UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Abel Mesfin Cherinet

Recommending Issue Reports to Developers Using Machine Learning

Master's Thesis (30 ECTS)

Supervisor(s): Ezequiel Scott (PhD)

Tartu 2019

Recommending Issue Reports to Developers Using Machine Learning

Abstract:

The development of a software system is often done through an iterative process and different change requests arise when bugs and defects are detected or new features need to be added. These requirements are recorded as issue reports and put in the backlog of the software project for developers to work on. The assignment of these issue reports to developers is done in different ways. One common approach is self-assignment, where the developers themselves pick the issue reports they are interested in and assign themselves. Practising self-assignment in large projects can be challenging for developers because the backlog of large projects become loaded with many issue reports, which makes it hard for developers to filter out the issue reports in line with their interest. To tackle this problem, a machine learning-based recommender system is proposed in this thesis. This recommender system can learn from the history of the issue reports that each developer worked on previously and recommend new issue reports suited to each developer. To implement this recommender system, issue reports were collected from 6 different opensource projects and different machine learning techniques were applied and compared in order to determine the most suitable one. For evaluating the performance of the recommender system, the Precision, Recall, F1-score and Mean Average Precision metrics were used. The results show that, from a backlog of 100 issue reports, by recommending the top 10 issue reports to each developer a recall ranging from 52.9% up to 96% can be achieved, which is 6 up to 9.5 times better than picking 10 issue reports randomly. Comparable improvements were also achieved in the other metrics.

Keywords:

Recommender system, task assignment, bug-triage, machine learning, text classification, Naïve Bayes, Support vector machines, K-nearest neighbor, Information retrieval

CERCS: P170 Computer science, numerical analysis, systems, control

Ülesannete soovitamine tarkvaraarendajatele masinõppe abil

Lühikokkuvõte:

Tarkvarasüsteemide arendust viiakse tihti läbi iteratiivse protsessina ning erinevad tööülesanded tekkivad siis kui leitakse defekte või tekib vajadus uue funktsionaalsuse järele. Need ülesanded salvestatakse probleemihalduse süsteemi, kust arendajad saavad sisendit oma tööle. Ülesannete jaotamine arendajatele võib toimude mitmel eri viisil. Üks populaarsemaid lähenemisi näeb ette, et arendajad valivad ise ülesandeid, mis neid huvitavad. Suurtes projektides võib see aga muutuda keeruliseks: ülesannete suure arvu tõttu on arendajatel raske aegsasti valida omale huvitav tööülesanne. Selle probleemi leevendamiseks esitatakse antud töös masinõppel põhinev soovitussüsteem, mis on võimeline probleemihalduse süsteemi ajaloost õppima milliseid ülesandeid on iga arendaja eelnevalt täitnud ja selle põhjal soovitada neile uusi ülesandeid. Süsteemi arendamiseks koguti 6 erinevast avatud lähtekoodiga projektist ülesandeid, kasutati erinevaid masinõppe meetodeid ja võrreldi tulemusi, et leida sobivaim. Soovitussüsteemi jõudluse hindamiseks kasutati täpsuse (precision), saagise (recall), f1-skoori (f1-score) ja keskmise täpsuse (mean average precision) mõõdikuid. Tulemused näitavad, et 100 tööülesande kirjelduse põhjal 10 igale arendajale sobivaima soovitamise puhul võib saavutada saagise 52.9% ja 96% vahel, mis on 6 kuni 9.5 korda parem 10 juhusliku töökirjelduse valimisest. Sarnased parandused saavutati ka teistes mõõdikutes.

Võtmesõnad:

Soovitussüsteem, probleemihaldus, masinõpe, teksti klassifitseerimine, naiivne Bayes'i algoritm, tugivektormasin

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

List of abbreviations

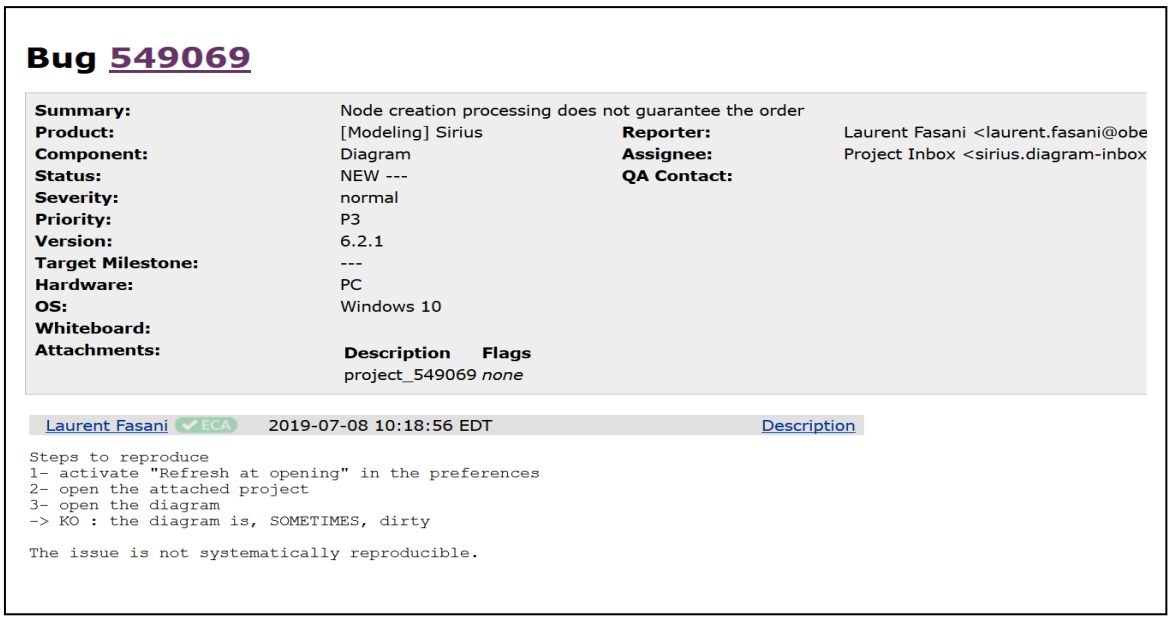
IRS	Issue recommender system	P	precision
NBY	Naïve Bayes	R	recall
KNN	K-nearest neighbour	F1	F1-score
SVM	Support vector machine	MAP	Mean average precision
NTB	NetBeans project	COV	Coverage
ECL	Eclipse project	RAND	Random recommender
FDT	Free Desktop project	TFO	Textual features only
MULE	MuleSoft project	TCF	Both textual and categorical features
MESOS	Apache Mesos project	@N	At recommendation size of N
TIMOB	Titanium SDK/CLI project	ML	Machine Learning
TF-IDF	Term frequency-inverse document frequency		
VSM	Vector space model		
DF	Document frequency		
TF	Term frequency		

1 Introduction

In the course of development of a software system, many change requests, bugs and new requirements arise at different stages and each of these issues needs to be documented using issue reports and made ready for fixing. The recording and management of software issues is mostly done through an issue tracking software. Bugzilla[1] and Jira[2] are some examples of such softwares widely used in open-source projects.

Bugzilla is a web-based general-purpose bug tracker and testing tool originally developed and used by the Mozilla project. However different open source software projects like Eclipse[3] have turned out to use this software to track issues reported on their software products. Jira is a similar issue tracking product developed by Atlassian that allows bug tracking and agile project management. Apache Software Foundation[4], an open-source community of developers, currently manages more than 350 open source projects using this issue tracker.

A basic issue report in these issue trackers has a title and description as textual fields. The description elaborates the issue in full detail while the title summarizes the issue in a short text, mostly a one-line sentence. Besides the title and description, an issue report also contains other categorical fields which are used to label, categorize and prioritize the issue. Some of the common metadata fields are assignee, reporter, issue type, priority, component, version etc. Figure 1 and 2 show an example of an issue report from the Jira and Bugzilla issue trackers of The Eclipse and Apache projects respectively.



Bug 549069

Summary:	Node creation processing does not guarantee the order		
Product:	[Modeling] Sirius	**Reporter:**	Laurent Fasani <laurent.fasani@obe
Component:	Diagram	**Assignee:**	Project Inbox <sirius.diagram-inbox
Status:	NEW ---	**QA Contact:**	
Severity:	normal		
Priority:	P3		
Version:	6.2.1		
Target Milestone:	---		
Hardware:	PC		
OS:	Windows 10		
Whiteboard:			
Attachments:	**Description** **Flags** project_549069 *none*		

Laurent Fasani ECA 2019-07-08 10:18:56 EDT Description

```
Steps to reproduce
1- activate "Refresh at opening" in the preferences
2- open the attached project
3- open the diagram
-> KO : the diagram is, SOMETIMES, dirty

The issue is not systematically reproducible.
```

Figure 1. An example of an issue report from the Bugzila issue tracker

of the Eclipse project

[1] https://www.bugzilla.org/about/
[2] https://www.atlassian.com/software/jira
[3] https://www.eclipse.org/
[4] https://projects.apache.org/

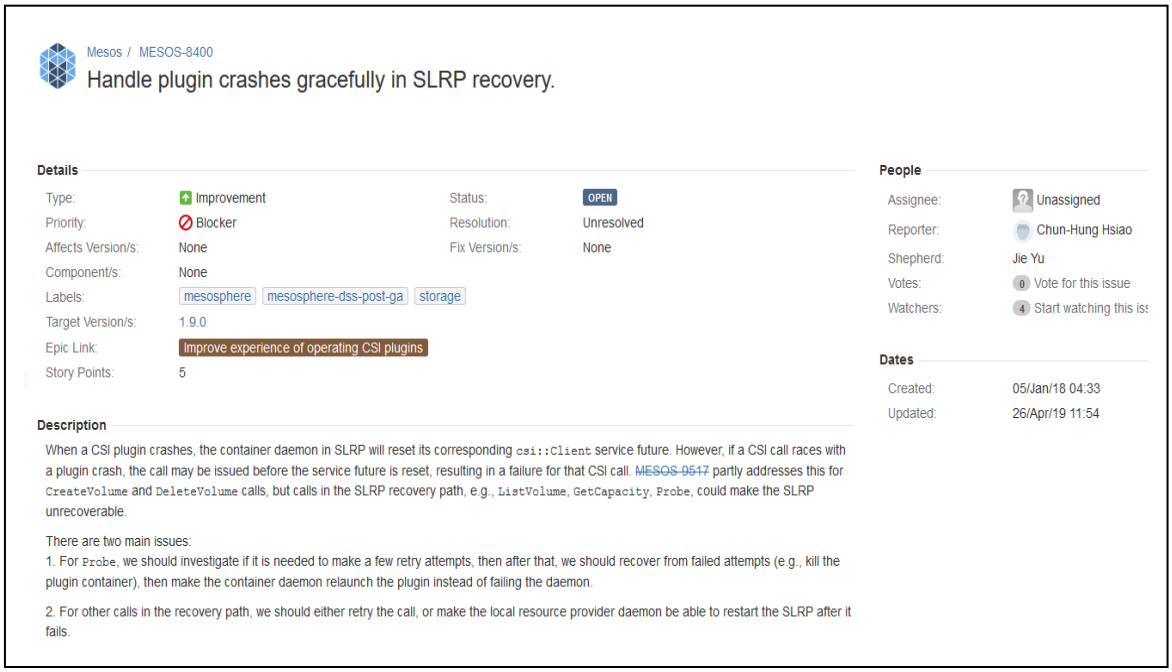


Figure 2. An example of an issue report from the Jira issue tracker of the Apache Mesos project

Once issues are well recorded and prioritized, they need to be assigned to a developer who will be responsible for their fixing. The assignment of issue reports to developers can be done in different ways. The traditional approach is through a separate person, who can be a project manager, team leader, or bug triaging person who will decide which developer should be assigned to which issue report. However, in more agile and self-organizing teams, self-assignment is widely practised [1, 2]. This means developers get to choose tasks and assign themselves. In large projects, where more developers collaborate and lots of issues get reported periodically, assigning issue reports can be difficult and time-consuming as it requires reading each issue report and choosing a suitable developer to assign them to.

Different research works have been done over the years to improve the issue report assignment process. Some of these research works have proposed an automated assignment approach where a developer is assigned to each issue report directly [3, 4, 5] while the others have proposed a developer recommender system that recommends a set of developers to a third person who will assign the issue report to one of those developers [6, 7].

The automatic assignment and developer recommendation approach proposed in previous works are not suited for a type of task assignment where developers are free to choose tasks and assign themselves (i,e self-assignment) because the decision of assigning tasks is made by either a software component or a third person respectively. Self-assignment is being widely practised these days and when practising self-assignment in large projects, developers face the same problem of having to browse through many issue reports to find the issue reports they prefer to assign themselves to.

Therefore, this study tries to improve this self-assignment practice, by implementing a machine learning-based issue recommendation system that can learn from previous assignment history of developers and recommend a shorter list of most relevant new issue reports for developers to choose from.

1.1 Problem Statement

When the number of open issue reports in a backlog is large, self-assignment becomes challenging for developers as they have to read more issue reports to find the next issue report they want to self-assign and this can be an unpleasant and time-consuming task.

Therefore, the aim of this study is to tackle this problem using a machine learning-based issue recommender system (IRS) that can learn from the assignment history of the previously fixed issues and recommend a shorter list of new issues suited to each developer so that developers can easily find the next issues to work on.

To implement such an IRS, three different traditional machine learning algorithms namely, K-Nearest Neighbour, SVM and Naïve Bayes, are compared to select the best one. As a data source, issue reports collected from 6 different open source projects which are based on the Jira and Bugzilla issue trackers are used. By evaluating the implemented IRS using performance metrics like Precision, Recall, Mean Average Precision and F1-score, this thesis work tries to answer the following research question,

1. What is the performance of an IRS using ML algorithms with respect to a random recommender?

2. How much does including features from the categorical fields affect the performance of the IRS.

3. What is the optimal recommendation size that maximizes the F1-score (i.e. giving a good balance between Precision and Recall)?

2 Related Works

There have been a number of research works done over the years which set out to improve the assignment process of issue reports to developers. To find these research works, online digital libraries like Google Scholars, IEEE and Springer have been accessed, as they are reliable sources for academic resources. In these digital libraries, search keywords like "issue/bug report recommendation", "developer recommendation for issue/bug reports", "task-assignment in software projects" and others were used to find some of the related works. Moreover, traversing through the references of the research works found in the first-round search, it was possible to collect more related works. Among the collected works, 7 papers which are most related to the topic of this study were selected for review.

The work by Murphy G et al. [3], as one of the very first works done on the topic, proposed a machine learning-based approach for automatic assignment of bug reports to developers. In their work, they treated the problem as a text categorisation task. For this, they used the description of the issue reports as an information source and represented it in a bag of words representation based on term frequency and used the Naïve Bayes algorithm to train a model on this representation that can classify issue reports among the developers and automatically assigned to them. They applied their approach to issue reports collected from the eclipse project and were able to achieve accuracy up to 30%.

Another similar work was done by Ahsan SN et al. [4] which also used machine learning to classify bug reports for automatic assignment. They tried to implement an automatic bug triage system using latent semantic indexing and support vector machine. Just like the work by Murphy G et al. [3], they relied totally on the description of the issue reports as an information source. However, they are different in two things. First, they used TF-IDF weighing based VSM representation for the issue reports and they also applied dimensionality reduction and latent semantic indexing methods for feature selection. The other difference is they used the SVM algorithm for classification. In this way, they were able to achieve up to 44.4% accuracy on bug reports collected from the Mozilla open-source project.

Nasim S et al. [5] used the frequency of each alphabet instead of terms in the bug short summary as features for 11 different classification algorithms to make the prediction on the developer to be assigned. They used the Eclipse JDT project for their experiments. The bug summaries in eclipse contain tags and a one-sentence description. They experimented their approach using only the tagged issue reports and using all collected issue reports. The best results they achieved was using only the tag information of the tagged issue reports which gave an accuracy of 62% accuracy with the J48 decision tree algorithm. However, not all issue report descriptions contain tags, in fact, from the issue reports they collected, less than half of the issue reports contained tags in their description, which makes it hard to completely rely on tags. Using all collected issue reports the best accuracy they were able to achieve was only 32%.

Not all research works on the issue report assignment problem relied on machine learning algorithms. For example, Tamrawi A et al. [6] proposed a fuzzy sets-based approach. In their approach, for every technical term in bug reports, they kept a record of a fuzzy set of the developer's relation to the term based on the issue reports fixed by the developers previously. For a new issue report, they ranked the developers based of their membership score to the fuzzy set of the new issue report, calculated based on the fuzzy set theory [8], and assigned the issue report to the developer with the highest membership score. Using this approach, they were able to achieve a top 1 accuracy of 37.81% and top 5 accuracies of on average with issue reports collected from the Eclipse project.

Another example is the work by Hu H et al. [7]. Their approach makes use of a Developer-Component-Bug (DCB) network structure to make developer recommendations. This network captures the relationship among developers, source code components and bug report and assigns a weight to each connection in the network. They made use of the VSM model to represent bugs and the keywords for this model are extracted from the summary of the bug report and the source code repository log of the commit corresponding to the bug fix. They made use of cosine similarity to calculate the relevance of each previous bug to the new bug. This relevance is then propagated through the DCB network to calculate the relevance of each developer to the new bug which in turn is used to recommend the top n developers. They Evaluated their approach on Eclipse, Mozilla, NetBeans and 2 other industrial projects. Their best result achieved 42.36% top1 accuracy and 73.85% top 5 accuracy on the Eclipse project.

In an effort to improve the representation of the textual description of issue reports, recent work by Mani S et al. [9] has applied a more advanced representation using deep learning. They used a deep bidirectional recurrent neural network to learn the semantics of the textual description of the issue reports in an unsupervised manner. Applying this type of representation on issue reports from the Chromium, Mozilla Core and Mozilla Firefox and by using Naïve Bayes, SVM, SoftMax and cosine distance-based classifiers for predicting developers, they reported improvements in top 10 accuracies with respect to the TF-IDF based bag of words representation, however their best results are still low for top 10 accuracy, which is 47%.

Rocha H et al. [10] proposed a tool called NextBug [1] which recommends similar bugs to each bug reports browsed by developers so that developers can find the next bug, they want to fix after they worked on a bug report. The tool has an IR and recommender component. The IR component uses the summary and description of issue reports computing a VSM representation using TF-IDF weighting scheme. The recommender component computes the similarity of bug reports by using a cosine similarity function and applying it on the VSM representations. This way, for every issue report visited by developers they were able to show the top similar issue reports as part of the visited issue report. Evaluating the tool on bug reports collected from the Mozilla Project they were able to achieve a precision of 31% approximately by recommending 1 up to 5 similar issue reports for each visited issue report.

The NextBug tool can be useful for developers when practising self-assignment however the recommended issue reports are specific to an issue report instead of a developer and developers have to visit an issue report they fixed before to find recommendations. This makes it different from what this thesis proposes because it is trying to recommend issue reports specific to developers.

In Table 1 the approaches used and results reported in the previous works reviewed is summarised. In general, it can be seen that there are limited research works that target to improve the self-assignment practice and the IRS we are implementing is expected to fill this gap. We also noticed that most focus has been given to the description and summary of issue reports as an information source, however, issue reports contain other categorical metadata information (e.g. components, issue report type, priority, reporter, ..., etc) that can be useful to analyse. Therefore, in this study, we tried to combine features from these categorical fields with features from the description and title of issue reports to be used to train the Machine learning algorithms and finally we analyse the effect of this on the performance of the IRS.

Table 1. Summary of related works

Paper	Tries to Improve task assignment by	Used Information source	Methods used	Best results
Murphy G et al. [3]	Automatic assignment	Summary and description	• BOW representation with TF • Naive Bayes classifier	Accuracy: 30%
Ahsan SN et al. [4]	Automatic assignment	Summary and description	• VSM representation with TF-IDF, • Latent semantic indexing • SVM classifier	Accuracy: 44%
Nasim S et al. [5]	Automatic assignment	Summary and description	• Representation with Frequency of alphabets • 8 different classification algorisms, Best result with J48 decision tree classifier	Accuracy: 32% Accuracy: 62%(using only tagged issue reports)
Hu H et al. [8]	Developers recommendation to third person assigner	Summary and description, commit logs, Source code components	• VSM representation with TF-IDF, • Cosine similarity • Developer-Component-bug network structure • ranking by score calculated from the network	Top 1 accuracy: 42.36% Top 5 accuracy: 73.85%
Mani S et al. [9]	Developers recommendation to third person assigner	Summary and description	• Representation using deep learning • SVM, SoftMax and Naïve Bayes classifiers	Top 10 accuracy: 47%
Rocha H et al. [10]	Similar Issue report recommendation	Summary and description	• VSM representation with TF-IDF • Ranking by Cosine similarity	Top 5 precision: 31%

3 Methodology

This section presents the procedures followed to answer the research questions of this study.

First, an IRS was built using the approach explained in section 3.1. The approach involves pre-processing and feature selection steps which are further elaborated in the sections that follow.

The IRS was evaluated using different machine learning algorithms on the collected issue reports by sampling multiple backlogs of size 100 issue reports at different places within the chronological order of the issue reports. The IRS was run to recommend issue reports to the developers from each backlog and the performance for each recommendation was evaluated using the precision, recall f1-score and mean average precision metrics. This experimental setup and the evaluation metrics used are explained in section 3.6.

Finally, to answer each research question, the following procedures were followed.

RQ1: What is the performance of an IRS using ML algorithms with respect to a random recommender?

To answer RQ 1, The IRS was evaluated using both textual and categorical features for each ML algorithm separately with top 10 recommendation sizes, and the best results for each metrics were compared against a random recommender to know how much more easily developers can find relevant issue reports using the ML-based IRS compared to selecting issue reports randomly.

RQ 2: How much did including features from the categorical fields affect the performance of the IRS.

To answer RQ 2, the IRS was evaluated for top 10 recommendation as in RQ 1, but using textual features only and the best results were compared with what was found using both categorical and textual fields (see RQ 1). In addition to that, a chi-squared statistical test is used to know which features are more associated with the class labels (i.e. the developers |assigned to the issue reports) and which fields of the issue reports contributed these features.

RQ3: What is the optimal recommendation size that maximizes the f1-score (i.e. giving a good balance between precision and recall)?

To find the size of recommendation with a better balance between precision and recall, the f1-score was used for comparison because it measures both the precision and recall in harmony. By varying the recommendation size from 1 to 25, we tried to find the recommendation size which achieves a maximum f1-score in each project using the three algorithms separately and by using both textual and categorical features.

3.1 Approach

The problem addressed by this thesis can be formulated as follows. Among a list of open issue reports R, how can we recommend the top N issue reports suitable for each developer, in a list of available developers D. Tackling this problem requires one to be able to rank issue reports for each developer, based on their probability of being assigned to that developer.

To calculate the probability of assignment of an issue report to a developer, the problem was first treated as a multi-class classification problem which means the issue reports were considered as items to be classified, the developers as the classes or labels to be assigned to these items and the goal was to classify these items among the classes with a probability

estimate for all classes. These probability estimates could then be used as a measure of the probability of assignment of issue reports to developers.

This problem can be referred to as a supervised multi-class text classification problem. It is a text classification because issue reports are basically tagged textual documents. It is supervised learning because previous issue reports, whose assigned developers are known, are available to learn from. The fact that the number of developers used as classes can be more than two also makes it a multi-class classification problem. Moreover, since we are doing the classification to rank issue reports and extract the most relevant ones for each developer, the problem can also be referred to as an information retrieval problem.

Among the different supervised machine learning algorithms out there that can be used for the classification task, 3 traditional algorithms were selected for experiment namely K-nearest Neighbour (KNN), Naïve Bayes (NBY) and Support Vector Machine (SVM). These algorithms were selected for comparison because they are easy to implement are commonly used algorithms for text classification problems in general [11, 12].

Before applying these ML algorithms, the issue reports pass through a set of pre-processing steps to extract terms from the textual and categorical fields that can be used to represent the issue reports. The pre-processing steps applied to the textual fields like the summary and description involve, converting to lower case, removal of numbers and punctuations, stop word removal, stemming, document indexing, dimensionality reduction.

After the pre-processing steps, the terms from the textual fields were used to build a VSM representation using their TF-IDF weights which is a common way of representing text in text mining [13]. Similarly, the terms from the categorical fields were converted into a numeric representation using one-hot encoding [14] and combined with the VSM representation of the textual fields.

After issue reports were represented numerically using terms extracted from the textual and categorical fields, the most relevant terms that can be used as features were selected using chi-squared which is a statistical feature selection method. Chi-squared was used as a feature selection method because it has been found to be a reliable feature selection method for text classification [15] and has achieved good performance compared to other methods in bug triaging [16].

After the features are selected, the three machine learning algorithms were separately used to implement a classifier on these features. The classification was done to estimates the probability of classification for all developers as a measure of the probability of assignment. Using these estimates, the issue reports are ranked for each developer and the top N issue reports with the highest probability of assignment are recommended to each developer.

Figure 3 shows an example of the general flow of the proposed approach. In this example, there are three developers and 10 unassigned issue reports which need to be assigned to these developers. After all issue reports pass through the pre-processing and feature selection steps, the machine learning section performs probabilistic classification on the unassigned issue reports using the previously assigned issue reports for training. The result of the classification is a matrix of the probability of assignment that has the issue reports in the row and the developers in the column and each cell RiDj represents the probability of the issue report Ri to be assigned to the developer Dj.

Lastly, the recommender section takes this to rank the issue reports for each developer and recommend the top 3 issue report in this case. For example, For the first developer D1, the issue reports are sorted as R5, R9, R7, R10, … etc, in decreasing order of their probability

of being assigned to D1. Therefore, the top 3 issue reports, R5, R9, and R7, will be recommended to developer D1.

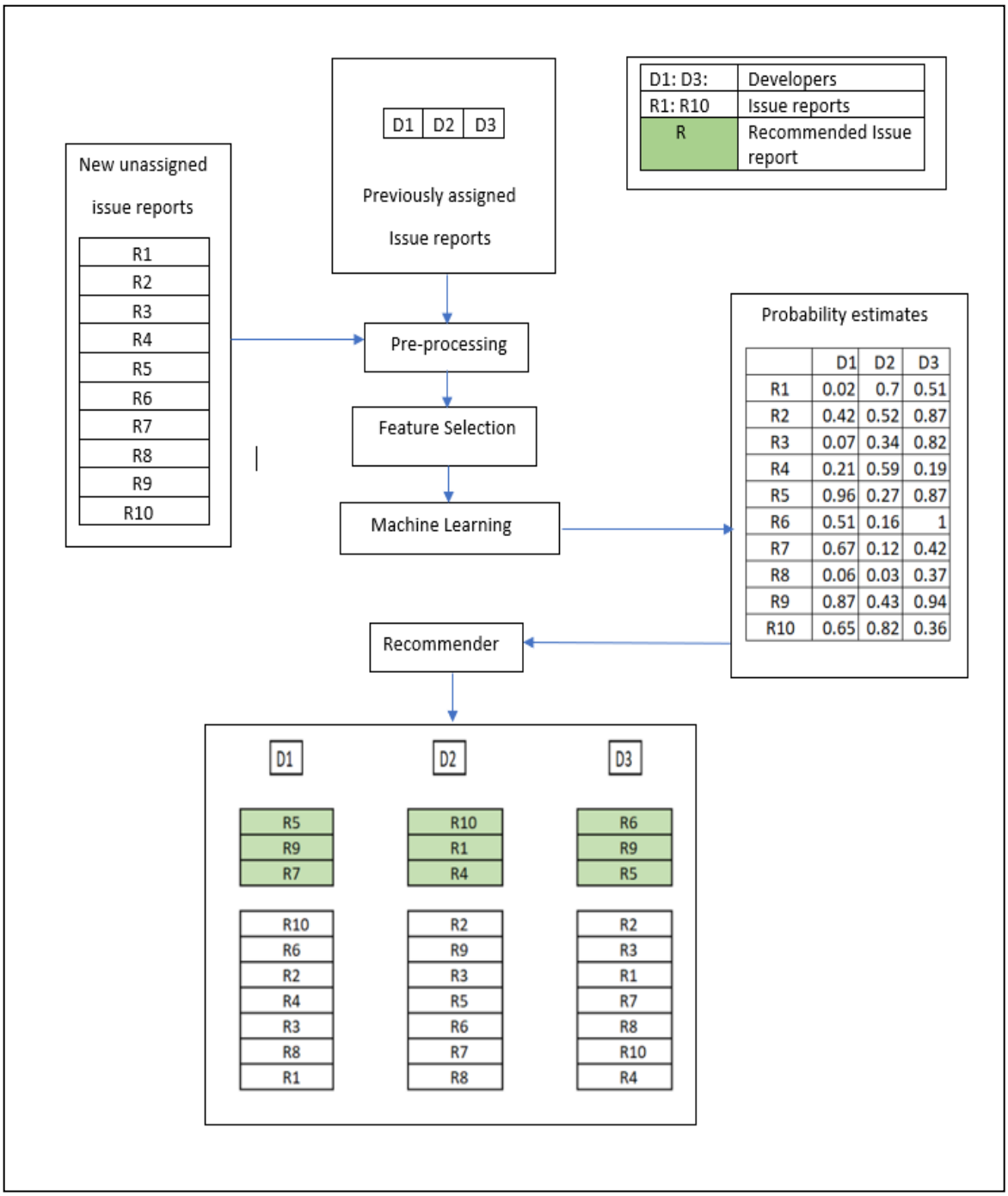


Figure 3. An Example of the general flow of the IRS,

3.2 Representation of issue reports

Two different representation techniques were applied to the issue reports. VSM was used to represent textual features such as description, title whereas One Hot Encoding was applied for categorical features.

VSM

The Vector space model (VSM) also referred to as a term vector model is an algebraic model for representing text documents numerically as a vector of terms and their weights. i.e. given a vocabulary of terms T, and set of issue reports R, the term vector for an issue report Rj is given by

$$Rj = (W1j,\ W2j,\ W3j, \dots, Wij) \quad \text{(Equation 1)}$$

Where Wij is the weight of a term Ti in the issue report Rj

The commonly used method to give weights to terms in a VSM representation is Term Frequency-Inverse Document Frequency (TF-IDF)[5] weighing method. In the TF-IDF weighing method, the weight of a term is proportional to the frequency of the term in a document but is offset by the number of documents that contain the word, which helps to adjust for the fact that some words appear more frequently in general. The TF-IDF is defined as the product of term frequency (TF) and inverse document frequency (IDF). For a term t and issue report r TF, IDF and TF-IDF are calculated with Equation 2, 3, and 4 respectively.

$$TF(t,r) = \frac{the\ frequency\ of\ t\ in\ r}{total\ number\ of\ terms\ in\ r} \quad \text{(Equation 1)}$$

$$IDF\ (t) = \log\left(\frac{total\ number\ of\ issue\ reports}{number\ of\ issue\ reports\ containing\ t}\right) \quad \text{(Equation 2)}$$

$$TF - IDF\ (t,r) = TF(t,r)\ \times\ IDF\ (t) \quad \text{(Equation 3)}$$

One Hot Encoding

One hot encoding is a widely used numeric representation of categorical features for use by machine learning algorithms. With the one-hot encoding method, a categorical feature is expanded into multiple dummy variables using its unique values. Then, for each example, one or more of the dummy variables will be set to 1 or 0 depending on whether the example has the value associated with the dummy variables or not. Figure 4 shows an example illustration of the one-hot encoding method when applied to a components field which has unique values UI, SDK, HTTP and LIB,

[5] http://www.tfidf.com/

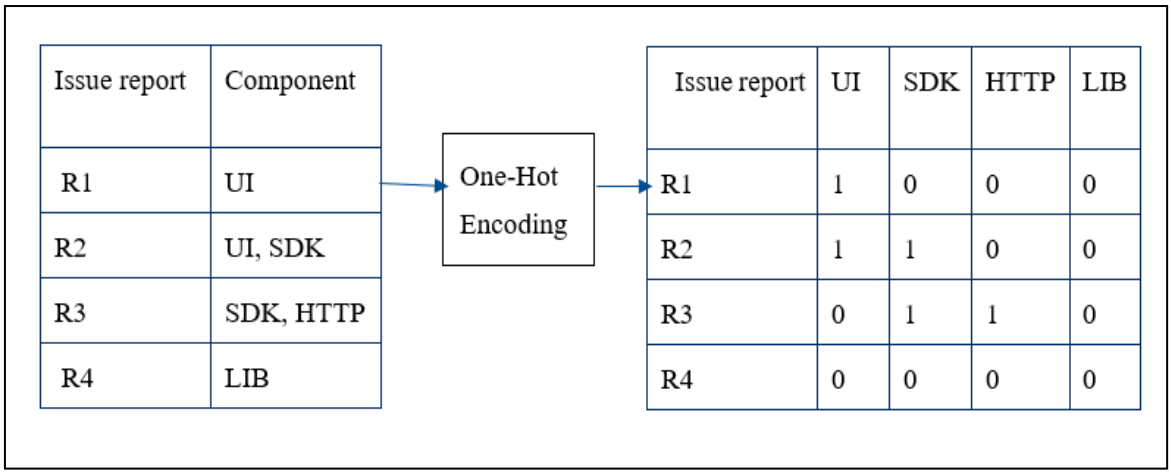


Figure 4. An Example of the One-Hot Encoding representation

3.3 Machine Learning Techniques

Three different algorithms were used separately to implement the IRS. These algorithms are discussed below.

Naïve Bayes (NBY)

Naïve Bayes is a simple learning algorithm that is based on the Bayes rule and a strong assumption that attributes are conditionally independent [17, 18]. Using the Bayes rule, for a document x with n features and class y, the probability of x being labelled as class y is given by,

$$P(y|x) = P(y)P(x|y)/P(x) \quad \text{(Equation 4)}$$

With the assumption that attributes are conditionally independent, $P(x|y)$ is calculated as

$$P(x|y) = \prod_{i=1}^{n} P(xi \mid y) \quad \text{(Equation 5)}$$

where xi is the value of the ith feature of x and P(x) is given by

$$P(x) = \prod_{i=1}^{k} P(ci)P(x|\, ci) \quad \text{(Equation 6)}$$

where k is the number of classes and ci is the ith class.

There are two variants of this algorithm widely used in document classification, namely the multivariate Bernoulli model and Multinomial model, based on how they calculate the probability of a document given a class. For this study, the Multinomial model was used because, unlike the Bernoulli model, this model considers the frequency of words in the document when calculating the probabilities which make it more suited for the VSM representation and it has also been found to generally have better performance [17, 18]. To train the Multinominal Naïve Bayes model the "naivebayes"[6] package in R was used.

K-Nearest Neighbour (KNN)

KNN is one of the simplest lazy machine learning algorithms used for classification which doesn't require any training phase. When applied in text classification [19], KNN determines a class for a new document using the class composition of the top k documents most similar to the new document. As an estimate of the probability of classes, the weighted proportion of those classes in the k nearest neighbours was used, which is given by,

$$P(yi|xi) = \frac{\sum_{xj \in KNN(xi)} Sim(xi, xj)\delta(yi, yj)}{\sum_{xj \in KNN(xi)} Sim(xi, xj)}$$
$$,\delta(yi, yj) = \begin{cases} 1 & yi = yj \\ 0 & yi \neq yj \end{cases} \quad \text{(Equation 7)}$$

where xj is a document of class yi in the k most similar documents to document xi, i.e $KNN(xi)$, of class yi. $\delta(yi, cj)$ is there to select only documents of class yi in $KNN(xi)$ and $Sim(xi, xj)$ represents the similarity of two documents xj and xi.

If a class doesn't exist in the k nearest neighbours, its probability estimate is calculated to be 0 with Equation 10. To make sure that enough classes have a non-zero probability estimate, k has to be made as big as possible, while also making sure that the recommendation recall is not significantly affected. Experimentally, the value of k was varied from 1 to 50 to select a general optimum value. This experiment is presented in section 4.1.1

To calculate the similarity of two-issue reports, the cosine similarity measure [16], a common similarity function for VSM, was used. The cosine similarity of two documents $A = (A1, A2, \ldots\ldots, An)$ and $B = (B1, B2, \ldots\ldots, Bn)$ is given by

$$Sim(A, B) = \frac{\sum_{i=1}^{n} AiBi}{\sqrt{\sum_{i=1}^{n} Ai^2}\sqrt{\sum_{i=1}^{n} Bi^2}} \quad \text{(Equation 8)}$$

To implement the KNN algorithm, the "text2vec"[7] R package, the package also used to build the VSM model, was used to calculate the cosine similarities and custom code was written to find the nearest neighbours and calculate the class probabilities.

[6] https://cran.r-project.org/web/packages/naivebayes/vignettes/intro_naivebayes.pdf

[7] http://text2vec.org/

Support Vector Machine(SVM)

SVM[20] is another supervised machine learning algorism widely used for classification and regression problems. SVM is basically a binary classifier which tries to find a hyper-plane in a vector space which best separates two classes of a dataset.

Such a hyperplane can be represented as a set of points x satisfying $\langle w, \phi(x)\rangle - b = 0$ where $\langle\rangle$ is the inner product, w represents a vector normal to the hyperplane, b represents the distance of the plane from the origin and ϕ is a kernel function. The goal of SVM is to find the value of w and b which maximize the margin γ given by,

$$\gamma = \min_{1<i<l} \|\langle w, \phi(x))) - b\| \qquad \text{(Equation 9)}$$

where l is the number of data points.

Fig 4(c) illustrates an example of a maximum margin hyperplane in a 2-dimensional vector space. For a new data point, SVM decides its class based on which side of the hyperplane the new data point lies.

The purpose of the kernel function in SVM is to transform the feature space into a form separable by a linear hyperplane as shown in Figure 5(a, b). There are different types of kernel functions used with SVM. Some examples are linear, radial, polynomial and sigmoid kernels. Among these kernels, The linear kernel is simple and widely used in practice [8].

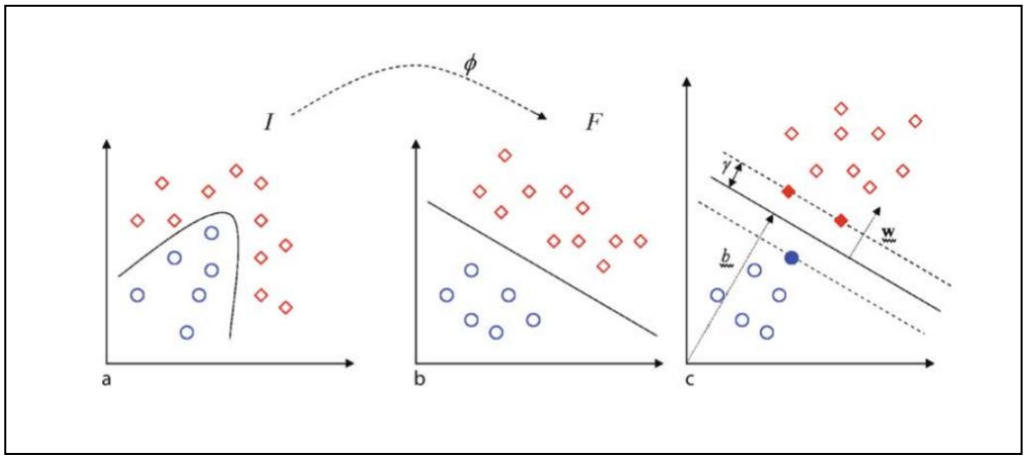


Figure 5. Support Vector Machines, Figure 1

(a) datapoints in a vector space. (b) datapoints in a vector space after kernel function is applied. (c) margin of the hyperplane.

To implement the linear SVM in R the "e1071"[8] library was used. This library is based on LIBSVM, a popular SVM library written in C++. LIBSVM uses the "one-against-one" approach to apply SVM to the multiclass classification problem. the "one-against-one" basically applies SVM to classify between all possible pairs of classes and takes the class with the maximum vote. The LIBSVM library also makes probability estimation for SVM using

[8] https://cran.r-project.org/web/packages/e1071/e1071.pdf

a pairwise coupling approach proposed in [21]. More details about the library can also be found in its practical guide [22].

3.4 Dataset

The issue reports for this study were collected from the Jira and Bugzilla issue tracking systems. Three projects were chosen for each issue tracking software making the total number of projects studied six. For the Bugzilla issue tracking system, the Eclipse (ECL)[9], NetBeans (NTB)[10] and Free Desktop (FDT)[11] projects and for Jira issue tracking system, the MuleSoft (MULE)[12], Apache Mesos (MESOS)[13] and Titanium SDK/CLI (TIMOB)[14] projects were studied. These projects were chosen because their issue repositories are freely accessible, they have enough issue reports to be analysed and have been widely researched in similar studies [20, 21, 22, 23].

Tabel 2. Collection of issue reports

Project	Status	Resolution status	Creation date	# of issue reports	# of assignees
NB	RESOLVED, VERIFIED, CLOSED	FIXED	01/01/2012 – 31/12/2012	8392	114
EC	RESOLVED, VERIFIED, CLOSED	FIXED	01/01/2011 – 31/12/2013	8310	244
FD	RESOLVED, VERIFIED, CLOSED	FIXED	01/01/2011 – 31/12/2012	9928	432
MULE	RESOLVED, DONE, CLOSED	FIXED, COMPLETED, DONE	01/01/2016 – 31/12/2017	3447	53
MESOS	RESOLVED	RESOLVED, DONE, FIXED	01/01/2014 – 31/12/2017	4055	185
TRIMOB	RESOLVED, CLOSED	FIXED, DONE	01/01/2011 – 31/12/2013	7745	114

[9] https://bugs.eclipse.org/bugs/
[10] https://netbeans.org/bugzilla/
[11] https://bugs.freedesktop.org/
[12] https://www.mulesoft.org/jira/projects/MULE/issues
[13] https://issues.apache.org/jira/projects/MESOS/issues
[14] https://jira.appcelerator.org/projects/TIMOB/issues

The issue repositories of these projects provide the functionalities to search, filter and export issue reports. Using these functionalities, it was possible to query and extract issue reports that are successfully resolved within a limited range of creation dates in the form of CSV files. Projects like MESOS have a limit of 1000 issue reports that can be downloaded at a time, so the issue reports were exported with multiple downloads by dividing the creation date range. To extract issue reports that are successfully resolved, certain values were set for the status and resolution status fields of the issue reports when executing the queries as shown in Table 2. Table 2 also summarizes how many issue reports were collected for each project.

Each issue report has a title, description and other metadata fields which characterize the issues. These fields are filled at different stages of the lifecycle of the issue report and some of them are prone to change. For the purpose of recommending new issue reports, one needs to use the attributes which are filled when a new issue is reported. By looking at the new issue report creation form for each project and also through a small survey on new unassigned issue reports of each project, it was possible to select a set of fields that can be analysed to recommend new issue reports which are summarised in Table 3.

Each project might also include custom fields in their issue tracker system, however, to be more standard across projects, only default fields provided by the issue tracker software are included. The selection of fields was done by limiting the columns to be included in the search queries when collecting the issue reports. For the Bugzilla based projects, there were no possibilities to export the description of the issue reports in the CSV format so the descriptions were extracted from the XML format and added into the CSV file by matching the ID of the issue reports.

Tabel 3: Field selection of issue reports

	Summary	Description	Reporter	Component	Issue Type	Version	OS	Hardware	Product	Priority	Severity
NB	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	X
EC	✓	✓	✓	✓	X	✓	✓	✓	✓	✓	✓
FD	✓	✓	✓	✓	X	✓	✓	✓	✓	✓	✓
MULE	✓	✓	✓	✓	✓	✓	X	X	X	✓	X
MESOS	✓	✓	✓	✓	✓	✓	X	X	X	✓	X
TIMOB	✓	✓	✓	✓	✓	✓	X	X	X	✓	X

3.5 Data cleaning

After selecting the fields to be analysed and collecting the issue reports accordingly, further cleaning steps were applied to select the issue reports relevant for analysis. Two cleaning steps were taken in this case,

- Removing issue reports with no assignee or whose assignee field does not refer to a developer.

- Removing issue reports fixed by inactive developers.

For projects based on Jira, some of the issue reports had empty values, so these reports were removed. For the issue reports collected from projects based on Bugzilla, the values for the Assignee field were all available. However, projects like EC have assignee field values which do not refer to a specific developer rather referring to a name of an inbox where issue reports are added to. Some examples of such assignee values are "Platform-UI-Inbox@eclipse.org", "jdt-doc-inbox@eclipse.org'', etc. Issue reports with such value of assignee were also removed.

In the second step, issue reports which are assigned to developers who have only been assigned to a few issue reports in the whole dataset were removed. We consider these developers to be inactive in the development process. To remove inactive developers, developers were arranged in increasing order of the number of issues they were assigned to and the cumulative sum of the number of issues was used to filter out developers whose cumulative sum of the number of issues assigned composes less than 10% of the total number of issue reports.

In the above two steps. 795 issue reports from NTB, 1833 issue reports from ECL, 4545 issue reports from FDT, 348 issue reports from MULE, 586 issue reports from MESOS and 729 issue reports from TIMOB were removed. The total number of issues and developers for each project after the two cleaning steps is summarized in Table 4.

Tabel 4: Number of issues and developers after data cleaning

project	# of issues	# of developers
NTB	7597	38
ECL	6477	50
FDT	5474	101
MULE	3099	18
MESOS	3469	56
TIMOB	7016	28

3.6 Pre-processing

The first pre-processing was applied to the textual fields which are the title and description of the issue reports. Before converting these textual fields into a VSM representation, a set of pre-processing measures were taken to remove less interesting words and characters and keep only the relevant ones. The basic pre-processing measures [23] taken in text mining were also applied in this study which will be presented as follows,

1. *Converting to Lowercase*: All alphabetic characters were converted to lowercase because we want a word to appear exactly the same every time it appears and are less interested in the capitalization of the words.

2. *Removal of Numbers and Punctuations*: Numbers and punctuations were removed because they will appear to be words to the machine and they create noises.
3. *Stop Words Removal*: Stop words are words that appear very frequently in all texts because of their nature (a, and, also, the, etc.). For this step, we used the 571 English stop words from the SMART information retrieval system[15].
4. *Stemming*: Stemming is the process of reducing words to root by removing inflexion through dropping unnecessary characters, usually a suffix. With this step, it is possible to remove common word endings (e.g., “ing”, “es”, “s”).
5. *Document Indexing*: This step involves extracting a set of unique terms in the whole set of issue reports and calculating their term frequency (TF) and document frequency (DF). TF represents the total frequency of a term in the issue reports and DF represents the number of issue reports containing a term.
6. *Dimensionality reduction*: This step involves reducing the size of the vocabulary used to represent documents by removing rare and too common words. For this study, using the DF values calculated in step 5, terms which do not appear in at least 3 issue reports or terms which appear in more than half of the issue reports were removed. This step assists the chi-square test based feature selection that will be applied later.
7. *After the above 6 steps*, a document term matrix (DTM) was created with the issue reports in the row, the extracted vocabulary of terms in the column and each cell containing the TF-IDF weights where each row of the DTM represents the VSM of the title and description of issue reports. To implement the above pre-processing steps the "tm"[16] and "text2vec"[17] R packages were used in combination. "tm" was used for the first 4 steps while text2vec was used for the last 2 steps and for the creation of the DTM.

The second pre-processing was applied to the categorical fields which involve the component, version, product, reporter priority etc. A matrix of similar structure as the DTM was constructed for these fields using the one-hot encoding approach explained in Figure 4.

Tabel 5: Number of terms after preprocessing

project	# terms from textual features	# terms from categorical features	total
NTB	5320	1597	6917
ECL	5077	1132	6209
FDT	6510	2546	9056
MULE	2609	355	2964
MESOS	3357	544	3901
TIMOB	4689	1851	6540

[15] https://www.lextek.com/manuals/onix/stopwords2.html
[16] https://cran.r-project.org/web/packages/tm/vignettes/tm.pdf
[17] http://text2vec.org/

Which means unique terms were extracted from the categorical fields and put in a column then for each issue report, the values for these columns were set to 1 if the issue report is labelled with the respective term and 0 if not. This matrix was then combined with the DTM to give the full representation of the issue reports. The total number of terms extracted in the pre-processing step for each project is summarised in Table 5.

3.7 Feature selection

After issue reports are represented using the terms extracted from the textual and categorical fields, the chi-squared test was used as a feature selection measure to remove non-informative terms according to dataset statistics.

The chi-squared test is a statistical test that measures the lack of independence between the terms and the classes, in a text classification problem. For a set of classes, $c = \{c1, c2, \ldots, ck\}$ The chi-squared statistics of a term t, also represented as x^2, can be calculated as,

$$x^2(t,c) = \sum_{t\epsilon\{0,1\}} \sum_{c} \frac{[\mathrm{n}_{tc} - n_t n_c/N]^2}{n_t n_c/N}$$

Where n represents the observation count for each state of t and c and N represents the total number of samples. From the chi-squared statistics, the Cramér's V coefficient was calculated to measure the association of each term with the classes. The Cramér's V coefficient is given by,

$$V = \sqrt{\frac{x^2/N}{k-1}}$$

When the value of x^2 for a term t is 0, it means that the classes are independent of or not associated with the term t therefor such terms were removed from the feature set. Using this measure, it was possible to decrease the number of terms used as features which are

Tabel 6: Number of terms after feature selection

project	# of terms from textual features	# of terms from categorical features	total
NTB	173	163	336
EC L	194	80	274
FDT	484	143	627
MULE	25	52	77
MESOS	24	59	83
TIMOB	118	96	214

summarized in Table 6. The chi-squared test and the calculation of the Cramer coefficient were performed using the "Fselector"[18] package in R.

3.8 Evaluation

To evaluate the IRS, a set of training and test sets were extracted. The test sets are used to represent a backlog of unassigned issue reports and the training sets are used to represent the fixed issue reports. To extract such training and test sets, the issue reports were first arranged by their creation date to keep their chronological order. After that, 10 different partitions of the issue reports were extracted by taking the first {10%, 20%, 30%, ..., 100%} of the issue reports as summarized in Table 7. For each partition, the last 100 issue reports were taken as a test set to represents a large backlog of open issue reports and the rest as a training set to represent previously fixed issue reports. The IRS is run on each partition and the final values for the evaluation metrics were taken as the average of the result from each partition. This was done for cross-validation of the result across the creation date range of the issue reports in each project.

Tabel 7. Dataset partitions for cross validation

Partition	NB	EC	FD	MULE	MESOS	TIMOB
First 10%	759	647	547	309	346	701
First 20%	1519	1295	1094	619	693	1403
First 30%	2279	1943	1642	929	1040	2104
First 40%	3038	2590	2189	1239	1387	2806
First 50%	3798	3238	2737	1549	1734	3508
First 60%	4558	3886	3284	1859	2081	4209
First 70%	5317	4533	3831	2169	2428	4911
First 80%	6077	5181	4379	2479	2775	5612
First 90%	6837	5829	4926	2789	3122	6314
100%	7597	6477	5474	3099	3469	7016

The IRS is then evaluated with these train and test sets using precision, recall, F1-score and mean average precision (MAP). These metrics were selected considering the fact that the IRS performs a classification task [24, 25].

For each developer, the precision measures how much of the issue reports recommended to the developer are relevant and the recall measures how much of the relevant issue reports have been recommended to the developer. Because the IRS produces a ranked list of issue

[18] https://cran.r-project.org/web/packages/FSelector/FSelector.pdf

reports, The MAP was used to evaluate the ordered precision of the recommendations given by the IRS. The F1 score was used to aggregate the precision and recall metrics.

If, among all the issue reports, m issue reports in total are relevant to a developer and if the developer is recommended N issue reports, the precision(P), recall(R), F1-score(F1) and average precision (AP) are calculated as,

$$\text{R@N} = \frac{\#\ of\ relevant\ issue\ reports\ @\ N}{\text{m}} \quad \text{(Equation 10)}$$

$$\text{P@N} = \frac{\#\ of\ relevant\ issue\ reports\ @\ N}{\text{N}} \quad \text{(Equation 11)}$$

$$F_1(N) = 2\frac{P(N)R(N)}{P(N)+R(N)} \quad \text{(Equation 13)}$$

$$AP@N = \frac{1}{\#\ of\ relevant\ issue\ reports\ @\ N}\sum_{k=1}^{N} P(k)rel(k) \quad \text{(Equation 14)}$$

The final values for the above metrics were calculated using the average of the result for each developer. The mean of the AP of each developer is what we call MAP.

4 Results

This section presents the results according to each research question.

4.1 RQ 1

The purpose of this section is to compare how the ML-based IRS compares to a random recommender.

4.1.1 Choosing a value of k for KNN using MAP

The choice of k for KNN is an important decision to make because it affects the performance of the classifier. To select a general value of k for KNN that can be used across the projects, the MAP was compared for a value of k ranging from 1 to 50 as shown in Figure 6. The MAP was measured for the complete ranking of issue reports (i.e. MAP@100) and both categorical and textual features were used. Taking the average of MAP from each project, it is found that a value k between 20 and 40 is a good choice for K. Therefore, for the rest of the results a value of k = 25 was used.

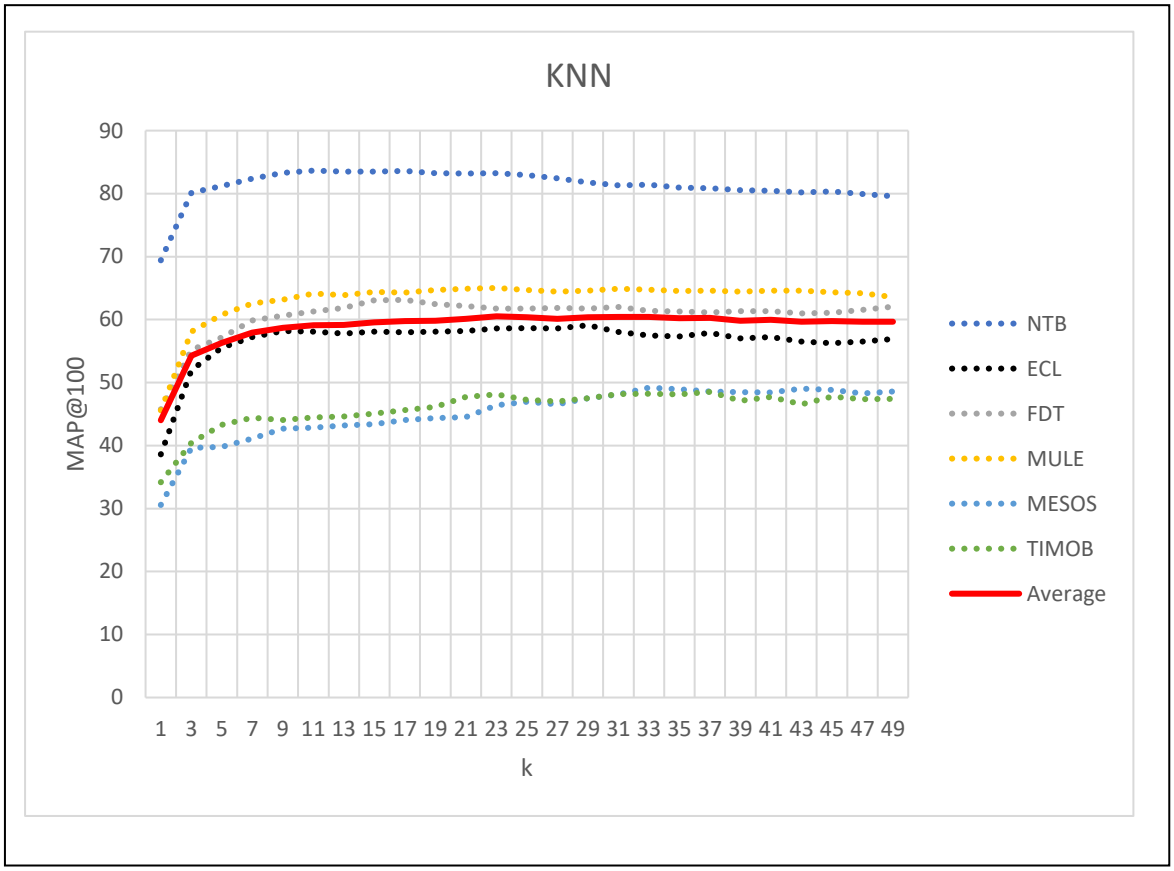


Figure 6: comparision of MAP@100 using KNN for k ranging from 1 to 50

For SVM, parameter tuning was done using the tune() functions in "e1071 library, however better result was achieved using the default parameters (e.g cost = 1 and epsilon = 0.1). Similarly,fornaive Bayes, the default Laplace smoozing value = 0.5 in the multinomial_naive_bayes() function of the "naivebayes" library was used was used.

4.1.2 Comparison for all metrics at N=10 using ML-Based IRS vs random recommender

This section presents in Table 8, the results for all metrics at a recommendation size N = 10 and using the three ML algorithms separately. In addition, it also presents the result achieved through a random recommendation(RAND). For this evaluation, both the textual and categorical features(TCF) were used.

Table 8. Result of P, R, F1, and MAP for N = 10 using TCF for each algorithms vs RAND

PROJ	ALG	P@10	R@10	F@10	MAP@10
NTB	KNN	29.2	93	44.4	86
	NBY	29.8	95.3	45.4	88.5
	SVM	30.5	96	46.3	90.5
	RAND	3.2	10.1	4.9	8.6
ECL	KNN	26.5	75	39.2	63.5
	NBY	27.5	76.6	40.5	63.9
	SVM	27.9	78.6	41.2	64.1
	RAND	3.7	9.7	5.4	9.4
FDT	KNN	20.2	77.2	32	63.5
	NBY	20.2	76.1	31.9	61
	SVM	21.2	81.5	33.6	66.6
	RAND	2.6	9.9	4.1	6.9
MULE	KNN	39.7	68.4	50.2	76.6
	NBY	42.6	73.2	53.9	78.7
	SVM	41.5	70	52.1	78.2
	RAND	6.7	10.3	8.1	14
MESOS	KNN	21.4	54.9	30.8	55.4
	NBY	23.4	59.1	33.5	58
	SVM	23.4	59.4	33.6	58.4
	RAND	3.3	9.4	4.9	6.7
TIMOB	KNN	33.4	49.6	39.9	55.5
	NBY	35.1	52.9	42.2	45.9
	SVM	33.9	49.9	40.4	43.8
	RAND	7.4	8.8	8	13.6

From Table 8, the highest results achieved for each metrics in every project from the result of KNN, NBY and SVM is highlighted. This value is compared with the result of a random recommender in Figure 7 below in the form of a bar plot and Figure 8 shows the ratio of the result for each metrics against the result of the random recommender.

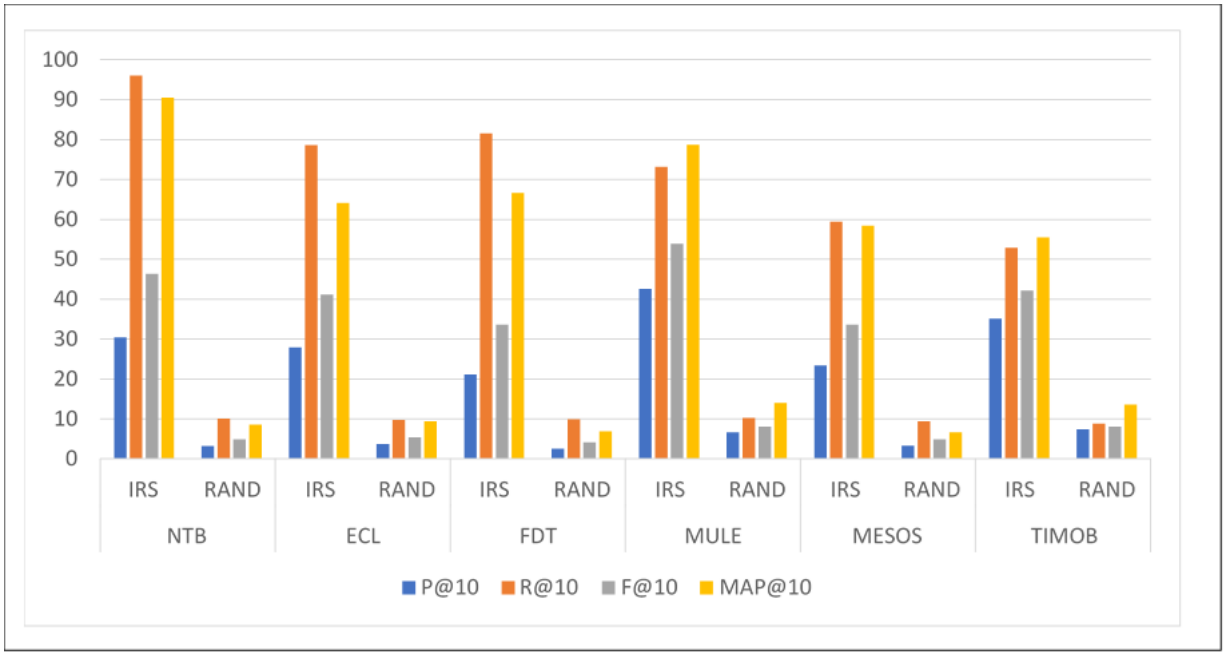


Figure 7. Comparision of best results of P, R, F1, and MAP for N = 10 using TCF vs RAND

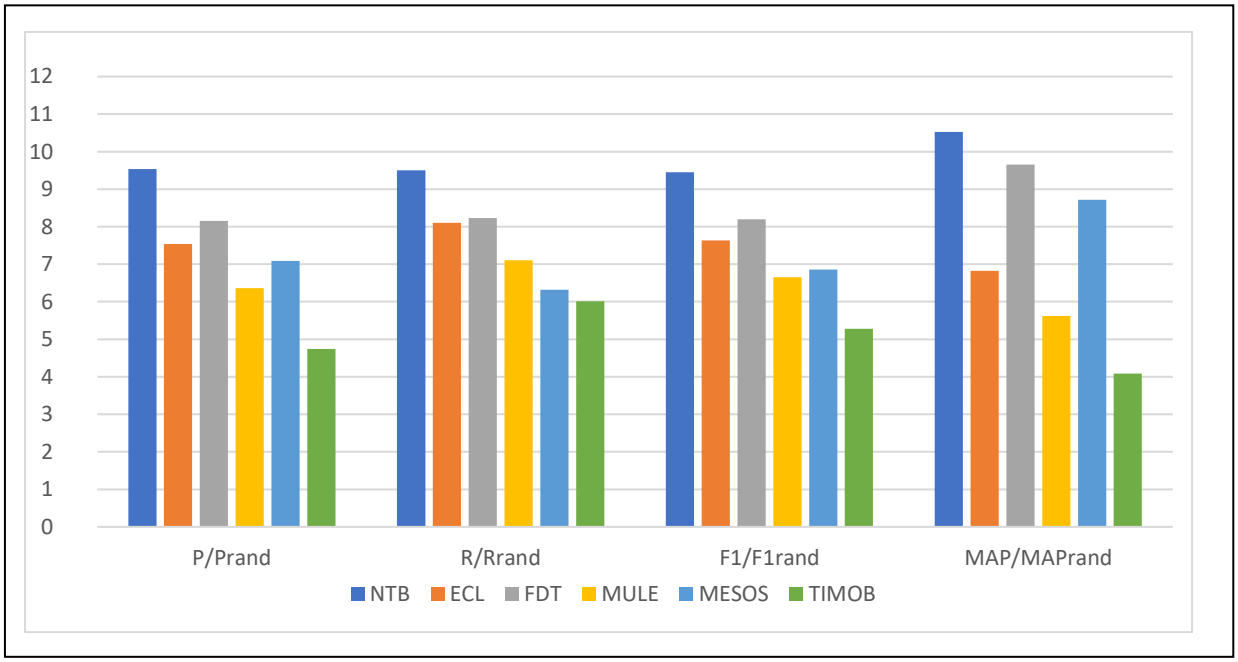


Figure 8. Comparision of ration of best result for P, R, F1, and MAP at N = 10 from KNN, SVM and NBY using TCF vs RAND

4.2 RQ 2

The purpose of the results under RQ 2 is to show how a comparison of all metrics using textual fields only versus using both textual and categorical fields as a source of features.

4.2.1 Comparison of results of all metrics at N=10 for TFO vs TCF

In this section, Table 9 presents, for each algorithm, the result of all metrics for the top 10 recommendation using textual fields only(TFO). It also includes the best results found using TCF in RQ 1 from the three algorithms which are highlighted in Table 8.

Table 9. Result of P, R, F1, and MAP for N = 10 for each algorithms using TFO vs best result using TCF

PROJ	ALG	P@10	R@10	F@10	MAP@10
NTB	KNN	18.2	57	27.6	60.9
	NBY	18.6	58.4	28.2	56.7
	SVM	18.7	57.8	28.3	58.7
	TCF - Best	**30.5**	**96**	**46.3**	**90.5**
ECL	KNN	15.2	38.1	21.7	38.9
	NBY	16.2	40.9	23.2	36.2
	SVM	16.9	42.4	24.2	37.9
	TCF - Best	**27.9**	**78.6**	**41.2**	**64.1**
FDT	KNN	16.3	60.5	25.7	51.9
	NBY	16.8	61.1	26.4	49.5
	SVM	17	64.4	26.9	49.5
	TCF - Best	**21.2**	**81.5**	**33.6**	**66.6**
MULE	KNN	16.1	26.7	20.1	35.7
	NBY	15.3	24.2	18.7	32.8
	SVM	15.5	23.3	18.6	34.7
	TCF-Best	**42.6**	**73.2**	**53.9**	**78.7**
MESOS	KNN	8.1	19	11.4	16.8
	NBY	8.2	17.7	11.2	17
	SVM	7.9	19.8	11.3	18.3
	TCF - Best	**23.4**	**59.4**	**33.6**	**58.4**
TIMOB	KNN	18.8	25.8	21.8	33.7
	NBY	18	25.9	21.2	32.8
	SVM	18.9	28.1	22.6	38.7
	TCF - Best	**35.1**	**52.9**	**42.2**	**55.5**

In Figure 9, the best results achieved for all metrics by using TFO which are highlighted in Table 9 are compared against the best results achieved using TCF in the form of a bar plot.

Next, In figure 10, the percentage increase achieves for all metrics between using TFO and TCF is also presented in a bar pot.

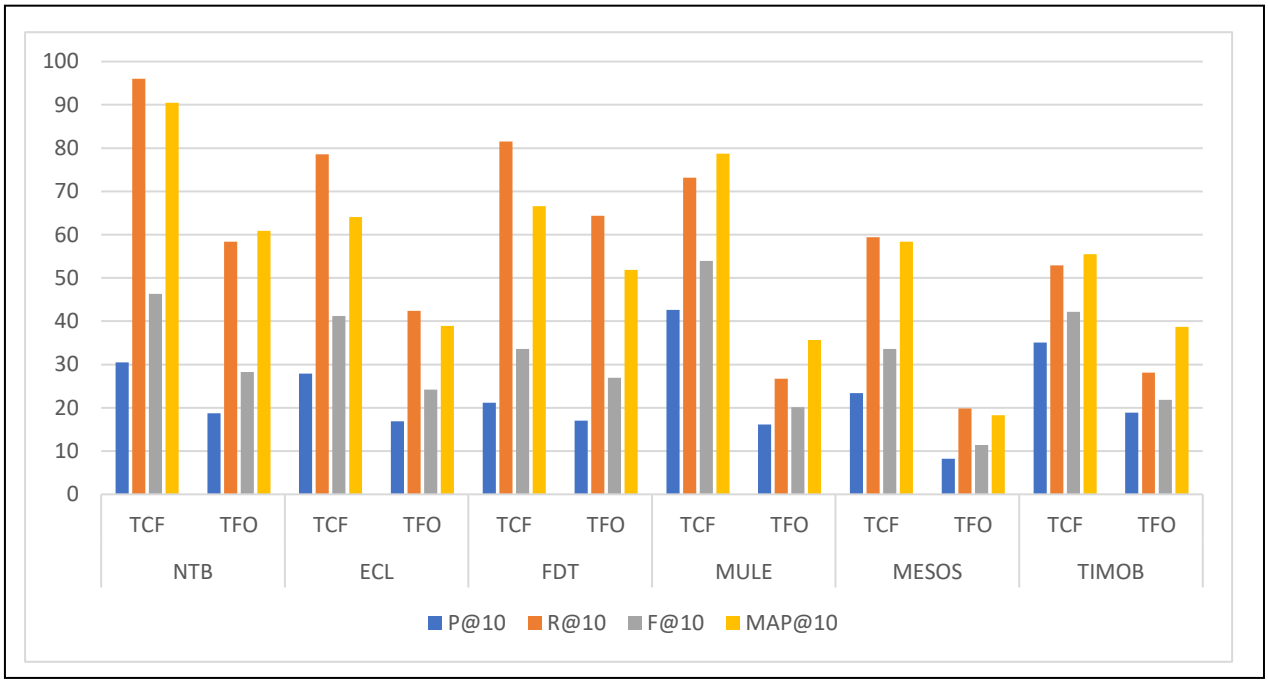


Figure 9. Comparision of best results of P, R, F1, and MAP for N = 10 using TFO vs TCF

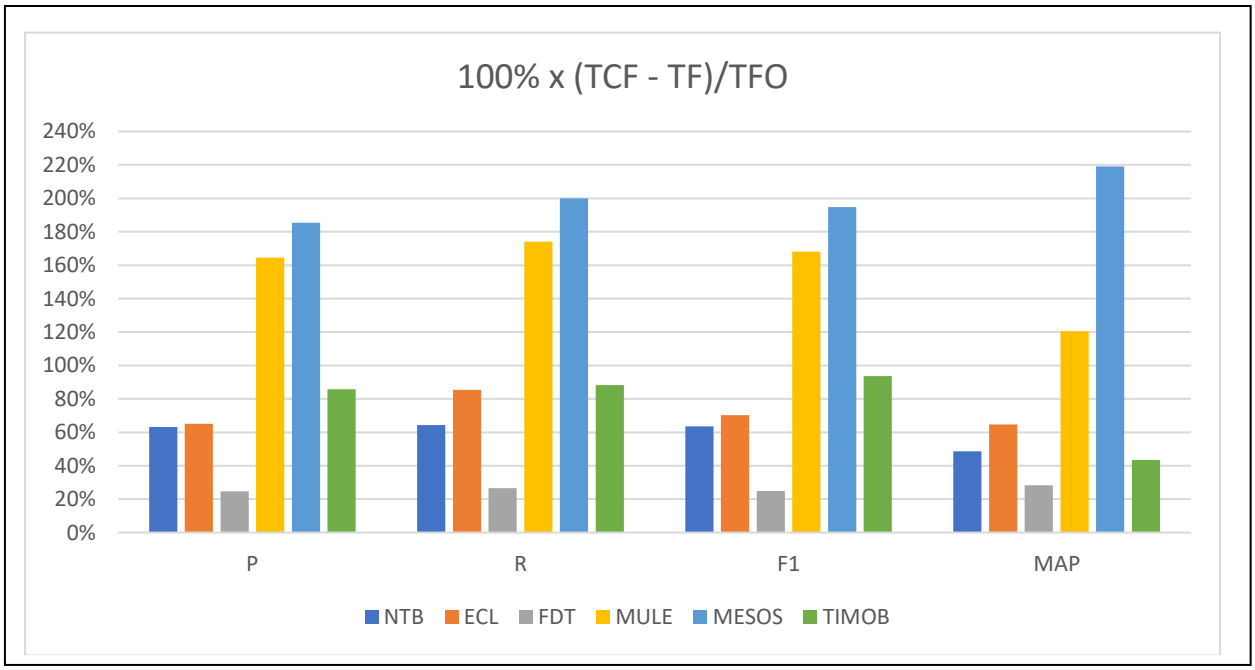


Figure 10. Comparision of % increase of P, R, F1, and MAP for N = 10 using TFO vs TCF

4.2.2 Comparison of the influence of features using the chi-squared statistics

In this section, the top 10 terms in the issue reports of each project with the highest Cramer's V coefficient values calculated from the chi-squared feature selection step and the fields of the issue reports they were extracted from is presented in Figure 11. Moreover, the average of the Cramer's V coefficient of all terms extracted from each field of the issue report is compared in Figure 12.

Top	NTB	ECL	FDT	MULE	MESOS	TIMOB
1	maven	david_williams	freedesktop	rodrigo_merinc	hausdorff	ios
2	javascript	releng	driver_intel	pablo_kraan	klueska	qe_tracking
3	versioncontrol	pde	spreadsheet	fgonz	mesosphere	cli
4	css_visual_tool	framework	xorg	afelisatti	window	cbarber
5	platform	ui	libreoffice	facundo_velazq	jieyu	cli
6	projects	core	mesa	juan_desimoni	haosdent_gma	android
7	parsing___inde	platform	dri	dfeist	vinodkone	blackberry
8	html_editor	equinox	drm_intel	lautaro_fernan	bbannier	release
9	web	remy_suen_gm	sna	mule4	mcypark	blackberri
10	inspection	tjwatson_us_ib	unspecified	pablo_lagreca_	andschwa	dthorp
	component	reporter	product	text	version	label

Figure 11. Comparision of average Cramer's V coefficient of terms from each field of the issue reports

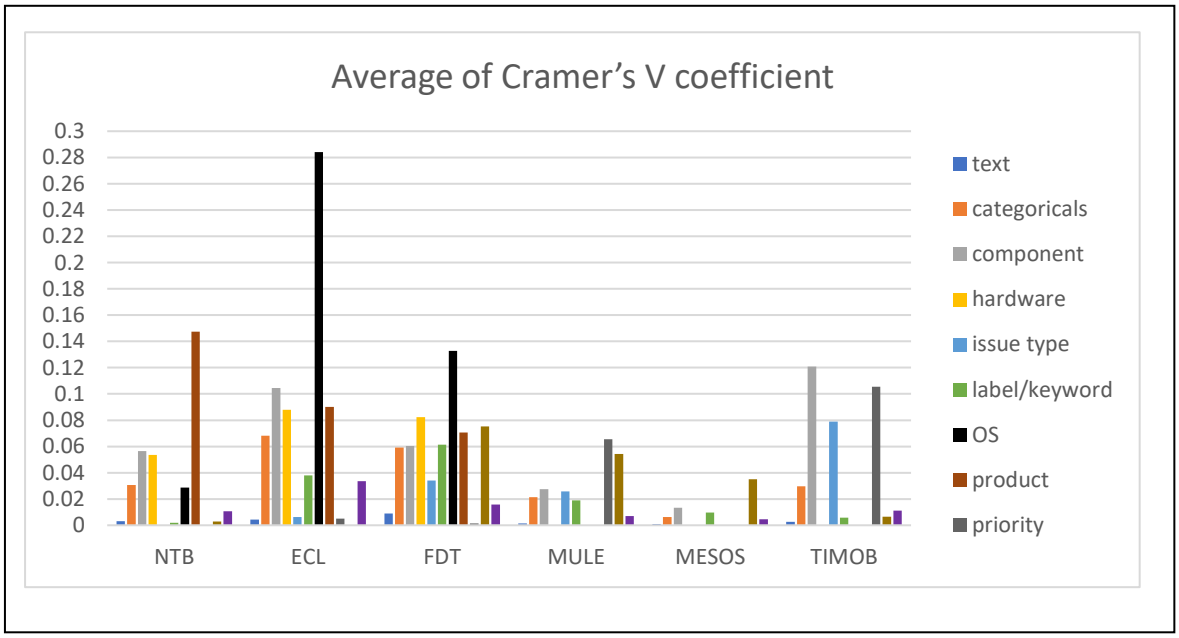


Figure 12. Top 10 influential terms and thier source field using Cramer's V coefficient

4.3 RQ 3

4.3.1 Comparison of all metrics for N = 1 up to 25 for each algorism using TCF

In this part of the result, the P, R, F1 and MAP values of the IRS are compared for recommendation size N ranging from 1 to 25 for each algorithm including a random recommender using TCF. It also includes the maximum F1-score points. The results for each algorithm and project are presented in the form of a line graph from Figure 13 to 18.

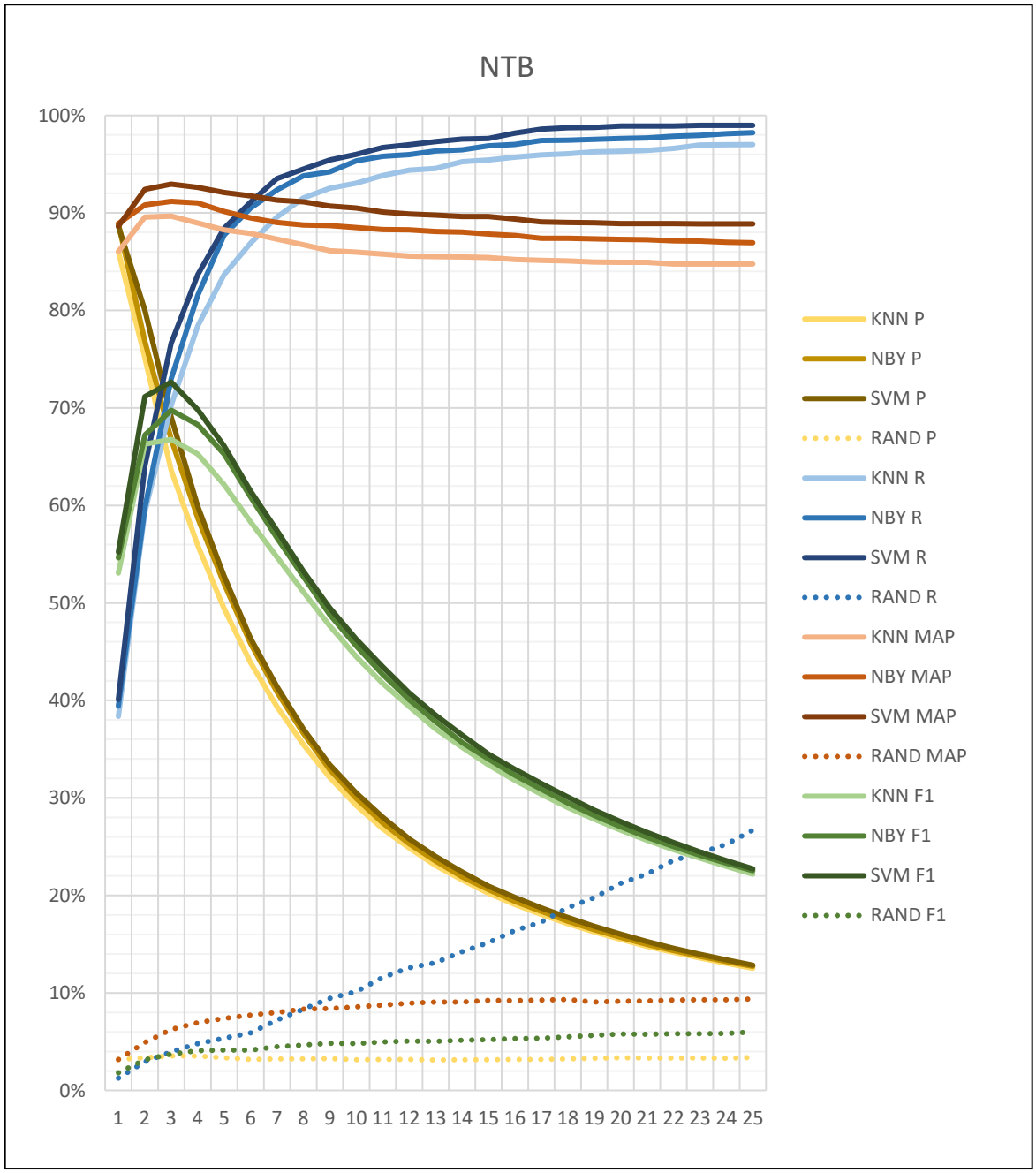


Figure 13. Comparision of all metrics as N increase from 1 to 25 using TCF for NTB

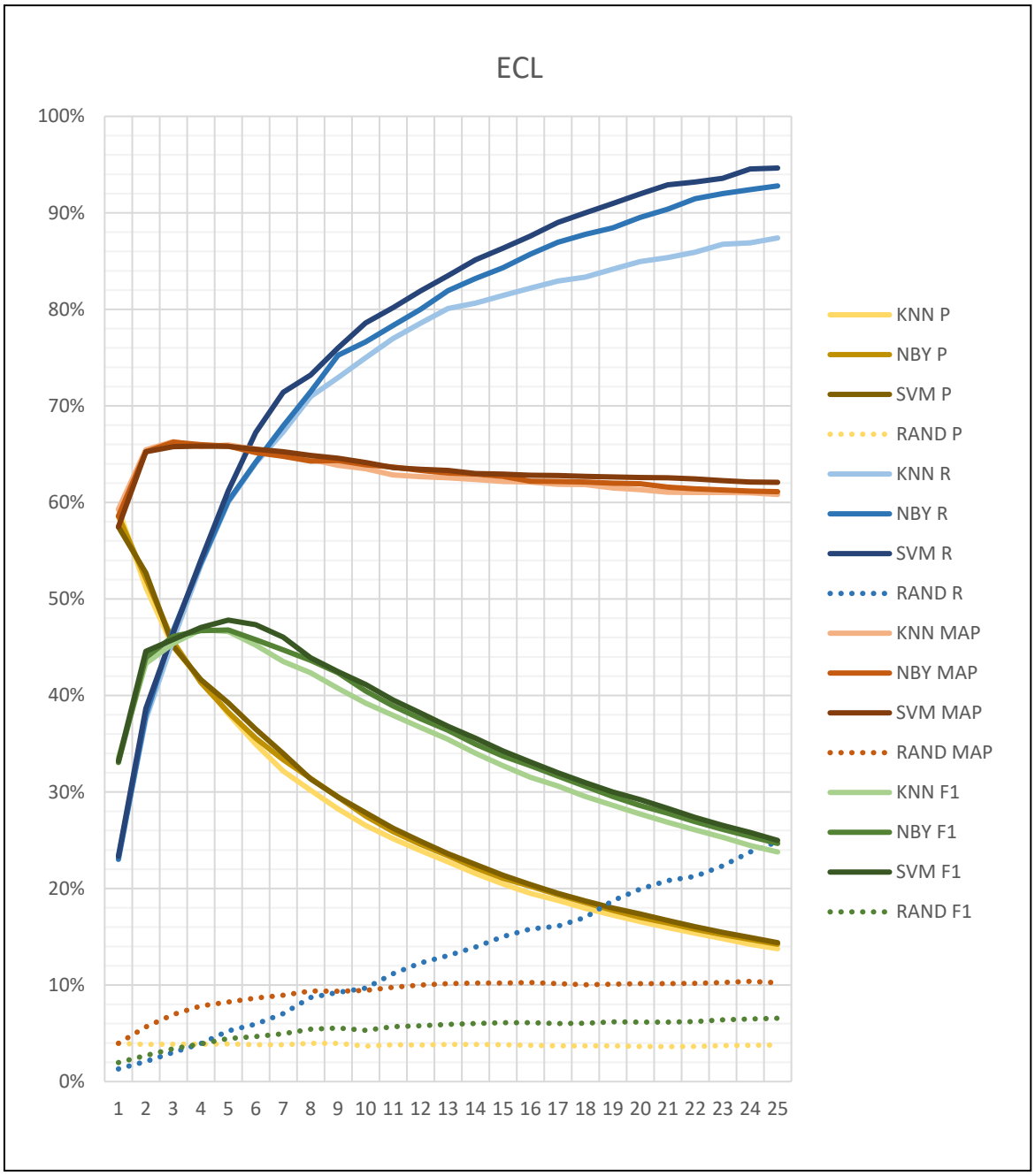


Figure 14. Comparision of all metrics as N increase from 1 to 25 using TCF for ECL

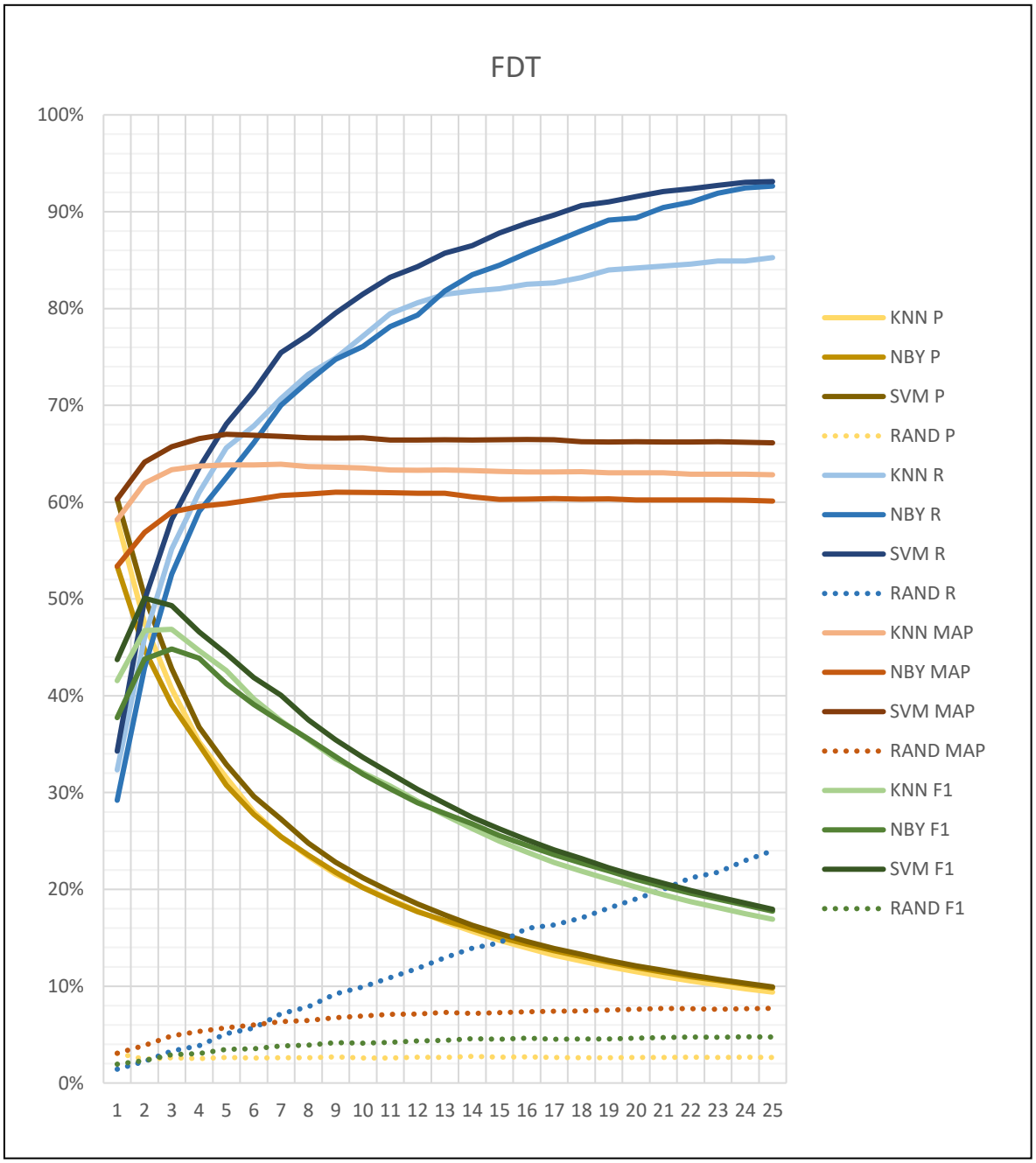


Figure 15. Comparision of all metrics as N increase from 1 to 25 using TCF for FDT

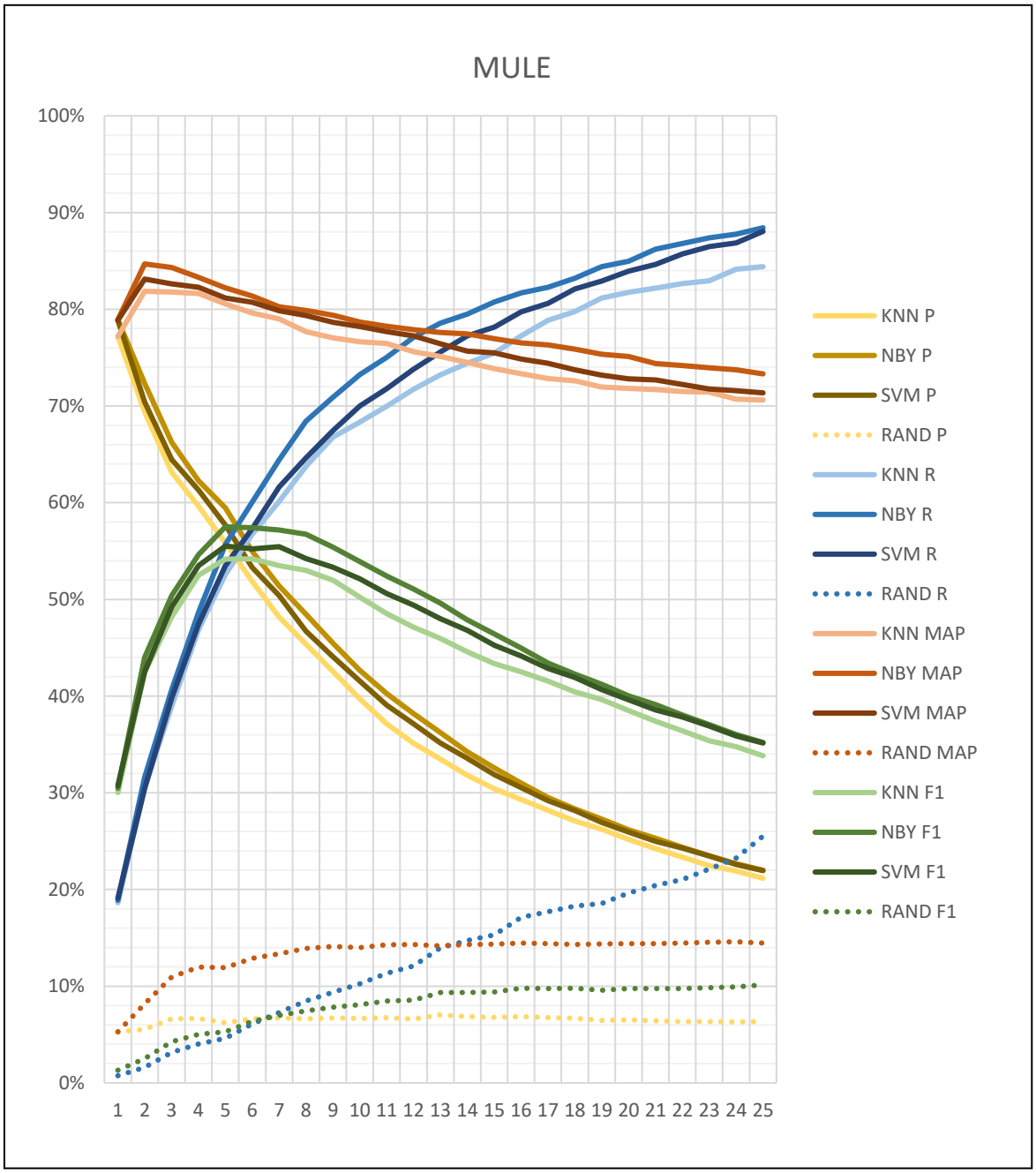


Figure 16. Comparision of all metrics as N increase from 1 to 25 using TCF for MULE

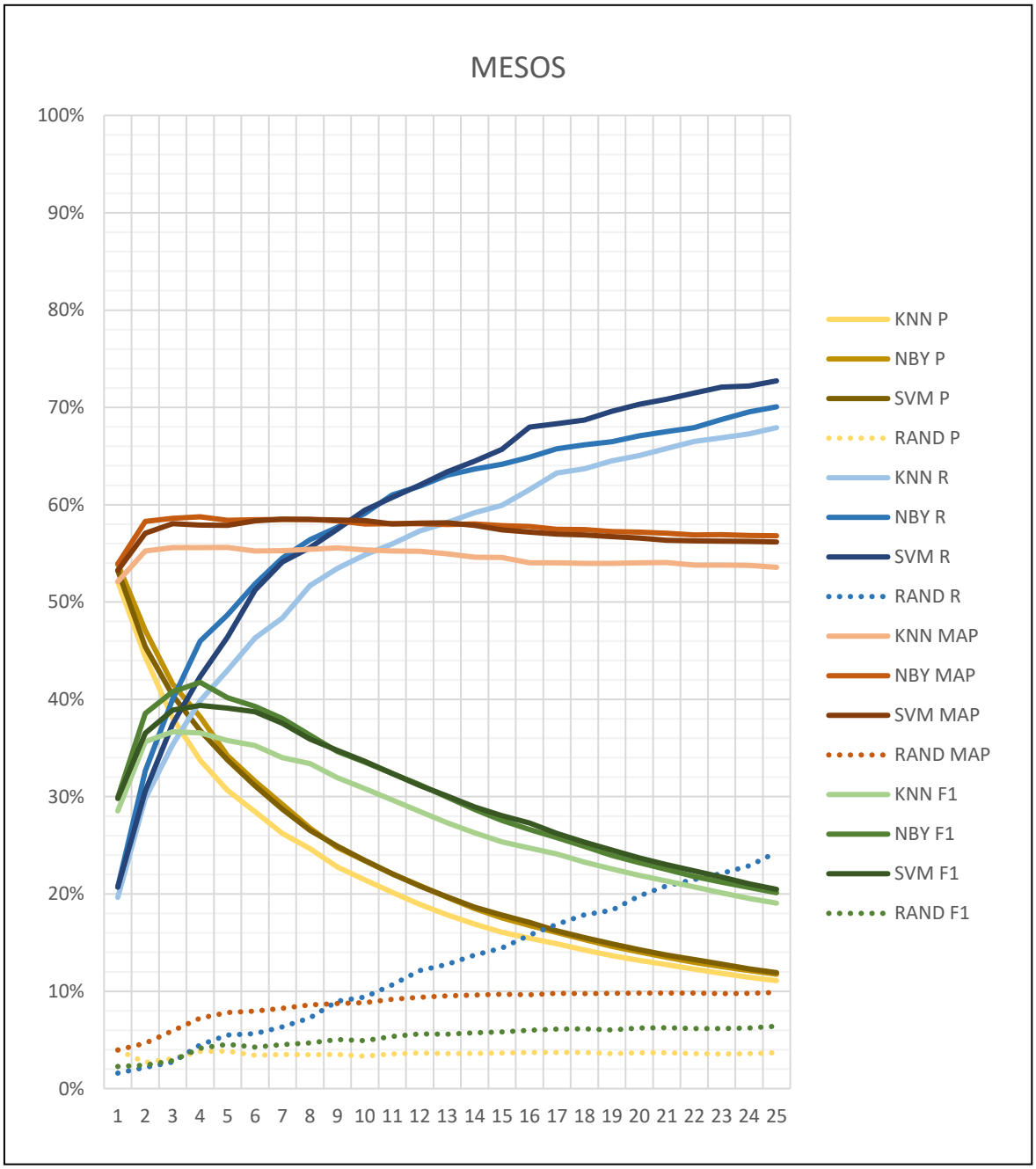


Figure 17. Comparision of all metrics as N increase from 1 to 25 using TCF for MESOS

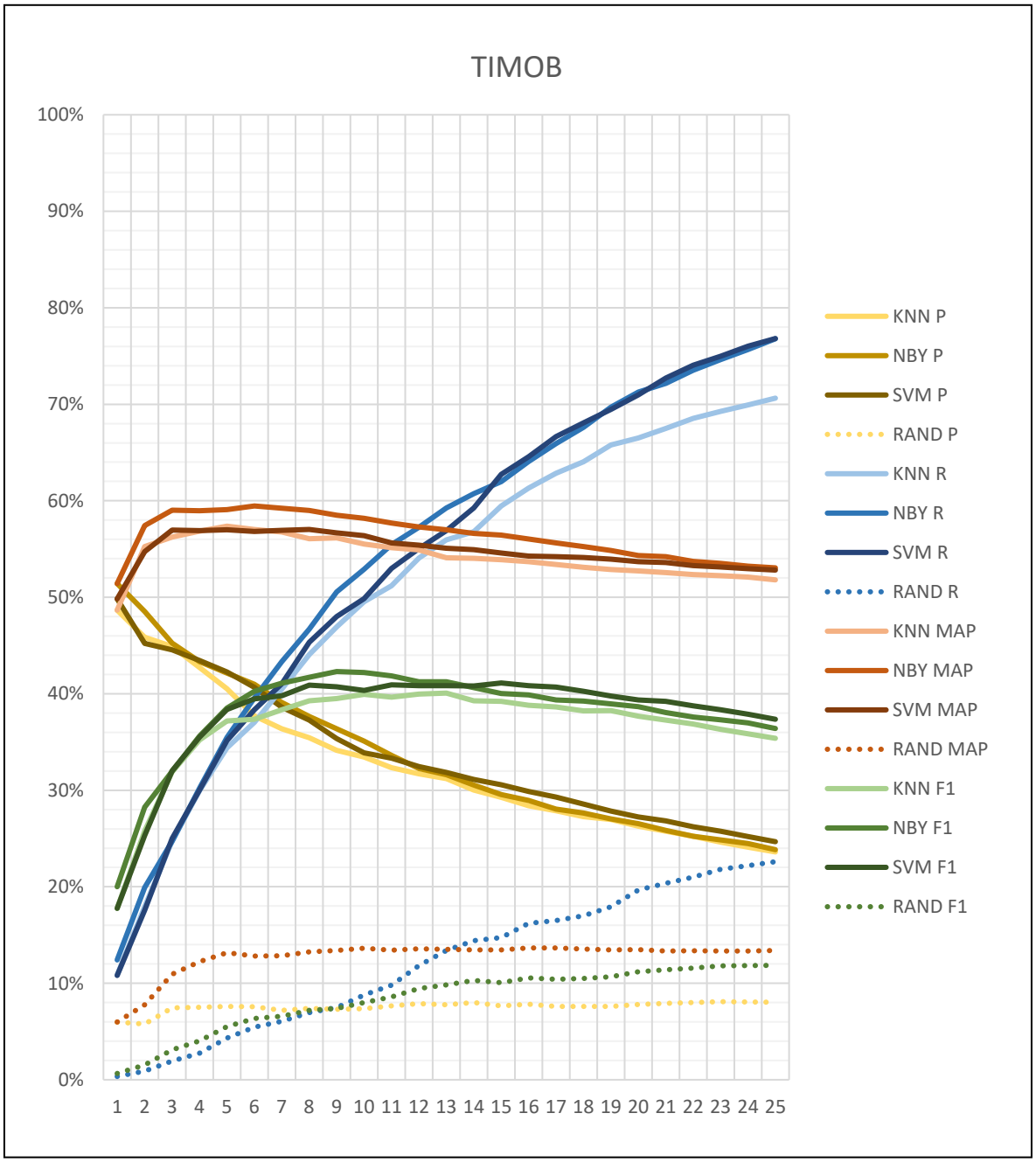


Figure 18. Comparision of all metrics as N increase from 1 to 25 using TCF for TIMOB

4.3.2 Comparison of all metrics at N = maximum F1-score point using TCF

This part presents the value of all metrics at maximum F1-sore for each algorism using both textual and categorical features in Table 10.

Table 10. Result of P, R, F1, and MAP for N = maximum F1 point for each algorithms using TCF

PROJ	ALG	N	P	R	F1(max)	MAP
NTB	KNN	3	63.61	70.27	66.77	89.67
	NBY	3	66.81	73	69.76	91.19
	SVM	3	69.06	76.63	72.65	92.95
ECL	KNN	4	41.63	53.48	46.81	65.79
	NBY	5	38.28	60.14	46.78	65.82
	SVM	5	39.23	61.2	47.81	65.82
FDT	KNN	3	40.73	55.17	46.87	63.34
	NBY	3	39.09	52.55	44.83	58.97
	SVM	2	50.26	49.9	50.08	64.12
MULE	KNN	6	51.85	56.74	54.19	79.62
	NBY	5	59.46	55.69	57.51	82.21
	SVM	5	57.68	53.51	55.52	81.16
MESOS	KNN	3	38.09	35.35	36.67	55.6
	NBY	4	38.22	46	41.75	58.76
	SVM	4	36.79	42.35	39.37	57.91
TIMOB	KNN	13	31.21	55.96	40.07	54.09
	NBY	9	36.37	50.55	42.3	58.5
	SVM	15	30.58	62.74	41.12	54.57

5 Discussion

In this section, the results presented in the previous section are discussed with respect to the research questions of this study.

RQ1: What is the performance of an IRS using ML algorithms with respect to a random recommender?

To answer this question, we have evaluated the performance of the IRS we implemented for top 10 recommendation using precision, recall, f1-score and mean average precision and compared these results with the result found by random recommendation in section 4.1.

The result shows that, using the ML-based IRS, it is possible to achieve best top 10 recall ranging from 52.9% up to 96 % for TIMOB and NTB projects respectively, which means developers can find more than half of the issue reports relevant to them in the top 10 issue reports recommended from a backlog of 100 unassigned issue reports. In NTB, the ratio of the top 10 recall of the ML-based IRS with the recall of a random recommendation is 9.5 which means developers are around 9 times more likely to find the issue reports that they are later assigned to from the top 10 recommendation of the IRS compared to randomly picked 10 issue reports. This ratio for the ECL, FDT, MULE, MESOS and TIMOB is 8.1, 8.2, 7.1, 6.3 and 6 respectively. The best results achieved for top 10 recall are by using the SVM algorism in the NTB, ECL, FDT and MESOS project and using Naïve Bayes in MULE and MESOS projects.

Similarly, the best precision achieved for top 10 recommendation ranges from 23.4% up to 42.6 in MESOS and MULE projects respectively. The precision range for top 10 recommendation is generally lower than 50 % which means developers were later assigned to less than half of the recommended issue reports. However, comparing this result with the precision of a random recommender, more or less the same improvement achieved for the recall was also achieved which ranges from 4.7 up to 9.5 for TIMOB and NTB projects respectively.

The reason why the precision value is lower than recall for top 10 recommendation is that the number of issue reports that each developer is assigned to from 100 unassigned issue reports on average is less than 10 in most projects and this sets a limit to the maximum precision that can be achieved. For example, the number of active developers in the NTB project is 36, which means each developer fixes approximately 3 issue reports from the 100 issue reports if they were equally divided among the developers. Even if the IRS includes all three issue reports in the top 10 recommendation, the precision that can be achieved is still 30%. In NTB, the top 10 precision achieved is 29.2% which is quite close to the expected precision if issue reports were equally divided among the developers.

The precision metric has limitations in telling how well the IRS performs, therefore, to fill this limitation, we additionally compared the MAP which measures not just how correct the recommendations are, but also how precisely the issue reports are ranked in the top 10 recommendations. For the top 10 recommendation, the MAP achieved ranges from 55.5% up to 95.5 % for TIMOB and NTB respectively, which is pretty much close to the range for top 10 recall. If we assume that, in NTB, each developer was assigned to an equal number of issue reports (i.e. 3 issue reports approximately), the 95.5 % MAP tells us that the IRS is not only able to include the 3 issue reports that each developer fixed in the 10 issue reports it recommends but also was able to give them a higher rank, most probably in the top 5 issue reports. Comparison of MAP of the ML-based IRS with a random recommender shows a ratio ranging from 4.1 up to 10.5 for TIMOB and NTB respectively.

RQ 2: How much did including features from the categorical fields affect the performance of the IRS.

For RQ 2, we implemented the IRS using only features extracted from the textual fields (i.e., the summary and description) and evaluated the performance using all metrics for top 10 recommendation and compared this result with what we found in RQ 1 using both the textual and categorical fields as presented in section 3.4. Taking the result for NTB as an example, the best result for top 10 recall achieved using textual features only is 58.4%; however, by including the categorical features a recall of 96% was achieved. This means an increase of 64.4%. Similarly, in the ECL, FDT, MULE, MESOS and TIMOB projects an increase of 85.3%, 26.5%, 174.2%, 200%, 88.2% was achieved. For precision, the percentage increase ranges from 24.7% for FDT up to 185.3% for MESOS. This shows that including the categorical fields when extracting features improves the performance of the IRS significantly.

In addition to this, we also used the result of the Cramer's coefficient calculated using chi-squared statistics in the feature selection step to get further insight into which fields of the issue report contributed the terms which are associated with the developers assigned to the issue reports. Figure 11 in section shows the top 10 terms with the highest Cramer's coefficient value. In can be seen that most of these terms come from categorical fields like component, reporter and product.

Comparison of the average of the Cramer's coefficient value of terms which are extracted from each field is also presented in Figure 12. From this figure, it can be seen that the field that is most associated with the assignee are the Product type(in NTB), the Operating System(in ECL and FDT), Software Component(in MULE and TIMOB) and the Reporter(in MESOS). It can also be seen that the average of the Cramers Coefficient for terms from the textual fields like summary and description is very low compared to the terms form the categorical fields.

RQ3: What is the optimal recommendation size that maximizes the f1-score (i.e. giving a good balance between precision and recall)?

For RQ 3, we compared how the performance of the IRS changes in terms of each evaluation metrics as the recommendation size varies using a line graph as shown from Figure 13 up to 18. From these graphs, it is possible to notice a similar trend in each project. As recommendation size increases, the precision value decreases while the recall value increases. This is expected because the chance of making incorrect recommendation increases but more relevant issue reports start to get included in the recommendation. The MAP, on the other hand, shows a small increase at the beginning and stays more or less constant. This is because the MAP is more dependent on the correctness of the top-ranked issue reports and its value doesn’t change significantly even if correct recommendations are made at a lower rank.

Since the precision and recall values have opposite behaviours, we compared the F1 score to find the optimum sizes of recommendation which balances the two. The trend for F1 score is, that it starts to increase at the beginning until it reaches a maximum and starts to decrease. The maximum value for the F1 score is oftentimes around the crossing point between precision and recall because it gives equal weight to precision and recall.

The size of recommendation which archives the highest F1-score is different from project to project as recorded in Table 10. In addition to the projects, each algorithm might also reach maximum F1 score at different points. For example, in NTB the maximum f1 score was reached at N = 3 for all algorithms while in TIMOB it was reached at N = 13, 9, and 15 for KNN, NBY and SVM respectively. Taking the algorithms that archives the highest maximum F1, the optimum sizes of recommendation found for NTB, ECL, FDT, MULE, MESOS and TIMOB are 3, 5, 2, 5, 4, 9 respectively. For NTB, ECL and FDT the best maximum f-score achieved was using SVM while in MULE, MESOS and TIMOB it was achieved using the Naïve Bayes algorithm.

Taking the algorithms that archives the highest maximum F1, the optimum sizes of recommendation found for NTB, ECL, FDT, MULE, MESOS and TIMOB are 3, 5, 2, 5, 4, 9 respectively. For NTB, ECL and FDT the best maximum F1-score was achieved using SVM and it amounts to 72.65%, 47.81%, and 50.08 respectively, while in MULE, MESOS and TIMOB it was achieved using the Naïve Bayes algorithm and it amounts to 57.51%, 41.75% and 42.3% respectively.

To generalize about the performance of the three different algorithms used to build the IRS it is possible to calculate the average of the result achieved by each algorithm in each project at the maximum F1-score point as shown in Figure 19 below. From this figure, it can be seen that on average SVM has the best performance in all metrics followed by NBY and KNN respectively. However, since the difference in performance is not very significant it is possible to use each of these algorithms to build an IRS.

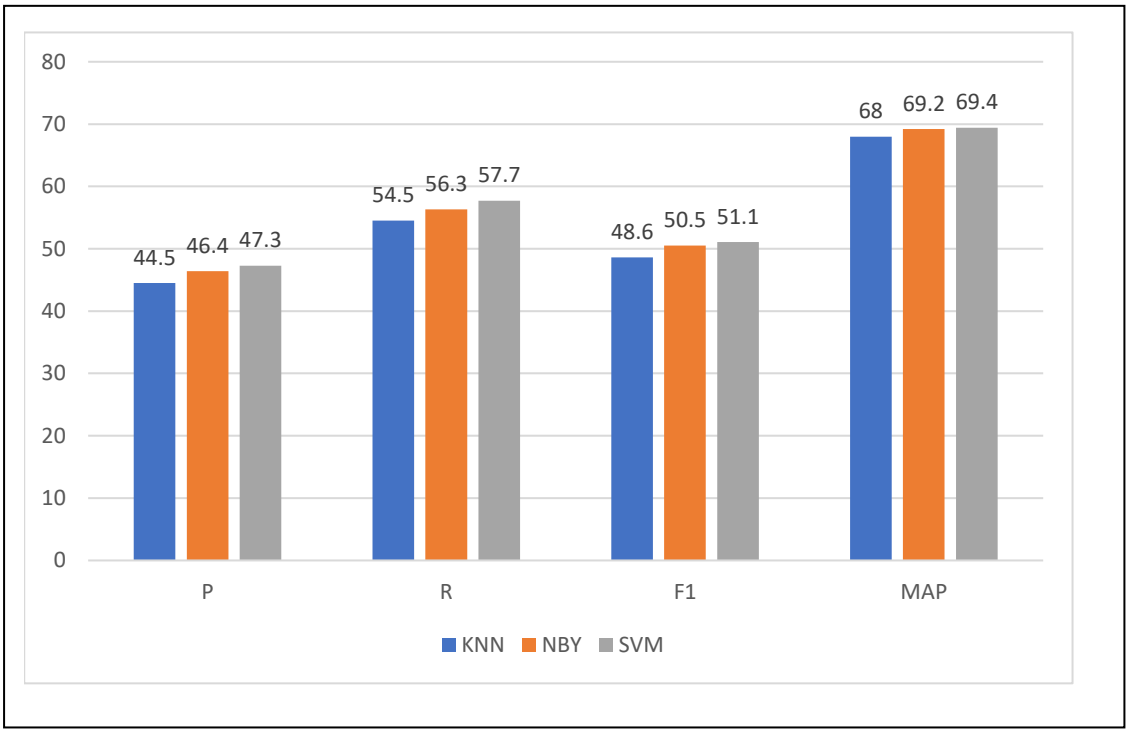


Figure 19. Average of all metrics for KNN, NBY and SVM across all projects @ N = maximum F1-score point

6 Conclusion

In this study, we have shown how traditional machine learning algorithms like KNN, Naïve Bayes and SVM can be used to implement an issue recommender system that can facilitate the self-assignment of issue reports by developers. Most previous works have used these algorithms to either do an automatic assignment or to recommend developers to a third person that does the assignment. In this study, we used these algorithms to perform probabilistic classifications and used their probability estimates to rank the issue reports for each developer and recommend the top ones.

By evaluating the issue recommender system in terms of precision, recall, f1-score and mean average precision metrics and comparing it with a random recommender, we have shown how easily developers can find relevant issue reports from the issues reports recommended to them using machine learning. The evaluation in terms of recall shows that developers are up to 9 times more likely to find the issue reports that they later fixed in the top 10 recommendations given by the issue recommender system compared to picking 10 issue reports randomly from 100 unassigned issue reports. The same improvements are also reflected in the other metrics.

Most previous works have also focused on the summary and description of the issue reports as an information source to implement their solutions for automatic assignment and developer recommendation. In this study, we have tried to include more features from categorical metadata fields like the software component, version and platform(ie.g operating system and hardware) the issue reports belong to, the reporter who created the issue reports and the priority and severity of the issue and a significant improvement in all metrics.

We have also shown how we can use metrics like f1-score to find an Optimum recommendation size that can achieve a good balance between precision and recall. The best maximum f1-score we achieved was 72.65% in the Netbeans project algorithm for top 3 recommendation size with precision and recall amounting to 69.06 % and 76.63% respectively. The worst maximum f1-score we achieved was 42.3% in the TIMOB project for top 9 recommendation size with precision and recall amounting to 50.55% and 42.3% respectively. The comparison of the three algorithms we evaluated shows that SVM has the best performance followed by Naive Bayes and KNN respectively with a very small difference in each evaluation metrics.

Overall, the issue recommender we evaluated has a reasonable performance to apply in an issue tracking software. In these issue tracking software, it can be used as an additional way of sorting issue reports according to each developer and to create a more customized backlog. It can also be used to send periodic recommendations of bugs to developers participating in open source projects. However, its performance can be improved by including by collecting feedback from developers on the recommendations using ratings, comments and likes.

7 References

[1] Crowston K, Li Q, Wei K, Eseryel UY, Howison J. Self-organization of teams for free/libre open source software development. Information and software technology. 2007 Jun 1;49(6):564-75.

[2] Hoda R, Noble J, Marshall S. Developing a grounded theory to explain the practices of self-organizing Agile teams. Empirical Software Engineering. 2012 Dec 1;17(6):609-39.

[3] Murphy G, Cubranic D. Automatic bug triage using text categorization. InProceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering 2004 Jun.

[4] Ahsan SN, Ferzund J, Wotawa F. Automatic software bug triage system (BTS) based on latent semantic indexing and support vector machine. In2009 Fourth International Conference on Software Engineering Advances 2009 Sep 20 (pp. 216-221). IEEE.

[5] Nasim S, Razzaq S, Ferzund J. Automated change request triage using alpha frequency matrix. In2011 Frontiers of Information Technology 2011 Dec 19 (pp. 298-302). IEEE.

[6] Tamrawi A, Nguyen TT, Al-Kofahi J, Nguyen TN. Fuzzy set-based automatic bug triaging (NIER track). InProceedings of the 33rd International Conference on Software Engineering 2011 May 21 (pp. 884-887). ACM.

[7] Hu H, Zhang H, Xuan J, Sun W. Effective bug triage based on historical bug-fix information. In2014 IEEE 25th International Symposium on Software Reliability Engineering 2014 Nov 3 (pp. 122-132). IEEE.

[8] Klir GJ, Yuan B. Fuzzy sets and fuzzy logic: theory and applications. Upper Saddle River. 1995:563.

[9] Mani S, Sankaran A, Aralikatte R. Deeptriage: Exploring the effectiveness of deep learning for bug triaging. InProceedings of the ACM India Joint International Conference on Data Science and Management of Data 2019 Jan 3 (pp. 171-179). ACM.

[10] Rocha H, Oliveira G, Maques-Neto H, Valente MT. Nextbug: A tool for recommending similar bugs in open-source systems. InV Brazilian Conference on Software: Theory and Practice–Tools Track (CBSoft Tools). SBC, Maceio, AL, Brazil 2014 (Vol. 2, pp. 53-60).

[11] Aggarwal CC, Zhai C. A survey of text classification algorithms. In Mining text data 2012 (pp. 163-222). Springer, Boston, MA.

[12] Patra A, Singh D. A survey report on text classification with different term weighing methods and comparison between classification algorithms. International Journal of Computer Applications. 2013 Jan 1;75(7).

[13] Hotho A, Nürnberger A, Paaß G. A brief survey of text mining. In Ldv Forum 2005 May 13 (Vol. 20, No. 1, pp. 19-62).

[14] Brownlee J. Why One-Hot Encode Data in Machine Learning? [Internet]. Machine Learning Process. 2017 July 28. Available from: https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

[15] Yang Y, Pedersen JO. A comparative study on feature selection in text categorization. In Icml 1997 Jul 8 (Vol. 97, No. 412-420, p. 35).

[16] Alenezi M, Magel K, Banitaan S. Efficient Bug Triaging Using Text Mining. JSW. 2013 Sep 1;8(9):2185-90.

[17] Lewis DD. Naive (Bayes) at forty: The independence assumption in information retrieval. In European conference on machine learning 1998 Apr 21 (pp. 4-15). Springer, Berlin, Heidelberg.

[18] McCallum A, Nigam K. A comparison of event models for naive bayes text classification. In AAAI-98 workshop on learning for text categorization 1998 Jul 26 (Vol. 752, No. 1, pp. 41-48).

[19] Tan S. Neighbor-weighted k-nearest neighbour for unbalanced text corpus. Expert Systems with Applications. 2005 May 1;28(4):667-71.

[20] Cortes C, Vapnik V. Support-vector networks. Machine learning. 1995 Sep 1;20(3):273-97.

[21] Wu TF, Lin CJ, Weng RC. Probability estimates for multi-class classification by pairwise coupling. Journal of Machine Learning Research. 2004;5(Aug):975-1005.

[22] Chang CC, Lin CJ. LIBSVM: A library for support vector machines. ACM Transactions on intelligent systems and technology (TIST). 2011 Apr 1;2(3):27.

[23] Srividhya V, Anitha R. Evaluating preprocessing techniques in text categorization. International journal of computer science and application. 2010;47(11):49-51.

[24] Chen M, Liu P. Performance Evaluation of Recommender Systems. International Journal of Performability Engineering. 2017 Dec 1;13(8).

[25] Herlocker JL, Konstan JA, Terveen LG, Riedl JT. Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems (TOIS). 2004 Jan 1;22(1):5-3.

Appendix

I. Code and Dataset

The dataset of the collected issue reports and the code used for each project can be found in the following Kaggle link: https://www.kaggle.com/abelmes/irs-dataset/

II. License

Non-exclusive licence to reproduce thesis and make thesis public

I, Abel Mesfin Cherinet

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

Recommending Issue Reports to Developers Using Machine Learning

supervised by **Ezequiel Scott (PhD)**

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Abel Mesfin Cherinet

08/14/2019