VISUAL REPRESENTATION OF BUG REPORT ASSIGNMENT RECOMMENDATIONS

SHAYLA AZAD BHUYAN
Bachelor of Science, BRAC University, 2011

A thesis submitted
in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

VISUAL REPRESENTATION OF BUG REPORT ASSIGNMENT RECOMMENDATIONS

SHAYLA AZAD BHUYAN

Date of Defence: December 18, 2019

Dedication

To an outstanding human being and an adorable father, Abul Kalam Azad Bhuyan. The first person ever who believed in me.

Abstract

Software development projects typically use an issue tracking system where the project members and users can either report faults or request additional features. Each of these reports needs to be triaged to determine such things as the priority of the report or which developers should be assigned to resolve the report. To assist a triager with report assigning, an assignment recommender has been suggested as a means of improving the process. However, proposed assignment recommenders typically present a list of developer names, without an explanation of the rationale. This work focuses on providing visual explanations for bug report assignment recommendations. We examine the use of a supervised and unsupervised machine learning algorithm for the assignment recommendation from which we can provide recommendation rationale. We explore the use of three types of graphs for the presentation of the rationale and validate their use-cases and usability through a small user study.

Acknowledgments

I would first like to thank my thesis advisor Dr. John Anvik. His office door was always open whenever I ran into any trouble or had questions about my research, writing, or anything. He always let me find my own way but guided me in the right direction whenever he thought I needed it. I learned everything about research from him. He is the best supervisor I could wish for.

The next person on this list would surely be Faisal Ahmed, for always believing in me, supporting me, and pushing me to achieve my goals.

I am very grateful to my M.Sc. supervisory committee member Dr. Wendy Osborn and Dr. Yllias Chali for their valuable feedback and time. I appreciate their effort. I want to thank the Natural Sciences and Engineering Research Council (NSERC) of Canada Discovery Grant and Alberta Innovates Technology Futures (AITF) Grant for providing me funding for the research work.

I want to thank my family, especially my mother and sister, for offering me their unfailing support and encouragement throughout my years of study. I am very grateful to my friends, who were always there for me when I needed any help, cheered for me, listened to me and advised me.

I would also like to thank those people who participated in my user study. I am deeply grateful for their time and effort.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The increasing need for Global and Distributed Software Development (GDSD) projects has also increased the demand to find and connect with people with needed expertise for a task. For any newly assigned developers in a medium to large GDSD project, the modules and libraries used in existing code bases are often unfamiliar or difficult to understand solely based on documentation. Being able to identify an expert in an area of the codebase or the use of any library is a potential solution to this cold start problem. For example, if a new developer or a tester wants to know about a specific part of a project or how to solve a specific bug, recommendation systems have been proposed as a means for improving the achievement of this goal [6, 27, 38, 41]. Also, different studies have shown that developers spend 50%-70% of their time communicating with other developers [11, 30, 39]. By knowing with whom to communicate, developers can increase their effectiveness. However, creating such a recommendation system requires effectively assessing the project members' expertise.

There are many ways a developer can acquire expertise in a particular project. In general, the more a developer works on a project, the more he or she gains expertise for that project. Expertise is primarily gained in two ways. First, developers acquire expertise as they develop code for a specific module in the codebase. This type of expertise is referred to as implementation expertise [3, 14]. Secondly, a developer gains expertise through the use of calls to other modules or components of the system or libraries. This type of expertise is referred to as usage expertise [18, 32]. Aside from these two, bug report analysis is also another way of finding experts [2, 7, 20, 22]. When using bug reports, the system can consider facts like how many times a developer solved an issue, how many of those issues are on the same kind of topics, how many of them are a high priority, how much time did the developer take to solve an issue and so on. In this work, bug reports are used to find expertise.

Researchers have been using machine learning algorithms to create classifiers [2, 17, 18, 22, 26]. The use of machine learning algorithms frequently involves careful tuning of learning parameters [33]. As a result, researchers can tune parameters until they are satisfied with the result of a classifier. Then the classifier can be used with real-time data to give predictions. The machine learning algorithm has been proven to be useful in creating a recommender system. The creation of a recommender system involves a series of steps. First, data needs to be collected from an appropriate data source. These datasets are then filtered to provide features for distinguishing different classes by a machine learning algorithm. For a bug report assignment recommender, these features may include the names of developers, how often they have committed code for a module in the project, and how many times they called or used specific components of the system within their code. Having established the features and class names, the machine learning algorithm is used to create the recommender.

In a typical expert recommender system, the recommendations are provided as a textual list without an explanation of the rationale for the recommendations. As described by Herlocker et al. [12], most current recommendation systems are similar to a black box where the transparency is not ensured. Providing transparency by incorporating the reasoning and data behind a recommendation is an essential feature of an effective recommendation system [6, 29]. As many recommendation systems use multi-dimensional data, such as historical user preferences, context-related information, and temporal information, making efficient visualization of the recommendations can show how different dimensions were applied while making recommendations to improve transparency [29] . Furthermore, the visualizations can improve a user's acceptance rate of recommendations by providing more information [6]. Trintarev et al. [37] surveyed a group of moviegoers and found that as much as the list of recommended movies is important, the explanations behind the recommendations are also equally important to the users.

Bug report[1] triage recommenders are an example of such a recommender system in software engineering. Bug report triage is the process where a project member, typically a project manager, decides what to do with a bug report. These decisions include which bug reports need to be resolved and which to ignore, the priority of the report, what is the estimated time to resolve the report, who should be assigned to resolve the bug report, and which "sprint" or product release will contain the changes. Bug report triage is a significant software maintenance problem for any reasonably sized and active software project. Projects may receive an overwhelming number of bug reports each day [2, 3, 39]. Also, bug report triage is a tedious task that often shifts development resources away from improving a product instead towards managing the project.

Bug report triage recommenders have been proposed as a means to reduce the overhead of the project. Within the area of bug report triage recommenders, assignment recommenders are the most commonly researched [2, 26, 41]. Typically, proposed assignment recommenders provide a textual list of recommended developers' names, without explanation given for the recommendations (e.g., [2, 36]). Although the expert list provided by the recommender is crucial, evidence and feedback from software developers indicate that the lack of explanation leads them to question and perhaps not trust the recommendations.

In this thesis, we present work towards providing accurate expert recommendations and transparency for bug report triage assignment recommendations using visual explanations. To the best of our knowledge, this area has not been explored within the area of recommender systems for software engineering (RSSE). We believe that part of the reason for this is that bug report assignment recommenders are created typically using machine learn-

[1]We use the term "bug report" to refer to any change request made for software development, including feature requests and tasks.

ing algorithms that make it hard to provide explanations. For example, one of the most commonly used algorithms is Support Vector Machines (SVM) [28, 42], which creates a non-probabilistic binary linear classifier. When training such a classifier, each instance in the training set is labeled as belonging to one or the other of two labels, and the built model assigns new examples to one label or the other. In a multi-class classifier, the classifier is typically built as a set of one-vs-all classifiers with a winner-takes-all strategy, or as a set of one-vs-one classifiers with a max-win strategy. In either case, determining the rationale for the recommendations is near impossible. More recent work on assignment recommenders have used the Random Forest algorithm [46, 45], but we found a similar problem concerning determining rationale. Instead, we have focused on assignment recommenders using `Multinomial Naïve Bayes` and `Topic Modeling`. We have created one classifier using Multinomial Naïve Bayes. To create the recommender, we first create a bag of words, then generate the conditional probability scores for each developer. We use the conditional probability scores as the expertise score for that respective developer. To find the most relevant word, we have used an information retrieval technique known as the TF-IDF. Finally, we used the expertise score, relevant words, and experts name to generate visual representations. In contrast to Multinomial Naïve Bayes, the use of Topic Modeling is a relatively new addition to this field. First, we create a `word2vec` model using the training dataset. Next, we randomly select five bug reports, and those five reports are used as initial centers of clusters. Using the vector values, we use the Word Movers Distance method and calculate the distance between bug reports. We then apply the K-means clustering method and move the bug reports from one cluster to another depending on their distances. We stop when not a lot of the reports are shifting from one cluster to another. At this point, each cluster represents a topic. We adopt an information retrieval technique to select topics. We determined experts depending on how many reports a developer has solved from a cluster. From this classifier, we visually represent the topic name, the number of reports a developer solved, and the developers' names.

To explore the use of visualization for recommender explanation, we chose to display assignment recommendations using three different graphical methods: stacked horizontal bars, pie charts, and a data table. Previous researchers have used different graphical representations for recommendations, including a circle pack layout for recommending talks to conference attendees [29], and a network representation to display recommendations based on the data extracted from Facebook [27]. Bostandjiev et al. [6] created a music recommendation system where the interface was interactive. The recommendations took the form of a horizontal bar chart, and the length of the bars indicated the recommendation score. When a user changed their preference for a particular artist or band, the recommendation strength changed in the interface in real-time.

To assess the impact of the use of the assignment recommendations and these visualizations, we conducted a small user study. We created a browser plug-in that communicated with a web service. We posted an advertisement on `reddit` to invite people to participate in our study. The criteria for participation selection were to have prior experience in either software engineering, machine learning, or bug report triage. In the end, our study contained fourteen (14) individuals where participants examined bug reports from an open-source project and used a web browser plug-in to receive visualized assignment recommendations. The goal of our work was to answer the following research questions:

RQ1: Does Topic Modeling (i.e., an unsupervised learning algorithm) provides an acceptable accuracy to be an alternative to Multinomial Naïve Bayes (i.e., a supervised learning algorithm) ?

RQ2: Do developers find visual explanations of assignment recommendations easier to understand than a text list?

RQ3: Do developers find visual explanations of assignment recommendations more trustworthy?

RQ4: What is the preferred recommended visualization technique by the developers?

This thesis is organized in the following manner. First, we provide some background information regarding our approach to creating and visualizing bug report assignment recommendations. Then, we present an overview of how the assignment recommenders are created and the visualizations we use. Next, we describe the setup of our user study, including details about the web browser plug-in and web service that is created to provide a visual explanation of the recommendations. Then we give the results of our study and a discussion of our findings. Before we present previous works that are found to be complementary to our approach, we discuss the lessons we learned, present threats to validity, and possible future directions. In the end, we conclude with a summary of the works that we proposed.

Chapter 2

Background

Representing assignment recommendations visually requires an understanding how such recommenders are created. In this section, we provide overviews of assignment recommender, machine learning, text processing, and term weighting.

2.1 Bug Report Assignment Recommenders

In order to create a bug report assignment recommender using machine learning algorithms, text and attributes from the reports are extracted. In the machine learning literature [2], these reports are known as *instances*, and the attributes of the instances are known as *features*. An important feature for supervised machine learning algorithms is the *label*, which describes the category or class to which the instance belongs. For an assignment recommender, this will be the name of the developer previously assigned to the report. In the case of unsupervised machine learning, the user determines the number of classes (i.e., the number of developers in the project), or clusters (i.e. if using a cluster-based unsupervised machine learning algorithm). Also, the user specifies the number of iterations to run the algorithm. Then, the classifier uses the features to iterate through grouping instances and recalculates a goodness metric for the groups until either there is little to no change in the groups or the iteration threshold is reached.

As the title (or summary) and description of the bug report provide the primary features, text classification is used to process this data into individual features. The Bag of Word (BoW) model can be used for text classification method. In this model, the text (such as a sentence or a document) is interpreted as a collection of independent terms. It disregards grammar and the order of words in that text but keeps the multiplicity. The BoW usually gets utilized in the processes of document classification, where the frequency of a word found in that document is used as a feature for training a classifier.

2.2 Machine learning

Machine learning is applying some form of artificial “learning”, where the “learning” is the ability to change an existing model based on new information. Machine learning refers to techniques that allow an algorithm to modify an internal model based on observations so that its accuracy or effectiveness increases. Although every machine learning algorithm is unique, all such algorithms generalize to the following steps which repeat until the result is considered satisfactory:

1. Apply the algorithm to a set of training examples.
2. Find results for testing examples.
3. Evaluate the results against a gold standard.
4. Tune the algorithm parameters.

Machine learning algorithms are often divided into two categories: *supervised* and *unsupervised*. Supervised machine learning is applied based on the knowledge learned from previous data to predict future results. The process begins by analyzing a training dataset where the categories of the training instances are known. In other words, each item in the training set is labeled. The algorithm then generates an inferred function (i.e., classifier) that makes predictions for labeled data that has not been used in training dataset. The predicted label is then compared to the known label to assess the effectiveness of the classifier. Support Vector Machines, Linear Regression, and Naïve Bayes are examples of commonly used supervised learning algorithms.

In contrast, unsupervised machine learning algorithms analyze a dataset to describe a hidden structure of unlabeled data. Unsupervised learning identifies patterns in the data and reacts based on the presence or absence of such patterns in each new piece of data. K-means, Topic Models, and mixture models are examples of commonly used unsupervised learning algorithms.

The main goal of this work is to investigate the visualization of recommendations for both a supervised and unsupervised machine learning algorithm. The two chosen algorithms were Multinomial Naïve Bayes and Topic Modeling.

2.2.1 Multinomial Naïve Bayes

Multinomial Naïve Bayes is a specialized version of Naïve Bayes that is designed to use with text documents [16]. Whereas the traditional Naïve Bayes algorithm would represent a document as the presence and absence of particular words, Multinomial Naïve Bayes explicitly models the word counts and adjusts the underlying calculations. Especially for small sample sizes, Multinomial Naïve Bayes classifiers can outperform the more powerful alternatives.

This equation is for Multinomial Naïve Bayes. It shows that the likelihood of a word ($P(w|c)$) given a class (c) equals the count of the word (w) occurring in the class divided by the count of all words in the class. The purpose of the addition of one (1) is to smooth the result. $|V|$ represents the vocabulary size.

$$P(w|c) = \frac{count_{(w,c)} + 1}{count_{(c)} + |V|} \tag{2.1}$$

2.2.2 Topic Modeling

A topic modeling approach begins with selecting n instances as centroids. The remaining instances are then grouped around the centroids based on a distance metric and using a k-means approach [5]. The centroids are recalculated, and clusters are reformed until the system reaches a stable state or a specified number of iterations is completed.

Traditionally, the Latent Dirichlet allocation (LDA) [26, 25] is used as a technique for topic modeling. However, in this work, the more modern method, known as Word Mover‘s Distance (WMD), is applied. This method uses semantic similarity. WMD applies the earth mover‘s distance technique.

2.2.3 Word Mover’s Distance(WMD)

Word Mover’s Distance (WMD) is based on word embedding techniques that learn meaningful representations for words from local coexistence in sentences. WMD uses some advanced embedding techniques like `word2vec` and `Glove`. `Word2Vec` is a predictive word embedding technique which changes a word into number-based vectors based on the target word. This model measures a target word’s vector value using other words from the same sentence. It uses a Neural Network whose hidden layer encodes the word representation. `Glove` is a count-based model that learns vectors or words from their co-occurrence information (i.e., how frequently they appear together in large text corpus).

Word Mover’s Distance (WMD) suggests that distances between embedded word vectors are, to some degree, semantically meaningful. It utilizes this property of word vector embedding and treats text documents as a weighted point cloud of embedded words. The distance between two text documents A and B is calculated by the minimum cumulative distance that words from the text document A needs to travel to match exactly the point cloud of text document B. This approach produces a more coherent document representation than a Bag of Word model.

2.3 Text Processing

Text processing is one of the most common tasks in many machine learning applications. Applications like a language translator, a sentiment analyzer, and spam filtering use text processing. Machine learning for natural language processing (NLP) and text processing involves using machine learning algorithms and some artificial intelligence to understand the meaning of any given text documents. NLP includes applying algorithms to identify and extract the natural language rules in such a way that the unstructured language information is converted into a form of a string which is comprehensible by the computers. After the texts is given, NLP utilizes algorithms to obtain the meaning incorporated with each sentence and collect the required data from them. There are several kinds of text processing libraries available for classification, tokenization, stemming, tagging, parsing, and semantic reasoning. Text processing contains a few steps:

- Tokenization — converting sentences to words.
- Removing unnecessary punctuation (-*$), tags.
- Removing stop words — removing frequent words such as “the”, “and”, “is” that do not have specific semantics.
- Stemming — reducing words to their root by removing inflection through discarding unnecessary characters, usually a suffix.

2.4 Information Retrieval(IR)

Information retrieval (IR) is the process of searching for documents or information in documents [9]. Document types included text, multimedia, HTML, etc. It has been fully studied in the last 40 years and applied to various domains, including software engineering [1]. Information retrieval is established on two principles: firstly, the data under search are unstructured, and secondly, unlike SQL queries in database systems, the queries in IR are formed mostly by keywords. Usually, information retrieval adopts probabilistic models (e.g. Naïve Bayes). A few concepts from information retrieval have been used in this work:

- Bag of Word(BoW)
- Term Frequency(TF)
- Inverse Document Frequency(IDF)

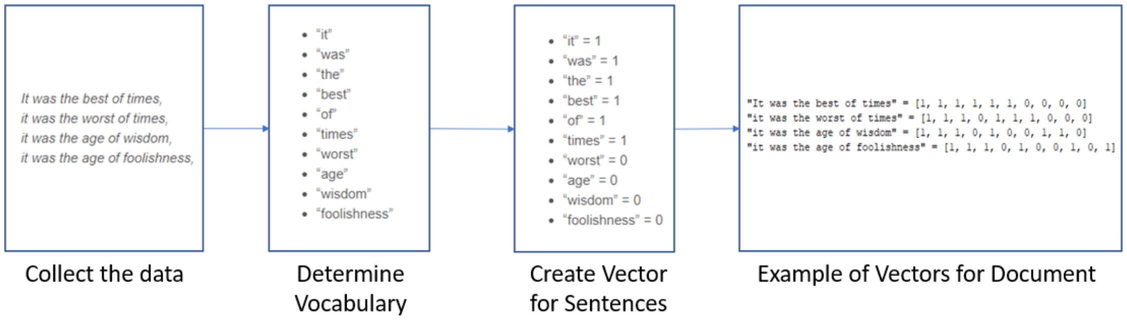

It was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness,
"it"
"was"
"the"
"best"
"of"
"times"
"worst"
"age"
"wisdom"
"foolishness"
"it" = 1
"was" = 1
"the" = 1
"best" = 1
"of" = 1
"times" = 1
"worst" = 0
"age" = 0
"wisdom" = 0
"foolishness" = 0
"It was the best of times" = [1, 1, 1, 1, 1, 1, 0, 0, 0, 0]
"it was the worst of times" = [1, 1, 1, 0, 1, 1, 1, 0, 0, 0]
"it was the age of wisdom" = [1, 1, 1, 0, 1, 0, 0, 1, 1, 0]
"it was the age of foolishness" = [1, 1, 1, 0, 1, 0, 0, 1, 0, 1]
Collect the data
Determine Vocabulary
Create Vector for Sentences
Example of Vectors for Document

Figure 2.1: Flowchart for BoW model creation.

- Term Frequency-Inverse Document Frequency(TF-IDF)

2.4.1 Bag of Word(BoW)

The Bag of Words (BoW) model is a simple algorithm used in NLP (Natural Language Processing). It can be used to extract features from the text. This framework contains a very simple and easy to use approach. It can be used in many ways to extract features from documents. It is known as "bag" of words as the order or structure of words is not stored. The main interest of this model is to find occurrences of a known word, not how many times or where in the document it occurred. Figure 2.1[2]. shows the steps to create a BoW model. At first, data is collected. After that, a list of unique words is created from that data. The next step is to score those words in each document so that these documents can be turned into a vector. These vectors can later be used as input or output for a machine learning classifier. Finally, a binary vector for each sentence of that document can be generated.

2.4.2 Term Frequency(TF)

The term frequency refers to the number of times that a term t occurs in document d [24]. As every document is different in length, the possibility of a term appearing more times in long documents than shorter ones is higher. Therefore, to normalize this value, the

[2]The image was created using information provided in this `https://machinelearningmastery.com/gentle-introduction-bag-words-model` website

term frequency is often divided by the document length:

$$TF(t) = \frac{Number\ of\ times\ term\ t\ appears\ in\ a\ document\ d}{Total\ number\ of\ terms\ in\ the\ document} \tag{2.2}$$

2.4.3 Inverse Document Frequency(IDF)

The inverse document frequency is a way to measure whether a term is usual or not in a given document collection. It can be calculated by dividing the total number of documents with the number of documents containing the term in the corpus [24].

$$IDF(t) = \ log_e \frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ term\ t\ in\ it} \tag{2.3}$$

2.4.4 Term Frequency-Inverse Document Frequency(TF-IDF)

The term frequency and inverse document frequency can be combined (i.e. multiplied) to obtain a new measure known as TF-IDF. This value shows how important a word is with respect to a given document in a collection of documents.

$$\text{TF-IDF}\ (t,\ d,\ D) = \text{TF}(t,\ d)\ \cdot\ IDF(t,\ D) \tag{2.4}$$

t: Term occurring in a document

d: One document from corpus

D: Total number of documents in the corpus

2.5 Summary

This chapter explained the basic concepts for understanding our work. We presented how a machine learning algorithm can be used to create a bug report assignment recommender, how machine learning algorithms work, and various text processing techniques.

Chapter 3

Methodology

In this section, an overview of the steps to create the assignment recommenders and create the visual explanations of the recommendations is presented. Figure 3.1 shows steps of our work. First, data from the bug reports need to be collected. Second, the data needs to be filtered and processed using text categorization techniques. After that, a supervised and unsupervised machine learning algorithm is applied to create two assignment recommender classifiers. Finally, the recommendations are represented visually.

3.1 Assignment Recommender Creation

The creation of an assignment recommender for a software project involves the following steps:

1. Data Collection.
2. Data Preparation.
3. Recommender Creation.

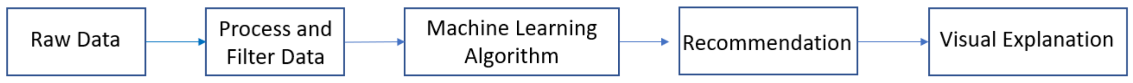


Figure 3.1: Flow chart for visualization using machine learning algorithm.

3.1.1 Data Collection

We use text classification for assignment recommender creation. The first step is to collect bug reports from a project's issue tracking system (ITS). For this work, we decided to use bug reports from a well known open-source bug repository: bugzilla.mozilla. As it is an open-source bug repository, there are hundreds of products for which developers, testers, users, and other project-related people submitted reports. From there, we used the bug reports from two products: `Bugzilla` and `Thunderbird`. Generaly speaking in machine learning, the more data we have to train the algorithm, the better the classifier can perform. Therefore, we gathered bug-reports going back five years from August 2019 (i.e., August 2014 to August 2019).

A bug report contains different kinds of information: name and other details of the product, status, reporter's name, contact information, assignee's name, type of bug, priority, summary, and comments. Not all information is useful. For an assignment recommender, information like the bug-report id, name of the reporter, name of the person who fixed the bug report, summary/title, and description of the report are useful. In the Bugzilla ITS there is no direct "description" of a bug report. Instead, we found that the first comment in any bug report is the description of that bug. In many software ITS, typically the name shown in the assignee can be a default email address or the reporter's name. As the name of the assignee is important for creating a supervised recommender, we found that the person who wrote the last comment is often the developer who fixed that bug. For training both of the machine learning algorithms, the reports whose status are either *Assigned* or *Resolved* with a *Fixed* resolution were used.

3.1.2 Data Preparation

In this work, over eight thousand (8,371) bug reports were used. Over five thousand (5,020) reports were used from the `Bugzilla` product, and the remainder (3,351) were from the `Thunderbird` product. Before initiating the data processing, we need to perform a few more steps. First, the first comment (i.e., bug report description) needs to be extracted and concatenated with the summary/title of the report. After that, for each report, the name of the person who wrote the last comment needs to be collected. Often instead of a name, the ITS uses the developer's full email address. We used these email addresses as the developer's name. Now each report has three core pieces of information: report ID, assignee, and text (i.e., summary and description together). These pieces of information are saved to be used later to train the classifiers. After all the necessary elements are accumulated, data processing can be started.

The first step of data processing is to break down the data-set into a sets of features. These features from the bug reports will be used by the machine learning algorithm to learn how to distinguish the different classes (i.e., developers). We performed the standard text processing actions of removing stop words and stemming. The remaining terms were then used to create a BoW model. The assigned developers' name is used to label the instances. And both of the classifiers are heavily dependent on the TF-IDF score, TF-IDf is calculated for each product separately.

3.1.3 Recommender Creation

The main aim of this work is to explain the recommendations of the assignment recommender in such a way that would help triagers to understand the reason for a person to be recommended as an expert. The triager can then assign new reports to the developer. To create an assignment recommender, two algorithms were investigated. A supervised machine learning algorithm (Multinomial Naïve Bayes) and an unsupervised machine learning algorithm (Topic Modeling).

3.1.3.1 Multinomial Naïve Bayes

We chose to use one of the most popular supervised machine learning algorithms: Multinomial Naïve Bayes. By using Multinomial Naïve Bayes theory, the classifier calculates conditional probability values. These values are then assigned to developers as their expertise score. When a new bug report arrives for triaging, the classifier uses that report's text information and calculates conditional probability values for that report. Based on the value, the experts are recommended.

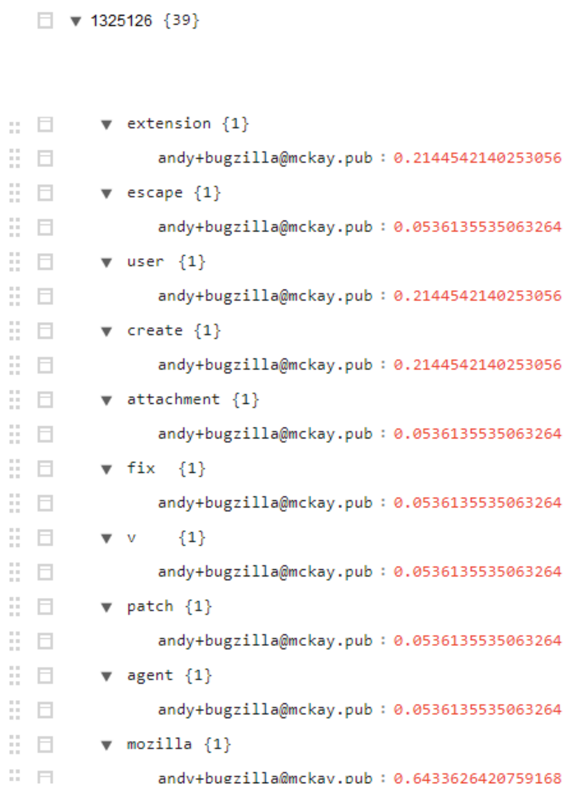
After processing the reports, developers that solved less than ten (10) bug reports are excluded from the list. As the intention is to find an expert, if a developer did not solve a minimum of ten (10) bug reports within the past five (5) years (i.e., two (2) reports per year), he/she should not be recommended as an expert. Hence, those developers and their solved bug reports are eliminated from the dataset. Less than 1% of bug reports from the datasets are removed by this process. After that, for every report the conditional probability value for each word and assigned developer is calculated. The Equation from 2.1 in the context of a bug report assignment recommender is as follows.

$$CP(Developer_D|Word_W) = \frac{count_{(Developer_D,Word_w)}+1}{count_{(Word)}+count_{(Developer_D,Words)}} * 1000 \tag{3.1}$$

Where:

- $Developer_D$: A developer `D`.
- $Word_W$: A specific word `W`.
- $CP(developer_D|word_W)$: The conditional probability value for `D` given `W`.
- $count_{(Developer_D,Word_w)}$: Number of time `W` appears in all bug reports of `D`.
- $count_{(Word)}$: All distinct words from the bug-report corpus.
- $count_{(Developer_D,Words)}$: Total number of words from the bug-report corpus for `D`.

The conditional probability value was calculated for each word and developer in every bug report. To make these values more understandable and presentable, they were scaled by one thousand (1000). Figure 3.2 shows an example of conditional probability values for all the words of a bug report (ID:1325128) for a developer named `andy+bugzilla@mckay.pub`,



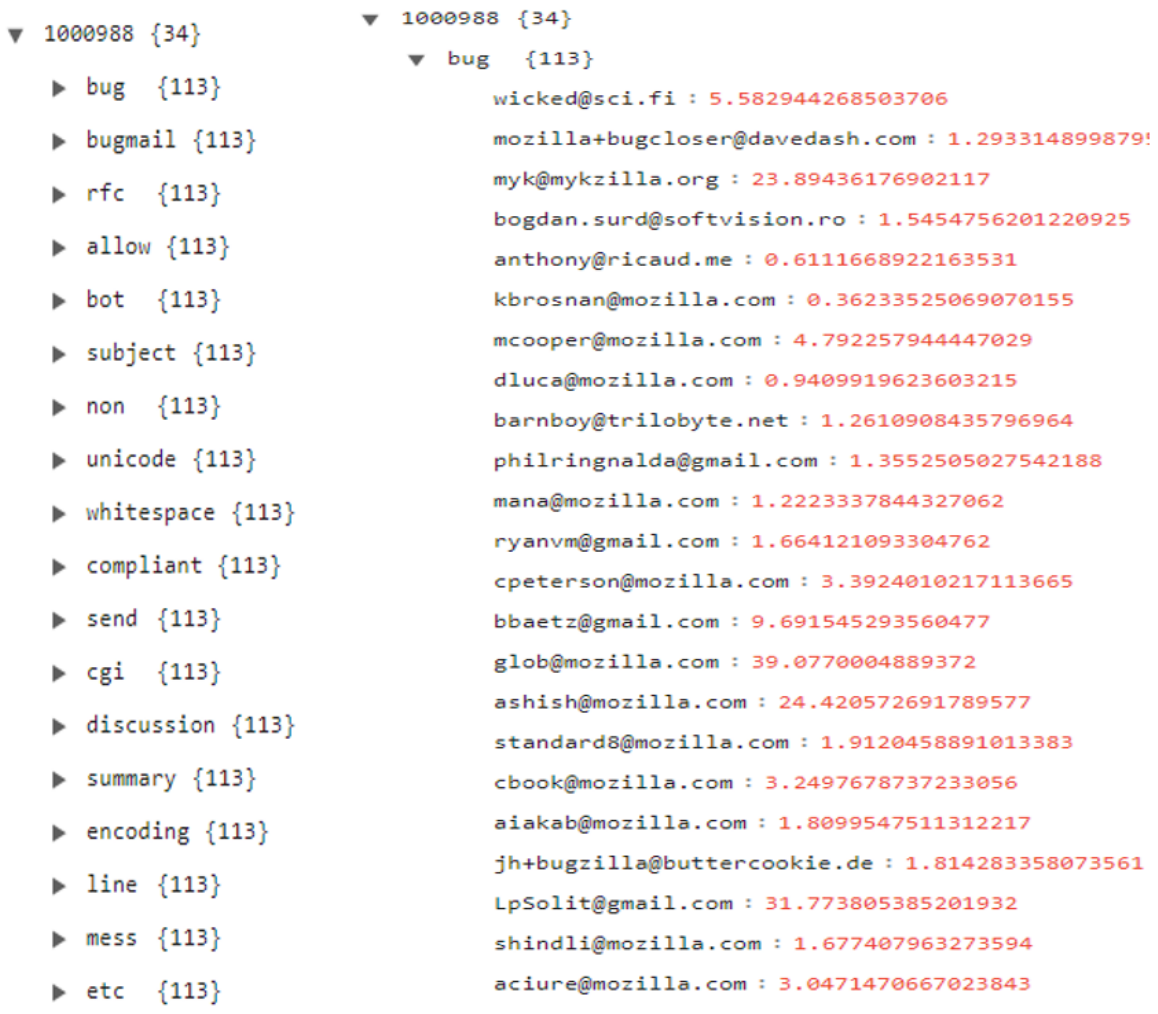
```
▼ 1325126 {39}

    ▼ extension {1}
        andy+bugzilla@mckay.pub : 0.2144542140253056
    ▼ escape {1}
        andy+bugzilla@mckay.pub : 0.0536135535063264
    ▼ user {1}
        andy+bugzilla@mckay.pub : 0.2144542140253056
    ▼ create {1}
        andy+bugzilla@mckay.pub : 0.2144542140253056
    ▼ attachment {1}
        andy+bugzilla@mckay.pub : 0.0536135535063264
    ▼ fix {1}
        andy+bugzilla@mckay.pub : 0.0536135535063264
    ▼ v {1}
        andy+bugzilla@mckay.pub : 0.0536135535063264
    ▼ patch {1}
        andy+bugzilla@mckay.pub : 0.0536135535063264
    ▼ agent {1}
        andy+bugzilla@mckay.pub : 0.0536135535063264
    ▼ mozilla {1}
        andy+bugzilla@mckay.pub : 0.6433626420759168
```

Figure 3.2: Example of conditional probability values scaled by 1000 for terms for a developer in a bug report.

where the values are scaled by 1000. For implementation convenience, we saved the data in such a way where bug report id would be the first entry. After that, for every word of that bug-id, we would keep a conditional probability value of all the developers for that product. Figure 3.3 shows the conditional probability values for all developers for one term in a bug report. The left-hand side of Figure 3.3, shows that this bug report has 34 words in it (after removing stop words and stemming). This product (`Bugzilla`) has one hundred and thirteen (113) developers who solved more than ten (10) reports. On the right-hand side of the figure, a sample of conditional probability scores for the developers is shown for only the first word.

Not every word in a bug report carries equal importance. The higher the value of TF-IDF, the more relevant the word is. So the TF-IDF list is sorted by their values from highest to lowest. Only the words from reports that appear in the first one thousand (1000) relevant words from the TF-IDF list are kept. In this way, the computation time for generating



```
▼ 1000988 {34}
    ▶ bug   {113}
    ▶ bugmail {113}
    ▶ rfc   {113}
    ▶ allow {113}
    ▶ bot   {113}
    ▶ subject {113}
    ▶ non   {113}
    ▶ unicode {113}
    ▶ whitespace {113}
    ▶ compliant {113}
    ▶ send  {113}
    ▶ cgi   {113}
    ▶ discussion {113}
    ▶ summary {113}
    ▶ encoding {113}
    ▶ line  {113}
    ▶ mess  {113}
    ▶ etc   {113}
```

```
▼ 1000988 {34}
    ▼ bug  {113}
        wicked@sci.fi : 5.582944268503706
        mozilla+bugcloser@davedash.com : 1.293314899879
        myk@mykzilla.org : 23.89436176902117
        bogdan.surd@softvision.ro : 1.5454756201220925
        anthony@ricaud.me : 0.6111668922163531
        kbrosnan@mozilla.com : 0.36233525069070155
        mcooper@mozilla.com : 4.792257944447029
        dluca@mozilla.com : 0.9409919623603215
        barnboy@trilobyte.net : 1.2610908435796964
        philringnalda@gmail.com : 1.3552505027542188
        mana@mozilla.com : 1.2223337844327062
        ryanvm@gmail.com : 1.664121093304762
        cpeterson@mozilla.com : 3.3924010217113665
        bbaetz@gmail.com : 9.691545293560477
        glob@mozilla.com : 39.0770004889372
        ashish@mozilla.com : 24.420572691789577
        standard8@mozilla.com : 1.9120458891013383
        cbook@mozilla.com : 3.2497678737233056
        aiakab@mozilla.com : 1.8099547511312217
        jh+bugzilla@buttercookie.de : 1.814283358073561
        LpSolit@gmail.com : 31.773805385201932
        shindli@mozilla.com : 1.677407963273594
        aciure@mozilla.com : 3.0471470667023843
```

Figure 3.3: Example of conditional probability values of a bug report for all developers.

recommendations could be reduced significantly.

3.1.3.2 Topic Modeling

We have decided to use topic modeling to create a K-means clustering algorithm as an example of an unsupervised machine learning algorithm. The algorithm creates clusters of bug reports based on the shortest distance with each other using the dataset. Then, the most relevant topic from each cluster can be selected and used as centroids of the clusters. When a new bug report arrives for triaging, the classifier calculates the distance between the new report and the centroid of each cluster. Based on the minimum distance, the report is labeled with cluster name. Experts from that cluster are then recommended to the triager by the classifier.

Similar to the Multinomial Naïve Bayes classifier creation process, after reports are processed, developers who did not solve a minimum of ten (10) reports are excluded. We use the Word Movers Distance method (WMD) to calculate this distance. This method can assess the ”distance” between two (2) documents even when they have no words in common. It uses `word2vec` vector embedding of words. In other words, instead of using exact word similarity like LDA, WMD uses meaning. By using the `word2vec` model, all of the words in the sentences are turned into vectors, which are later used to find the distance between two documents.

To start, five (5) bug reports are selected randomly, and these reports would be used as centroids to create clusters. Figure 3.4 shows a representation of those clusters as a pie chart. The distance between each report and these five (5) reports are measured. Later, reports are grouped into the clusters according to their minimum distance to each centroid. A short TF-IDF scored list is created to find the most relevant words for each cluster. This list of terms now represents the centroid of each cluster. Again the distance between each bug report and the centroids of each cluster are calculated, and reports are moved to new clusters as needed. The process is repeated ten (10) times. After that, the clusters are con-

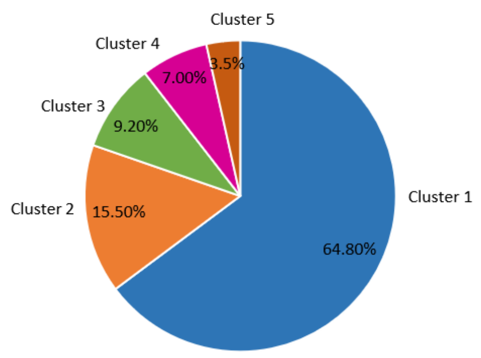


Figure 3.4: Experimental pie chart.

sidered stable. Each cluster can have thousands of bug reports, and the same developer might have solved most of them. The names of developers for each cluster are determined and sorted according to how many times they have appeared in the cluster. In short, developers are ranked in each cluster by the number of reports they resolved. Figure 3.5 shows the flowchart of how the topic model classifier provides recommendations.

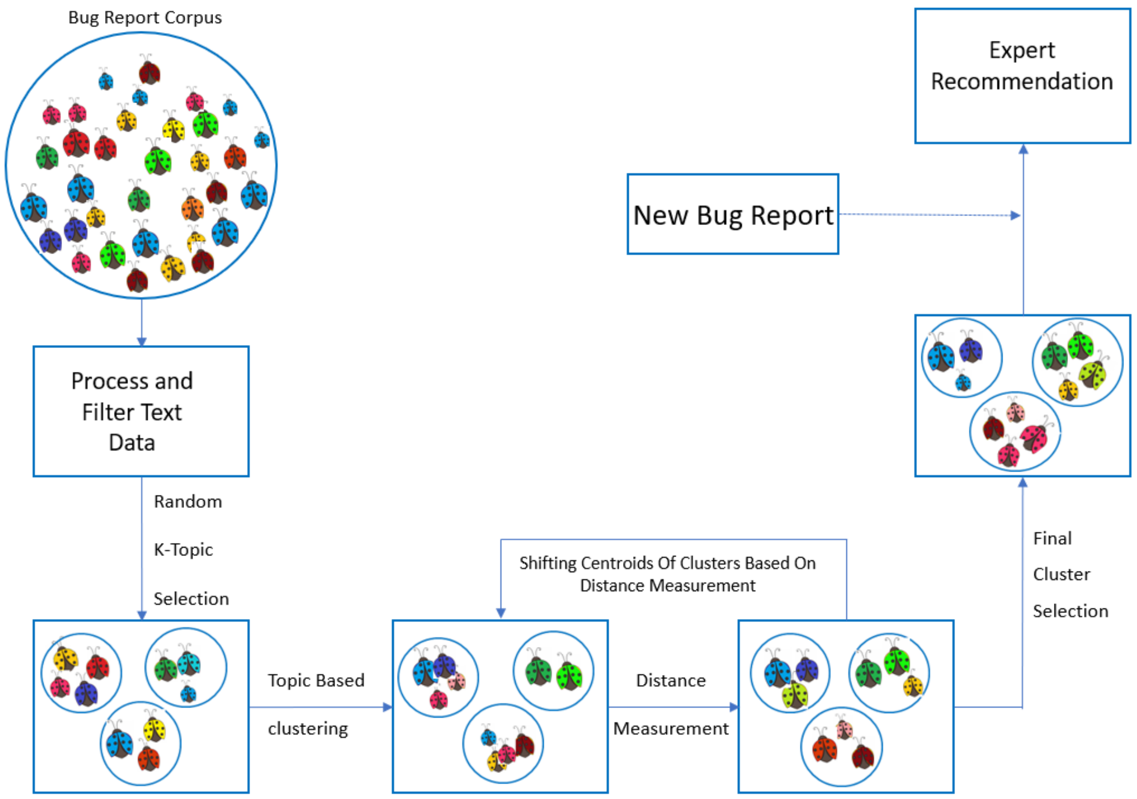


Figure 3.5: Topic modeling Classifier workflow.

3.2 Visualization of Assignment Recommendations

After the recommenders are created, the next step is to use the information provided by the recommender in a visually understandable way. The aim is to generate graphs in a way that a triager would see the graph and understands why the specific developers are recommended. In order to determine how many expert recommendations are sufficient, we conducted a small experiment. Table 3.1 shows that if the assignment recommender gives only one recommendation, for both classifiers, the chances of being accurate is less than 25%. When the number of recommended developers increases by five (5), the accuracy rate increases significantly. But, as the number of recommended developers increases to ten (10), we can see that the accuracy rate increases but less significantly. Also, visually representing ten developers in one graph proved to be confusing and unclear (Figure 3.6). Therefore, the assignment recommenders should provide five expert recommendations.

Table 3.1: Accuracy of the classifiers based on the number of recommendations.

Number of Recommendations	Multinomial Naïve Bayes	Topic Modeling
One Recommendation	24.31%	6.61%
Five Recommendations	68.20%	44.63%
Ten Recommendations	81.30%	49.87%

3.2.1 Visually Representing Recommendations (Multinomial Naïve Bayes)

Recall that in the case of the Multinomial Naıve Bayes classifier, when getting a recommendation for a new bug report, the system returns the most relevant words, corresponding conditional probability scores for all of the developers, and the developers' names. All of these pieces of information are used to graphically represent the recommendations. As the system returns information for each developer, the amount of information can be large. Therefore, the developers are ranked based on their highest conditional probability score.

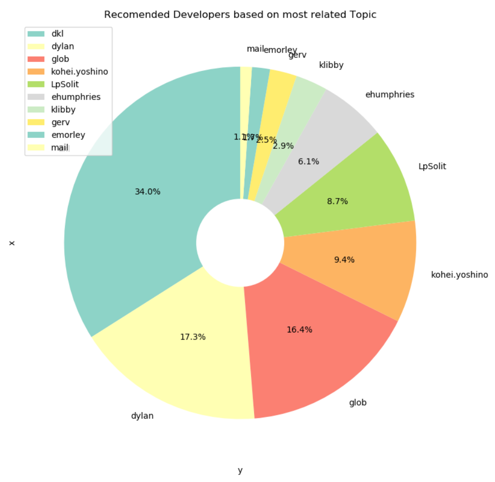


Figure 3.6: Pie for ten developers.

Based on the results shown in Table 3.1, the first five developers whose overall score is the highest are recommended as experts.

In most cases, bug reports have a lot of relevant words. Figure 3.7 shows an example of how it would look if all the words from a bug report were shown. Figure 3.8 shows an example for the same report as Figure 3.7. The pie chart was also chosen to be one of the experimental visualizations. As the pie chart can only display two values meaningfully (a percentage and a label), a pie chart is created for each developer (Left side of the Figure 3.9) and a summary pie chart is created (Figure 3.10). Note that, from the right hand side of Figure 3.9 shows a similar problem with displaying the values for all terms. Again, only the top five (5) terms are shown to the user. There would also be five more pie charts, for each recommended developer and their score for the relevant words from the bug report. Here also we faced similar issues.

3.2.2 Visually Representing Recommendations (Topic Modeling)

For topic modeling, when getting a recommendation for a new bug report, the system provides the overall size of each cluster, the names of the developers from each cluster, and the number of bug reports they resolved for the cluster. For example, from Figure 3.4, we

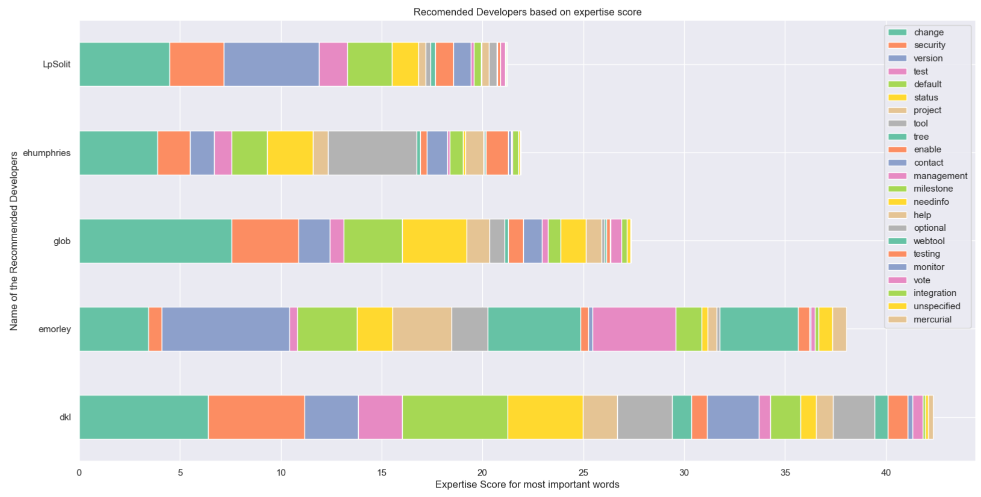


Figure 3.7: Stacked bar graph for all words from a bug report for a Multinomial Naïve Bayes classifier.

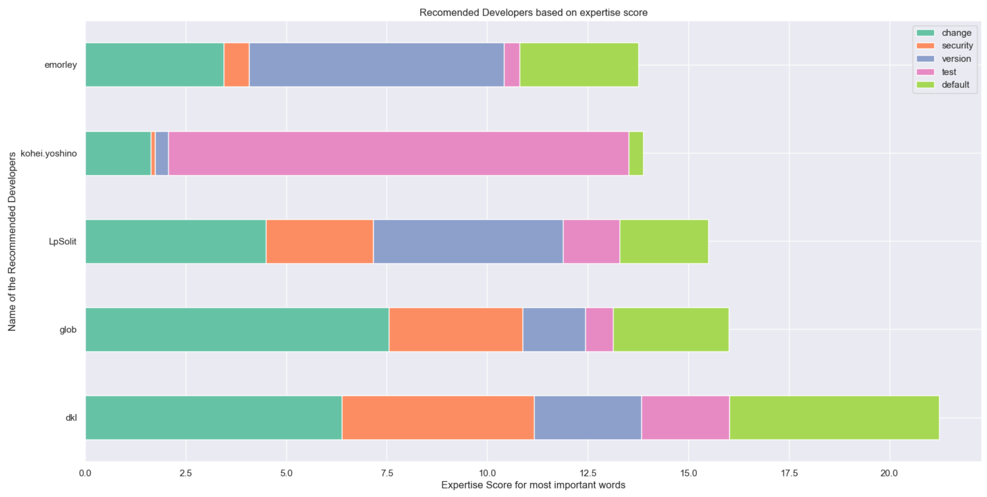
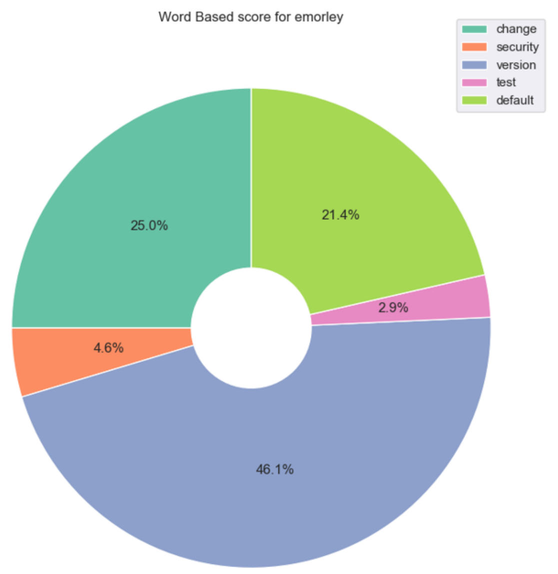


Figure 3.8: Stacked bar graph for the top five most relevant words from a bug report for a Multinomial Naïve Bayes classifier.



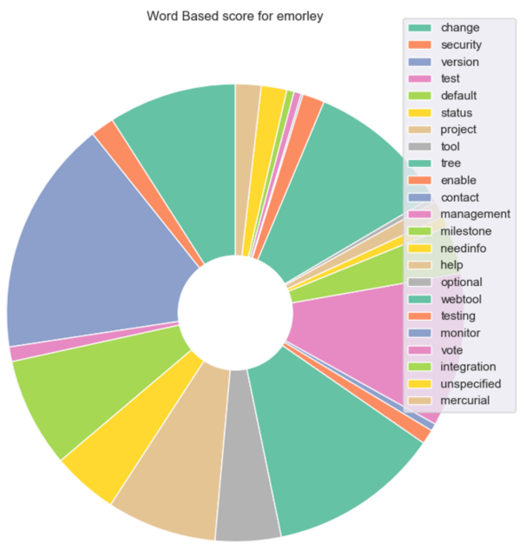


Figure 3.9: Example of conditional probability value of a bug report with the top five (left) terms and all terms (right) for Multinomial Naïve Bayes classifier for a developer.

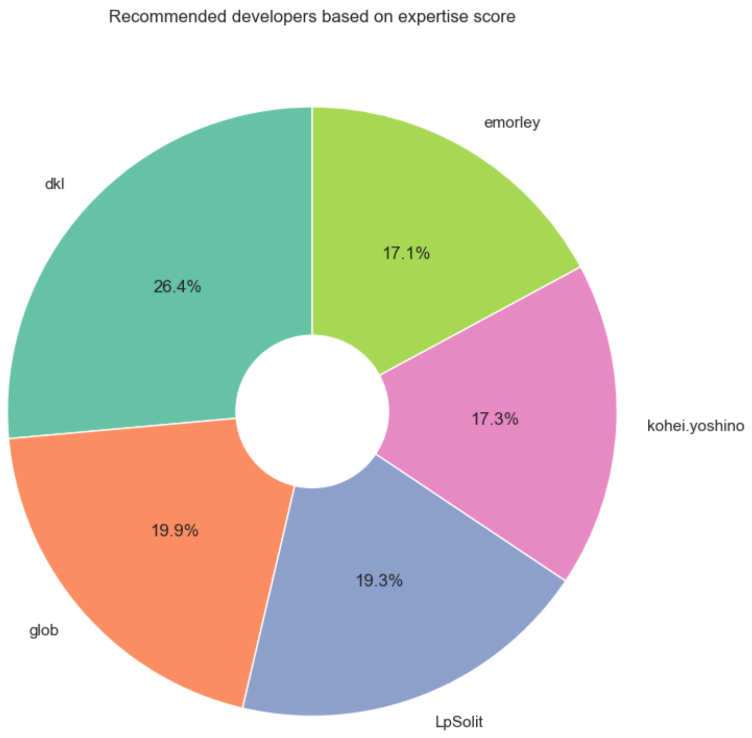


Figure 3.10: Pie chart for the top five ranked developers of a bug report for a Multinomial Naïve Bayes classifier.

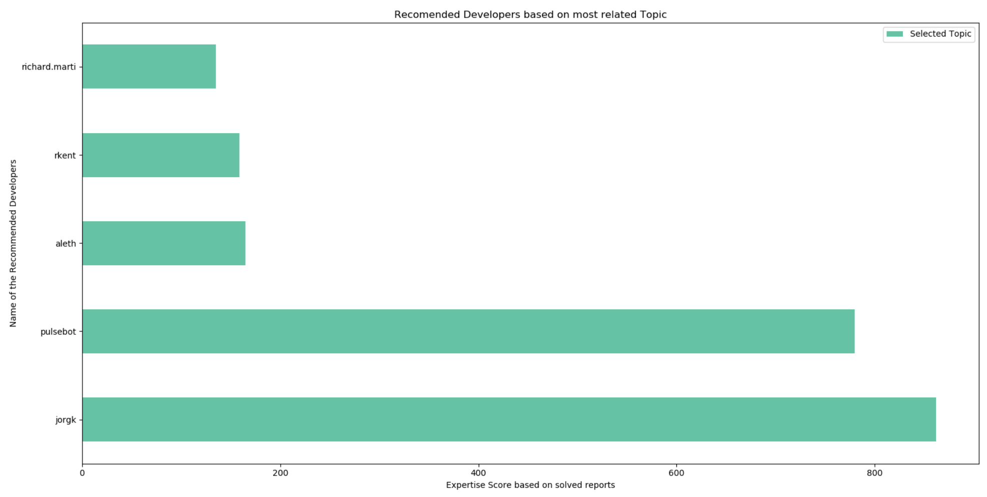


Figure 3.11: Bar chart for the selected topic for Topic Modeling classifier.

can see that cluster 1 is the largest cluster. Therefore, the system would select cluster 1 as our recommended cluster. For this cluster, the system would provide the name of the developers and the number of reports they solved. Figure 3.11 shows the recommended developers for the selected cluster. The Y-axis of the graph shows the name of the developers, and X-axis shows how many bug reports from that cluster were solved by each developer. Figure 3.12 shows the same recommended developers using a pie chart. The chart shows the name of the developers and the percentage of the number of reports they solved.

3.2.3 Font-End Representation

After the extension is installed and the user navigates to bugzilla.mozilla, it is ready to provide recommendations. To provide an assignment recommendation for a new bug report, the user needs to click the “Recommend Expert” (Figure 3.13) button. When the user clicks that button, multiple steps of the recommendation process start. First, the data preparation steps are applied to the report (i.e., the report is turned into a “bag of words” or a vector of features). Next, the features of the new bug report are given to the trained classifier.

In the case of the Multinomial Naïve Bayes classifier, for each potential developer,

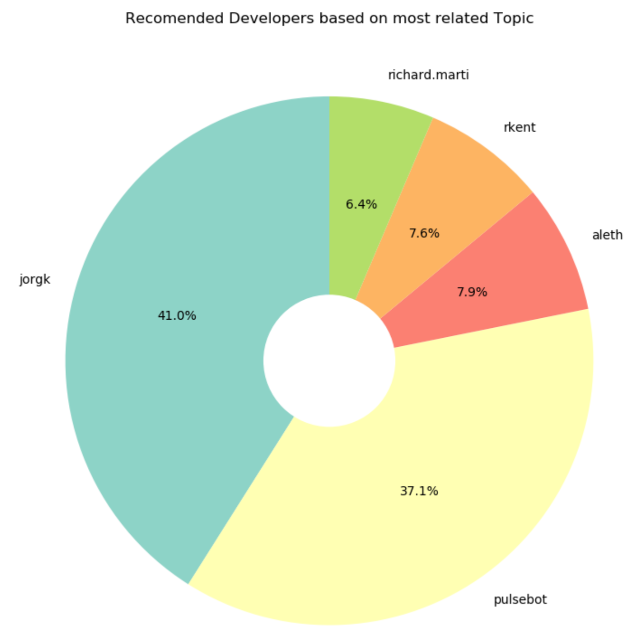


Figure 3.12: Pie chart for for the selected topic for Topic Modeling classifier.

the set of features that are common between their instances and the new bug report are collected. Then the sum of conditional probability scores of each of these features is determined, which provides the expertise score of that developer for that bug report. Finally, developers are ranked based on expertise scores. Figure 4.4 shows three types of visual representation. The pie chart is showing the important features of each report based on their conditional probability values. If a user clicks on the pie, a new web page (Figure 4.5) opens. This new page shows a pie chart for each developer, and each pie chart shows the overall conditional probability values for the corresponding recommended developers. The data-table shows the values for each important feature and developers. The stacked bar chart, however, shows developers horizontally where a different color represents each feature. The developer who has the highest total conditional probability values for all the selected features is at the very bottom.

For the Topic modeling classifier, the cluster with the shortest distance to the new bug report is the selected cluster, and the ranked list of developers for that cluster forms the recommendation list. Figure 4.6 shows the same three types of visual representations for Topic Modeling. The pie chart shows the recommended developer names and their solved bug report rate for the specific cluster. The data table gives the list of developer names with the exact number of reports that the developer solved related to that cluster. The stacked bar chart also shows the developer names horizontally with their score. The color of the bar, however, is related to the selected cluster. Based on the cluster, the color of the bar would change.

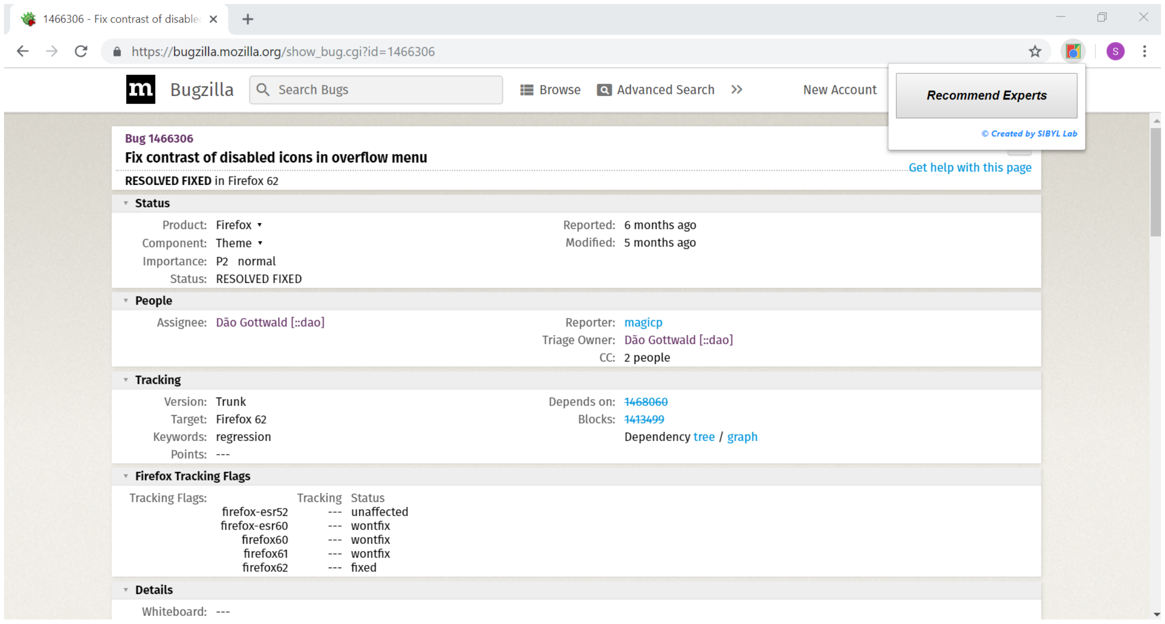
3.3 Web Extension Creation

When provided with a new bug report, the Multinomial Naïve Bayes and Topic Modeling techniques are used to recommend developers to resolve the bug report. To present the visual explanations, a web browser plug-in is used to provide a front-end interface to the user, which connects to a back-end server web service for providing the visualizations. The REST (Representation State Transfer)[3] architecture was adopted to develop the API that establishes the communication between the front-end and the back-end.

3.3.1 Web Browser Plug-in

The front-end part of this project involves a browser plug-in or extension, which is a small software program typically used to customize users' browsing experience. Extensions can be general-purpose, where a user can use them for any website. On the other hand, extensions can also be developed that are specific to certain websites. In other words, such an extension will only get activated while browsing a specific website.The latter approach was used to create a browser extension designed to activate for bugzilla.mozilla. The goal was to create the extension in such a way so that it would be simple and user-friendly. For the simplicity of design, this extension consists of only a button for users to obtain recommended experts. Figure 3.13 shows how the extension looks when a user decides to

[3] A



Bug 1466306
Fix contrast of disabled icons in overflow menu
RESOLVED FIXED in Firefox 62

Status

Product: Firefox
Component: Theme
Importance: P2 normal
Status: RESOLVED FIXED
Reported: 6 months ago
Modified: 5 months ago

People

Assignee: Dão Gottwald [::dao]
Reporter: magicp
Triage Owner: Dão Gottwald [::dao]
CC: 2 people

Tracking

Version: Trunk
Target: Firefox 62
Keywords: regression
Points: ---
Depends on: ~~1468060~~
Blocks: ~~1413499~~
Dependency tree / graph

Firefox Tracking Flags

Tracking Flags:

	Tracking	Status
firefox-esr52	---	unaffected
firefox-esr60	---	wontfix
firefox60	---	wontfix
firefox61	---	wontfix
firefox62	---	fixed

Details

Whiteboard: ---

Figure 3.13: Using Extension.

use it. By clicking the `Recommend Experts` button, the user sends a request to the backend server. The browser extension was created for Google Chrome.

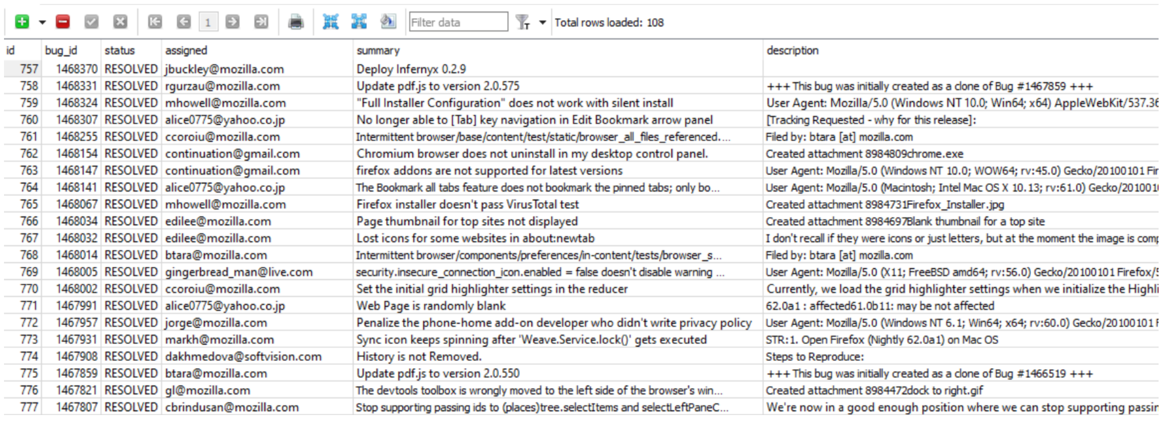
3.3.2 Web Service

In this work, we adopted one of the most common architectures for creating a web-based application known as REST (Representation State Transfer). To create the back-end server the DJANGO framework was used.

A SQL (Structured Query Language) server was used as the back-end database. The database contains 4 tables[4].

- Bugzilla bug report information,
- Thunderbird bug report information,
- TF-IDF score for Bugzilla, and
- TF-IDF score for Thunderbird.

[4] A



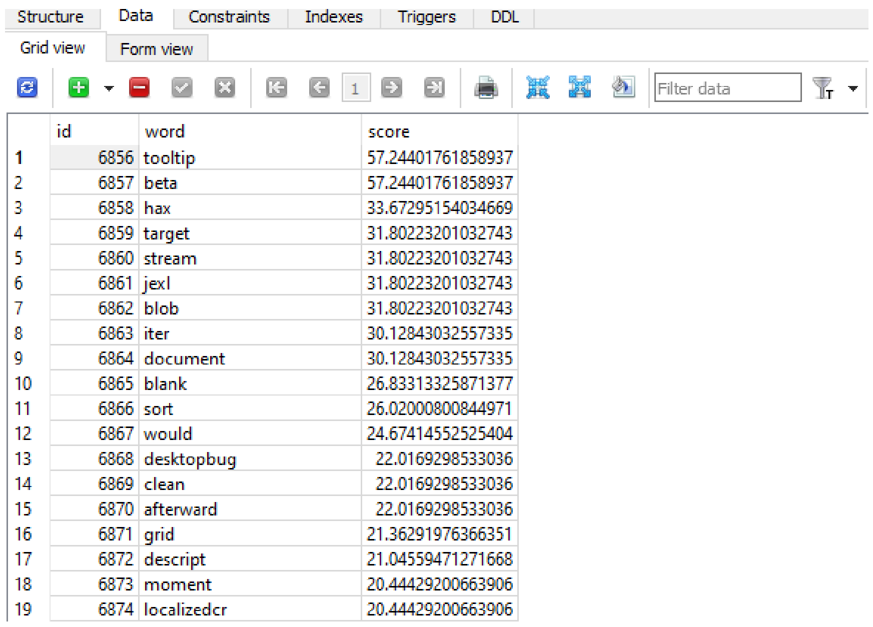
id	bug_id	status	assigned	summary	description
757	1468370	RESOLVED	jbuckley@mozilla.com	Deploy Infernyx 0.2.9	
758	1468331	RESOLVED	rgurzau@mozilla.com	Update pdf.js to version 2.0.575	+++ This bug was initially created as a clone of Bug #1467859 +++
759	1468324	RESOLVED	mhowell@mozilla.com	"Full Installer Configuration" does not work with silent install	User Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.3
760	1468307	RESOLVED	alice0775@yahoo.co.jp	No longer able to [Tab] key navigation in Edit Bookmark arrow panel	[Tracking Requested - why for this release]:
761	1468255	RESOLVED	ccoroiu@mozilla.com	Intermittent browser/base/content/test/static/browser_all_files_referenced....	Filed by: btara [at] mozilla.com
762	1468154	RESOLVED	continuation@gmail.com	Chromium browser does not uninstall in my desktop control panel.	Created attachment 8984809chrome.exe
763	1468147	RESOLVED	continuation@gmail.com	firefox addons are not supported for latest versions	User Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:45.0) Gecko/20100101 Fi
764	1468141	RESOLVED	alice0775@yahoo.co.jp	The Bookmark all tabs feature does not bookmark the pinned tabs; only bo...	User Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:61.0) Gecko/201001
765	1468067	RESOLVED	mhowell@mozilla.com	Firefox installer doesn't pass VirusTotal test	Created attachment 8984731Firefox_Installer.jpg
766	1468034	RESOLVED	edilee@mozilla.com	Page thumbnail for top sites not displayed	Created attachment 8984697Blank thumbnail for a top site
767	1468032	RESOLVED	edilee@mozilla.com	Lost icons for some websites in about:newtab	I don't recall if they were icons or just letters, but at the moment the image is com
768	1468014	RESOLVED	btara@mozilla.com	Intermittent browser/components/preferences/in-content/tests/browser_s...	Filed by: btara [at] mozilla.com
769	1468005	RESOLVED	gingerbread_man@live.com	security.insecure_connection_icon.enabled = false doesn't disable warning ...	User Agent: Mozilla/5.0 (X11; FreeBSD amd64; rv:56.0) Gecko/20100101 Firefox/
770	1468002	RESOLVED	ccoroiu@mozilla.com	Set the initial grid highlighter settings in the reducer	Currently, we load the grid highlighter settings when we initialize the Highl
771	1467991	RESOLVED	alice0775@yahoo.co.jp	Web Page is randomly blank	62.0a1 : affected61.0b11: may be not affected
772	1467957	RESOLVED	jorge@mozilla.com	Penalize the phone-home add-on developer who didn't write privacy policy	User Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:60.0) Gecko/20100101
773	1467931	RESOLVED	markh@mozilla.com	Sync icon keeps spinning after 'Weave.Service.lock()' gets executed	STR:1. Open Firefox (Nightly 62.0a1) on Mac OS
774	1467908	RESOLVED	dakhmedova@softvision.com	History is not Removed.	Steps to Reproduce:
775	1467859	RESOLVED	btara@mozilla.com	Update pdf.js to version 2.0.550	+++ This bug was initially created as a clone of Bug #1466519 +++
776	1467821	RESOLVED	gl@mozilla.com	The devtools toolbox is wrongly moved to the left side of the browser's win...	Created attachment 8984472dock to right.gif
777	1467807	RESOLVED	cbrindusan@mozilla.com	Stop supporting passing ids to (places)tree.selectItems and selectLeftPaneC...	We're now in a good enough position where we can stop supporting passi

Figure 3.14: Table with Bug report information.

Figure 3.14 and 3.15 shows example of two tables that we created to store data. These tables contain all of the necessary information from bug reports (e.g. bug-id, assigned developers' names, reported by, status, summary, description)

When the front end (i.e. browser plugin) requests a recommendation for a particular bug report, the extension sends an HTTP request to the back-end server's REST endpoint. The HTTP request contains the id of the bug report for which to obtain recommendations using this information, features are extracted and fed to the machine learning classifiers. An example request to obtain the bug report information is

https://bugzilla.mozilla.org/rest/bug/1466306, where 1466306 is the id of the bug report for which a recommendation has been requested. Once the recommendations are obtained from the two classifiers, the back-end server processes the result to form an HTTP response object, which is then sent back to the front-end. For Multinomial Naïve Bayes , this HTTP response object contains conditional probability scores of each word in the report for each developer. The words are in ranked order with the expertise score for each top-recommended developer. For topic modeling, it returns the developers' names and the number of bug reports solved by the corresponding developer in ascending order.



	id	word	score
1	6856	tooltip	57.24401761858937
2	6857	beta	57.24401761858937
3	6858	hax	33.67295154034669
4	6859	target	31.80223201032743
5	6860	stream	31.80223201032743
6	6861	jexl	31.80223201032743
7	6862	blob	31.80223201032743
8	6863	iter	30.12843032557335
9	6864	document	30.12843032557335
10	6865	blank	26.83313325871377
11	6866	sort	26.02000800844971
12	6867	would	24.67414552525404
13	6868	desktopbug	22.0169298533036
14	6869	clean	22.0169298533036
15	6870	afterward	22.0169298533036
16	6871	grid	21.36291976366351
17	6872	descript	21.04559471271668
18	6873	moment	20.44429200663906
19	6874	localizedcr	20.44429200663906

Figure 3.15: Table with TF-IDF score.

3.4 Summary

This chapter explained the methodology used to create the two assignment recommenders. Figure 3.1 showed steps of our work. First, the data from the bug reports are collected. Second, the data is filtered and processed using text classification techniques. After that, machine learning algorithms are applied to create two bug report assignment classifiers. Finally, the recommendations are represented visually. Multinomial Naïve Bayes was chosen as a supervised machine learning algorithm and Topic Modeling as an unsupervised machine learning algorithm for creating the two classifiers. Both of these classifiers have the potential to represent their results visually. An information retrieval technique (TF-IDF score) was used to find the most relevant words. However the goal of this work is not just to create an assignment recommender, but also to present the recommendations in such a way so that a project member can understand why the specific recommendations were provided. Two graphs, a stacked bar chart and a pie chart, were selected to visually represent the recommendations, in addition to a data table. A browser plug-in (i.e., a Google Chrome extension) was created to interact with a REST web service which provides the visualizations.

Chapter 4

Evaluation and Result

The focus of this work is to investigate the explanation of assignment recommendations using visual representation. In order to do this, assignment recommenders needed to be created such that their recommendations can be explained. Therefore, an analytical evaluation of the two recommenders is presented before presenting the details and results of the empirical evaluation used to answer our research questions. Recall that our research questions are:

RQ1: Does Topic Modeling (i.e., an unsupervised learning algorithm) provides an acceptable accuracy to be an alternative to Multinomial Naïve Bayes (i.e., a supervised learning algorithm)?

RQ2: Do developers find visual explanations of assignment recommendations easier to understand than a text list?

RQ3: Do developers find visual explanations of assignment recommendations more trustworthy?

RQ4: What is the preferred recommended visualization technique by the developers?

4.1 Analytical Evaluation of Recommenders and Results

In this section, we explain how both of the classifiers were evaluated analytically. When getting recommendations for a new bug report, the recommender processes the report, categorizes the texts and creates a BoW. Then both of the classifiers use that BoW to recommend developers. Both of our assignment recommender systems provide five experts (recommended developers) names.

Our bug report corpus has more than eight thousand (8,371) bug reports; more than five thousand (5,020) reports are from `Bugzilla` product and the rest (3,351) are from `Thunderbird`. For evaluating the recommenders, cross-validation evaluation method techniques were used.

4.1.1 Evaluation Procedure

Cross-validation is a resampling procedure that is used to evaluate machine learning classifiers on any given dataset. The procedure has only one parameter known as *K*. *K* refers to the number of groups into which a given dataset is to be split. This is why the procedure is known as k-fold cross-validation. In this work, the value of K was set to 5. Hence, we used five-fold cross-validation. For k-fold cross-validation, the whole data is first randomly shuffled, and then split into k groups (in our case five). First, one set is chosen as the test-set, and the remainder are training sets. Each bug report in the testing set is then given to the recommender for classification and the bug report would be selected randomly and used as a new bug report for the classifiers. The rest of the 4 groups would be used as training dataset. For both Multinomial Naïve Bayes and Topic Model classifier, we used the cross-validation to evaluate.

TF-IDF was used to find relevant words in our assignment recommender. For both of the classifiers, we compared the words from bug reports with the TF-IDF list. The TF-IDF list provides scores of all the words from the bug report corpus. Using the whole list would be redundant. Therefore, a shortlist was made where the classifiers would only use the first few hundred words (based on their higher values) for comparison. We tested with the five hundred (500), one thousand (1000), and fifteen hundred (1500) most relevant words from the lists based on their TF-IDF score. We found that if we use the first five hundred words, even though the comparison time is significantly reduced, most of the test reports would have few or no words to use as input for the classifiers. With fifteen hundred words, the comparison time increases significantly, and many words from the bug report remain as the input, which increases the recommendation time. But with one thousand words, the comparison time is not too long, and bug reports still have words to use as input in the classifiers. Therefore, one thousand most relevant words (based on their TF-IDF score) from the whole bug-reports corpus to compare is found to be adequate. However, there still remains a possibility that a bug report can only have one relevant word or even none. An experiment was conducted to assess how often the recommenders were able to provide multiple terms, one term, or no terms for an explanation when the vocabulary size is limited.

4.1.2 Result

To determine the accuracy of the classifiers, the following approach was used: as the classifiers give a list of recommended developers, if one of the developers matches the actual developer that solved the report, we count that as a success. The top-1 (Accuracy@1), top-5 (Accuracy@5), and top-10 (Accuracy@10) accuracies were evaluated. Table 4.1, 4.2, and 4.3 shows the percentages of the accuracy for both the classifiers for both projects.

Table 4.1: Accuracy@1 for Multinomial Naïve Bayes and Topic modeling.

Product Name	**Multinomial Naïve Bayes**	**Topic Modeling**	**Average (Product-wise)**
Bugzilla	27.12%	8.24%	17.62%
ThunderBird	21.49%	4.97%	13.23%
Average (Classifier-wise)	24.31%	6.61%	

Table 4.2: Accuracy@5 for Multinomial Naïve Bayes and Topic modeling.

Product Name	**Multinomial Naïve Bayes**	**Topic Modeling**	**Average (Product-wise)**
Bugzilla	78.08%	71.67%	74.88%
ThunderBird	58.31%	17.59%	37.95%
Average (Classifier-wise)	68.20%	44.63%	

Table 4.3: Accuracy@10 for Multinomial Naïve Bayes and Topic modeling.

Product Name	**Multinomial Naïve Bayes**	**Topic Modeling**	**Average (Product-wise)**
Bugzilla	87.51%	73.62%	80.57%
ThunderBird	75.08%	26.12%	50.60%
Average (Classifier-wise)	81.30%	49.87%	

Table 4.1 shows that, when an assignment recommender recommends only one developer, the accuracy is very low (24.31% for Multinomial Naïve Bayes and 6.61% for Topic Modeling). But if the assignment recommender recommends five developers, the average rate for the Multinomial Naïve Bayes classifier becomes 68.20%, and for the Topic Modeling classifier, 44.63%. Finally, if the number of recommended developers increases from five to ten, the average accuracy rate for Multinomial Naïve Bayes becomes 81.30% and for Topic Modeling, 49.87%

To evaluate whether the assignment recommenders is always providing the visual explanations, it was decided that any recommendation where either less than five (5) developers were recommended or less than five terms were available for explanation was not sufficient for recommendation. For the Multinomial Naïve Bayes classifier, both situations were investigated.

Even though number of terms for explanations are not shown in the case of Topic Modeling classifier, we decided to test for all three cases similar to the Multinomial Naïve Bayes classifier.

Table 4.4: Explanation sufficiency for recommendation for Bugzilla.

	Multinomial Naïve Bayes	**Topic Modeling**
More than 5 developers and less than 5 terms	100%	100%
Less than 5 developers and more than 5 terms	98.20%	99.76%
Less than 5 developers and less than 5 terms	92.30%	99.98%

Table 4.5: Explanation sufficiency for recommendation for Thunderbird.

	Multinomial Naïve Bayes	**Topic Modeling**
More than 5 developers and less than 5 terms	63.81%	100%
Less than 5 developers and more than 5 terms	63.35%	100%
Less than 5 developers and less than 5 terms	63.35%	99.73%

Table 4.4 shows that given a bigger dataset, both the Multinomial Naïve Bayes classifier and the Topic modeling classifier can provide graphical explanations for any given

scenarios. However, Table 4.5 shows that given a smaller dataset, the Topic Modeling classifier outperforms the Multinomial Naïve Bayes classifier significantly. In the cases of the supervised machine learning classifiers, the more labeled data that is given as input to train the classifier, the better the predictions classifier can make. However, in the cases of the unsupervised machine learning classifiers, data labeling is not necessary as the classifier learns from the natural structure of the data. Hence, the classifier would always generate an output.

4.2 Empirical Evaluation and Result

4.2.1 Evaluation Procedure

In this section, we present the details of the empirical study performed to evaluate the effectiveness of the approach for explaining assignment recommendations visually. This evaluation consisted of conducting a user study where we sought answers to questions about ease of understanding, recommendation trust, and preferred visualization. First, we present details about the web application created for our study. Then, we present the methodology of our study in which the web application was used.

4.2.1.1 Web Application

The web application used in our study consisted of two parts: a web browser plug-in and a web service.

Web Browser Plug-in

To present a participant with visual explanations of the assignment recommendations, we created a web browser plug-in, specifically a Google Chrome extension[5]. Web-browser plug-ins come in two varieties: those that customize a user's general browsing experience (e.g., Adblocker) and those that are activated only for a specific website (e.g., Amazon.com shopping list). Our plug-in is of the latter type.

[5]https://chrome.google.com/webstore/detail/recommend-expertise

To use the plug-in, first, the user must open a bug report in the web browser from a Bugzilla server. As our study used reports from Mozilla projects, the plug-in was configured to only work with reports from that project's ITS (Issue Tracking System). Next, the user clicks on the plug-in in the browser and clicks on a button labeled "Recommend Experts." This action sends a request to the web service with the bug report's id and opens a new browser window containing the response from the web service - an HTML page showing the assignment recommendations in a visual form. Figures 4.2, 4.3, 4.4 and 4.6 shows the four visualizations that are returned by the web service.

Web Service

When given the bug report id on Mozilla's ITS, the web service queries the ITS for the summary and description of the requested report. Stop words are removed and stemming applied to the text before being passed to a classifier. The results from the classifier are then used to create the visualizations. To avoid information overload, only the top ten (10) recommended developers are shown.

The web service was created using the REST (Representation State Transfer) model and implemented using the Django framework.

4.2.1.2 User Study

Our user study[6] consisted of three parts:

1. A demographic survey.
2. Presentation of the visualizations with an accompanying survey.
3. A post-usage survey.

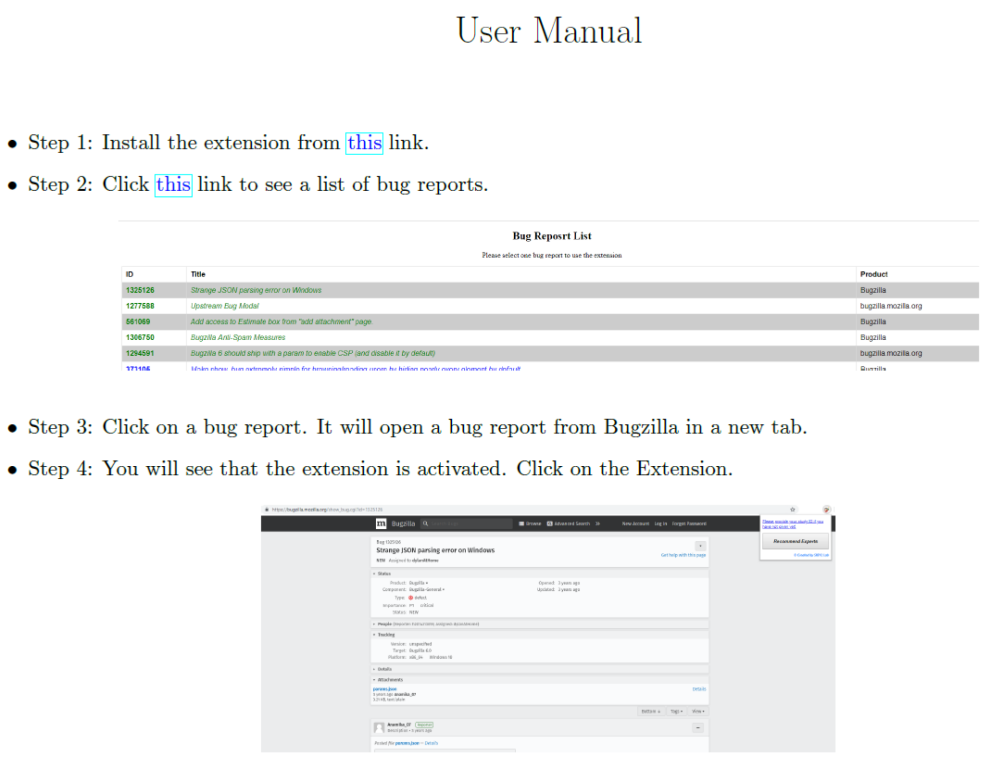
The demographic and post-usage survey was conducted using Qualtrics, and the visualization survey integrated into the web pages generated by the web service.

[6]The study was reviewed by the ethics committee of the University of Lethbridge and the assigned protocol number 2019-070.

To recruit participants for our study we posted in channels such as, `r/learnmachinelearning` and `r/AskComputerScience` on `reddit.ca` asking for software engineers to participate. The criteria for participation were to have either be in a two-year computer science post-graduate degree (i.e., in an M.Sc.-like program or equivalent) or have more than one year of software development experience in open source or commercial projects. Interested participants were asked to contact the primary researcher for a study id and further instructions.

Participants were given six items in the response email: their study id, a link to download the web browser plug-in, a link to the plug-in user manual (see Figure 4.1), links to the demographic and post-usage survey, and a link to a web page containing links for the Mozilla bug reports used in the study[7].

Participants were asked to complete the demographic survey first, then install the browser plug-in and go through the list of bug reports, and then complete the post-usage survey.



User Manual

- Step 1: Install the extension from this link.
- Step 2: Click this link to see a list of bug reports.

Bug Reposrt List

Please select one bug report to use the extension

ID	Title	Product
1325126	Strange JSON parsing error on Windows	Bugzilla
1277588	Upstream Bug Modal	bugzilla.mozilla.org
561069	Add access to Estimate box from "add attachment" page.	Bugzilla
1306750	Bugzilla Anti-Spam Measures	Bugzilla
1294591	Bugzilla 6 should ship with a param to enable CSP (and disable it by default)	bugzilla.mozilla.org
371106	Make show_bug extremely simple for browsing/reading users by hiding nearly every element by default	Bugzilla

- Step 3: Click on a bug report. It will open a bug report from Bugzilla in a new tab.
- Step 4: You will see that the extension is activated. Click on the Extension.

https://bugzilla.mozilla.org/show_bug.cgi?id=1325126

Bug 1325126
Strange JSON parsing error on Windows
NEW Assigned to dylan@hardison.net

Status
Product: Bugzilla
Component: Bugzilla-General
Type: defect
Importance: P1 critical
Status: NEW
Opened: 3 years ago
Updated: 3 years ago

Tracking
Version: unspecified
Target: Bugzilla 6.0
Platform: x86_64 Windows 10

Attachments
params.json

Figure 4.1: Excerpt from the user manual.

[7] http://thesis-final.fuam9wh3cq.us-west-2.elasticbeanstalk.com/reportlist/

Demographic Survey

Participants were asked to complete a demographic survey to allow us to understand who was participating in the study. Participants were asked the following questions:

1. To which gender identity do you most identify?
2. What is the highest degree or level of school you have completed?
3. What is your job function?
4. How many years of experience do you have with programming?
5. What level of experience do you have with triaging bug reports?
6. How frequently do you log a bug in an issue tracking system?
7. What level of experience do you have with using machine learning algorithms?

Presentation of Visualizations

To assess the effectiveness of the visual representation of recommendations, each participant was given a set of links to a set of pre-selected bug reports from the Mozilla project. The selected bug reports were randomly chosen from those that had a resolution and status of `FIXED` and `RESOLVED` respectively. Each participant was given the same fifteen bug reports to use.

After clicking the link for one of the bug reports, the participant was taken to the actual bug report in the Mozilla ITS. The participant would when click ”Recommend Experts” in plug-in and the web service would provide the recommendations using one of four randomly-selected visualizations (i.e. one of Figures 4.2, 4.3, 4.4, and 4.6). The participant would also be asked one of two sets of questions depending on the presented visualization.

The intent of presenting Figures 4.2 and 4.3 was to present participants with a single visualization for both algorithms. In this way, we could determine if participants preferred the use of one visualization approach over another (i.e., bar chart vs. pie chart). The intent of presenting Figures 4.4 and 4.6 was to determine if participants preferred a particular type of classifier.

For the visualizations that presented results from the two different classifiers (Multinomial Naïve Bayes and Topic Modeling), participants were asked:

1. Do you think these visualizations increase your understanding of the recommendation?
2. How much do you trust these recommendations? (1 being not trustworthy at all to 5 being you trust this fully.)
3. Do you think these visualizations provide you with enough information?
4. If no, what do you think the visualization is missing. Explain in a few words.

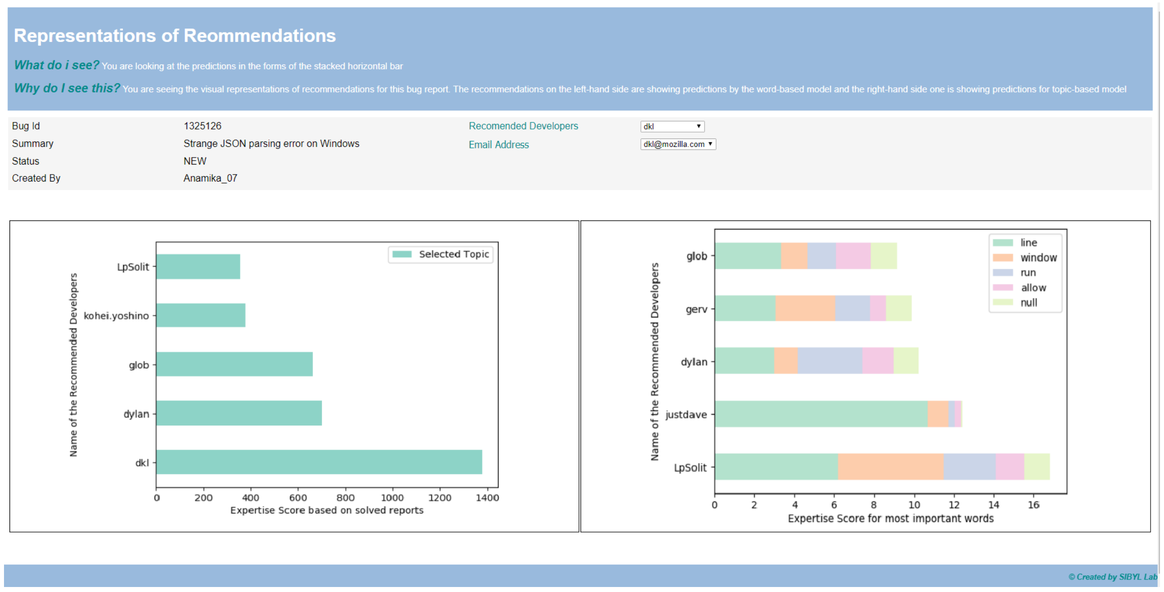
And for the visualizations where the same classifier was used, but the visualization differed (i.e. bar vs. pie vs. data-table), the participant was asked:

1. Do you trust these recommendations?
2. Do you think these visualizations provide you enough information?
3. If no, what do you think the visualizations are missing. Explain in a few words.

Post-Usage Survey

After participants finished using the browser plug-in on the fifteen bug reports, or however many they chose to do, they were asked to complete the post-usage survey. This survey contained the following questions about their thoughts about our approach to providing visual explanations of bug report assignment recommendations.

1. How important to you is the visual explanation of the recommendation?
2. If you were a triager, how useful would you find an assignment recommender?

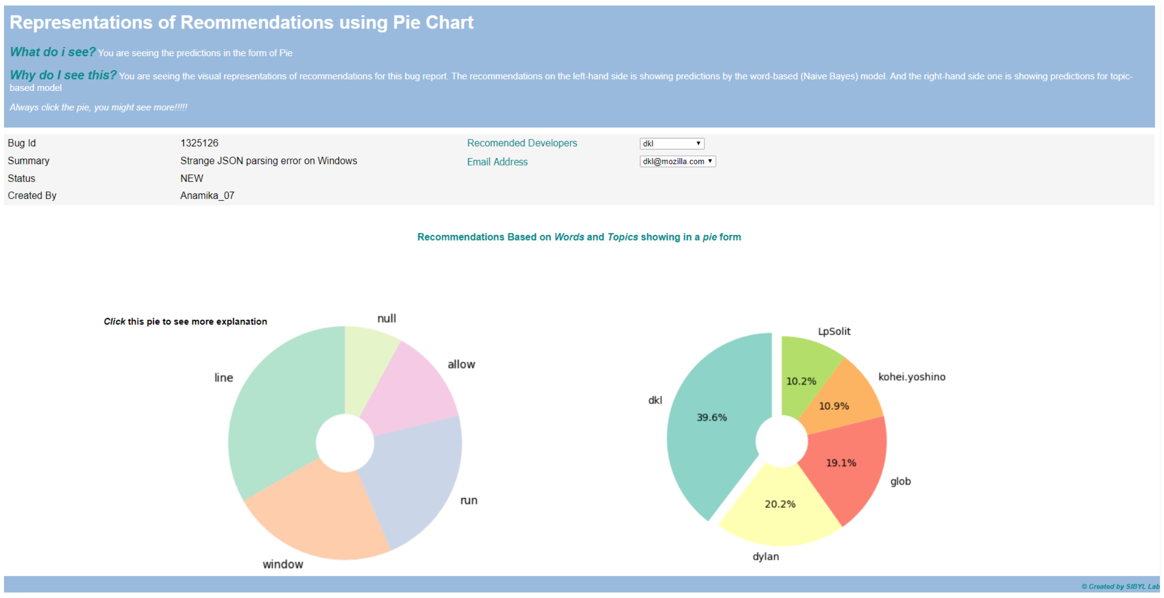

Representations of Reommendations

What do i see? You are looking at the predictions in the forms of the stacked horizontal bar

Why do I see this? You are seeing the visual representations of recommendations for this bug report. The recommendations on the left-hand side are showing predictions by the word-based model and the right-hand side one is showing predictions for topic-based model

Bug Id	1325126	Recomended Developers	dkl
Summary	Strange JSON parsing error on Windows	Email Address	dkl@mozilla.com
Status	NEW		
Created By	Anamika_07		

Figure 4.2: Visualization for recommendations in stacked bar form.


Representations of Reommendations using Pie Chart

What do i see? You are seeing the predictions in the form of Pie

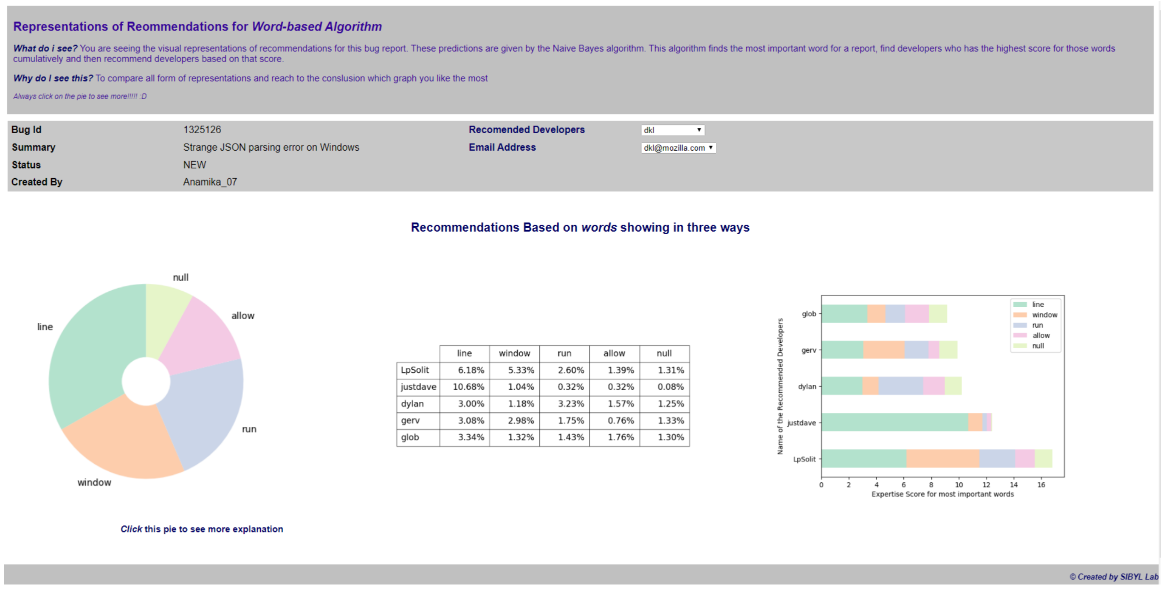
Why do I see this? You are seeing the visual representations of recommendations for this bug report. The recommendations on the left-hand side is showing predictions by the word-based (Naive Bayes) model. And the right-hand side one is showing predictions for topic-based model

Always click the pie, you might see more!!!!!

Bug Id	1325126	Recomended Developers	dkl
Summary	Strange JSON parsing error on Windows	Email Address	dkl@mozilla.com
Status	NEW		
Created By	Anamika_07		

Recommendations Based on *Words* and *Topics* showing in a *pie* form

Figure 4.3: Visualization for recommendations in pie chart form.



Representations of Reommendations for *Word-based Algorithm*

What do i see? You are seeing the visual representations of recommendations for this bug report. These predictions are given by the Naive Bayes algorithm. This algorithm finds the most important word for a report, find developers who has the highest score for those words cumulatively and then recommend developers based on that score.

Why do I see this? To compare all form of representations and reach to the conslusion which graph you like the most

Always click on the pie to see more!!!!! :D

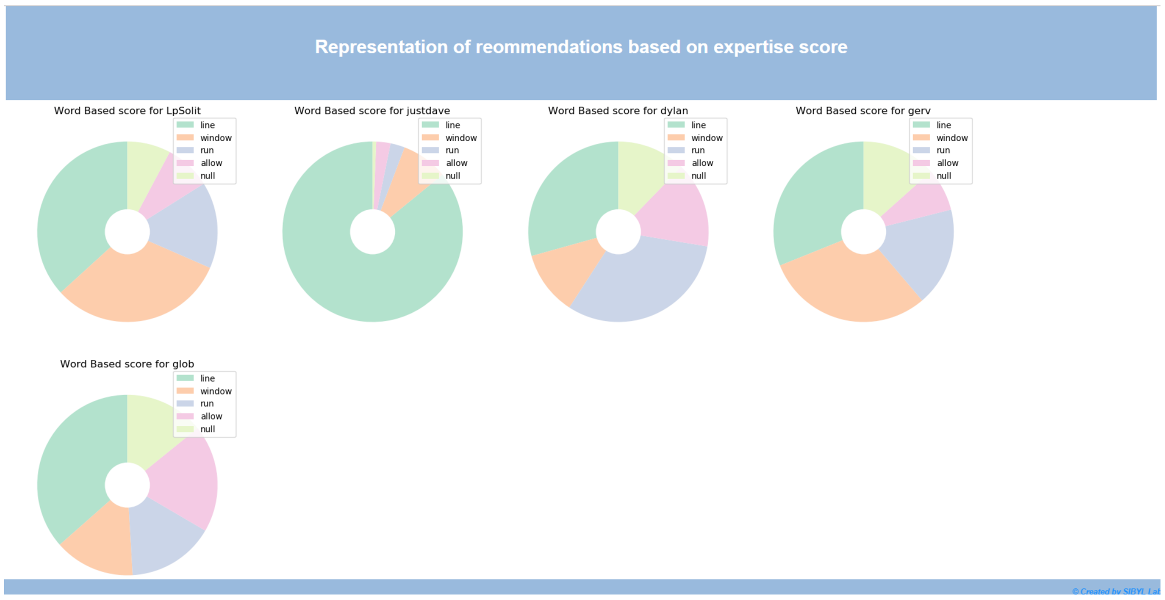
Bug Id	1325126	**Recomended Developers**	dkl ▾
Summary	Strange JSON parsing error on Windows	**Email Address**	dkl@mozilla.com ▾
Status	NEW		
Created By	Anamika_07		

Recommendations Based on *words* showing in three ways

	line	window	run	allow	null
LpSolit	6.18%	5.33%	2.60%	1.39%	1.31%
justdave	10.68%	1.04%	0.32%	0.32%	0.08%
dylan	3.00%	1.18%	3.23%	1.57%	1.25%
gerv	3.08%	2.98%	1.75%	0.76%	1.33%
glob	3.34%	1.32%	1.43%	1.76%	1.30%

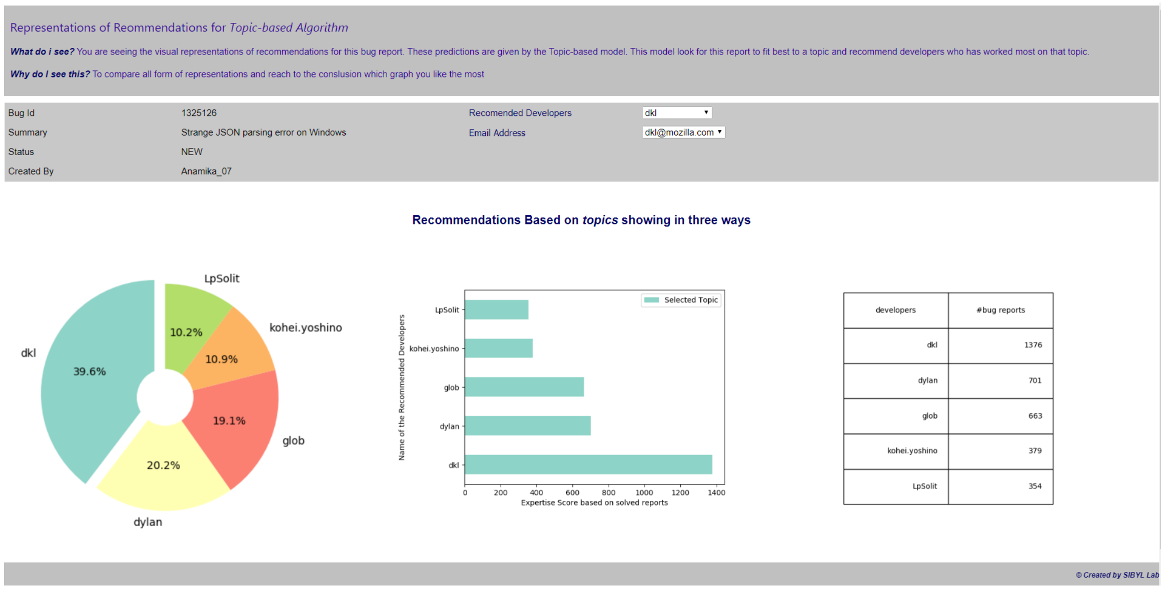
***Click* this pie to see more explanation**

Figure 4.4: Visualization of recommendations using Naïve Bayes classifier.



Representation of reommendations based on expertise score

Figure 4.5: Extended explanation of a pie chart for Naïve Bayes classifier.



Representations of Reommendations for *Topic-based Algorithm*

What do i see? You are seeing the visual representations of recommendations for this bug report. These predictions are given by the Topic-based model. This model look for this report to fit best to a topic and recommend developers who has worked most on that topic.

Why do I see this? To compare all form of representations and reach to the conslusion which graph you like the most

Bug Id	1325126	Recomended Developers	dkl ▾
Summary	Strange JSON parsing error on Windows	Email Address	dkl@mozilla.com ▾
Status	NEW		
Created By	Anamika_07		

Recommendations Based on *topics* showing in three ways

developers	#bug reports
dkl	1376
dylan	701
glob	663
kohei.yoshino	379
LpSolit	354

© Created by SIBYL Lab

Figure 4.6: Visualization of recommendations using Topic Modeling.

3. How would you improve the explanation of the recommendations?

4. Do you think that one visualization is enough?

5. Which combination of visualizations do you want?

Also, participants were asked if they wanted to be informed about the results of the study at a later time. If so, the participant could provide their email addresses.

4.2.2 Results

We now present the results of our user study. We found that, on average, a participant took an hour to complete the study.

4.2.2.1 Demographics of Participants

We had a diverse set of participants for our study, leading us to believe that our results are likely generalizable.

We had fourteen (14) participants in our study, with three (3) students, three (3) quality analysts, five (5) application developers, one (1) project manager, one (1) application architects, and one (1) other were given. The participants identified as 64% male and 36%

Table 4.6: Results for visualization preference.

Question	Stacked Bar Chart	Pie Chart
Do you think these visualizations increase your understanding of the recommendation?	77.00%	76.09%
How much do you trust these recommendations? (1 being not trustworthy at all to 5 being you trust this fully.)	3.46	3.41
Do you think these visualizations provide you enough information?	79.00%	77.78%

female. Just over half (57%) of the participants had a post-undergrad degree (Masters or Ph.D.). Only one participant had less than three (3) years of software development experience, with 50% having between four (4) and nine (9) years of development experience and the remainder (43%) having more than nine years of development experience. Most of the participants (71%) reported having logged a bug report, which indicates that most of them had some form of firsthand knowledge of how bug report assignment works. When asked about their level of familiarity with machine learning, two (2) reported themselves as beginners, with the rest considering themselves to have advanced knowledge about machine learning.

4.2.2.2 Presentation of Visualisation of Assignment Recommendations Result

Recall that participants were asked a different set of questions depending on if we were assessing preference for a type of classifier or assessing preference for a type of visual explanation.

Table 4.6 shows the results for the questions where we were trying to determine if there was a preference for one visualization over another. From this table, we can see that there was a slight preference for the stacked bar chart over the pie chart. We can also see that more than 70% of participants felt that these visualizations provided enough information.

Figure 4.7 shows that the participants trusted the recommendations from the Topic Modeling classifier more than those from the Naïve Bayes classifier.

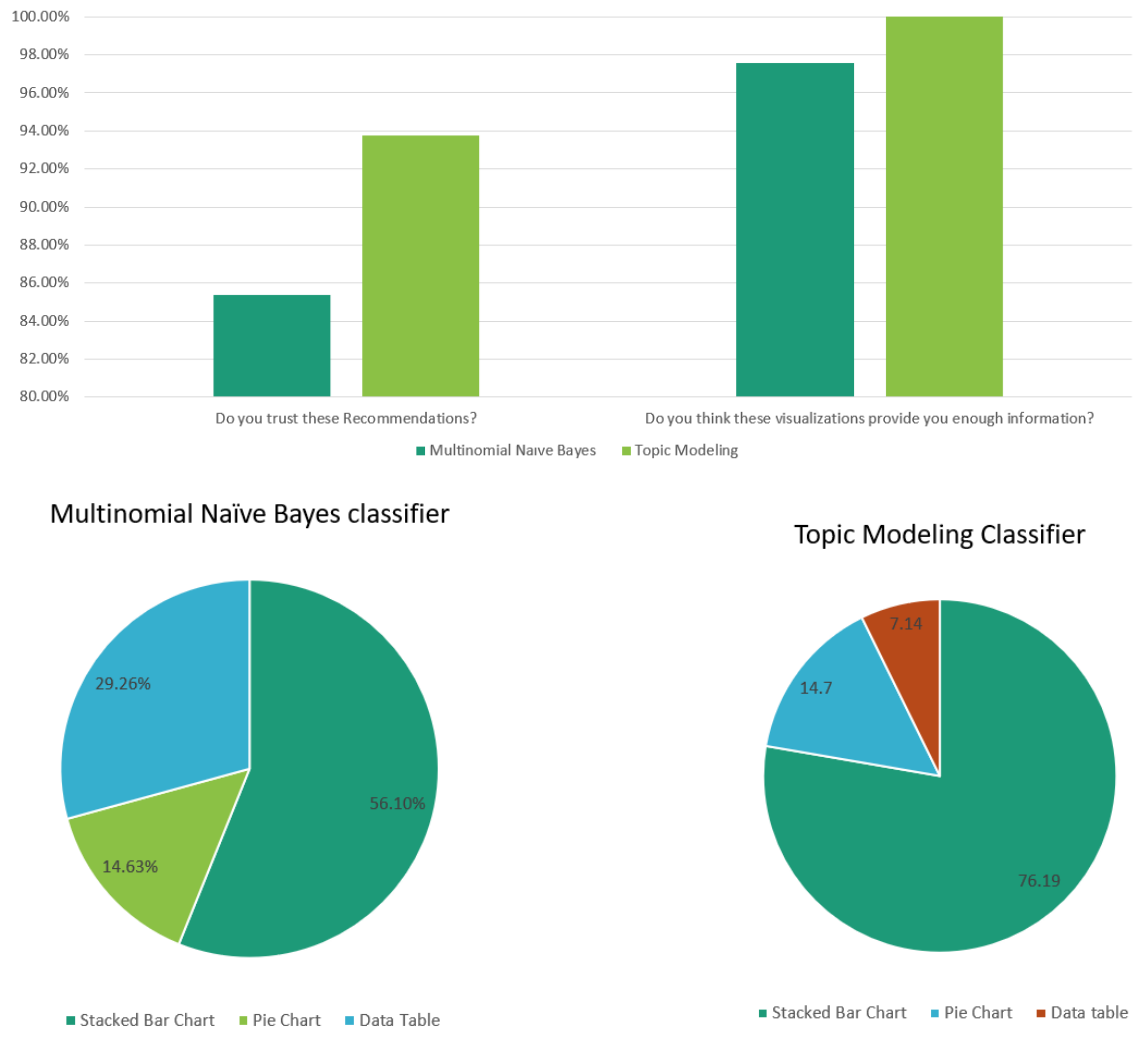


Figure 4.7: Results for classifier preference.

4.2.2.3 Post-Usage Survey Result

Figure 4.8 shows the results from our post-usage survey. The results show that more than half of the participants wanted to see more than one visual representation of the recommendations. Also, three-quarters of the respondents (75%) felt that it was very or extremely important to represent recommendations with explanations in a visual manner and that if they were a triager, they would find an assignment recommender useful.

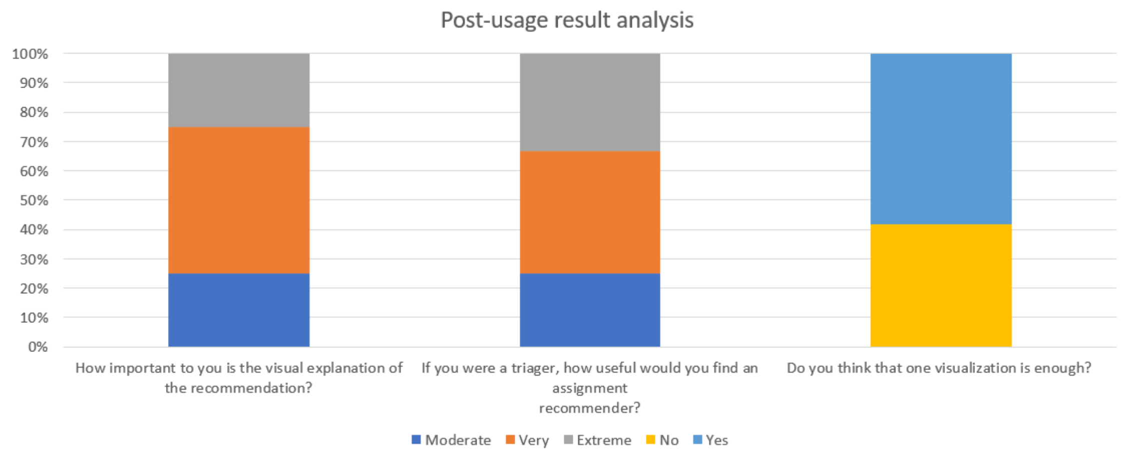


Figure 4.8: Post-usage result analysis.

4.3 Summary

Our main focus is to find the answers to our four research questions. To evaluate the system thoroughly, we evaluated the system in two ways: analytically and empirically. The analytical approach showed the system is sufficiently accurate and can produce enough information to provide an explanation for the recommendation in nearly all cases. We used k-fold cross-validation as the evaluation approach and accuracy as the evaluation metric. If the system gives us a positive result, we can say that our assignment recommender system can be reliable. From Tables 4.1, 4.2, and 4.3, we can see that the Multinomial Naïve Bayes classifier is more accurate than the Topic Modeling classifier.

The empirical evaluation (i.e., user study) evaluates the system from the user's perspective. By conducting this study and analyzing the results, we found that participants trust recommendations with the provided explanations with more than 80% of participants considering these recommendations as trustworthy. Also more than 75% of participants found that these visualizations are easy to understand.

Chapter 5

Discussion

In this section, a detailed discussion of our results in the context of the research questions is presented, including final conclusions. The sections is concluded with a discussion of the threats to the validity of this work.

5.1 Research Questions

In this section, we present a discussion of our findings towards answering our research questions. Recall that the research questions for this work are:

RQ1: Does Topic Modeling (i.e., an unsupervised learning algorithm) provides an acceptable accuracy to be an alternative to NB (i.e., a supervised learning algorithm)?

RQ2: Do developers find visual explanations of assignment recommendations easier to understand than a text list?

RQ3: Do developers find visual explanations of assignment recommendations more trustworthy?

RQ4: What is the preferred recommended visualization technique by the developers?

5.1.1 RQ1: Supervised Vs. Unsupervised Machine Learning Algorithm

To create the assignment recommender, we investigated creating a supervised machine learning algorithm (Multinomial Naïve Bayes) and an unsupervised machine learning algorithm (Topic Modeling). In the case of supervised machine learning algorithms, the data needs to be labeled, and then the algorithm uses that labeled input data to predict the output. For unsupervised machine learning algorithms, data does not need to be labeled, and it learns from the natural structure of the input data. As the aim for this work is to create an assignment recommender that provides recommendations that can be visually explained. Multinomial Naïve Bayes and Topic Modeling were chosen as the algorithms. Using these two examples, we can find which type of algorithm is more suitable.

From Tables 4.1, 4.2, and 4.3 we can see that the accuracy is higher for the Multinomial Naïve Bayes classifier than for the Topic Modeling classifier. Also from Table 4.1, 4.2, and 4.3, it is shown that the accuracy rate for the Bugzilla project is higher than the Thunderbird project regardless of the algorithm used. As mentioned previously, the data used for training the Thunderbird classifiers was less than that used to train the Bugzilla classifier(3351 vs. 5020). Therefore, we can state that for an unsupervised machine learning algorithm, the larger the training data-set is better the accuracy. Nevertheless, the supervised algorithm can provide reasonable accuracy using a smaller data-set.

Tables 4.4 and 4.5 show that when it comes to the explanation sufficiency of the recommender, the unsupervised machine learning classifier performs better than a supervised one.

From Figure 4.7, we can see that when it comes to the visual representation of the recommendations, participants trusted Topic Modeling more than Multinomial Naïve Bayes. However, in considering this result, it needs to be understood that participants were presented with more information with the Multinomial Naïve Bayes classifier than the Topic Modeling classifier. With the Multinomial Naïve Bayes classifier, the participant was presented with the relevant words, whereas this information was not provided with the Topic Modeling classifier. This added layer of information may have made the Multinomial Naïve Bayes visualizations more challenging to understand.

In the summary, we can say that, given a large enough data-set, Topic Modeling (or an unsupervised learning algorithm) can provide an acceptable accuracy as an alternative to

Naïve Bayes (or a supervised learning algorithm). However, if we do not consider the size of the training data-set, a supervised machine learning algorithm provides more accurate recommendations.

5.1.2 RQ2: Ease of Understanding Visual Explanations

Recall that, Table 4.6 shows that more than 76% of the participants find visual explanations of assignment recommendations easier to understand. From the participants' scoring pattern, we found that the more they used the extension, the better they understood the explanations. We think that is why 24% of users do not find that easier to use. In short, we can say that participants do find it easier to understand the recommendation with the visual explanations than a text list.

5.1.3 RQ3: Trust of Visual Explanations

We considered the trustworthiness of visual explanation as a critical issue. If people do not trust the explanations, they will not use it in the future. Therefore, the visual explanation must provide enough information to be trustworthy to the participants. We asked whether they found that these visualizations provide them enough information to make a bug report assignment decision. More than 77% of times participants think they provide enough information. From the participants' comments, it can be stated that they wanted to know about the topic for Topics Modeling based recommendations. That is why almost 23% of the time, they felt that the information is not enough. While answering the question about trustworthiness when presenting any visual explanations, Figure 4.7 and Table 4.6 shows that most participants felt that these visual explanations of assignment recommendations are trustworthy.

5.1.4 RQ4: Prefered Visualization

We wanted to know the preferred recommended visualization technique by the developers, from the three styles of visual graphs: stacked bar chart, pie chart, and data-table.

For word-based recommendations, participants have to click on the pie to find more explanations. Some of the participants did not find this to be user-friendly. When it comes to explaining recommendations using the data-table, one participant said: “it is not interesting.” But the stacked bar chart explains the recommendations easily with different colors and without being redirected to a new web page. Therefore, from Figure 4.7, we can see that participants preferred the stacked bar chart over any other graphical representations.

5.1.5 General Comments from Participants

The post-usage survey collected the general thoughts and impressions of visual explanation for assignment recommendation. We asked the participants whether they thought that the visual explanation of recommendations is essential. From Figure 4.8, we can see that 85% of the participants think it is very important to explain what the system is recommending visually. When it comes to the usefulness of an assignment recommender, 85% of the participants believe an assignment recommender is very useful. From Figure 4.8, we can also see that almost all participants want to see more than one kind of visual explanations. We also asked for their comments/suggestions on how to improve the explanation of these recommendations. Many of the participants wanted to see the selected terms from the cluster in topic-based recommendations. They suggested that if the terms were provided, it would be easier for them to understand why the bug report falls into that selected cluster. One participant commented that for word-based visual recommendations, we should increase the number of words we are showing. Another participant suggested adding a tooltip-text (i.e., a mouse hover event) to make the websites more responsive. However, all of the participants commented that they understood the ideas and principles of the work, and with the addition of a few more features, the representations can be even more responsive, informative, and attractive.

5.2 Threats to Validity

In this section, we present a discussion on external and internal threats to the validity of our work.

5.2.1 External Validity

Although in our study, we trained our classifier using data from a single Mozilla product - Bugzilla, we do not feel that this limits the generalizability of our results. As our focus was on the representation of the recommendations, not the accuracy of the recommendations, our results are not dependent on the project used. Generalizability related to using an open-source project vs. a commercial project, and few projects vs. many projects are regarded to be a concern.

5.2.2 Internal Validity

Although we found that the accuracy of our created recommender suffered due to an imbalanced data set (i.e., the developer who solved the greatest number of the bug reports is always in the list recommended developers), this is unlikely to have affected the results of our investigation. As none of the participants were from the Bugzilla project, the accuracy of the recommender was not a factor in their responses.

The inclination to generalize our results to a larger set of projects is bound by the limited size of our data set. Our experiments were applied to two projects, and although we gathered data for 5 years, one of the projects still did not have many bug reports. In addition, both projects are open source, which means it has its own distributed development structures. We can not state that our findings can be equally applicable to another kind of software projects that do not use the open-source platform.

Like any machine learning algorithm, both algorithms have the same problem. A developer who solved the greatest number of the bug report is always in the list recommended developer. For example, a person named *dkl* solved 1376 bug reports, whereas the next highest solver solved almost half. As a result, *dkl* is always on the list. However, our algorithms bring other developers to the list, so even though one person may always be there, the user has other recommended developers too.

During the recruitment of participants, we tried to cast as broad of a net as possible. As a result, some of the participants were known to us. However, as the collected data identified participants only by study id, and we did not seek to discover who was whom, we do not know which responses were from these people. Also, participants were asked not to contact us personally to let us know their personal views during the study. In other words, we tried not to influence the result in any way.

Nevertheless, there is a possibility that our results may suffer from social desirability bias (i.e., “please the researcher” bias). Based on the trend where participants initially reported that they had low trust in the recommendations, and then the trust level improved, we do not feel that such bias had a significant impact. However, we cannot discount this possibility from our study results.

5.3 Summary

In this chapter, we discussed the four research questions of our work, as well as threats to the validity of our conclusions. We found that, given a large enough dataset, an unsupervised learning algorithm can provide an acceptable accuracy as an alternative to a supervised learning algorithm. From the study, it can be stated that most of the participants found the visual explanations of assignment recommendations easier to understand than a list of names. Also, the visual explanation provided enough information to be trustworthy to most of the participants. Among the three types of representations, the stacked bar chart was favored by most of the participants. Finally, the users wanted to see more than one visualization and that the visualizations be more interactive.

Even though there is a possibility that our results may suffer from social desirability bias, the user study results show that the more users used the extension, the more positive reaction they showed towards the tool.

Chapter 6

Related Work

The following sections present previous works on expert recommendations using different software artifacts, the use of machine learning algorithms in the recommendation system, the usefulness of explaining the recommendations, and visual explanations of recommendations.

6.1 Expert Recommendation

Recommendation systems are mainly an extension of the normal process of recommendation by word-of-mouth to other people. Usually, every recommendation system can provide a narrow or personalize recommendations based on users' expectations. Creation of user profile and learning from them, experimentation with different recommendation algorithms, and design user interface so that the system is more user-friendly, this kind of research has become a popular idea among researchers from the early 90s [4, 8, 13]. Glance et al. [8] states that it is essential to have a recommendation system that gives users suggestions related to their needs. They were one of the first people that researched recommendation systems, and their usage and effects in different fields. Even though they thought recommendation techniques should be based on indexing, retrieval, and relevance feedback, but they were more focused on the last one. Their goals were not just to create a recommender tool using recommendation algorithms and user profile construction methods but also focusing on the issues that motivate users to use the recommender system in an organization.

As the researchers wanted to support the workplace community and help automate the process of sharing recommendations, they implemented and deployed a research prototype named “Knowledge Pump.” They implemented their prototype in Java as a client-server system. Users could use it as an applet in a web browser. By being available in both web-browser and Java, their tool was cross-platform. When the first-time user created a profile in the Knowledge Pump, they were asked multiple questions. For example, they were to choose a set of advisors from current users whom they think are most trustworthy as referrals, and their domain of interest from a set of given communities. As a browser applet, the Knowledge Pump provided users with two main functionalities. The first one was a shared bookmarking system. This system let a user bookmark a web-page that they think might be useful for other members of the organization. It helped users search and browse through the set of shared bookmarks. It had a “what’s new” button that allowed users to see the most popular items shared among the communities. The second one was a list of recommended articles, case studies, customer solutions, and web-pages which get updated every day. When a user reloaded the web browser, Knowledge Pump provided a list of “What’s Recommended?” where each recommendation had stars. These represent the likeliness of whether Knowledge Pump thinks the user would find the article interesting. It also included a list of reviewers that previously rated and commented on the items. The Knowledge Pump also provided a link where users could see each reviewer’s ratings, comments, and time of evaluation for the item. The researchers conducted a thorough user study for one year, where 66 people participated in that study. They found that only a portion of these users were active participants. These active contributors contributed more than they received from the Knowledge Pump. Even though their shared bookmarks idea was unique, the researchers thought they needed to work more on that. But overall, they found that the recommender system is highly essential for sharing knowledge support.

To create an expert recommendation system in software engineering domain, researchers have been using a different kinds of software artifacts.

In any software company, whether it is big-scale or not, global or not, software engineers get a good number of the bug reports for their ITS. These reports then need to be triaged, which means to sort the reports based on their duplicate entries, importance, and later assigning these reports to a developer. If there is a new project coming on the farm, the manager also has to decide which people have the most knowledge to be part of that project team. In order to help them, recommendation systems have become a commonly researched part of software engineering. Anvik and Murphy [3] and Ma et al. [18] used source code as an artifact to provide expert recommendations. Both of their works insisted on the fact that the more a developer works on the code base of a project, the more knowledge they gather, which makes them expert for that project.

Recommendation system can help developers by giving a list of experts that can review the code they write for any project or even a specific bug report. In most software companies, reviewing code by patches is a common practice for Quality Assurance (QA). After one or more people review each patch, the code is usually merged into the Version Control System (VCS). Hannebauer et al. [10] performed an empirical study and showed that using a recommender system can mitigate the problems of finding relevant reviewers. They compared six algorithms based on modification expertise and two algorithms based on review expertise on four major Free/Libre Open Source Software (FLOSS) projects. They used File Path Similarity (FPS) and Weighted Review Count (WRC) algorithms to predict based on review expertise. Later they evaluated their results with modified expertise algorithms like Line 10 Rule, Expertise Cloud, Degree-of-Authorship (DOA), and many more. They found that for every set of evaluation, higher accuracy is provided from the review expertise algorithms. Not only that, but also, they found that WRC is more reliable, less character sensitive, and shows better results than FPS to recommend expert reviewers.

In reality, for many software companies, the main unit of work is the number of solved bug report. Knowing which developers have code base expertise (implementation expertise) can help to solve a particular bug report has many usages. Anvik and Murphy [3] used two approaches to recommend expert developers for solving a new bug report. The first one is to find who checked-in last in the source repository, and the second one is to use the value of “status” and “assigned to” fields in a bug report. For the initial approach, they created expertise sets for a bug report by using source repository check-in logs in three steps. The first step is to create a link between the bug report and the source repository. The result is a list of source files with information about the changeset of the report. The next step is to find the containing module for each source file in the changeset. From there, they wanted to find a list of developers’ names who committed changes to these modules. This list can be acquired by using the “Line 10” heuristic. Finally, this set is filtered to find the set of relevant developers. For the second approach, the researchers used bug reports and bug networks. A bug report contains different types of information in the form of predefined fields, free-form texts, attachments, and dependencies. To create a list of experts from a bug report, the researchers first created a bug network that contains the report, and then they extract the names of the people in the carbon-copy (CC) list, who added comments to the report and who fixed the bug. They created lists in every step and finally merged them to form a list of experts for the bug report. This set of the recommended expert was also filtered to remove irrelevant developers. The expertise set created by these two given approaches was compared to the expertise sets produced by the project experts. Both approaches can give better results depending on what was expected, i.e., if it is necessary to find expertise set with as few false positive as possible, then the source repository approach using changeset gives a better result. The researchers also found that the bug report can be a better substitute for the source repository approach using changesets, mainly when the source repository data is not that precise.

Previous researches were done to find expert recommendations based on expertise artifacts such as files, packages, and software repositories. However, these expertise artifacts are usually specific to a given project. Schuler et al. [32] proposed mining usage expertise to make it possible to get independent project expertise that can be used across different projects. Their method to recommend experts for code worked with very little or no source revision history, which eventually helped new developers to find experts for a specific project.

Ma et al. [18], on the other hand, proposed a method where they built two types of expertise profiles. The two profiles were for changed methods and inserted methods for each developer. Based on how many times and when a developer changed method, a scoring formula was used to build an implementation expertise profile. Similarly, a usage expertise profile score was determined based on the depth and breadth of the change in the method. Profiles and queries were then used as input to heuristics, which would then generate a ranked list of developers with the most expertise. After every commit to the version control system, the expertise profile would be updated. To evaluate their approach, they considered a recommendation successful when an actual commit author was found in the recommendation list. We adopted a similar approach in our analytically evaluation.

6.2 Use of machine learning

A Recommender can be useful for developers while writing codes. It is a prevalent practice among programmers to use API (Application Programming Interfaces). But not all the API has proper documentation or inspect code examples. As a result, using API correctly and accurately from unknown libraries and frameworks is not easy for all APIs. Zhang et al. [44] proposes an automated tool named `Precise`. This tool, instead of recommending only API method calls, predicts all the parameters for that method. It also helps to simplify the API usage by helping developers with code completion. By proposing recommendation, precise helps the programmer to reduce the effort of finding some pieces of information from examples. It also helps them to map them in the right places. Precise built a usage database by evaluating the parameter usage instances and their context. After getting the recommendation request, it then ran a query in the database. It uses the context as key and fetches a bunch of abstract usage patterns. Then it shortens the list, sorts it, and finally provides the recommendation. For finding the similarity of recommended methods, they used a customized K-Nearest Neighbors Algorithm (KNN) algorithm. The kind of recommendations made by Precise are, in general, more complicated, hard to find or compose and not recommended by the actual Eclipse JDT. At the same time, the user study showed that Precise is indeed a useful tool. But it was also suggested to include an explanation for their ranking strategy as well as it should integrate better with other techniques.

To help the triagers with bug report triaging, Anvik et al. [2] proposed a semi-automatic approach to recommend a list of developers to whom the bug report should be assigned. They used Eclipse and Firefox project bug reports for training purposes. Their approach consisted of four steps. First of all, they characterized the bug reports. They decided to use both the one-line summary and description. In this step, they created a feature vector that shows the frequency of the terms in texts. The second step is to assign a label. They used a project-based heuristics method for labeling to avoid issues like generic “assigned-to” label. For example, in the case of the Firefox project, it’s the last one who approved the patch and for Eclipse, the last person to change status. The third step is to choose a set of bug-reports to train the algorithm. In order to select the trained data-set, they filtered the whole data-set three times. They discarded all the reports which do not have a useful label. They also disregarded reports for which the developer no longer works in the company. Then they filtered out all those developers, and their bug reports, who did not solve a minimum of three (3) reports each month for the most recent three months. The last part of their proposed method wa to decide which machine learning algorithm is applicable in this case. They used Naïve Bayes , Support Vector Machines (SVM), and C4.5 algorithm. From the result of all these three algorithms’ precision and recall value, they finally decided to use SVM. Their precision with SVM on the Eclipse and the Firefox project data-set was 57% and 64% respectively.

Researchers have found that Naïve Bayes is an exceptionally well-performing text classification algorithm and have frequently adopted the algorithm in recent work. For text classification, Naïve Bayes has the two most efficient and frequently used model: multi-variate Bernoulli model and multinomial model. McCallum and Nigam [19] described these two models in their paper and showed the differences between these two models. They showed that multi-variate Bernoulli performs well with small vocabulary sizes; however, if the vocabulary size is large, then the multinomial performs better.

However, Topic Modeling is relatively new in the assignment recommender system. Nguyen et al. [25] used topic modeling to find duplicate bug reports. They proposed a new approach, named DBTM. This approach uses both information retrieval and topic-based features to identify duplicate bug reports. The model takes advantage of both IR based features and topic-based features. These features focused on finding the textual dissimilarity between duplicate reports. Their topic model finds not only the similar technical issues of the bug report but also the semantic similarities between duplicate reports based on topic structures. DBTM is a combination of two components: T-Model and BM25F model, where T-Model is an extension of Latent Dirichlet Allocation (LDA), and BM25F is an advanced document similarity function based on weighted word vectors of documents. In their approach, each bug report is contemplated as a textual document that narrates one or more technical issues. This model used LDA to depict the topic structure for a bug report. LDA also finds the duplication relations among the bug reports. The topic selection of bug reports gets affected by the topic distribution of that report as well as problematic topics for which the report was submitted. They also applied the Ensemble Averaging technique to combine both IR and topic modeling. Finally, they found that DBTM is scalable and efficient in detecting duplicate bug reports accurately. In the end, they concluded by saying that their system can improve the current state of the approach by 20% as DBTM utilizes both IR and topic modeling-based features.

In recent years topic modeling was also used in bug report triaging. Bug report triaging can be a tedious task. In order to help software engineers/managers with bug report triaging, Xia et al. [40] proposed a new framework that maps every word(term) from a bug report to their corresponding topics (topic). They created a special topic modeling algorithm named Multi-Feature Topic Model (MTM). Their model was created using Latent Dirichlet Allocation (LDA). MTM maps the terms with topics by using product and component information from bug reports. They have also proposed a new method named TopicMiner. When a new bug report comes in, the TopicMiner takes into account the topic distribution of that report and recommends an expert based on how many reports they solved to that related topic. They combined their MTM and TopicMiner and evaluated their solution on 5 bug report data-sets. For the top one recommendation, their model gets accuracy more than 45%, and for the top 5, it is higher than 75%. Finally, they compared their approaches with some of the currents approaches [35, 43, 34, 23], and found that their approach precedes those approaches.

6.3 Importance of explanation

Usually, the recommender system works as a black box. It does not explain the list. As a result, people who do not have good knowledge of algorithms might get confused. Explaining the reason that why a recommendation system recommends a list of people/-movie/song is important to many users. Trintarev et al. [37] surveyed a focused group of moviegoers to show what users think of a recommendation system. In total, 11 people participated in their study. These moviegoers were divided into two groups. They wanted to know how participants would like to be recommended or discouraged from watching a movie. Form the study, they came to a few decisions. First, while explaining the recommendations, features should be selected according to the user's preference. Second, in explanation, features should also be selected based on context. Third, the shorter the feature list, the better. Fourth, a credible source of recommendations is important to users. Moreover, the result showed that the most appropriate explanation also depends on some other factors such as mood and source.

6.4 Visualizing Expertise

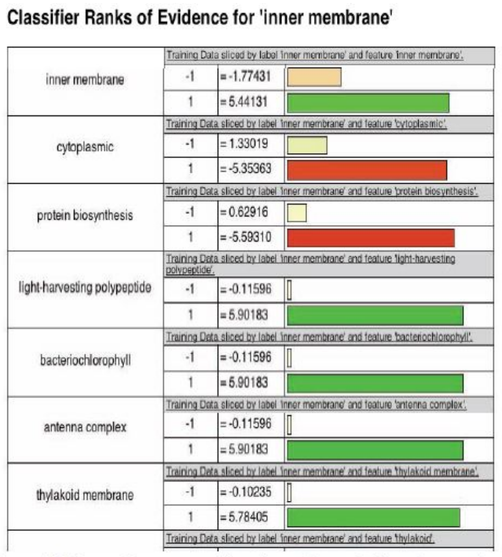
Some classic recommendation systems for software engineering like Expertise Browser [21] and Mylar [15] realized the visualization was important.

Expertise Browser (ExB) [21] is a tool that uses data from change management systems to locate people with desired expertise who are distributed in a large geographical area. It is a web-based platform where information is displayed in the form of HTML pages. Users can run the query to find the people who have the adequate expertise for an individual product. This tool also provides the expertise profile of an expert. Rather than letting a recommender system to provide a list of experts, this tool allowed users to choose the most prolific experts by browsing all the available expert. To measure the degree of experience of an expert, they use Experience Atoms (EAs). They use deltas, which keeps tracks of the changes made in the code. They use Modification Requests (MRs) that indicated to make a change to the software entity for a single purpose at the correct time by using deltas. If EAs failed to measure the expertise, they would introduce a substantial degree of support. Expertise Browser can be used as a Java Applet. In order to evaluate their system, they ran a user study. The researchers implemented ExB in several projects of two organizations where each of them has more than one workstation. From the study, the researchers found that in most cases, the user finds it extremely helpful that they can see experts profile, which gives them an explanation of why this person is an expert. The user gave feedback that also suggests that they wanted even more explanations.

Kersten and Murphy [15] decided to use the degree of interest (DOI) model. They created a tool named Mylar, an Eclipse plugin, which automatically encodes the context of the programmer's task in a DOI model and shows it in IDE views. The default highlighting scheme uses colored shading to indicate the programmer's relative interest in the element. Mylar records developers' activity and whenever a developer chooses or edits a program element, Mylar escalates the interest level of that specific element. Mylar's interest model encodes the relevance of systematic program elements rather than recording the unsystematic places where a document is edited. It does not capture navigation paths but correlates editing and navigation activity with program elements. As Eclipse already indicates currently selected elements, Mylar shows them in bold font. Using Mylar has four benefits. First of all, interest-based filtering is possible. As a result, the relevant files and libraries are more visible. Second, from a large number of elements, the specific problem of interest is highlighted. Third, the Mylar editor can provide a choice of actively folding and unfolding elements based on the interest filtering. Finally, this view is updated to provide an idea of how high-interest elements fit into the crosscutting structure of the system. They performed a user study to see how real-life programmers find it useful. Six senior IBM Toronto lab programmers participated in this study. All developers stated that the representation of the context of their task by the model was entirely correct and transparent. Even though highlighting was necessary, a few of the subjects stated that they did not like the color used to highlight. Subjects found the automatic filtering and auto-expansion mode useful.

Lu et al. [17] constructed five machine-learning classifiers to predict the most accurate subcellular localization of proteins from animals, plants, fungi, Gram-negative bacteria, and Gram-positive bacteria. Their predictor was a part of the Proteome Analyst (PA) web-service. After using their classifier, Proteome Analyst could than also be used to create new sub-cellular classifiers using custom training data. PA used labeled training data to build a simple Naïve Bayes classifier, which generates a probability for each label. They have used a stacked bar chart to show their prediction graphically.

Poulin et al. [31] described a framework to explain decisions made by different classifiers. They demonstrated their framework by using implementations of Naïve Bayes, Linear Support Vector Machine, and Logistic Regression classifiers. ExplainD used a simple graphical way of representation. This representation gives five types of explanations: Decision, Evidence of that decision, Speculation of that decision, Ranks of evidence, and finally, the source of that evidence. They used an example of a classification model used by a physician on the diagnosis of obstructive Coronary Artery Disease (CAD). They showed in their work that ExplainD could be added to any classifier that is an additive model. They merged their idea with Proteome Analyst(a popular bioinformatics application) and found that their graphical explanation helped both experienced and less-experienced users to find an error in PA. Their graphical representations are shown in figure 6.1. It helped users to build trust in classifier's results, increase the knowledge on the relationship between feature values and decisions for the classifier. As it shows visual representations of the decisions, it gives the user a visual idea of the classifier's result, and that helps them to find errors as well.


Classifier Ranks of Evidence for 'inner membrane'

inner membrane	Training Data sliced by label 'inner membrane' and feature 'inner membrane'.		
	-1	= -1.77431	
	1	= 5.44131	🟩
cytoplasmic	Training Data sliced by label 'inner membrane' and feature 'cytoplasmic'.		
	-1	= 1.33019	
	1	= -5.35363	🟥
protein biosynthesis	Training Data sliced by label 'inner membrane' and feature 'protein biosynthesis'.		
	-1	= 0.62916	
	1	= -5.59310	🟥
light-harvesting polypeptide	Training Data sliced by label 'inner membrane' and feature 'light-harvesting polypeptide'.		
	-1	= -0.11596	
	1	= 5.90183	🟩
bacteriochlorophyll	Training Data sliced by label 'inner membrane' and feature 'bacteriochlorophyll'.		
	-1	= -0.11596	
	1	= 5.90183	🟩
antenna complex	Training Data sliced by label 'inner membrane' and feature 'antenna complex'.		
	-1	= -0.11596	
	1	= 5.90183	🟩
thylakoid membrane	Training Data sliced by label 'inner membrane' and feature 'thylakoid membrane'.		
	-1	= -0.10235	
	1	= 5.78405	🟩
	Training Data sliced by label 'inner membrane' and feature 'thylakoid'.		

Figure 7 Naïve Bayes application of capability 3 - *ranks of evidence* in the ExplainD prototype.

Figure 6.1: Graphical representation of ExplainD.

6.5 Summary

Using a recommendation system in different sectors has become very popular from the very early 1990s. Researchers have found significant potential in this field and started using it in different fields like matchmaking, commercial product suggestions, related research paper suggestions, and so forth. In software engineering, a recommendation system can be beneficial to developers while writing code for any project by suggesting such items as an

API or a developer's name who worked in this kind of project before or developers who can review their codes. Recommender systems can be helpful to quality analysts by providing names of developers who have solved the same kind of bug reports before. Similarly a recommender system can help project managers to solve bug triage issue in the more efficient ways. When it comes to creating a recommender system, using a machine learning algorithm has been one of the most popular choices. For visualizing the recommendations, users want to know why and how the recommendations are made. In the software engineering sector, usage of visualizing is still an under-researched area.

Chapter 7

Conclusion

In this thesis, we wanted to assist bug report triagers for assigning a new bug report. We want to create an assignment recommender that provides accurate recommendations. Instead of just the usual black-box approach where a triager or any user can only see a list of recommendations, we wanted to visually present our recommendations in such way so that anyone can see the graphs and understand why these people have been recommended. We determined that our goal for this work is to find answers to four research questions:

RQ1: Does Topic Modeling (i.e., an unsupervised learning algorithm) provides an acceptable accuracy to be an alternative to Multinomial Naïve Bayes (i.e., a supervised learning algorithm)?

RQ2: Do developers find visual explanations of assignment recommendations easier to understand than a text list?

RQ3: Do developers find visual explanations of assignment recommendations more trustworthy?

RQ4: What is the preferred recommended visualization technique by the developers?

We used one supervised (Multinomial Naïve Bayes) and one unsupervised (Topic Modeling) algorithms to create two assignment recommenders. We used a Bag of Words model, TF-IDF score from information retrieval techniques, along with the basic probabilistic equations to create the Multinomial Naïve Bayes classifier. We have used the `Word movers distance` model that uses a `word2vec` technique, as well as TF-IDF, while creating the topic model classifier. When it comes to the number of recommended experts, we provided five developers names as the experts. The logic behind that is, a lot of the time the first few developers can be unavailable. By creating our assignment recommender to be as accurate as possible, we answered our 1st research question.

The next goal of this work was to investigate the use of visualization for explaining bug report assignment recommenders. To accomplish this, we created a web service that provides explanations of assignment recommendations using either a bar chart, a pie graph, or a table. We also investigated whether explaining a supervised learning (Multinomial Naïve Bayes) or unsupervised learning (Topic Modeling) recommender was preferred. We found that developers did prefer visual explanations, with 75% of participants stating that the visual explanations increased their understanding of the assignment recommendations. We also found that developers gained trust in the recommendations over time. Finally, we found that the developers preferred a stacked bar chart.

7.1 Future Directions

When designing the visualization for the Topic Modeling classifier, we felt that including the topic names and related words, the developer names and expertise score in one graph would be confusing. However, from comments provided by participants, we discovered that our representation of recommendations for the Topic Modeling classifier was not detailed enough. Many participants commented that they also wanted to see the topics for this type of recommender. A possible solution suggests by a participant is to have a tooltip that provides extra information to the triager on what are they seeing. Another possibility could be to allow the triager to click on a pie piece and have a pop-over that provides further details as with our Multinomial Naïve Bayes visualization.

Bibliography

[1] Franz L Alt. *Advances in computers*, volume 2. Academic Press, 1961.

[2] John Anvik, Lyndon Hiew, and Gail C Murphy. Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering*, pages 361–370. ACM, 2006.

[3] John Anvik and Gail C Murphy. Determining implementation expertise from bug reports. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 2. IEEE Computer Society, 2007.

[4] Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997.

[5] Florian Beil, Martin Ester, and Xiaowei Xu. Frequent term-based text clustering. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 436–442. ACM, 2002.

[6] Svetlin Bostandjiev, John O'Donovan, and Tobias Höllerer. Tasteweights: a visual interactive hybrid recommender system. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 35–42. ACM, 2012.

[7] Michael Fischer, Martin Pinzger, and Harald Gall. Analyzing and relating bug report data for feature tracking. In *WCRE*, volume 3, page 90, 2003.

[8] Natalie Glance, Damián Arregui, and Manfred Dardenne. Making recommender systems work for organizations. In *Proceedings of PAAM'99*, pages 19–21. Citeseer, 1999.

[9] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[10] Christoph Hannebauer, Michael Patalas, Sebastian Stünkel, and Volker Gruhn. Automatically recommending code reviewers based on their expertise: An empirical comparison. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 99–110. ACM, 2016.

[11] James D Herbsleb, Helen Klein, Gary M Olson, Hans Brunner, Judith S Olson, and Joe Harding. Object-oriented analysis and design in software project teams. *Human Computer Interaction*, 10(2-3):249–292, 1995.

[12] Jonathan L Herlocker, Joseph A Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 241–250. ACM, 2000.

[13] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 194–201. ACM Press/Addison-Wesley Publishing Co., 1995.

[14] Da Huo, Tao Ding, Collin McMillan, and Malcom Gethers. An empirical study of the effects of expert knowledge on bug reports. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pages 1–10. IEEE, 2014.

[15] Mik Kersten and Gail C Murphy. Mylar: a degree-of-interest model for ides. In *Proceedings of the 4th international conference on Aspect-oriented software development*, pages 159–168. ACM, 2005.

[16] Ashraf M Kibriya, Eibe Frank, Bernhard Pfahringer, and Geoffrey Holmes. Multinomial naive bayes for text categorization revisited. In *Australasian Joint Conference on Artificial Intelligence*, pages 488–499. Springer, 2004.

[17] Zhiyong Lu, Duane Szafron, Russell Greiner, Paul Lu, David S Wishart, Brett Poulin, John Anvik, Cam Macdonell, and Roman Eisner. Predicting subcellular localization of proteins using machine-learned classifiers. *Bioinformatics*, 20(4):547–556, 2004.

[18] David Ma, David Schuler, Thomas Zimmermann, and Jonathan Sillito. Expert recommendation with usage expertise. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 535–538. IEEE, 2009.

[19] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer, 1998.

[20] Audris Mockus, Roy T Fielding, and James D Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3):309–346, 2002.

[21] Audris Mockus and James D Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, pages 503–512. IEEE, 2002.

[22] G Murphy and D Cubranic. Automatic bug triage using text categorization. In *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*. Citeseer, 2004.

[23] Hoda Naguib, Nitesh Narayan, Bernd Brügge, and Dina Helal. Bug report assignee recommendation using activity profiles. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 22–30. IEEE Press, 2013.

[24] David Nettleton. *Commercial data mining: processing, analysis and modeling for predictive analytics projects*. Elsevier, 2014.

[25] Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N Nguyen, David Lo, and Chengnian Sun. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 70–79. ACM, 2012.

[26] Tung Thanh Nguyen, Anh Tuan Nguyen, and Tien N Nguyen. Topic-based, time-aware bug assignment. *ACM SIGSOFT Software Engineering Notes*, 39(1):1–4, 2014.

[27] John O'Donovan, Barry Smyth, Brynjar Gretarsson, Svetlin Bostandjiev, and Tobias Höllerer. Peerchooser: visual interactive recommendation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1085–1088. ACM, 2008.

[28] Kenta Oku, Shinsuke Nakajima, Jun Miyazaki, and Shunsuke Uemura. Context-aware svm for context-dependent information recommendation. In *Proceedings of the 7th international Conference on Mobile Data Management*, page 109. IEEE Computer Society, 2006.

[29] Denis Parra. Beyond lists: studying the effect of different recommendation visualizations. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 333–336. ACM, 2012.

[30] Dewayne E Perry, Nancy A. Staudenmayer, and Lawrence G Votta. *People, organizations, and process improvement*, 1994.

[31] Brett Poulin, Roman Eisner, Duane Szafron, Paul Lu, Russell Greiner, David S Wishart, Alona Fyshe, Brandon Pearcy, Cam MacDonell, and John Anvik. Visual explanation of evidence with additive classifiers. 21(2):1822, 2006.

[32] David Schuler and Thomas Zimmermann. Mining usage expertise from version archives. In *Proceedings of the 2008 international working conference on Mining software repositories*, pages 121–124. ACM, 2008.

[33] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[34] Kalyanasundaram Somasundaram and Gail C Murphy. Automatic categorization of bug reports using latent dirichlet allocation. In *Proceedings of the 5th India software engineering conference*, pages 125–130. ACM, 2012.

[35] Ahmed Tamrawi, Tung Thanh Nguyen, Jafar M Al-Kofahi, and Tien N Nguyen. Fuzzy set and cache-based approach for bug triaging. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 365–375. ACM, 2011.

[36] Cédric Teyton, Marc Palyart, Jean-Rémy Falleri, Floréal Morandat, and Xavier Blanc. Automatic extraction of developer expertise. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, page 8. ACM, 2014.

[37] Nava Tintarev and Judith Masthoff. Effective explanations of recommendations: user-centered design. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 153–156. ACM, 2007.

[38] Katrien Verbert, Denis Parra, Peter Brusilovsky, and Erik Duval. Visualizing recommendations to support exploration, transparency and controllability. In *Proceedings of the 2013 international conference on Intelligent user interfaces*, pages 351–362. ACM, 2013.

[39] Chadd C Williams and Jeffrey K Hollingsworth. Bug driven bug finders. In *Proceedings of the International Workshop on Mining Software Repositories*, pages 70–74. IET, 2004.

[40] Xin Xia, David Lo, Ying Ding, Jafar M Al-Kofahi, Tien N Nguyen, and Xinyu Wang. Improving automated bug triaging with specialized topic model. *IEEE Transactions on Software Engineering*, 43(3):272–297, 2016.

[41] Jialiang Xie, Qimu Zheng, Minghui Zhou, and Audris Mockus. Product assignment recommender. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 556–559. ACM, 2014.

[42] Jin An Xu and Kenji Araki. A svm-based personal recommendation system for tv programs. In *2006 12th International Multi-Media Modelling Conference*, pages 4–pp. IEEE, 2006.

[43] Geunseok Yang, Tao Zhang, and Byungjeong Lee. Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports. In *2014 IEEE 38th Annual Computer Software and Applications Conference*, pages 97–106. IEEE, 2014.

[44] Cheng Zhang, Juyuan Yang, Yi Zhang, Jing Fan, Xin Zhang, Jianjun Zhao, and Peizhao Ou. Automatic parameter recommendation for practical api usage. In *Proceedings of the 34th International Conference on Software Engineering*, pages 826–836. IEEE Press, 2012.

[45] Heng-Ru Zhang and Fan Min. Three-way recommender systems based on random forests. *Knowledge-Based Systems*, 91:275–286, 2016.

[46] Hengru Zhang, Fan Min, and Shenshen Wang. A random forest approach to model-based recommendation. *JOURNAL OF INFORMATION &COMPUTATIONAL SCIENCE*, 11(15):5341–5348, 2014.

Appendix A

Appendix Example

A.1 Back-End Database

We have created two sets of table as we used two products. Figure A.1 and Figure A.2 shows the database schema for or back-end server. Each set contains two tables. One table contains all the word and their TF-IDF score, where other table contains bug report related in formations. The table with bug report information contains information such as, bug report ID, assigned developers name, summary and description of that report, combined text (where summary and description is concatenated together).

A.2 REST

REST (Representation State Transfer) is a set of rules that programmers follow when they create the API for their web application. One of these rules states that users should be able to get a piece of data (called a resource) when they link to a specific URL. Each URL is known as a request while the data sent back to the user is the response. Furthermore, each request consists of four things: the endpoint, the method, the headers, and the data (or body). The endpoint is the URL that a user requests for. The method represents what type of requests the user sends to the server. Headers are used to provide information to both the client and the server. And finally, the data (body) contains information that the user wants to be sent to the server.

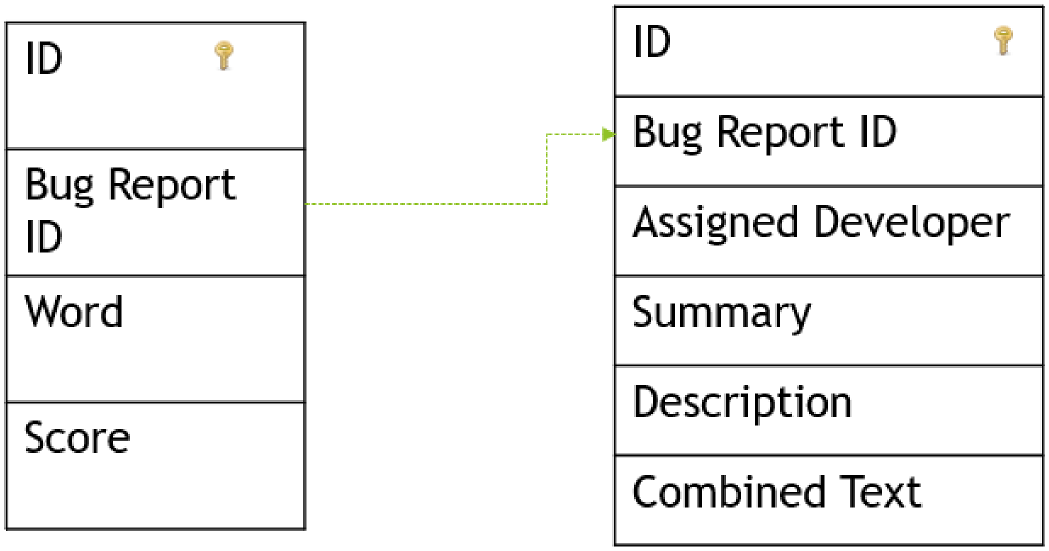


Figure A.1: Database Schema for Thunderbird bug report.

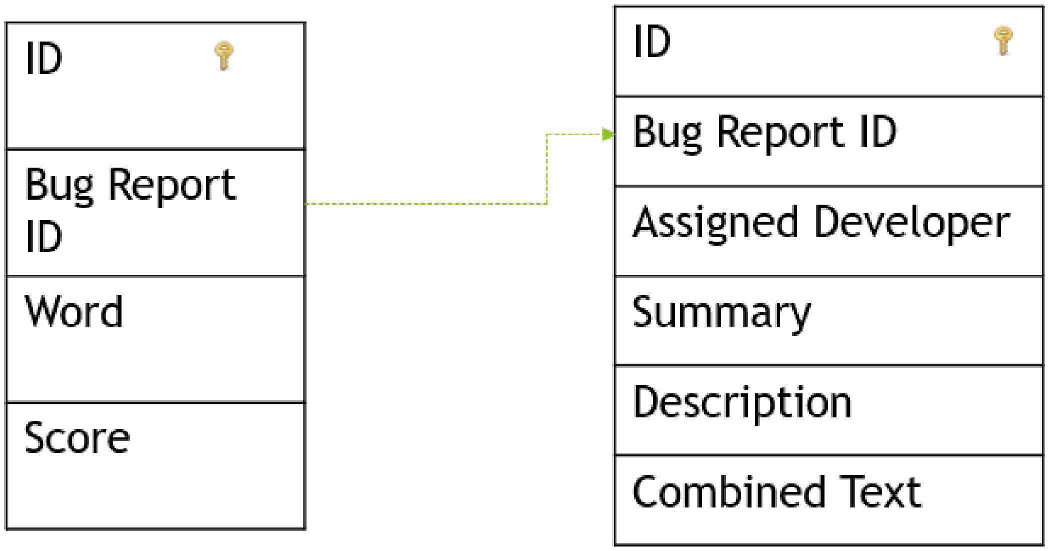


Figure A.2: Database Schema for Bugzilla bug report.