the histogram would have the right-most bin representing all similarities. On the other hand, a loose cluster would have histogram where all pair-wise similarities are minimum and the similarities would tend to be counted in the lower bins.

To prevent selection of the redundant sentences in to the summary, we should be careful to keep each cluster as coherent as possible. In other words, the objective would be to maximize the number of similarities in the high similarity intervals. To achieve this goal in an incremental fashion, we should judge the effect of adding a new sentence to a certain cluster. If the inclusion of this sentence is going to degrade the distribution of the similarities in the clusters very much, it should not be added, otherwise it is added. But assignment of sentences to clusters based on similarity distribution enhancement may create problem with the perfect clusters. The sentence may be rejected by the perfect cluster even if it has high similarity to most of the sentences in the cluster. So, the quality of a similarity histogram representing cluster cohesiveness is judged by calculating the ratio of the count of similarities above a certain similarity threshold to the total count of similarities. The higher this ratio, the more coherent the cluster is.

If n be the number of the documents in a cluster, the number of pair-wise similarities in the cluster is n(n+1)/2. Let S={sim_i ,: i=1,..., m} be the set of pair-wise sentence similarities in a cluster, where m = n(n+1)/2. The histogram of the similarities in the cluster is given by

H={hi , i=1, ..., n_b}

hi = count(sim_k) $sim_{li} <= sim_k < = sim_{ui}$,

where,

n_b: the number of bins in a histogram

hi : the count of sentence similarities in bin i

sim_{li}: the lower similarity bound of bin i

sim_{ui}: the upper similarity bound of bin i

The histogram ratio of a cluster is calculated using the following formula:

$$HR = \sum_{i=T}^{nb} hi \,/ \sum_{j=1}^{nb} hj$$

$$T = S_T * n_b$$

S_T = the similarity threshold

T = bin number corresponding to the similarity threshold

Algorithm for clustering:

1) Say, Clist is a cluster list which is initially empty

2) Convert all the input documents in a collection of sentences, Slist. Each sentence S in Slist is indexed by the document number and the sentence number

3) For each sentence S in Slist do for each cluster c in Clist do

3.1 Store the histogram ratio of the cluster c to a variable before adding s to c, that is,

$HR_{old} = HR_c$

3.2 Simulate adding s to c to check whether addition of s to c would severely degrade or improve the histogram ratio (coherence) of c. Let the simulated histogram ratio be HRs

```
        3.3  If ($HR_s \geq HR_{old}$) or (( $HR_s \geq HR_L$ )and ($HR_{old} - HR_s < eps$ ))
        then add s to c and exit from the inner loop to avoid any chance of assigning the same sentence to more than one cluster.
        3.4 If s is not added to any cluster, then create a new cluster c, add s to c and add c to Clist.
    end for (inner loop)
end for (outer loop)
```

Naïve Bayes Classifier

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. It is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the colour, roundness, and diameter features.

In machine learning we are often interested in selecting the best hypothesis (h) given data (d). In a classification problem, our hypothesis (h) may be the class to assign for a new data instance (d). One of the easiest ways of selecting the most probable hypothesis given the data that we have that we can use as our prior knowledge about the problem. Bayes' Theorem provides a way that we can calculate the probability of a hypothesis given our prior knowledge.

Bayes' Theorem is stated as:

$$P(h|d) = (P(d|h) * P(h)) / P(d)$$

Where,

P(h|d) is the probability of hypothesis h given the data d. This is called the posterior probability.

P(d|h) is the probability of data d given that the hypothesis h was true.

P(h) is the probability of hypothesis h being true (regardless of the data). This is called the prior probability of h.

P(d) is the probability of the data (regardless of the hypothesis).

You can see that we are interested in calculating the posterior probability of P(h|d) from the prior probability p(h) with P(D) and P(d|h).

After calculating the posterior probability for a number of different hypotheses, you can select the hypothesis with the highest probability. This is the maximum probable hypothesis and may formally be called the maximum a posteriori (MAP) hypothesis.

This can be written as –

$$MAP(h) = max(P(h|d))$$

Literature Review

Classification of software bugs using bug attribute similarity (CLUBAS) is proposed by N. K. Nagwani et. al. [1]. The proposed work can be divided into five steps- data pre-processing, text clustering, frequent term calculations, taxonomic terms mapping techniques and performance evaluation. In the first step data are pre-processed by eliminating stop words and applying stemming over the textual bug attributes. In the second step text clusters are created using textual similarity between the attributes summary and description for each pair of bugs. In the third step cluster labels are generated calculating the frequent and meaningful terms from each cluster text data and assign them to that cluster. In the fourth step cluster labels are mapped against the bug taxonomic terms to identify appropriate categories of the clusters. In the fifth step performance is evaluated by calculating accuracy, precision, recall and F-measure.

The algorithm CLUBAS shows stability and performs better than Classification using Clustering (CC), Support Vector Machine (SVM) and J48 when this algorithm is implemented on the Android bug repository. It maintains the F-measure value more than 0.9 for each experiment using different number of samples. When the precision, recall and F-measures are important CLUBAS gives the better and stable results irrespective of number of samples and software bug repositories. But both Naïve Bayes (NB) and Naïve Bayes Multinomial (NBM) performs better in terms of accuracy than CLUBAS.

To improve the performance of CLUBAS advance text pre-processing techniques can be implemented to optimize the clustering and classification work and also modern text clustering and classification can be implemented.

Recent approaches in automatically classifying bug reports as bug and non-bug are based on text mining. Antoniol et al. [2] have investigated the automatic classification of bug reports by utilizing conventional text mining techniques, which demonstrated the feasibility. They have used title and description of bug reports. They have collected about 1800 bug reports from Mozilla, Eclipse and Jboss. The approach consists of three pipeline phases. First, they have manually classified the bug reports as bug or non-bug. Second, the classifier is trained by the labelled data. Third, they have predicted the bug reports of test data using the trained classifier. In this paper they use Alternating Decision Tree (ADTree), Naïve Bayes (NB) and linear logistic regression classifier and evaluate their performance by using 10-fold cross validation.

Here, for logistic regression and ADTree, the precision of bug and the recall of non-bug are increased when the number of features are increased though decreased the recall for bug, for Mozilla repository. But NB exhibits limited improvement with increasing the number of features. For eclipse the performance of three classifiers is similar though the improvement is very limited with the increasing of the number of features. For JBoss the three classifiers show poor performance when the number of features are low like 20. For all repository logistic regression performs better than other classifiers.

In the following paper they classified the bug reports only as bug and non-bug. Here, further classification can be possible like as logical bug, security bug, UI related bug etc. and the performance of the classifiers are not so high. So, there are some opportunity to increase the performance.

To assigning the right bug to right developer Muhammad [3] proposed an automated approach for software bug classification. Their work can be split into three major steps pre-processing, feature selection and classification. In pre-processing step at they eliminate the stop words and punctuations. After that they use porter stemming algorithm for stemming the vocabulary. In feature selection step they use Chi-Square and Term Frequency Inverse Document Frequency. And Naive Bayes text classifier is used for bug classification.

Using the following approach they get maximum 86% accuracy. Highest accuracy is obtained when the training to testing ratio is 1:11. The following approach gives the better result than Support Vector Machine (SVM) when the training data is small. From the processing time point of view it performs better than SVM. But when the training dataset is large SVM performs better. Here the performance can be increased by using other feature selection algorithm and the synonym dictionary can be used to improve the performance.

Feature extraction and comparison of event model using Naïve Bayes approach for bug classification is proposed by S J Dommati et al [4]. They give priority on feature extraction, noise reduction and classification of network bugs. They distribute their work as feature extraction and pre-processing, probabilistic framework for classification and performance measure. The goal of a bug feature extractor is to automatically extract features from bug information in bug repository. They consider title, description and crash file attachment as valuable attributes. In this paper they use Information Gain Measure as feature selection. They use Bernoulli and Multinomial Naïve Bayes for classification. Their assumption was there is strong independence of features. In the Bernoulli, a bug is represented as a binary vector over the space of features. On the other hand the Multinomial captures feature frequency information in bugs. They get 60% and 78% accuracy for Bernoulli and multinomial respectively. More bug specific feature selection and better classification can be used for better performance.

Summarization of multiple text documents based on sentence clustering is proposed by K Sarkar [6]. Their work is split into three main steps clustering sentences, cluster ordering and selection of representative sentences. In the clustering step they use uni-gram matching based similarity measure and similarity histogram based sentence clustering.

J Xuan et al. [7] proposed a semi supervised text classification approach. Using this approach some deficiency can be avoided like labelling of bug reports in existing supervised approach. Their approach combines Naïve Bayes classifier and expectation maximization to take advantage of both labelled and unlabelled bug reports. They demonstrate two basic phases in their approach. In the first phase to train a classifier with labelled bug reports. In their approach they use Naïve Bayes classifier for the following reasons:

- Naïve Bayes is a probability classification which shows the better performance on text form of bug reports.
- Sorting them with probability is a common method for providing recommendation.

- It is easy to extend expectation maximization with a probability weight based on Naïve Bayes.

The second phase is expectation maximization with both labelled and unlabelled bug reports. Evaluation and labelling the bug reports on unlabelled subset is the expectation step and the maximization step is to rebuild the classifier with the label of all bug reports. Until the performance of classifier does not improve, the iterations of building classifiers is repeated.

References

1. Nagwani, Naresh Kumar, and Shrish Verma. "CLUBAS: an algorithm and Java based tool for software bug classification using bug attributes similarities." *Journal of Software Engineering and Applications* 5.06 (2012): 436.
2. Antoniol, Giuliano, et al. "Is it a bug or an enhancement?: a text-based approach to classify change requests." *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*. ACM, 2008.
3. Javed, Muhammad Younus, and Hufsa Mohsin. "An automated approach for software bug classification." *Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on*. IEEE, 2012.
4. Dommati, Sunil Joy, Ruchi Agrawal, and S. Sowmya Kamath. "Bug Classification: Feature Extraction and Comparison of Event Model using Na\" ive Bayes Approach." *arXiv preprint arXiv:1304.1677*(2013).
5.
6. Sarkar, Kamal. "Sentence clustering-based summarization of multiple text documents." *International Journal of Computing Science and Communication Technologies* 2.1 (2009): 325-335.
7. Xuan, Jifeng, et al. "Automatic bug triage using semi-supervised text classification." *arXiv preprint arXiv:1704.04769* (2017).