AUTOMATED SOFTWARE ISSUE TRIAGE IN LARGE SCALE INDUSTRIAL CONTEXTS

by
ETHEM UTKU AKTAŞ

Submitted to the Faculty of Engineering and Natural Sciences
in partial fulfilment of
the requirements for the degree of Doctor of Philosophy

Sabancı University
June 2021

AUTOMATED SOFTWARE ISSUE TRIAGE IN LARGE SCALE INDUSTRIAL CONTEXTS

Approved by:

Date of Approval: June 28, 2021

ABSTRACT

AUTOMATED SOFTWARE ISSUE TRIAGE IN LARGE SCALE INDUSTRIAL CONTEXTS

ETHEM UTKU AKTAŞ

Computer Science and Engineering Ph.D DISSERTATION, June 2021

Dissertation Supervisor: Assoc. Prof. Cemal Yılmaz

Keywords: issue triaging, text analysis, image analysis, explainable machine learning, change point detection

Software issue reports are the documents describing the problems users face when using a software product and software issue triage is the process of validating and assigning these issue reports. In practice, issue triage is carried out manually by experts or developers. In large scale industrial contexts, hundreds of software products exist and hundreds of issue reports are filed every day. It takes a great amount of human effort to triage these reports and failure to solve them on time results in customer dissatisfaction. In this thesis, we automate the issue triage process by using data mining approaches and share our experience gained by deploying the resulting system in a large scale industrial setting. Deployment of such a system presented us not only with an opportunity to observe the practical effects of automation, but also to carry out user studies, both of which have not been done before in this context. Furthermore, we developed and empirically evaluated methods on how to create human-readable, non-technical explanations for the predictions made, and on how to monitor and detect deteriorations in accuracies in an online manner. In our efforts to improve the performance, we analyzed the incorrectly assigned issue reports. We realized that many of them have attachments with them, which are mostly screenshots, and such reports generally have short or insufficient descriptions for the problem. Based on these observations, we further carried out studies on how to ensure that we detect the missing information in the descriptions of issue reports automatically and how we can use the attached screenshots as an additional source of information, in order to improve the performance of the automation.

ÖZET

BÜYÜK ÖLÇEKLİ ENDÜSTRİYEL BAĞLAMLARDA OTOMATİK YAZILIM OLAY KAYDI YÖNETİMİ

ETHEM UTKU AKTAŞ

Bilgisayar Bilimi ve Mühendisliği DOKTORA TEZİ, HAZİRAN 2021

Tez Danışmanı: Doç. Dr. Cemal Yılmaz

Anahtar Kelimeler: olay kaydı triyajı, metin analizi, imaj analizi, açıklanabilir makine öğrenmesi, değişim noktası tespiti

Yazılım olay kaydı raporları, kullanıcıların bir yazılım ürününü kullanırken karşılaştıkları sorunları anlattıkları belgelerdir. Bu raporları doğrulama ve atama sürecine ise yazılım olay kaydı triyajı denir. Uygulamada, kayıtların triyajı, uzmanlar veya yazılım geliştiriciler tarafından manuel olarak gerçekleştirilir. Büyük ölçekli endüstriyel bağlamlarda yüzlerce yazılım ürünü mevcuttur ve bu ürünleri kullanırken yaşanan sorunlarla ilgili her gün yüzlerce olay kaydı açılmaktadır. Bu raporların triyajı büyük miktarda insan eforu gerektirmektedir ve kayıtların zamanında çözülememesi müşteri memnuniyetsizliğine neden olmaktadır. Bu tezde, veri madenciliği teknikleri kullanarak süreci otomatikleştirdik ve sistemi devreye alarak edindiğimiz tecrübeleri paylaştık. Otomasyonun pratik etkilerini gözlemlemek ve aktarmakla kalmadık, aynı zamanda kullanıcılarla vaka çalışmaları yürüttük. Ayrıca, yapılan tahminler için teknik olmayan, kullanıcılar tarafından kolaylıkla anlaşılabilecek açıklamaların otomatik olarak nasıl oluşturulacağına ve tahminlerin doğruluğundaki bozulmaların çevrimiçi bir şekilde nasıl tespit edileceğine dair yöntemler geliştirerek değerlendirdik. Sistem performansını iyileştirmek için hatalı atanan olay kayıtlarını inceledik. İlgili kayıtlara çoğunlukla ekran görüntüleri de eklendiğini ve sorunla ilgili kısa veya yetersiz açıklamalar girildiğini tespit ettik. Bu gözlemlere dayanarak, otomasyonun performansının artırılması amacıyla, açıklamalarda eksik bilgi olduğunu otomatik olarak nasıl tespit edebileceğimize ve ekli ekran görüntülerini de ek bir bilgi kaynağı olarak nasıl kullanabileceğimize yönelik çalışmalar yürüttük.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor Associate Professor Cemal Yılmaz. Without his guidance and persistent help this dissertation would not have been possible. He frequently reminded me to take a step back and look at the problems from a new perspective. Our weekly meetings were always thrilling for me, which I always looked forward to, frequently resulting with fresh and creative ideas. I am tremendously fortunate to have Prof. Berrin Yanıkoğlu, Assoc. Prof. Kemal Kılıç, Assoc. Professor Vahid Garousi and Asst. Professor Ayşe Tosun as committee members. It was compelling to write for such a committee. I would also like to thank Prof. Albert Levi, who guided me expertly through my computer science education. I would not be able to make it here without his belief in the work that I was to conduct.

This thesis came true thanks to the incentives provided by Softtech Inc. At Softtech, I was involved in the Machine Learning Research Group, where at times I worked with Ümit Ülkem Yıldırım, Mehmet Çağrı Çalpur and Aziz Göktepe. Our collaborative work to improve our skills in this domain made me move forward faster. Also many thanks to Ebru Çakmak and Cihad İnan with whom we worked on improving validity and quality assessment of issue report descriptions.

Finally, many many thanks to my wife Zehra Sule Aktaş, who accepted the fact that I was away many times and had to work more to complete this dissertation. Without her support and patience I would not be able to make it here. My lovely children, Deniz and Toprak, I love them so much, I am so proud to have them. My brother, Assoc. Prof. Abdullah Onur Aktaş in the field of Philosophy, I miss those days of youth we had together, he was the reason I thought I could make an academic achievement. My mother, Muhibe Aktaş, I always felt her love and her belief in me through my life which kept me strong at both hard and good times. Her enthusiasm for painting inspires me for more to achieve. My father, Erdoğan Aktaş, who we lost recently, I know that he is proud of me, I will always hear his voice and feel his love inside me.

The last period of this thesis coincided with the Covid-19 pandemic. It is hard for all of us to keep motivated in such a period and I would like to thank everyone who did not spare their support, including the names of whom I may have forgotten to write here.

To my mother Muhibe, father Erdogan, and brother Onur,
To my wife Sule, and children Deniz and Toprak
For their love and patience through this work and through life

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

When a software system produces an unexpected result or when an additional feature is requested from the system, a report is submitted to the vendor. These reports, which are often referred to as *issue reports*, *bug reports*, or *problem reports*, include all the information necessary for the vendor to resolve the reported issues. Once a report is received, the vendor carries out a number of tasks until the issue is resolved, including determining the validity of the reported issue and figuring out whether the same/similar issues have been reported in the past (Antoniol, Ayari, Di Penta, Khomh & Guéhéneuc, 2008; Bettenburg, Just, Schröter, Weiss, Premraj & Zimmermann, 2008; Bettenburg, Premraj, Zimmermann & Kim, 2008; Jalbert & Weimer, 2008; Lamkanfi, Demeyer, Giger & Goethals, 2010; Menzies & Marcus, 2008; Pandey, Sanyal, Hudait & Sen, 2017; Wang, Zhang, Xie, Anvik & Sun, 2008).

An integral part of this process is to assign the issue reports to the development teams or to the individual developers, who are responsible for resolving the reported issues. In the remainder of the thesis, we refer to this task as *issue report assignment* (or *issue assignment*, in short).

Issue assignment is important as incorrect assignments can increase the turnaround time for resolutions. This is because incorrectly-assigned issue reports would typically bounce back and forth between the development teams and/or individual developers until the correct assignee is located, i.e., *issue tossing*. Therefore, issue tossing can cause a great deal of wasted time (Bhattacharya, Neamtiu & Shelton, 2012; Jeong, Kim & Zimmermann, 2009).

In this work, we automate the process of issue assignment and share our experience gained by deploying the resulting system, called *IssueTAG*, at *Softtech*[1]. Softtech is the largest software company of Turkey owned by domestic capital. Being an ISO-9001-certified subsidiary of the largest private bank in Turkey, called *IsBank*[2], Softtech receives an average of 350 issue reports every day from the field. As most

[1] https://softtech.com.tr

[2] https://www.isbank.com.tr

of these reports concern the business-critical systems, they need to be handled with utmost importance and urgency.

Automated issue assignment is indeed not a new idea (Anvik, Hiew & Murphy, 2006; Bhattacharya et al., 2012; Dedík & Rossi, 2016; Helming, Arndt, Hodaie, Koegel & Narayan, 2010; Jonsson, Borg, Broman, Sandahl, Eldh & Runeson, 2016; Lee, Heo, Lee, Kim & Jeong, 2017; Lin, Shu, Yang, Hu & Wang, 2009; Murphy & Cubranic, 2004; Wang et al., 2008). All of the existing works, however, carry out the assignments in a retrospective and offline manner by simply treating the actual issue databases as historical data. We have, on the other hand, deployed IssueTAG in the aforementioned business-critical industrial setup, which not only presented us with an unprecedented opportunity to observe the practical effects of automated issue assignment in practice (Chapter 5), but also enabled us to define new problems and develop novel solution approaches (Chapters 6-9).

1.1 Research Questions

The research questions we address in this thesis are as follows:

- RQ1: What are the advantages and disadvantages of deploying an automated issue report assignment system in an industrial context? (Chapter 5)
- RQ2: How can the usability of such a system be further improved? (Chapters 6-7)
- RQ3: How can the effectiveness (i.e., assignment accuracy) of such a system be further improved? (Chapters 8-9)

To address RQ1, we first deployed IssueTAG at Softtech on Jan 12, 2018, which has been making all the issue assignments since then (Chapter 5). We then evaluated it by not only carrying out quantitative studies, measuring the reductions in both the manual effort and the turnaround time for resolving issues, but also qualitatively by conducting user studies (Chapter 5).

To address RQ2 (Chapters 6-7), we used two valuable observations we have made after the deployment of IssueTAG. One observation was that the stakeholders tended to demand some explanations as to why certain issue reports (especially, the incorrectly assigned ones) were assigned to them (Chapter 6). This was important for the

stakeholders as incorrect assignments were increasing their workloads, which could adversely affect their performance metrics. Another observation was that since the issue report database was evolving continuously, the assignment accuracy of the deployed system was needed to be monitored, such that the deteriorations could be detected in an online manner (Chapter 7). This was important as it allowed corrective actions, such as recalibrating the assignment models, to be taken in time.

Note that these observations are not necessarily specific to the industrial setup used in this thesis. That is, the same or similar observations can be made for other setups. Note further that the aforementioned features can improve the perceived usability of an automated issue assignment system by gaining the trust of the stakeholders. Therefore, we formulated 2 sub-research questions under RQ2:

- RQ2.1: How can the assignments made by the system be explained to even non-technical stakeholders? (Chapter 6)

- RQ2.2: How can the deteriorations in assignment accuracies be automatically detected? (Chapter 7)

To address RQ3, we used the insights we gained by carrying out in-depth analyses of the results obtained from the deployed system. One observation we made was that a common problem with the poorly written issue reports was that they often did not contain all the information that they were supposed to contain, such as steps to reproduce the reported issues, expected behavior, and observed behavior (Chapter 8). Therefore, ensuring that issue reports do not have any missing information can, not only help resolve the reported issues, but also help improve the assignment accuracy~(Chaparro, Lu, Zampetti, Moreno, Di Penta, Marcus, Bavota & Ng, 2017a; Song & Chaparro, 2020). Another observation we made was that issue reports with attachments, especially the ones, which use screenshots as attachments, tend to contain less information in their textual descriptions, compared to the ones without any attachments (Chapter 9). Therefore, we conjecture that using attachments (i.e., screenshots) as an additional source of information for issue assignment can help improve the assignment accuracies.

Note that, as was the case before, these observations are not necessarily specific to the industrial setup used in this thesis; the same or similar observations can be made for other setups. Therefore, we formulated 2 sub-research questions under RQ3:

- RQ3.1: How can the missing information in manually written reports be identified? (Chapter 8)

- RQ3.2: How can the attached screenshots in issue reports be used for further

improving the accuracy of the system? (Chapter 9)

1.2 Contributions

The contributions of this dissertation can be summarized as follows:

Automating Issue Report Assignment (Chapter 5). We have cast the problem of issue assignment as a classification problem and evaluated a number of well-known classifiers by using the actual issue reports submitted to Softtech, all of which are written in Turkish. We have then picked the best classifier, developed the IssueTAG system around it, and deployed the resulting IssueTAG system. Since its deployment on Jan 12, 2018, IssueTAG has made more than 250,000 assignments as of June, 2021 (i.e., the assignment of all the issue reports submitted since then).

As we have already discussed, automated issue assignment using data mining approaches is not a new idea~(Anvik et al., 2006; Bhattacharya et al., 2012; Dedík & Rossi, 2016; Helming et al., 2010; Jonsson et al., 2016; Lee et al., 2017; Lin et al., 2009; Murphy & Cubranic, 2004; Wang et al., 2008). On the other hand, to the best of our knowledge we are the first one to actually deploy such a system, which presented us not only with an unprecedented opportunity to observe the practical effects of automated issue assignment, but also with an opportunity to carry out user studies, which (to the best of our knowledge) have not been done before in this context.

We first observed that it is not just about deploying a data mining-based system for automated issue assignment, but also about designing/changing the assignment process around the system to get the most out of it. We, in particular, made simple, yet effective changes in the manual issue assignment process employed at Softtech (Chapter 5).

We then observed that the accuracy of the assignments does not have to be higher than that of manual assignments in order for the system to be useful. This is further validated by the user studies we carried out on actual stakeholders (Section 5.5). In a nutshell, although the daily assignment accuracy of IssueTAG was slightly lower than that of manual assignments (0.831 vs. 0.864), it reduced the manual effort required for the assignments by about 5 person-months per year and improved the turnaround time for resolving the reported issues by about 20% (Section 5.4).

Furthermore, about 79% of the stakeholders participated in our user study "agreed" or "strongly agreed" that the system was useful (Section 5.5).

We next observed that deploying a data mining-based approach for automated issue assignments, requires the development of additional functionalities, which are not necessarily foreseen before the deployment. We have, in particular, developed two additional functionalities (i.e., explaining assignments and detecting deteriorations in assignment accuracies), both of which, to the best of our knowledge, have not been evaluated before in the context of issue assignment.

Last but not least, we observed that stakeholders do not necessarily resist change. In particular, we did not receive any objection at all to the deployment of IssueTAG. We believe that this was because all the stakeholders believed that they would benefit from the new system and none of them felt threatened by it (Section 5.6). We, furthermore, observed that gradual transition helped stakeholders build confidence in IssueTAG, which, in turn, facilitated the acceptance of the system (Section 5.6).

Explaining Assignments (Chapter 6). One functionality we needed was to create human-readable, non-technical explanations for the assignments made by the system. This was indeed a need we came to realize when we received several phone calls from the stakeholders shortly after the deployment of IssueTAG, demanding explanations as to why certain issue reports (especially, the incorrectly assigned ones) were assigned to them.

Note that this was a novel problem we were able to identify only after the deployment of IssueTAG; hadn't we deployed the system, we could not have identified it as a problem of practical importance. As a matter of fact, this problem was not studied before in the context of automated issue assignment.

Note further that explaining assignments is not a trivial problem at all, especially when the underlying data mining models are not human readable. To this end, we have generated model-agnostic explanations by using LIME – an algorithm for explaining the predictions of a classification or regression model~(Ribeiro, Singh & Guestrin, 2016). LIME has been used for similar purposes in different domains before, including healthcare and genetics~(Azodi, Tang & Shiu, 2020; Nanayakkara, Fogarty, Tremeer, Ross, Richards, Bergmeir, Xu, Stub, Smith, Tacey & others, 2018; Reyes, Meier, Pereira, Silva, Dahlweid, Tengg-Kobligk, Summers & Wiest, 2020). We, however, used LIME for the first time to explain the reasons behind automated issue assignments.

We, furthermore, evaluated the proposed approach by conducting user studies using actual issue assignments made by IssueTAG. In a nutshell, the participants have

found 95% of the explanations, each of which was created for a distinct issue assignment, helpful in understanding the rationale behind the assignments.

Monitoring Deteriorations (Chapter 7). Another novel problem we identified after deploying IssueTAG was to determine the deteriorations in the accuracy of the assignments obtained from the system. This turns out to be an important task as detecting deteriorations can help take the corrective actions, such as recalibrating the underlying assignment models, in time.

Note that the issue database maintained by IssueTAG continuously evolves as new issue reports are submitted on a daily basis. Furthermore, the incorrect assignments made by the system are fixed manually by the stakeholders closing the corresponding issue reports. As a matter of fact, the stakeholders take this task quite seriously as the number of issue reports closed is one of their performance metrics. Consequently, at a given point in time, the assignment accuracy obtained by the system can automatically be calculated by using the all the issue reports that have been closed so far.

All these in mind, we have developed and empirically evaluated an approach, which uses PELT – an online change point detection approach, to monitor and detect the deteriorations in daily assignment accuracies. PELT has been used in many application domains, including financial time series and oceanographic data (Hocking, Schleiermacher, Janoueix-Lerosey, Delattre, Bach & Vert, 2013; Killick, Fearnhead & Eckley, 2012; Lavielle & Teyssière, 2007). In this work, we, on the other hand, use it (to the best of our knowledge) for the first time in the context of automated issue assignment to detect the deteriorations in the assignments made by a data mining model.

In the experiments, we have evaluated the proposed approach by using the actual assignment accuracies collected from the field as well as by using simulations where both sudden and gradual drops in accuracies were tested. For the former, the proposed approach identified some deteriorations that actually aligned with the important historical events. For the latter, the proposed approach detected the deteriorations in all the experimental setups before the mean accuracy dropped more than 5 (out of 100) points.

Detecting Missing Information in Issue Reports (Chapter 8). We have also developed and empirically evaluated an approach for determining missing information in manually created issue reports, so that the stakeholders, who create these reports, can be provided with an appropriate feedback to improve the quality of the submitted reports. In particular, we focused on three types of information, which

one expects to find in an issue report, namely steps to reproduce the reported issue (S2R), expected behavior (EB), and observed behavior (OB).

Automatic identification of missing information in issue reports is not a new idea. Indeed, Chaparro et al. (2017a) proposes an approach to achieve this for the issue reports written in English by using part of speech (POS) tagging. We, however, do this for the issue reports written in Turkish. Consequently, one important way our approach differs from the aforementioned one is that as Turkish is an agglutinative language, we, instead of using POS tagging, base our analysis on morphological analysis. In the experiments, the proposed approach correctly identified the OB, EB and S2R sentences in the actual issue reports submitted to Softtech with an F-measure of 0.82+.

Using Screenshot Attachments for Issue Assignment (Chapter 9). Last but not least, we have developed and empirically evaluated an approach to use screenshot attachments included in the issue reports as an additional source of information for further improving the assignment accuracy.

Interestingly enough, although 57.41% (23,564 out of 41,042) of all the issue reports we examined had at least one screenshot and we observed that the issue reports with screenshot attachments tend to contain less information in their descriptions (i.e., fewer words, compared to the ones without attachments), to the best of our knowledge, this type of attachments have not been leveraged for issue assignments before.

In the experiments conducted by using actual issue reports submitted to Softtech, the proposed approach increased the assignment accuracy of the issue reports with attachments by 2 points (i.e., from 0.84% to 0.86%).

1.3 Organization of the Dissertation

This thesis consists of the following chapters: Chapter 2 provides the background for this thesis; Chapter 3 presents the related work and open issues in the literature; Chapter 4 presents the large scale industrial case and describes the issue handling process employed in the industrial case; Chapter 5 presents how we deployed an automated issue report assignment system, and results and insights after deployment; Chapters 6 and 7 are related to improving the usefulness of automation, in

other words, how to explain the automated assignments and how to monitor the deteriorations in accuracies, presenting the approaches we developed and empirically evaluated; Chapters 8 and 9 are related to our efforts in improving the effectiveness of the deployed system, in other words how to detect the missing information in issue report descriptions and how to leverage the information conveyed in attached screenshots, together with the evaluations of the approaches we developed; Finally, Chapter 10 discusses the threats to validity and Chapter 11 concludes the thesis and presents the future work to be conducted.

2. BACKGROUND

In this chapter, we introduce an overview of issue report management and the basic techniques used in thesis on automated issue triage. We first describe the issue triaging process in software engineering domain. Then, we present the information retrieval and natural language processing techniques we used in automating the process. Two important concepts we discuss in deploying a machine learning based system in production are explainability of the predictions and monitoring the system's performance, so the related methods we used are described in this section. We further introduce a new source of information, attached screenshots, to improve automated issue triage, where we used Optical Character Recognition (OCR), image classification and multimodal machine learning techniques. We present the basic techniques and tools we used in introducing this new piece of information in our experiments.

Note that our aim is to present the related concepts such that an industry practitioner or a reader unfamiliar with them gets a basic understanding of the methods and tools used in this thesis. We refer the main books we followed in writing the chapter, for the interested reader who wants to get into more detail on the concepts: (Zeller, 2009) for issue report management; (Bishop, 2006) and (Alpaydin, 2010) for machine learning techniques; (Schütze, Manning & Raghavan, 2008) for information retrieval and NLP techniques; and (Goodfellow, Bengio, Courville & Bengio, 2016) for deep learning techniques.

2.1 Issue Report Management

As a software program is deployed in production, it has to work smoothly to meet the users' expectations. When their expectations are not met, such as an error occurs or an unexpected behaviour is observed, the user informs the vendor about

the problem. Detecting problems in production is the most expensive place to detect them, since they cause customer dissatisfaction. Software life cycle enforces them to be detected before deployment, such as when developing the program or when testing it (unit tests, functional tests, acceptance tests). However, not all such reports are actual problems or bugs. An issue report sent to the vendor may be a misunderstanding, an enhancement request or a question as well.

When the vendors receive the issue reports, after validation and finding the software team to solve them, they reproduce the problem, isolate the problem circumstances, and then locate and fix the defect. Finally, they deliver the fix to the user. In large scale industrial contexts with millions of customers, this process cannot be handled manually. Furthermore, the managers should be able to answer questions such as: "Which issue reports are currently open?", "Are they valid and do they have enough information to reproduce and fix?", "Which teams should fix them?" "Which ones are the most severe and need to be fixed first?", "Did they occur in the past as well?". In short, "How can we organize the large scale issue report management process?".

2.1.1 Issue Reports

To understand the details of a problem and reproduce it, and furthermore, to determine if the problem is fixed, an *issue report* is needed. The problem should be described in the report from the developer's perspective. The textual features of an issue report are *one-line summary* and *description*. Other important features are *attached files*, *product*, *version*, *platform*, *component* and *operating system*.

2.1.1.1 One-line summary

The essence of the problem is provided by the user with one-line summary. Some examples from the industrial case are: "Problem in updating the customer", "Error while reading a file on the pay screen", "... amount is incorrect- It is different from the amount received in ... file". One-line summary reflects on what the problem is and where it is. It should help the vendors on deciding the severity and thus, the urgency of the problem.

2.1.1.2 Description

The user writes the details of the problem in the description part. It is expected that the user is specific about the problem and provides only the necessary details. Since the vendor will first need to reproduce the problem, it should include the steps that the user went through. Next, the user should also report what s/he observed and what s/he was expecting to observe. The quality of the information provided in description is important both for the human and the automated issue triager.

Steps to Reproduce: It is a description of what was done before the problem was observed, in other words, the steps that the developer should go through to reproduce the problem. It is the most important part of the description feature (Bettenburg et al., 2008), since if the developer can not reproduce the problem, s/he may not be able to solve it at all. The reporter must be sure to include the crucial circumstances for the issue to be reproduced. Sometimes, the problems may be specific to the production environment such as a database crash. Simplifying the steps to reproduce for such cases so that the problem persists, for example including the necessary lines for an SQL statement, will help the developer more in fixing it.

Observed Behaviour: It is the behaviour of the software product that the users observe in contrast to their expectation, in other words, symptoms of the problem. An example from the industrial case is: "Customer contact information cannot be updated. The error in the attached screenshot is received."

Expected Behaviour: It describes what should have happened according to the user. Mostly, the expected behaviour is not described explicitly in issue reports, since the users think that it is implied in the observed behaviour such that they expect the error should not have occurred at all.

2.1.1.3 Attached Files

The users attach many types of files to issue reports as an easy way to demonstrate the observed behaviour and to help the developer better understand the issue. The files may be text documents, spread sheets, image files, etc. The users may or may not include attached files, also it is possible to include more than one file. The most frequent attachments are image files, which we refer as *screenshots*. Mostly, screenshots are taken at the time when the error occurs.

2.1.1.4 Other features

The software *product* and its *version* information can help the developer to create a local copy of the program in the version as used by the user. As problems may occur because of the interactions between *components* or because of the *platforms* or *operating systems* used, such information may be useful for the developer to reproduce the problem as well.

The *severity* describes the impact of the problem while the *priority* is the means for triagers to define what should be done first. For a software product to be shipped, it is expected that all the severe problems that are determined in testing phase are fixed.

Users, developers or triagers can also add *comments* to issue reports, to ask questions to the users to get more information about the issue, to ask for attached files, to add initial findings, to discuss on possible causes or how the problem should be fixed, etc.

2.1.2 Issue Report Management Process in Industrial Contexts

A basic flow of issue report management process is depicted in Figure 2.1. The process is business critical in large scale industrial contexts and depending on the organizations' needs, the flow may have some variations. Organizations rely on this daily problem solving procedure to be able to successfully troubleshoot customer problems. These issues can be so critical that in cases such as crashes or data loss, they need to be resolved immediately.

Mostly there exist customer support teams in large scale organizations and the problems that the customers face in production are reported to these teams. They are non-technical staff and their main responsibility is to deal with the incoming issue reports and resolve them using basic troubleshooting guides. If they cannot resolve the incoming problems with these guides, they are responsible to dispatch the issues to the related entity in the organization. The customer support team is referred to as Level 1 employees, indicating the level of technical competency they have.

On the software development side, there is usually an application support team whose main responsibility is to reduce the workload of software engineers by resolv-

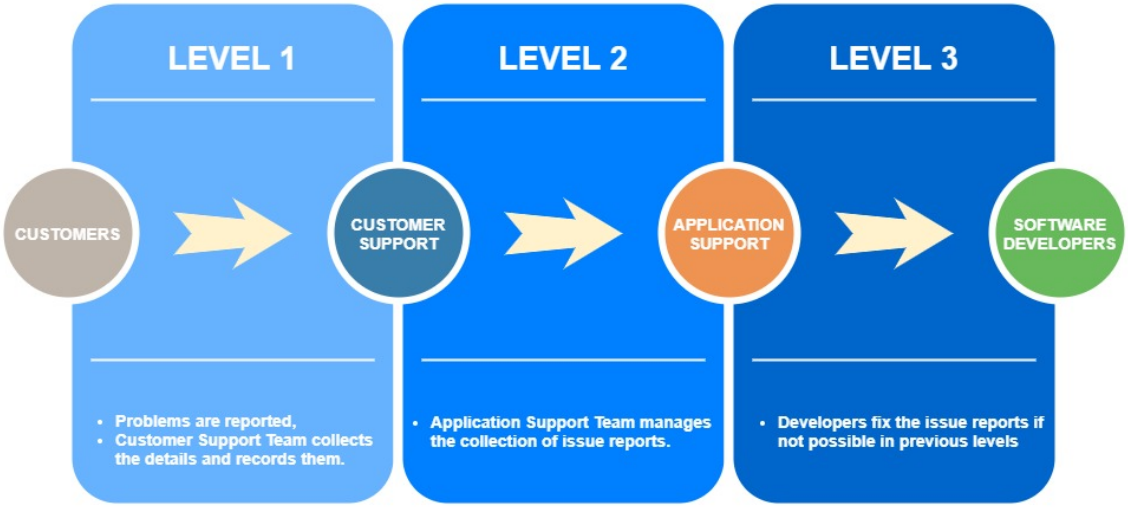


Figure 2.1 Sample issue report management process in large scale industrial contexts

ing the issues that do not require code changes. They are referred to as Level 2 employees, that is, in terms of technical competency, they are somewhere between Level 1 customer support team and Level 3 software engineers. They first check if the report is in the correct place, in other words, assigned to the correct team. If the decide that it is related to another team, they reassign it. When they decide that it is in the correct place, they check the validity of the report and check if it has enough information in its description to solve it. If needed they may ask for more information, such as screenshots. If the report is valid and has enough information to resolve it, they try to solve it, by using basic troubleshooting guides. If the application support team can resolve the reported issue, they close the report on behalf of the software team. Otherwise, they notify the development team about the newly reported issue by, for example, assigning it to a software engineer in the team or by creating a task for the team and linking it to the issue report.

At the third level, when the issue report cannot be solved with basic troubleshooting guides in previous levels, the software developers analyse the reports, find the related piece of code causing the error and fix it. The changed software is now ready to be delivered to the customer. Passing through the delivery procedure of the software company, such as testing, packaging and version management, the new code is delivered to the customer.

The process is business critical in large industrial contexts. Failure to solve the problems on time is costly and results in customer dissatisfaction. Since hundreds of such reports are received daily in these contexts, there may exist tens to hundreds of people employed working in the issue report resolution process. Any removal of manual work in this flow will help the triagers and decrease costs.

2.2 Natural Language Processing

Our main input in automating issue triage is textual data. In this section, we introduce the basic natural language processing techniques used in this thesis.

2.2.1 Preprocessing

We have a set of issue reports with textual data and each word in the document set has to be processed before any further representation, such as tokenization, removal of stop words and normalization. In this section we describe the preprocessing steps for the input text.

In language processing, *tokenization* is the task of chopping a string of input characters up into pieces that are called tokens. At the same time, some characters such as punctuations may also be thrown. To tokenize an input string of words, it looks trivial to chop on white space. Furthermore, we want to do the exact same tokenization of documents and input queries by processing with the same tokenizer to guarantee the matching of a sequence of characters in a document with the same sequence typed in a query.

Some words in the corpus may be of little value for the task that we want to carry out and we choose to exclude them from the vocabulary entirely. These words are called *stop words*. An apparent strategy for determining a stop word list is to count each word in the corpus, sort them according to their frequency and manually select starting from the most frequent considering the relevance to the domain. Then, after tokenization, the words in the stop word list will be discarded while processing the text. Using a stop word list will reduce the number of words that the system has to store and has little harm in its performance.

There exist many cases that two character sequences are not the same but you want them to match. Token *normalization* is the process of standardizing the tokens so that the tokens match despite the differences in character sequences. The standard way of normalization is to create and use equivalence classes named after one member of the set. A common form of normalization is to convert all the characters in the vocabulary to lower case. Thus, the use of both “Credit” and “credit” in different documents will match. Also, in languages like Turkish, there exist some letters used

to distinguish different sounds. Such letters are: “u” and “ü”, “o” and “ö”, “ı” and “i”, “c” and “ç”, “s” and “ş”, “g” and “ğ”. For reasons like speed, laziness or limited software, some users may have entered these letters in their corresponding forms. Thus, while preprocessing, these letters may be transformed in a standard way so that different written forms of such words are matched, for example “İş” and “Is”.

2.2.1.1 Stemming and Lemmatization

The textual data used in this work is written in Turkish, which is an agglutinative language, meaning that new words can be formed by simply adding suffixes to the root words, and sometimes you can even express a whole sentence with only one word. Thus, we need analysers specific to the Turkish language in this work to better process the input data.

We observe different forms of the words in documents such as “attach”, “attached”, “attaching”. In such situations, it may be appropriate to further normalize these words by converting their different uses to a common base form, which may be done by either *stemming* or *lemmatization*. Stemming is a heuristic approach where the ends of the words are chopped off in the hope of finding the root whereas lemmatization aims to find the root of the word properly using a vocabulary and morphological analysis of words. Lemmatization aims to return the basic form of the words known as lemma. This procedure may as well be harmful in cases such as, you actually want to differentiate between the different uses of words whose roots are the same. An example is while the root of “operational”, “operating” and “operative” are the same, which is “operate”, using their root form may result in loss of precision (Schütze et al., 2008).

2.2.1.1.1 Morphological Analysis Morphological analysis decomposes a word into its smallest possible functional units, its root and suffixes. It is an important concept for many NLP applications, such as spell checking, parsing, machine translation, dictionary tools and similar applications (Oflazer, 1994). Decomposing words into its smaller units is important especially for agglutinative languages such as Turkish or Finnish. In order to better understand the complexity in these languages, some one word examples with their possible roots and suffixes separated are provided below. Note that these examples are selected from the relevant banking domain, where software errors are described.

- Bul-un-a-ma-dı. [Verb + Passive + Ability + Negative + Past Tense + Third

Person Singular] : It could not be found.

- Çalış-tır-a-ma-dı-k. [Verb + Causative + Ability + Negative + Past Tense + First Person Plural] : We could not run it.
- Ek-te-ki [Noun + Locative + Adjective] : Attached

For the first example, the word has "*bul (find)*" as the root verb, adding the suffix "*-un*" makes it a passive verb, the suffix "*-a*" gives the meaning of ability, the suffix "*ma*" gives the meaning of negativity and finally the suffix "*-dı*" adds the past tense with third person singular. For the next example, the word has "*çalış (run)*" as its verb root, adding the suffix "*-tır*" makes it a causative verb (as in "*make it run*"), the suffix "*-a*" gives the meaning of ability, the suffix "*-ma*" gives the meaning of negativity, the suffix "*-dı*" adds the past tense and finally "*-k*" adds the first person plural to the meaning. In our work, morphological analysis of verbs is important since there exist patterns that we observe to be commonly used in issue report descriptions as provided in the first two examples. However, there are cases where the root of nouns or adjectives express some patterns as well. The final example is a case for that, where the word has "*ek* (attachment)" as its noun root, adding the suffix "*-te*" makes it a locative marked noun phrase and "*-ki*" suffix makes the word an adjective.

Morphological analysis is a complex task in Turkish and other agglutinative languages, and it is also a necessity for applying text based analysis in these languages. A well known two level morphological analyzer for Turkish was developed by Oflazer (Oflazer, 1994). In this work, Oflazer describes the rules and finite state machines for Turkish morphology. Based on this initial work, morphological analyzers were developed for Turkish such as TRmorph (Cöltekin, 2010) or for Turkic languages such as Zemberek (Akın & Akın, 2007).

2.2.2 Bag of Words Representation

After preprocessing the issue reports, we obtain a set of words representing each document, where some of these words may be occurring more than once. Logically, a word that appears more in a document should be more representative of that document. Thus, we assign a weight to each term in the document depending on the number of occurrences. The simplest way to achieve this is to count the number of occurrences for each term t in document d. This weighting scheme is called as *term frequency* and denoted by $tf_{t,d}$.

So, each term t in document d is represented by a number $tf_{t,d}$, which is simply the number of words in that document. This way of representation each document in the corpus is called *Bag of Words* representation in the literature, where the order of the terms in the document is ignored and only the number of their occurrences is taken into account. Thus, a sentence in an issue report description *"File A is downloaded faster than file B"* is identical to *"File B is downloaded faster than file A"*. Still, it makes sense to represent documents with similar set of terms similarly.

A document with n occurrences of a specific term is more relevant than a document with 1 occurrence of the term, but according to the term frequency weighting it is n times relevant, which is a bit overestimation. An alternative is the *log frequency weight* of the term t in document d calculated with the following formula:

$$w_{t,d} = \begin{cases} 1 + log(tf_{t,d}) & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Thus, a more reasonable weighting is achieved by taking the logarithm of the term frequency.

Term frequency weighting acts as if the words in the document are of equal importance, are they really? In the previous section, we removed the stop words, which shows that their importance is not the same at all. For example, in the context of software issue reports, words like "error", "fault" or "incorrect" may appear in many of the documents. So, it makes sense to introduce a mechanism that lowers the effect of the words that appear too often in the collection.

For this purpose, the idea of *document frequency*, df_t is introduced, which is the number of documents in the collection that contains the term t. A question that comes into mind is "Why do not we use the number of times the term appears in the collection instead?", which can be defined as *collection frequency* or cf_t. In fact, we use df_t since our purpose is to discriminate the effect of the word between documents, so we need a document-level statistic rather than a collection-level statistic. In other words, the collection frequencies of two terms may be similar, however one might appear in few documents while the other may appear in too many documents.

So, how do we involve the document frequency in trying to decrease the effect of a term for a document if it occurs in many documents in the whole corpus? We define the *inverse document frequency (idf$_t$)* for this purpose, where N in the following

formula is the total number of documents in the collection:

$$idf_t = log(\frac{N}{df_t}), \tag{2.2}$$

To calculate a combined weight for each term in each document, we combine term frequency and inverse document frequency given by the formula:

$$\text{tf-idf}_{t,d} = tf_{t,d} \times idf_t. \tag{2.3}$$

In summary, $tf - idf_{t,d}$ is the weight assigned to the term t in document d, the value of which increases as it appears more times in a document or appears in fewer documents.

2.2.3 Vector Space Model

We may view each document, or issue report in our corpus as a *vector*, where each term in the corpus is a component of the vector, the value of which can be found by 2.3 (or a similar weighting method). This way of representing each document in a corpus is known as the *vector space model* and is the first step to a number of tasks in information retrieval such as similarity scoring, document classification and document clustering.

In vector space model, a document in the corpus can be thought of as a point in the multi-dimensional space, with axes related to the terms. Thus, a document d, is defined as a vector $\vec{V}(d)$ and in case of $tf - idf$ weighting scheme, its value is calculated as:

$$\vec{V}(d)_t = (1 + log(tf_{t,d})) \times log(\frac{N}{df_{t,d}}) \tag{2.4}$$

The vector space model has some advantages over the boolean model, where the retrieval is based on whether or not the documents contain the terms. Both are simple models based on linear algebra since each document is represented as a vector.

On the other hand, with the vector space model representation, the term weights are continuous, which allows continuous similarity scores between documents and ranking accordingly. On the other hand, both models look for an exact match between the terms and does not take into account semantic relevance between the terms. Also, they assume that the terms are statistically independent and the order of the terms is lost, which are the main limitations.

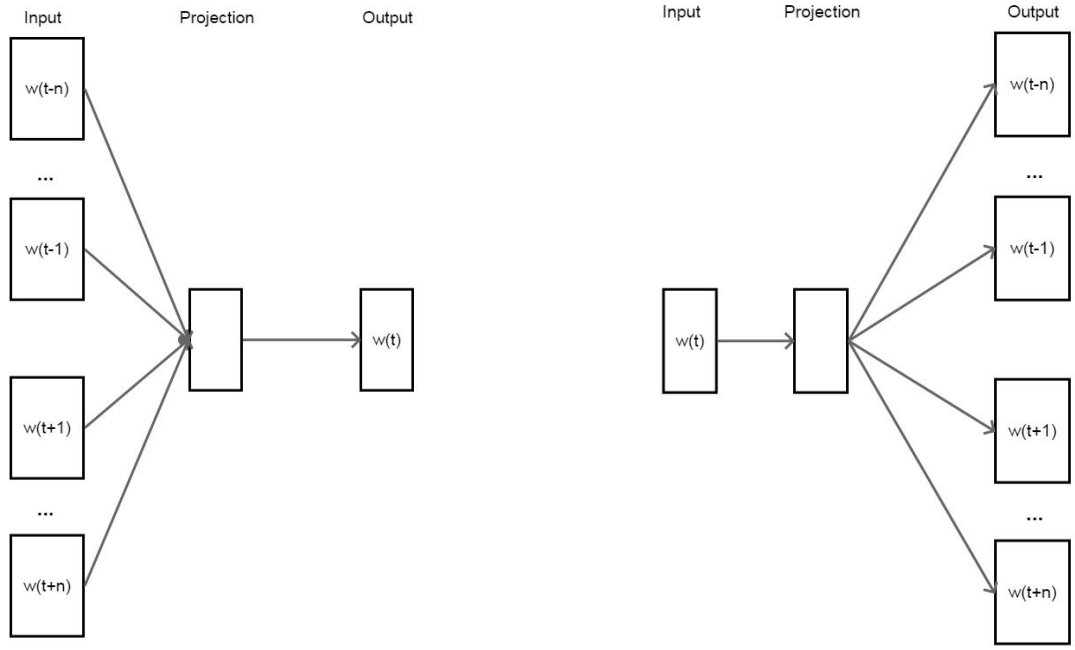
2.2.4 Word Embeddings

In the bag of words model, the context of the words are not taken into account. It treats each word independently, and it does not allow an algorithm to find out that two words may be similar in the contexts that they are used.

According to the distributional hypothesis, proposed by Harris in 1954 (Harris, 1954), words that appear in a similar context tend to have a similar meaning. The distributional hypothesis gave rise to distributional semantic models (DSMs), where words are represented as n-dimensional vectors of real numbers. In this way, words that appear in similar contexts have similar vector representations.

Word embeddings are a class of techniques where each word is represented as a real-valued vector in a predefined vector space (for example, a 300-dimension vector space), and words with similar meanings have similar representations. The most popular of these techniques are Continuous Bag of Words model (CBOW) and Skip-gram model (Mikolov, Chen, Corrado & Dean, 2013; Mikolov, Sutskever, Chen, Corrado & Dean, 2013), which are generally known as *Word2Vec* models. In CBOW model, current word is predicted using its context, whereas in Skip-gram model given the current word, the surrounding words are predicted. The architectures of these models are given in Figure 2.2. In the figure, $w(t)$ represents the word at position t, or the current position, and $w(t \mp i)(i = 1, 2, ..., n)$ represents the words positioned at $t \mp i$, or around the current word. The context window or the number of words before and after the current word are fixed in these models.

Two other techniques to mention for obtaining word embedding representations are *GloVe* (Pennington, Socher & Manning, 2014) and *fastText* (Bojanowski, Grave, Joulin & Mikolov, 2017; Joulin, Grave, Bojanowski & Mikolov, 2017). *GloVe* does not just rely on the local context information of words (as in *Word2Vec*), but it also incorporates global statistics information regarding the word co-occurrences to obtain the word vectors. *fastText* is another extension to *Word2Vec*. However, instead



(a) (b)

Figure 2.2 a) CBOW model architecture, where the training objective is to find word representations for predicting the current word based on the context. b) Skip-gram model architecture, where the training objective is to find word representations for predicting surrounding words.

of learning words directly, it represents each word as an n-gram of characters. It has an advantage such that while *Word2Vec* and *Glove* fail to provide representations for words not in the vocabulary, *fastText* works well with rare words as well.

2.3 Text Classification

Text classification is an active research field in natural language processing. It has many real world applications such as: Detection of sentiment in movie reviews or social media posts, classifying an email as spam or not, topic detection in news, such as politics, sports, economy, health, etc.

Many of these classification tasks may be solved manually. A librarian classifies the books according to their topics for example. However, it is expensive to classify textual documents manually when they grow large in number. The first solution that comes into mind to automatically classify text is to write rules. For example,

a set of keywords can be used indicating a class. However, finding out the keywords in a specific field comes with the necessity of finding someone specialized in that area, and if the domain is very dynamic, in other words, if new classes may emerge in the future, or new keywords can be added to the set, it will be cumbersome to maintain such a database.

Machine learning based text classification is an alternative, and many times a cheaper method compared to manual and rule based text classification. The rules that determine the class of the input document is automatically learned using the past data, in other words training data. However, we need a manual labelling of the past data this time, which may still be preferred to maintaining a knowledge base of rules. In fact, labelling is an easier task than rule based classification, because any one can understand if a news document is about sports for example. Most of the time, you do not even need to label, because labelling may already be a part of an existing process. For example, the reviewer may already be assigning scores for a movie, while at the same time writing comments. In our industial case, the issue reports are solved by specific teams or software developers and we already know the team/developer who solved the software problem described.

In this chapter, we first define the text classification problem formally and then describe the must-know methods for this thesis used in classifying input textual data.

2.3.1 Definition of the Text Classification Problem

Consider a document space $\mathcal{D}$, and a fixed set of classes $\mathcal{C} = \{c_1, c_2, ...c_n\}$. We are also given a training set $\mathcal{T}$ of labeled documents $(d, c) \in \mathcal{D} \times \mathcal{C}$. An example of such a document in our domain is $(d, c) =$ *("I get the attached error message while downloading the file", TeamA)*, where the related issue report is labeled as, or has been resolved by TeamA.

Our aim in classifying the documents is to learn a classifier f using a learning algorithm, which maps documents to classes:

$$f : \mathcal{D} \to \mathcal{C} \tag{2.5}$$

We call this type of learning as *supervised learning*, because we already have the labels for the training documents, which is, in a way, supervising us to learn the classier function. The learning method is denoted as ϕ, which takes the training set $\mathcal{T}$ as input and returns the classification function f:

$$\phi(\mathcal{T}) = f \tag{2.6}$$

We use the same names for learning methods and classifiers most of the time. For example, when we say "Support Vector Machines is very powerful in classifying text", we talk about the learning method, and when we say "In our experiments, Support Vector Machines achieved 80% accuracy", we talk about the classifier.

When we learn the function f using the training data, we test it on an unknown dataset, which we call as *test set*. Our aim in classification is to achieve high accuracy on the test set. Achieving high accuracy on the training set does not guarantee a similar performance in production. We also assume that training and test sets are similar, or coming from the same distribution.

2.3.2 Naive Bayes

Naive Bayes is a probabilistic learning method, so in this section, we first provide a basic probability theory and then, we introduce text classification with Naive Bayes.

Variable E represents an event from a space of possible events. However, we do not know for sure if an event is true in the real world, so we talk about the probability of the event E, $0 \leq P(E) \leq 1$. We describe $P(E,T)$ as the joint probability of both events E and T occurring, and $P(E|T)$ as the conditional probability of event E given that event T occurred. The relationship between joint and conditional probabilities is provided below:

$$P(E,T) = P(E \cap T) = P(E|T)P(T) = P(T|E)P(E) \tag{2.7}$$

Equation 2.7 states that the probability of a joint event equals the probability of one of the events multiplied by the probability of the other event given that the first

event occurs.

The complement of event E is written as $P(E^{\complement})$ and similarly we have:

$$P(E^{\complement},T) = P(T|E^{\complement})P(E^{\complement}) \tag{2.8}$$

The *partition rule* in probability theory states that if the probability of event E can be divided into an exhaustive set of disjoint sub-events, then $P(E)$ is the sum of the probabilities of these sub-events. A special case of this rule is given below:

$$P(E) = P(T|E) + P(T^{\complement}|E) \tag{2.9}$$

Using the above equations, we derive the *Bayes' Rule*:

$$P(E|T) = \frac{P(T|E)P(E)}{P(T)} = \Big[\frac{P(T|E)}{P(T|E)P(E) + P(T|E^{\complement})P(E^{\complement})}\Big]P(E) \tag{2.10}$$

In the above equation, $P(E)$ is the *prior probability* of E, or an initial estimate of the likelihood of E when we do not have any other information and $P(T|E)$ is the conditional probability or likelihood of T given that E holds. With Bayes' rule, we derive a *posterior probability* $P(E|T)$, based on the likelihood of T occurring in the two cases of E occurring or not.

Now, in supervised text classification with the multinomial Naive Bayes model, we aim to find the probability of an input document belonging to a class given the input terms that the document has. Since it is probable that the document belongs to more than one classes, we choose the class with the highest probability. We can calculate the probability of a document d belonging to a specific class c in the following way:

$$P(c|d) \propto P(c) \prod_{1 \leq i \leq n_d} P(t_i|c) \tag{2.11}$$

$P(c)$ in the above equation is the prior probability of a document classified as c and $P(t_i|c)$ is the probability of term i occurring in a document classified as c. t_i are the

terms or tokens in document d and n_d is the number of such tokens.

Our goal is to find the most probable class for a document, so the best class is the *maximum a posteriori (map)* class:

$$c_{map} = \arg\max_{c \in \mathcal{C}} \hat{P}(c|d) = \arg\max_{c \in \mathcal{C}} \hat{P}(c) \prod_{1 \leq i \leq n_d} \hat{P}(t_i|c) \tag{2.12}$$

Multiplying many conditional probabilities may result in underflow, which is, not being able to represent the last digits in floating point representation. Thus, many applications implement logarithms of probabilities, because multiplication is eliminated since $log(xy) = log(x) + log(y)$. Then, the equation turns into:

$$c_{map} = \arg\max_{c \in \mathcal{C}} [log\hat{P}(c) + \sum_{1 \leq i \leq n_d} log\hat{P}(t_i|c)] \tag{2.13}$$

$Log\hat{P}(c)$ is the weight indicating the relative frequency of c, and $log\hat{P}(t_i|c)$ terms are the weights that indicate how much the terms t_i are related to the class c. So, their sum is a measure of how much evidence we have for the document belonging to the class c.

2.3.3 Support Vector Machines

Support Vector Machines (SVM) is a vector space based machine learning method that performs at the state-of-the-art levels in text classification. The goal in SVM is to find a decision boundary between two classes that maximizes its distance to any point in the training data. In this section, we first introduce SVM for the two-class linearly-separable datasets, and then extend the model for non-separable data and multi-class problems.

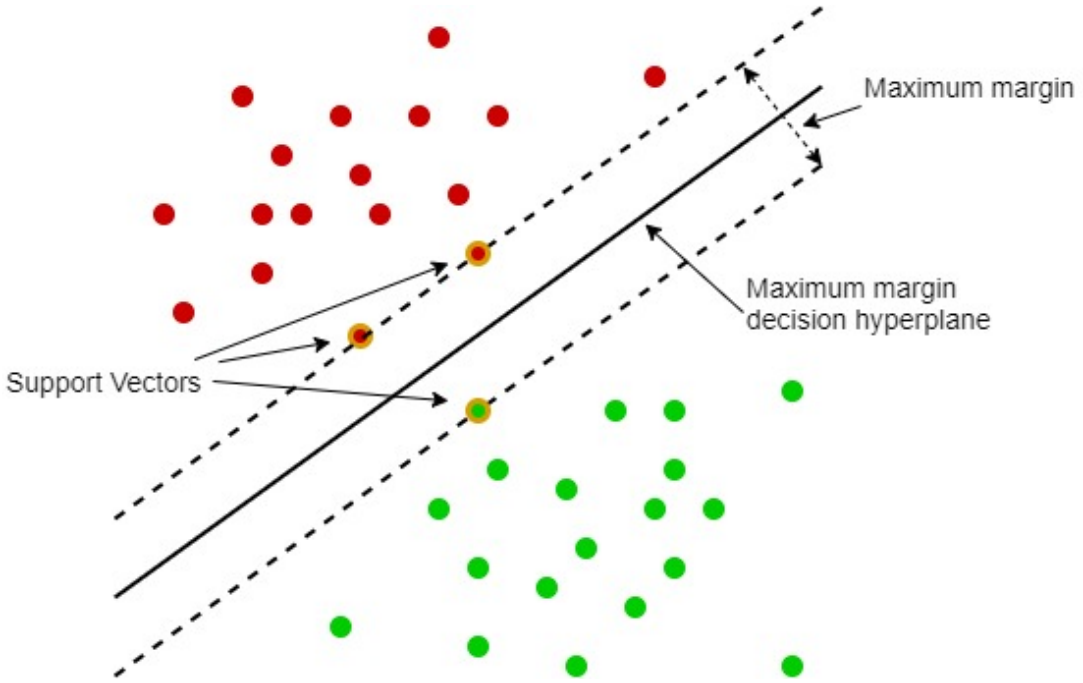


Figure 2.3 The margin and support vectors for a sample problem.

2.3.3.1 Linearly separable case

Figure 2.3 is an example of a two-class, linearly separable dataset. In fact, there exist lots of possible linear separators for the two classes of data. The criterion for SVM to look for a decision surface is it must be maximally far away from any data point. The distance between the decision surface and the nearest data points is called the *margin* of the classifier. Hence, SVM needs only a small subset of data points in order to decide on the separating hyperplane. These data points are called *support vectors*. Other data points have no part in deciding on the decision surface.

Any hyperplane can be written mathematically as:

$$w_1x_1 + w_2x_2 + ... + w_nx_n + b = 0 \tag{2.14}$$

All points on the hyperplane satisfy the above equation, thus all points on the hyperplane satisfy $\vec{w}^T\vec{x} + b = 0$. If for a data point $\vec{x_i}$, the result is greater than zero, then the point is on one side of the hyperplane, and if it is less than zero, then it is on the other side. For SVMs, we always define the two class labels, or y_i for a data

point x_i as +1 or -1. Then, the linear classifier is:

$$f(\vec{x}) = sign(\vec{w}^T\vec{x} + b) \tag{2.15}$$

where a value of -1 indicates one class, while +1 indicates another class. The *functional margin* of the i^{th} example in the dataset with respect to the hyperplane is defined as: $y_i(\vec{w}^T\vec{x_i} + b)$. However, we need to scale this distance in terms of the units of the weight vector, $|w|$, since without a constraint we can always make the functional margin as big as we want. Thus, the functional margin is normalized by the magnitude of the weight vector w, to get the *geometric margin* of a training example. As a result, the distance from example $\vec{x}$ to the separator is denoted as r and can be calculated as:

$$r = y\frac{\vec{w}^T\vec{x} + b}{|\vec{w}|} \tag{2.16}$$

We can scale the functional margin as we please and for convenience in solving large SVMs, we can require that the functional margin is at least 1 for all the data vectors: $y_i(\vec{w_T}\vec{x_i} + b) \geq 1$. For the support vectors the relation is an equality, so, the geometric margin is $\rho = 2/|\vec{w}|$. Maximizing the geometric margin is the same as minimizing $|\vec{w}|$, thus the standard formulation of the SVM as a minimization problem is:

- minimize $\frac{1}{2}\vec{w}^T\vec{w}$
- subject to $y_i(\vec{w}^T\vec{x_i} + b) \geq 1$, for all $(\vec{x_i}, y_i)$

By adding extra variables that are called *Lagrange multipliers*, where each multiplier α_i is associated with each constraint $y_i(\vec{w}^T\vec{x_i} + b) \geq 1$ in the primal problem, we construct the *dual problem*:

- maximize $\sum \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i\alpha_j y_i y_j \vec{x_i^T}\vec{x_j}$
- $\sum_i \alpha_i y_i = 0$
- $\alpha_i \geq 0$ for all $1 \geq i \geq N$

The solution to the dual problem is provided below:

$$\vec{w} = \sum \alpha_i y_i \vec{x_i} \tag{2.17}$$

$$b = y_k - \vec{w^T} \vec{x_k} \quad \forall \vec{x_k} : \alpha_k \neq 0 \tag{2.18}$$

Note that going from the primal form to dual form is out of the scope of this thesis. Each non-zero α_i in equation 2.17 indicates that the corresponsing x_i is a support vector. Thus, the classification function then becomes:

$$f(\vec{x}) = sign(\sum_i \alpha_i y_i \vec{x_i^T} \vec{x} + b) \tag{2.19}$$

So, we can make decisions on which class each new example x belongs to, by comparing x with only the support vectors x_i.

2.3.3.2 Soft Margin Classification

In real life, most of the time the classes are not linearly separable. Even if they are, we might want to ignore some of the noisy cases so as to have a better solution separating the bulk of the data. We do this by paying a cost for each of the misclassified example depending on how far it is from meeting the requirement in equation $y_i(\vec{w_T}\vec{x_i} + b) \geq 1$. We introduce slack variables z_i to implement this. A non-zero z_i will let x_i not meet the margin requirement at a cost proportional to the value of z_i. Introducing the slack variables, the SVM optimization problem then becomes:

- minimize $\frac{1}{2}\vec{w}^T\vec{w} + C\sum_i z_i$
- subject to $y_i(\vec{w}^T\vec{x_i} + b) \geq 1 - z_i$, for all $(\vec{x_i}, y_i)$

The optimization problem is now a trade off between how fat the margin can be, and how many data points allow this margin and the C parameter is the *regularization term* or the *penalty*. When it is too high, the penalty is high for non-separable data points, and when it is low, we may have underfitting.

The dual problem is the same as the separable case except for the regularization parameter C bounding the size of the Lagrange multipliers for the support vector data points: $0 \leq \alpha_i \leq C$ and the solution to the dual problem is provided below:

$$\vec{w} = \sum \alpha_i y_i \vec{x_i} \tag{2.20}$$

$$b = y_k - (1 - z_i) - \vec{w^T} \vec{x_k} \quad for \quad k = \arg\max_k(\alpha_k) \tag{2.21}$$

2.3.3.3 Non-linear SVMs

What if the training data does not let us classify with a linear classifier? Such a case is depicted in Figure 2.4. The top dataset cannot be classified with a linear classifier, however, when it is mapped to a higher dimensional space (the bottom dataset) using a quadratic function, it can be classified with a linear classifier. We call this mapping to a higher dimension space as the *kernel trick*. SVM relies on the dot product of data point vectors, $K(\vec{x_i}, \vec{x_j}) = \vec{x_i^T} \vec{x_j^T}$. So, the SVM classifier is:

$$f(\vec{x}) = sign(\sum_i \alpha_i y_i K(\vec{x_i}, \vec{x_j}) + b) \tag{2.22}$$

Suppose that every data point is transformed into a higher dimensional space via: $\phi : \vec{x} \rightarrow \phi(\vec{x})$, then the dot product becomes: $\phi(\vec{x_i})^T \phi(\vec{x_j})$. If this dot product could easily be computed in terms of the original data points, we could simply use the function's value in Equation 2.22, which we call the *kernel function*.

Two commonly used kernel functions are polynomial kernels and radial basis functions. Polynomial kernels are of the form: $K(\vec{x}, \vec{y}) = (1 + \vec{x}^T \vec{y})^d$. The case where $d = 1$ is a linear kernel, and $d = 2$ is a quadratic kernel, which are very common. The most common form of a radial basis function is a Gaussian distribution:

$$K(\vec{x}, \vec{y}) = e^{\frac{-(\vec{x} - \vec{y})^2}{2\sigma^2}} \tag{2.23}$$

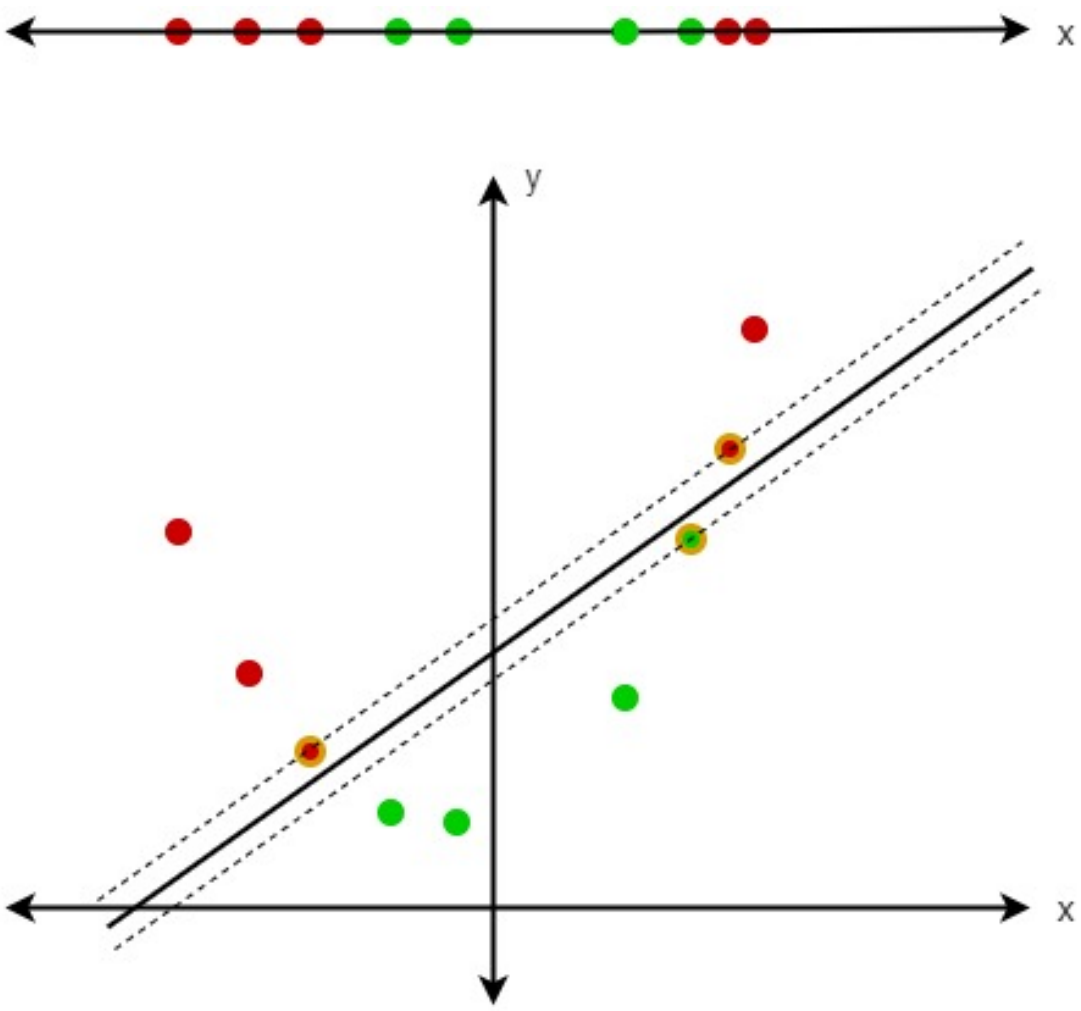


Figure 2.4 Data projected to a higher dimensional space to make it linearly separable.

In general, SVMs are robust and high performing text classifiers (Joachims, 1998). They generalize well in high dimensional spaces such as text, and a linear kernel is recommended since textual data is often linearly separable.

2.3.4 Convolutional Neural Networks

Neural networks are a set of algorithms that are composed of a large number of interconnected processing elements called neurons working together to solve a problem. In artificial neural networks (ANN), each neuron receives some inputs, computes the weighted sum of them, transforms the result into an output with the use of an activation function, and finally passes the result to other neurons. The last layer in an ANN produces the output of the system.

A convolutional neural network (CNN) is a special type of neural network, which is characterized by a mathematical operation called *convolution* in at least one of the layers. It was first applied for two dimensional image data, but it can also be applied on one dimensional data, as in the case of text represented with word vectors. CNNs involve the multiplication of a set of weights with an array of input data, similar to a traditional neural network. Promising results were obtained in classification of textual data with CNNs (Kim, 2014).

For the textual input case, let $w_i \in R^k$ be the k-dimensional word vector corresponding to the i^{th} word in the input text. Also let $w_{i:i+j}$ represent the concatenation of the consecutive words from i to $i+j$. The set of weights is another array, called *filter* or *kernel*. Let $f \in R^{hk}$ be the filter where h is the window size, or number of words in the window.

A *dot product* is applied between the filter and a filter sized part of the input array repeatedly, by sliding one step each time inside the input array. Note that, *dot product* is the element-wise multiplication of two arrays. An example of such filtering is depicted in Figure 2.5.

The components of the resulting array is then summed giving a single value. In other words, we obtain a feature t_i from a window of words $w_{i:i+h-1}$ by:

$$t_i = \phi(f \cdot w_{i:i+h-1} + b) \tag{2.24}$$

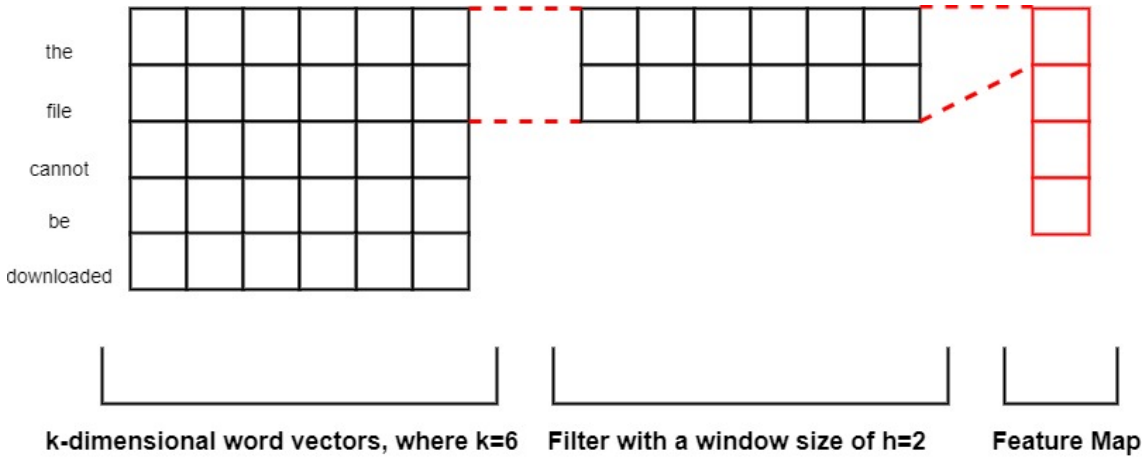


Figure 2.5 An example filter applied on word embedding representation of input text.

where b is a bias term and ϕ is a non-linear function such as ReLU, or rectified linear activation function, which outputs zero if the input is negative, otherwise it outputs the input itself. We apply this filter to each possible window of words in the sentence with n words and produce the *feature map*:

$$t = t_1, t_2, ..., t_{n-h+1} \tag{2.25}$$

Max pooling may be applied over the feature map, which is simply taking the maximum value $\hat{t} = max(t)$, so as to capture the most important feature.

Note that we have described applying one filter here, but it is possible to apply more than one filter on the input text to obtain multiple features. Also, we have described a *single-channel* architecture, but we may build two or more channel architectures, where there may be more than one input of word vectors processed as described here, and the results are added to obtain the feature values t_i.

At the end of convolutional neural networks, there exist fully connected layers where the output is the probability distribution over classes, as in ANNs.

2.3.5 Recurrent Neural Networks

Another type of neural network that uses sequences of inputs like text is recurrent neural networks (RNN). The basic idea is to add memory and feedback to the net-

work, which helps better model the sequence data. The word vector representation is fed to the network and at each step it performs the same computation for every element in the sequence using the output from the previous computation as well. At each step, RNN computes:

$$h(t) = \phi(u \times h(t-1) + w \times x_t) \tag{2.26}$$

where u and w are weights and ϕ is any non-linear function such as ReLU. $h(t)$ is the output while x_t is the input at time t. While RNNs can model sequence of data, its main drawback is in accessing information from a long time ago, such as a word from several steps back. Also, *vanishing gradient problem* is encountered with RNNs, which is, the gradient is vanishingly small, preventing the weight from changing its value and making the network hard to train (Pascanu, Mikolov & Bengio, 2013).

2.3.5.1 Long Short Term Memory Network

Long Short Term Memory (LSTM) network is a modified version of RNN, which is easier to train and captures the dependencies back in time. In an LSTM network, a *cell* and three gates are present at each module: *Input gate, forget gate and output gate*. The key component of the LSTM is the cell and it remembers values over arbitrary time intervals. Each of the three gates regulate the flow of information and have the same equation form as in 2.26, while each has a different set of parameters and the function ϕ is the sigmoid function. The input gate discovers which value from input to use to update the memory, while the forget gate discovers what information to discard. The output gate uses the input and the memory of the unit to decide on the output.

A forward LSTM takes the input sequence of words as it is, while a backward LSTM takes the input in the reverse order. We use backward LSTMs to capture the dependencies of a word on future words. A bi-directional LSTM is the concatenation of forward and backward LSTMs.

2.4 Explaining Predictions

Understanding the rationale behind the predictions made by a machine learning based system is important, since users of the system will take actions based on these predictions. Even if a wrong classification is made, people will trust the system more if they understand why such a prediction is made. In rule based automated systems, it is easier to explain why a software product behaves in a certain way. Even if the documentation is not well enough, you can check the code to interpret its behaviour. As machine learning systems get complex, explaining predictions becomes harder. While algorithms such as Naive Bayesian are easier to understand and explain, algorithms such as Support Vector Machines are more complex and it's harder to explain their predictions.

Most of the time, models are evaluated based on the accuracy achieved on the validation dataset. However, real life data is often different than the training and test data, and furthermore accuracy achieved is not the only metric indicating the product's success. If people do not understand the rationale behind the predictions, they may not use the software product in real life.

Riberio et al. (Ribeiro et al., 2016) differentiate between two types of trust: Trusting a prediction and trusting a model. They define "trusting a prediction" as whether users trust individual predictions to take the actions based on it and "trusting a model" as whether users trust the model to behave in reasonable ways. To solve the problem of trust for individual predictions, they developed a tool called LIME (Local Interpretable Model-Agnostic Explanations) that can explain the predictions of a classifier. We propose to use their approach in explaining the predictions made by the issue report triaging system and describe the approach in this section.

LIME is a novel model-agnostic algorithm that explains the predictions of a classification or regression model (Ribeiro et al., 2016). In a nutshell, it takes an issue report, the assignment made for this report, and the underlying classification model as input, represents the report as a bag of words, and samples instances around the report by drawing subsets of the words in the bag uniformly at random. Then, the samples are weighted by their proximities to the original issue report and fed to the classification model to label them. Next, all the samples together with their associated labels are used to learn a linear model comprised of K terms, which distinguishes the labels. Finally, the linear model learnt is reported as an explanation for the assignment.

2.5 Change Point Detection

One of the problems to consider with an automated machine learning based system is how often to retrain the model. Especially in an industrial setting, changes may occur in the organization, some new applications may be deployed and some may be retired. So, training a model for issue triaging once and using it for a long time will not be possible in a large industrial setting. If we can calculate the daily accuracies of predictions made, we can then follow the performance of the system, in other words if it deteriorates or not, and decide when to retrain. The approach we propose to detect deteriorations in the predictions made, is a statistical analysis technique called *change point detection*, which aims to identify when the underlying model of a signal changes (Truong, Oudre & Vayatis, 2020). We propose to calculate the accuracies of correctly classified issue reports at the end of each day and use the resulting time series data to detect if there is any change in its mean.

Binary Segmentation (Scott & Knott, 1974) is one of the most popular methods for change point detection. However, it is an approximate search method and its computational cost is *O(nlogn)*. Exact search methods exist as well, such as *Optimal Partitioning* (Jackson, Scargle, Barnes, Arabhi, Alt, Gioumousis, Gwin, Sangtrakulcharoen, Tan & Tsai, 2005) which is based on dynamic programming. However, its computational cost is $O(n^2)$. We propose to use *Pruned Exact Linear Time* (PELT) method (Killick et al., 2012) to detect the changes in the mean of prediction accuracy. It is an exact search method based on *Optimal Partitioning* but involves a pruning step within the dynamic program and under mild conditions its computational cost is linear in the number of data points.

2.6 Image Processing and Classification

One-line summary and description attributes of issue reports may lack enough information for the developers to understand and fix the problem. Users may prefer to attach the screenshots of the errors they receive instead of writing a detailed explanation of what happened. For better triaging such reports, the image attachments should also be considered together with other textual data. In this section, we briefly describe the techniques we used for extracting the useful information in

these screenshot images.

2.6.1 Optical Character Recognition

The most useful information in an attached screenshot is the textual data written there, although the layout of the screen may also be relevant if the software products have distinguishable image properties. The textual data to be extracted from an image in our case may be the error message received, the screen name, or the names of the fields on the screen. *Optical Character Recognition (OCR)* is the conversion of such images of typed, handwritten or printed text into machine-encoded text. It is an active research field related to computer vision.

OCR engines have been developed for various tasks such as extracting the textual data in business documents, car plate recognition, traffic sign recognition, textual versions of printed documents, etc. In our experiments, we used a well-known OCR technology, Tesseract (Smith, 2007). It is one of the most popular open source OCR engines. Very briefly, Tesseract operates as follows (Smith, 2007): It first analyses and stores the outlines in the images and gathers them together as Blobs. Blobs are then organized into text lines and text lines are broken into words. Next, a first pass of text recognition process attempts to recognize each word and words that are satisfactory are passed to an adaptive trainer. A second pass attempts to recognize the words that were not recognized satisfactorily in the first pass. Fuzzy spaces are resolved and text is checked for small caps. Finally, digital text is provided as output.

Extracting the textual data in the screenshots, we used them as another textual source of information for the classifying algorithms in our experiments.

2.6.2 Very Deep Convolutional Networks

In our industrial case, the software products use same GUI infrastructures. For example, the screens that the branches use have similar layouts whether it is a credit card application, personal loan application or customer information management. So, we do not expect that the image features are relevant for issue triage in our case, but to find out if these features have any contribution in improving automated

issue triage, we have also experimented on classifying the screenshots attached to the issue reports.

Convolutional neural networks (CNN) described previously in Section 2.3.4 were first developed for image classification. Instead of a one dimensional input as in the textual input case, they get a two dimensional input from an image and process it similarly. CNNs are comprised of two simple components: Convolutional layers and pooling layers. However, there may be many ways of arranging these layers for a specific image classification task. Also, it may take weeks to train these models on large image datasets. A solution to this problem is to use the model weights trained on benchmark datasets. This process is called *transfer learning* and has the benefit of decreasing the training time.

The image classification model we use in our experiments is VGG, which was named after the lab which developed it, *Visual Graphics Group* at Oxford (Simonyan & Zisserman, 2015). The architecture got the first and the second places in the localisation and classification tasks respectively in the ImageNet Challenge in 2014. Two variants of the model are mostly used, which are VGG-16 and VGG-19. They are named after the number of layers they have in their architecture. The VGG-16 model that we used in our experiments, includes five convolutional blocks having a total of thirteen convolutional layers, followed by three fully connected layers. The model is used to train 1,000 classes originally and in order to adapt for our case, we do not include the final fully connected layers in our experiments, but we add a new output layer specific to our classification task.

VGG model is heavily used for computer vision tasks with transfer learning as a starting point. The key aspects of the VGG model are: It uses very small convolutional filters, stacks convolutional layers before using a pooling layer which together define a block, repeats the convolution and pooling layers, uses very deep (16 or 19 layer) models.

2.6.3 Multimodal Learning

For the issue reports with attached screenshots, if we want to use textual modalities, one-line summary and description, together with the visual modalities, attached screenshots, one of the new techniques to combine these features is *Multimodal Machine Learning*, which is a new and exciting research area. In real life, before we make decisions, we consider multiple sources of input, such as sound, im-

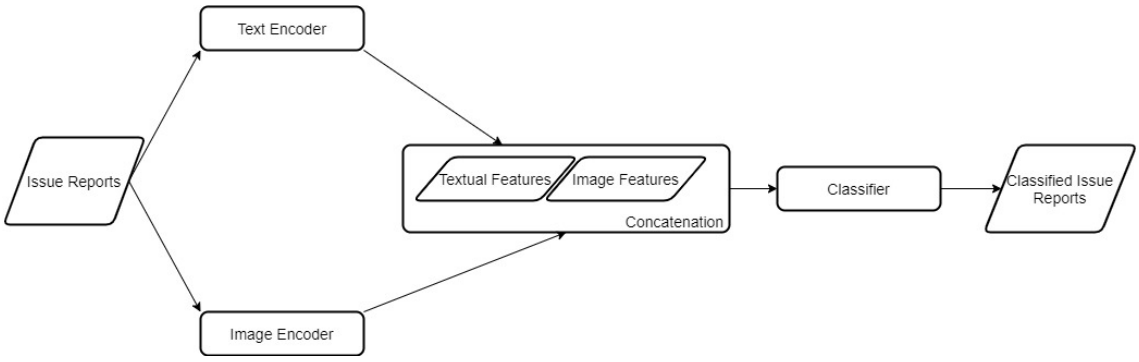


Figure 2.6 The multimodal model architecture used in the experiments.

age, text, smell, etc. Although we are sceptical about the possible contribution of image features in automated issue triage for our case, we considered experimenting with *multimodal framework (MMF)* tool provided by Facebook AI (Singh, Goswami, Natarajan, Jiang, Chen, Shah, Rohrbach, Batra & Parikh, 2020). We used the basic architecture provided by the tool which is depicted in Figure 2.6.

Briefly, the text encoder uses the fastText embeddings (Bojanowski et al., 2017; Joulin et al., 2017) to transform the input textual data, one-line summary and description, and the image encoder uses the ResNet image recognition model (He, Zhang, Ren & Sun, 2016) to transform the input images. The resulting features are concatenated and treated as a new feature vector. Finally, a fully connected layer is used for classification.

2.7 Evaluation Metrics

For comparing classification algorithms, we used Accuracy, Precision, Recall and F-measure as the evaluation metrics. Accuracy (A) is defined as the number of correctly classified issue reports divided by the total number of issue reports in the test set. So, given a matrix of number of cases C_{xy}, where the first index x represents the actual class that the issue report belongs to and the second index y represents the predicted class, then accuracy (A) is defined as:

$$A = \frac{\sum_i C_{ii}}{\sum_i \sum_j C_{ij}} \tag{2.27}$$

For a specific class x, precision (P_x) is the correctly predicted issue reports divided by all of the issue reports predicted as that class, recall (R_x) is the correctly classified issue reports divided by all of the issue reports actually belonging to that class, and F-measure (F_x) is the harmonic mean of precision and recall. Weighted F-measure (F) is reported in this thesis.

$$P_j = \frac{C_{jj}}{\sum_i C_{ij}} \tag{2.28}$$

$$R_j = \frac{C_{jj}}{\sum_i C_{ji}} \tag{2.29}$$

$$F_j = \frac{2 \times P_j \times R_j}{P_j + R_j} \tag{2.30}$$

3. RELATED WORK

Automated issue assignment is indeed not a new idea. Many approaches have been proposed and empirically evaluated by using both open- and closed-source projects~(Anvik et al., 2006; Bhattacharya et al., 2012; Dedík & Rossi, 2016,1; Helming et al., 2010; Jonsson et al., 2016,1; Lin et al., 2009; Murphy & Cubranic, 2004; Wang et al., 2008). All of the existing works, however, carry out the assignments in a retrospective and offline manner by simply treating the actual issue databases as historical data. We, on the other hand, deployed the proposed approach and shared both the results we obtained and the lessons we learnt regarding the practical effects of automated issue assignment in the field.

These works use a variety of approaches to make the assignments, including Naive Bayes classifiers (Anvik et al., 2006; Murphy & Cubranic, 2004), Bayesian Networks (Jeong et al., 2009), Support Vector Machines (Anvik et al., 2006; Jonsson et al., 2016), and information retrieval-based approaches (Canfora & Cerulo, 2006; Chen, Wang & Liu, 2011; Kagdi, Gethers, Poshyvanyk & Hammad, 2012; Linares-Vásquez, Hossen, Dang, Kagdi, Gethers & Poshyvanyk, 2012; Nagwani & Verma, 2012; Shokripour, Kasirun, Zamani & Anvik, 2012; Xia, Lo, Wang & Zhou, 2013; Xie, Zhang, Yang & Wang, 2012), Expectation Maximization (Anvik, 2007), Nearest Neighbor classifiers (Anvik & Murphy, 2011), Decision Trees (Ahsan, Ferzund & Wotawa, 2009), Random Forests (Ahsan et al., 2009), REPTrees (Ahsan et al., 2009), Radial Basis Function Networks (Ahsan et al., 2009), Neural Networks (Helming et al., 2010) and Ensemble-based classification (Jonsson et al., 2016).

Some of the aforementioned works use natural language explanations present in issue reports for assignments, such as one-line summary and description (Ahsan et al., 2009; Alenezi, Magel & Banitaan, 2013; Anvik et al., 2006; Anvik & Murphy, 2011; Baysal, Godfrey & Cohen, 2009; Bettenburg et al., 2008; Bhattacharya et al., 2012; Canfora & Cerulo, 2006; Chen et al., 2011; Helming et al., 2010; Jeong et al., 2009; Jonsson et al., 2016; Lin et al., 2009; Linares-Vásquez et al., 2012; Matter, Kuhn & Nierstrasz, 2009; Murphy & Cubranic, 2004; Nagwani & Verma, 2012; Park, Lee, Kim, Hwang & Kim, 2011). Others also leverage categorical information, such as

product, component, and version (Ahsan et al., 2009; Jonsson et al., 2016; Lin et al., 2009; Park et al., 2011).

Different sources of information have also been used. For example, Tamrawi, Nguyen, Al-Kofahi & Nguyen (2011) model the technical expertise of individual developers and use these models together with the information about the developers who recently made changes in the code base. Wu, Zhang, Yang & Wang (2011) infer a social network model of the developers using the comments they make on historical issue reports as well as the comments automatically generated at the time of the source code commits, to help with the assignments. Baysal et al. (2009) use developers' preferences as an additional source of information, which are expressed by the ratings the developers gave for the issues they resolved. Using the skill profiles of the developers as input, Karim, Ruhe, Rahman, Garousi & Zimmermann (2016) worked on the problem of assigning tasks to the developers for bug fixing.

In this thesis, our main source of information was natural language descriptions present in the issue reports, more specifically the one-line summaries and descriptions. We did not use any categorical information, e.g., product, component, and version information, because such information was not included in the issue reports; there were no fields in the issue reporting tool, requesting these types of categorical information. The reason was that with the collection of software products maintained by Softtech, which heavily interact with each other in a business-critical environment, sharing many resources, such as databases, file systems, and GUI screens, the boundaries of the products from the perspective of issue reporting were not clear at all.

3.1 Issue Report Assignment

Various work has been conducted on issue report assignment. All of the work presented in this section have been carried out on past data in a retrospective manner, and none of them discuss the results and insights after deployment, different from our work.

Murphy & Cubranic (2004) reported that issue triage takes an increasing amount of time in large open source projects. They proposed to categorize textual input with machine learning techniques for the assignment of the reports to the individual developers. They selected 15,859 reports from the Eclipse project and achieved 0.3

accuracy with a Naive Bayes classifier.

Anvik et al. (2006) proposed a semi-automated issue report assignment approach, where the triager selects from the recommended set of developers to whom the issue will be assigned. For recommending the developers, they proposed to use a supervised machine learning algorithm. For comparing and tuning the algorithms, they used data from two of the Bugzilla repositories, namely Eclipse and Firefox. They selected 8,655 and 9,752 reports for each of them respectively. They used one line summary and the full text description as input for the machine learning algorithms. The algorithms they selected are Naive Bayes, Support Vector Machines (SVM) and C4.5. Naive Bayes is a probabilistic algorithm which provides a baseline in their work, SVM is selected for its success in text classification and C4.5 is a popular decision tree algorithm. They obtained a top-1 precision of more than 0.5 for the Eclipse project and 0.64 for the Firefox project. To validate the results, they further evaluated their approach with the gcc project with 2,629 reports for training and 194 reports for testing. However, they obtained a top-1 precision of 0.06, which they attribute to the characteristics of the project such as one developer dominating the report resolution process.

Bettenburg et al. (2008) showed that automated triaging can be improved by adding the information in issue duplicates. By filtering the developers who have solved at least 50 issue reports, they obtained a *master* data set of 8,623 reports, and an *extended* data set of 23,990 reports consisting of both master reports and their corresponding duplicates from the Eclipse project. By taking into account the creation dates of the reports, they distributed them into 10 equal size folds for the master data set. For the extended data set, they added the duplicates to the folds where their master is in, so they did not preserve the order of the creation of the data sets. For each fold they used the previous folds' reports for training and tested on the related fold. They used title and description as inputs and used tf-idf for vector representation, then used SVM and Naive Bayes algorithms for prediction. They showed that as more data is used in each fold, the prediction accuracy increases, and as duplicates are added to the training set, better performance is obtained on the same test set. In this thesis, instead of dividing the data set into folds with equal sizes, we obtain the folds by taking into account the creation month of the issue reports. We also do not remove the duplicates from the data set, however the corresponding duplicates may not be in the same fold as the master issue report since we only take into account the creation date.

Ahsan et al. (2009) extracted 1,983 resolved issue reports from the Mozilla project to assign them to software developers using the input textual data. After pre-processing

the data, they used tf and $tf-idf$ methods for the representation of the data. They, furthermore, used feature reduction based on the term's global frequency threshold, and feature extraction based on latent semantic indexing (LSI). LSI is a document indexing method to produce low dimensional representation by taking into account the semantic relationship of the words. The machine learning algorithms they used are Decision Tree (DT), Naive Bayes (NB), RBF Network (RB), Random Forest (RF), REPTree (RT), Support Vector Machine (SVM) and Tree-J48 (J48). They obtained the best results with LSI technique and SVM algorithm with a 0.44 accuracy.

Incorrectly-assigned issue reports may bounce back and forth between the teams or developers until they are assigned to the correct team/developer. The reassignment of issue reports is called issue tossing. Jeong et al. (2009) introduced a graph based model based on Markov chains by capturing the issue tossing history. They first obtained top-n developer prediction list from a machine learning model and then identified a more appropriate top-n developer list using the tossing relationship. From a prediction list of P=$\{p_1, p_2, ..., p_n\}$, they created a second list RP=$\{p_1, t_1, p_2, t_2, ..., p_n, t_n\}$ where t_i is the developer with the strongest tossing relationship with p_i. The strongest tossing relationship was found by using the tossing probability of each edge in the graph. They selected the first n developers from the new list. Note that with this approach, when the number of predictions is 1, the list does not change. Since our system does not recommend a list but directly assigns the best prediction, their approach is not applicable to our case. To validate their model, they used the title and description attributes and obtained vector representations for the reports and then used Naive Bayes and Bayesian Networks as the training models. The data set they used is a benchmark set by Bettenburg et al. (2008). They used the first $165,385$ Eclipse issue reports for training and $165,397$ to $211,822$ for testing. Similarly, they selected the first $345,343$ Mozilla issue reports for training and $345,345$ to $429,903$ for testing. For each data set and algorithm, they compared the results with and without the use of tossing graphs for the top-2 to top-5 predictions (Note that their approach is also not applicable when the number of predictions is 1). The best accuracy they obtained with their approach is 0.77 for the top-5 predictions with the Eclipse data set.

Bhattacharya et al. (2012) improved the previous work by Bettenburg et al. (2008) and Jeong et al. (2009) by using product and component attributes together with summary and description to improve issue triage. More specifically, they improved the tossing graphs of Jeong et al. (2009) by adding developer expertise (product and component attributes of the issue reports they solved) to the graph edges and developer activity to the graph nodes, and they used incremental learning by increasing

the size of the training set in each iteration similar to the work of Bettenburg et al. (2008). Note that, in our work we do not have the product and component information at the time of assignment, since the boundaries of these products/components are not clear at all. For example, many GUI screens interact with the core banking system, which is maintained by a different set of development teams. Bhattacharya et al. (2012) used Mozilla data set such that the issues ranging from number 37 to 549,999 are covered and the Eclipse data set such that the issues ranging from number 1 to 306,296 are covered. The classifiers they selected are Naive Bayes, Bayesian Network, C4.5 and SVMs with polynomial and rbf kernel functions. They did not improve for the top-1 prediction as their approach needs a list of at least two predictions as in the work of Jeong et al. (2009). As stated before, in our work we only use the best prediction since we do not recommend a team to the human triagers, but instead we assign the reports without the intervention of the triager, so their approach is not applicable to our case. The best accuracy they obtained is 0.86 for the top-5 predictions with the Eclipse data set when they use all of the folds for training.

Karim et al. (2016) formulated the problem of bug assignment to developers as a multi-objective problem (minimization of fix time and cost), assigning issue reports to several developers at the same time. In this model, for each competency area for an issue report, a developer is assigned. They used source code files to extract the competency of developers, and the number of times they changed a file in each area is assumed to represent their competency in that area. To find the effort required for a new report, they designed a function that uses the average solution times for previously closed similar reports. They estimated the time to fix an issue report, using the estimations of the competency of developers in different areas and the effort needed in each area. They used a genetic algorithm that aims to minimize time and cost and maximize expertise. They evaluated the performance of the proposed approach for 2,040 reports from 19 open-source projects of the Eclipse platform and showed that they could reduce the time to fix a report by 39.7%.

One difference between our work and the aforementioned work is that we have an online system, which assigns the issue reports to the stakeholders right after an issue report is created by using the information that is available at the time of the report creation. The aforementioned work, on the hand, assumes the presence of a batch assignment process where the project manager decides which reports to fix at the beginning of a release. Furthermore, at Softtech, aligned with the general agile development practices, every developer within a development team is expected to solve almost all the issue reports received for a product that the team is responsible for. Therefore, the skill profiles of the developers within a team are

expected to be close to each other. Indeed, within a team, developers typically either pick the issue reports that they want to work on or they take turns to resolve the issues reported, rather than the issue reports are being assigned to the individual developers. Consequently, IssueTAG, rather than assigning reports to individual developers, assign them to the development teams, who are responsible for resolving these issues. More discussion on this can be found in Chapter~5. We, however, believe that skill profiles can also be used at the level of development teams to improve assignment accuracies by identifying the teams that are responsible for related, but different products/modules.

Lin et al. (2009) conducted a case study on a proprietary software project, called SoftPM, a tool for software process management. In their work they both used textual (summary and description) and nontextual information (such as the priorities and the submitters of the reports) and compared the results. For the textual input they used SVM models, while for the non-textual input they used decision tree models. They extracted 2,576 issue reports written in Chinese and achieved a better accuracy of 0.63 with textual features for their assignment to individual developers. Our work differs from this work since we assign the reports to development teams.

Helming et al. (2010) also assigned work items, such as issue reports and tasks, to individual software developers. They proposed and evaluated two approaches: a model-based approach and a data mining-based approach. The model based approach used simple statistics about the software developers handling work items linked to particular functional requirements. The data-mining based approach evaluated a number of classifiers, using tf-idf scores obtained from the textual information present in work items as input. UNICASE (a system for unified software engineering research tools), DOLLI (a system for facility management), and Kings Tale (a browser-based computer game) were the three small-scale software projects they used to evaluate their approach. The accuracy of the model-based approach was between 0.58 and 0.83, while the accuracy of the data mining based approach was between 0.29 and 0.43. They used a maximum of 1,191 work items in this study, which may be the reason for the low accuracy levels of the data-mining based approach.

Jonsson et al. (2016) operated in large scale industrial contexts; one in the automation domain and another in the telecom domain. For the automation domain, they used 15,113 reports assigned to 67 different software teams. For the telecom domain, they used a maximum number of 10,000 issue reports assigned to 64 different software teams. They used both textual and nominal features in this study, where the textual features are one-line summary and full description of the report

and the nominal features are affiliation of the submitter, site from where the report was submitted, revision of the product that the bug/issue was reported on and the priority. They obtained the best results with an ensemble technique called stacked generalization, with assignment accuracies between 0.50 and 0.89. We find this study similar to ours, since it was conducted in a large scale industrial context with many products and many teams, and the reports were assigned to the teams. Some of the design decisions in this thesis, i.e., assigning issue reports to development teams, rather than individual developers and determining the time locality of training data, are inspired from their work. They, however, carry out the assignments in a retrospective and offline manner by simply treating the actual issue databases as historical data.

Dedík & Rossi (2016) experimented both on a proprietary project in software technologies domain and an open source project, namely Mozilla Firefox. They used textual information present in the issue reports to obtain the tf-idf scores first. 2,424 issue reports with 35 developers for the proprietary project and 1,810 issue reports with 20 developers for the open source project, were used in this work. They reported that SVM models were quite effective in both cases. They achieved an accuracy of 0.53 for the proprietary project and 0.57 for the open source project.

Lee et al. (2017) proposed to use word embeddings and Convolutional Neural Networks to classify issue reports. In the proposed model, they used a two channel CNN, where one of the channels used summary as input while the other used description of the issue report as input. They conducted their experiments on four anonymized industrial projects and three open source projects, JDT, Platform and Firefox. For the industrial projects, they used 1,257, 6,256, 14,583 and 3,152 reports, while for JDT, Platform and Firefox they used 1,465, 4,825 and 13,667 reports, respectively. They regard the developers who fixed more than 10 reports as active and create another data set after filtering the reports that these developers solved. The number of different active developers to assign the reports are 18, 76, 67 and 12 for the industrial projects, and 15, 55 and 142 for the open source projects. They tested CNN on the data sets with and without active developers for top-1 to top-5 accuracies and compared the results with the performance of human triagers. When we consider the top-1 accuracies in their tests for the four industrial projects (which is more of our interest), the human triagers performance is 0.42, 0.26, 0.35 and 0.79, while CNN performs 0.73, 0.58, 0.86 and 0.73 in the same order on the active developer data set. Our work differs from theirs since we assign the reports to teams, while they assign to developers. We believe that what is missing in Lee et al. (2017)'s work is the comparison of a previously successful data mining model with the proposed model, although they noted that their preliminary experiments with simpler models

did not perform well against the CNN model. As a result, we do not know how a Linear SVC model applied on tf-idf vectors would perform in their case. As Fu & Menzies (2017) indicate, benefits of a deep learning model need to be assessed with respect to its computational cost. Such work should be baselined against a simpler and faster alternative, which is missing in Lee et al. (2017)'s work.

3.2 Issue Report Validation

Issues in the tracking system can be enhancements, re-factoring activities or organizational issues rather than bugs to be fixed (Antoniol et al., 2008). Some of them can be the result of missing information by the user who reported it. Detecting such issues using supervised machine learning techniques and possibly recommending the reporter for some actions to take, can be important in eliminating the possible issues and thus reducing the number of issue reports in the bug tracking system.

Antoniol et al. (2008) worked on classifying the issue reports into corrective maintenance and other kinds of activities, with the goal of building an automated classifier. For this purpose, they first extracted 1,800 reports from Mozilla, Eclipse and JBoss projects and classified them manually as bug or non-bug. As a result of manual classification, 0.45 of issue reports in Mozilla project, 0.32 in Eclipse and 0.58 in JBoss were reported to be bug, while the rest to be non-bug. The percentages of non-bug issue reports were very high indeed for each project. Using the relevant words to bug/non-bug classification, they used tf-idf vectorization and obtained accuracies between 0.77 and 0.82 using the Naive Bayes, ADTree and Logistic Regression classifiers.

Pandey et al. (2017) worked on a similar problem and classified the issue reports as bug or non-bug to develop a tool with the aim of automatically labelling them and thus reducing the cost of valuable developer time. They used a manually classified corpus of 5,587 selected issue reports from three open source projects of Apache, namely HttpClient, Lucene and Jackrabbit. 305 of 745 reports in HttpClient project, 697 of 2,441 reports in Lucene project and 937 of 2,401 reports in Jackrabbit project were classified as bug. They used the summary as input and converted each text into a document term matrix. The algorithms they used for classification were Naive Bayes, Linear Discriminant Analysis, K-Nearest Neighbors, Support Vector Machine (SVM) with various kernels, Decision Tree and Random Forest. Depending on the

project, Random Forest and SVM achieved the best results with an accuracy of 0.75 to 0.83.

Kallis, Di Sorbo, Canfora & Panichella (2019) introduced Ticket Tagger, a GitHub app to label issue reports created on any GitHub repository's issue tracker, as bug, enhancement or question. For this purpose, they used the subject (title) and description features as input, represented them as bag of n-grams (a set of sequences of n consecutive words) and classified them with *fastText* (Joulin et al., 2017), which is a multi-class linear neural model. They evaluated the tool on 30,000 issue reports extracted from 12,112 GitHub projects, distributed uniformly as bug, enhancement or question. The 10-fold cross validation F1-score results were 0.83, 0.82 and 0.83 for bugs, enhancements and questions respectively.

3.3 Issue Report Quality

Poorly written issue reports slow down developers because it takes more time and it is harder to identify the problem from such reports (Zimmermann, Premraj, Bettenburg, Just, Schroter & Weiss, 2010). Zimmermann et al. (2010) conducted a survey among developers and reporters of Apache, Eclipse and Mozilla projects. They identified the items developers use mostly, and find as most important when fixing bugs as well as the items reporters provide mostly, the most difficult ones to provide, and the most helpful that they consider. *Steps to reproduce* (S2R), *Observed behaviour* (OB), and *Expected Behaviour* (EB) stand out to be the most widely used by developers and mostly provided by reporters. However, the results also show that there is mismatch between what information developers consider as important and what reporters provide. Thus, from an information-centric point of view, they recommend to develop tools to provide feedback on the quality of an issue report and what information is missing. They further asked the survey participants to rate the quality of issue reports on a five-point Likert scale ranging from very poor to very good. Using the 289 rated issue reports, they developed classifiers using either binary features (such as presence of attachments) or continuous features (such as readability) as input, to predict whether an issue report is bad, neutral or good in its quality. They evaluated the classifiers within and cross projects. The best result in terms of accuracy within projects is 0.50 with the SVM regression model with a linear kernel for the Apache project, while the best result is 0.49 with SVM classification model with a linear kernel trained on Apache and tested on Eclipse

project.

Chaparro et al. (2017a) conjectured that a well-defined set of discourse patterns is used when describing observed behaviour , expected behaviour or steps to reproduce, thus automatic detection of these patterns is possible with high accuracy. To validate their hypothesis, they first manually analyzed 2,912 reports and found out that most reports contain OB (0.94), however only some of them explicitly describe EB and S2R (0.35 and 0.51, respectively). Out of these 2,912 issue reports, they further analyzed 1,091 reports at sentence or paragraph level and extracted 154 discourse patterns. Out of 154 discourse patterns, 90 correspond to OB, 31 correspond to EB and 33 correspond to S2R. Note that they worked on nine projects to extract these patterns and the number of patterns extracted varies from project to project. Facebook has the most number of discourse patterns (67 OB, 16 EB and 19 S2R patterns) and Wordpress-A has the least number of discourse patterns (44 OB, 11 EB and 14 S2R patterns).

Chaparro et al. (2017a) observed that the mostly occurring OB patterns are negative sentences with auxiliary verbs (such as, "*are not, can not, did not, does not, etc.*"), sentences with verb phrases having error related nouns (such as, "*The GUI gives the following error:*"), and sentences with non-auxiliary negative verbs (such as "*hang on*"). For the EB pattern, the most observed one is sentences with modal terms "*shall, should*". And for the S2R pattern, the most common pattern is listing the actions performed either as items in paragraphs or in single sentences. In a subset of 1,821 reports, they evaluated their approaches to detect the missing information. The machine learning based approach trained on discourse pattern features performs best in detecting missing EB with a *within-project* F1-score of 0.89, and the machine learning based approach trained on discourse patterns and n-grams (to capture the vocabulary used in descriptions) performs best in detecting missing S2R with a *within-project* F1-score of 0.75. Note that in their work, they first eliminate non-bug (NB) issue reports such as questions, enhancements, feature requests or operational tasks.

Song & Chaparro (2020) developed a tool called BEE, that analyzes the titles and descriptions of GitHub issue reports. BEE first checks if the issue describes a bug (as opposed to a question or an enhancement), and then labels the sentences as OB, EB and/or S2R. It is developed as a GitHub app and can be installed in any repository to provide feedback on the incoming issue reports to improve their quality. The first analysis of the BEE tool is to classify the report as bug or not, which uses Ticket Tagger (Kallis et al., 2019) described in the previous section. Then, if the report is a bug, BEE labels each sentence of the report as OB, EB or S2R, if they

convey the related information. To evaluate the tool's performance in identifying the OB, EB and S2R, they work on 5,067 issue reports compiled from their earlier work (Chaparro, Florez & Marcus, 2017,1; Chaparro, Florez, Singh & Marcus, 2019; Chaparro et al., 2017a) with 116,084 manually analyzed sentences.

Song & Chaparro (2020) implement three binary SVMs in their work, meaning that they classify each sentence three times as OB or not, EB or not, and S2R or not. Thus, each sentence can have more than one label. Not all of the 116,084 manually analyzed sentences contain the OB, EB or S2R information. 0.12 of these sentences describe OB, 0.02 describe EB and 0.06 describe S2R, while 82% describe other information. So, the input data is highly imbalanced. To overcome the imbalanced data problem, they tune the paramater j of SVMs, which balances the cost factors for incorrect predictions and and use oversampling for sentences containing OB, EB and S2R with the SMOTE tool (Chawla, Bowyer, Hall & Kegelmeyer, 2002). Using 10-fold cross validation, they evaluate BEE in terms of detecting OB, EB and S2R in sentences, and in terms of detecting the missing elements in entire issue reports. They achieve a recall performance of 0.88, 0.98 and 0.91 in detecting OB, EB and S2R respectively in sentences. The recall performance of detecting missing information is 0.34, 0.89 and 0.7 for OB, EB and S2R respectively. Although the performance of detecting missing OB is low, only 0.02 of the reports lack this information. Thus, the effect of this misclassification is anticipated as negligible. Given these results, the tool is considered useful to improve issue report quality, while a few cases can be expected with false alerts.

While BEE is a useful tool for most of the repositories in GitHub where the reports are written in English, its performance could be improved by considering other languages as well, especially for agglutinative languages, where the patterns should be detected with morphological analysis.

3.4 Other Problems Related to Issue Triaging

An avenue for future research is to carry out industrial-strength studies to evaluate the efficiency and effectiveness of the other related approaches in the field, including duplicate detection, severity identification, and effort prediction.

One type of approach aims to identify duplicate issue reports, which can help developers with 1) figuring out the number of actual issues reported; 2) assigning

priorities; and 3) debugging (Podgurski, Leon, Francis, Masri, Minch, Jiayang Sun & Bin Wang, 2003). Generally speaking the problem of duplicate identification is cast to a clustering problem where similar reports are grouped together with the assumption that similar descriptions report the same (or similar) issues (Bettenburg et al., 2008; Jalbert & Weimer, 2008; Podgurski et al., 2003; Wang et al., 2008).

Since more information can be gathered through these duplicates, they are not considered as harmful for the issue resolution process (Bettenburg et al., 2008). Runeson, Alexandersson & Nyholm (2007) used NLP techniques and similarity measures to detect duplicate reports in their case study. They first used standart tokenization, stemming and stop word removal for the textual data from these reports, and used a thesaurus search for finding the synonyms and for correcting the spelling errors. Then, they represented the words in a vector space model by using term-frequencies, and found the duplicate ones with cosine, jaccard and dice similarity measures. In their work, they also took into account the time frame the reports are submitted to narrow their search. They showed that 2/3 of the duplicates could be found with the proposed method. They also reported that two defects stemming from the same fault may use different vocabulary. In such cases, it would be relevant to ask if the prediction performance of a duplicate detection system would increase by taking into account the attached screenshots and the error messages in these files.

In mission critical systems, it is important to correctly determine the severity level of an issue report (Lamkanfi et al., 2010; Menzies & Marcus, 2008), where the attached screenshots would be another use case to improve prediction. Menzies & Marcus (2008) developed a tool called SEVERIS to predict the severity level of issues that the test engineers encounter while software testing at NASA. They first processed the textual data gathered from the project and issue tracking system (PITS), applied tf-idf to represent the words and used InfoGain measure for dimensionality reduction. Then, they used a rule learner in SEVERIS to generate the rules to predict the severity of an issue report, in the form $if..then...$ Applying their approach on five anonymous PITS projects, they reported the F-measures on each severity level in these projects. Even in the case with a few number of tokens as input, they obtained high F-measure performance.

The effort needed to solve the issue reports is another problem to be dealt with, for better project management (Giger, Pinzger & Gall, 2010; Weiss, Premraj, Zimmermann & Zeller, 2007; Zhang, Gong & Versteeg, 2013). Weiss et al. (2007) used text similarity and nearest neighbour approaches to search for similar reports and used their average time as a prediction for the fix time of the current issue report. They indicated the use of attachments to further enhance their approach as future work.

3.5 Open Issues/Gaps in the Literature

One gap in the literature is that none of the existing works have so far deployed an automated issue assignment system. Therefore, the practical effects of such a system, including its effectiveness in reducing the manual effort and improving the turnaround time for issue resolutions, have not been studied before. We, in this work, deploy IssueTAG in a business-critical industrial context, which not only provides us an unprecedented opportunity to observe the practical effects of an issue assignment system, but also helps us identify some novel problems (Chapter~5).

One novel problem we have identified after deploying IssueTAG was the need for explaining the automatically made assignments in a non-technical manner. Although explaining the assignments can further improve the trust in an automated assignment system, and at the same time, help stakeholders/end-users improve the quality of the issue reports that they generate, this problem has been overlooked for in this context. We, on the other hand, develop an approach in this work for explaining the assignments in terms of the natural language descriptions present in the issue reports, so that stakeholders can relate to them (Chapter~6).

Another novel problem we have identified after deploying IssueTAG was the need for detecting the deteriorations in the assignment accuracies. Although, for automated assignment systems, as the underlying issue databases evolve continuously, the detection of deteriorations is a practical concern allowing corrective actions, such as the re-calibrations of the models, to be taken in time, this problem has been also overlooked for in the literature. We, on the other hand, develop an approach in this work for automatically detecting deteriorations in daily assignment accuracies in an online manner (Chapter~7).

During an in-depth analysis of the results obtained from IssueTAG, we observed that one common problem with the submitted issue reports was the presence of missing information, such as the steps required to reproduce the issues, expected behavior, and observed behavior. Note that the absence of such information cannot only prevent the developers from reproducing the reported issues, but also make it difficult to correctly assign them. Therefore, automated identification of missing information in issue reports is of practical importance. To this end, there are some existing works in the literature~(Chaparro et al., 2017a; Song & Chaparro, 2020). These works, however, determine the missing information in the issue reports written in English by using part of speech (POS) tags. We, however, observe that in agglutinative languages, such as Turkish, which is the focus of this work, much of

the information that needs to be analyzed is in the verbs together with the suffixes attached to the verbs. In this work, we, therefore, develop an approach based on morphological analysis of verbs to determine the missing information in the reports written in Turkish (Chapter~8).

Another observation we make is that many of the actual issue reports we analyzed have screenshot attachments and that the assignment accuracy of the reports with such attachments tend to be smaller than that of the ones without any attachments. An in-depth analysis revealed that one reason behind this phenomenon could be that the reports with screenshots attachments tend to contain less information in their descriptions (i.e., fewer words) as much of the information is already included in the attachments. To the best of our knowledge, however, there is no work in the literature leveraging screenshots attachments for issue triaging. In this work, we develop an approach, which uses screenshot attachments as an additional source of information for issue report assignment (Chapter~9).

4. LARGE SCALE INDUSTRIAL CASE

In this section, we introduce the context of the research, specifically the issue triage process at *Softtech*[1] and *IsBank*[2], to better understand the problems that arise in such an environment.

IsBank is the largest private bank in Turkey with 7.5 million digital customers, 25 thousand employees, 6566 ATMs (Automated Teller Machines), and 1314 domestic and 22 foreign branches, providing a large variety of banking and financial services.

Being an ISO-9001-certified subsidiary of IsBank, Softtech is the largest software company of Turkey owned by domestic capital. Softtech provides customer-oriented, business-critical solutions to IsBank by using universally-recognized lean techniques and agile processes with a diverse set of programming languages, platforms, and technologies. Some of the technologies used by Softtech include COBOL, Java, C#, C++, mainframe platforms, mobile/wearable platforms, security- and privacy-related technologies, natural language processing technologies, speech technologies, image/video processing technologies, and artificial intelligence technologies.

When the wide range of software systems maintained by Softtech couple with the large user base owned by IsBank, who depend on these systems to carry out their day-to-day businesses, Softtech receives an average of 350 issue reports from the field every day (around 90 thousand reports per year). The reported issues range from bank clerks having software failures to bank customers facing software-related problems in any of the bank channels, including online, mobile, and ATM.

Most of the reported issues concern business-critical software systems. Therefore, both Softtech and IsBank need to handle these issues with utmost importance and urgency. To this end, two dedicated teams of 80 full-time employees in total, namely IT Help Desk (IT-HD) and Application Support Team (AST), are employed, the sole purpose of which is to manage the reported issues.

---

[1] https://softtech.com.tr

[2] https://www.isbank.com.tr

4.1 IT Help Desk

The IT-HD team is employed at IsBank and it consists of 50 full-time, (mostly) non-technical clerks, who are internally referred to as Level 1 employees, indicating the level of technical competency they have.

When a bank employee or a bank customer faces an IT-related issue, they call IT-HD on the phone. The IT-HD clerk listens to the issue, collects the details as needed, records them, and resolves the reported issue right away if it is an issue that can be resolved by an IT-HD clerk, such as the ones documented in basic troubleshooting guides. If not, the clerk is responsible for dispatching the issue to the proper entity/unit in the company. In the case of a software-related issue, the clerk files an issue report to Softtech.

4.2 Application Support Team (AST)

The AST team is employed at Softtech and it consists of 30 full-time, Level 2 employees. That is, in terms of technical competency, the AST employees are somewhere between Level 1 IT-HD clerks and Level 3 software engineers. AST employees are embedded in development teams, which are consisted of software engineers. The same AST member can work with multiple development teams and a development team can have multiple AST members.

The sole responsibility of an AST member embedded in a development team is to manage the collection of issue reports assigned to the team. When a new issue report is assigned to a development team, the AST member embedded in the team is typically the first one to examine the report. If the AST member can resolve the reported issue, he/she resolves it and then closes the report on behalf of the team. Otherwise, the AST member notifies the development team about the newly reported issue by, for example, assigning it to a software engineer in the team or by creating a task for the team and linking it to the issue report.

Note that AST members, although they are not considered to be software engineers, can still resolve some of the reported issues as not all of these issues may require changes in the code base. Some issues, for example, are resolved by running

pre-existing scripts, which can automatically diagnose and fix the problems or by manually updating certain records in the database. Therefore, the ultimate goal of the AST members is to reduce the workload of software engineers by resolving the issues that do not require code changes.

4.3 Issue Reports

An issue report, among other information, such as the date and time of creation, has two parts: a one-line summary and a description, both of which are written in Turkish. The former captures the essence of the issue, whereas the latter describes the issue, including the observed behaviour (OB) and expected behavior (EB) of the system , and provides the steps to reproduce (S2R) the reported issue (Bettenburg et al., 2008).

Another important feature in these issue reports are the attached files. We observed that most of the these reports contain attachments, which are mostly screenshots taken at the time of the software failure. They are visual artifacts that show the unexpected behaviour and include the error messages observed at the time of software failure; and other screenshots that show information related to the steps to reproduce the error. We observed that users frequently prefer to take screenshots and attach to the report rather than explicitly describing the issue in detail in the description part. So, attached screenshots are also important features to take into account in automated issue triage.

Note that the issue reports do not have any field conveying categorical information at the time of creation. The reason is that the collection of software products maintained by Softtech are heavily interacting with each other in a business-critical environment, sharing many resources, such as databases, file systems, and GUI screens. Therefore, the boundaries of these products/components from the perspective of issue reporting and management are not clear at all.

For example, a single GUI screen can have multiple tabs, each of which is maintained by a different development team. A single tab can, in turn, have a number of widgets, each of which is under the responsibility of a different team. Almost all of the GUI screens interact with the core banking system, which is maintained by a different set of development teams. The core can be accessed via different banking channels, such as online, mobile, ATM, and SMS (Short Message Service), each of which has

a dedicated set of development teams. Last but not least, financial transactions are typically carried out by using multiple GUI screens, widgets, and channels, crossing the boundaries of multiple development teams.

4.4 Life Cycle of An Issue Report at Softtech

The issue reports are managed by using Maximo[3] at IsBank and by using Jira[4] at Softtech. Issue reports created at IsBank may be software, hardware or data related, which all exist in Maximo. If the report is software related and cannot be solved by IT Help Desk, it is forwarded to Softtech and created in Jira as well.

The life cycle of an issue report at Softtech (created at Jira) is rather simple, it starts being marked as "Open". When the AST team members start to work on the report, they change the status as "In Progress", validate the report and confirm that enough information is included to solve it. At this stage the AST team member may decide that the report belongs to another team and reassign it, may solve the report with his/her previous knowledge, may not be able to solve it and assign to a software engineer (Level 3). When the report is resolved, the team member updates the status as "Resolved". The related solution information passes to Maximo and the status report becomes "Resolved" as well. If the reporter decides that the problem is not solved, s/he reopens it and the status becomes "Reopened" at both Maximo and Jira and the process restarts again.

[3]https://www.ibm.com/products/maximo

[4]https://www.atlassian.com/software/jira

5. AUTOMATING ISSUE REPORT ASSIGNMENT (RQ1)

Softtech receives an average of 350 issue reports from the field every day. It turned out that the manual issue assignments carried out by the IT-HD clerks were costly and cumbersome, mainly due to the large number of issues received on a daily basis and the relatively large number of development teams that these issues should be assigned to (an average of 53 development teams at any point given in time). Furthermore, incorrect assignments, were not only causing friction between IT-HD and AST, but also increasing the turnaround time for resolutions as the respective issue reports tented to bounce back and forth between IT-HD and AST until the right development team was located.

In this chapter, we address our first research question (RQ1): "What are the advantages and disadvantages of deploying an automated issue report assignment system in an industrial context?"

To this end, we first cast the problem of issue assignment to a classification problem and evaluate a number of existing classifiers by using the actual issue reports submitted to Softtech to determine the classification approach to be used in the deployed system. We then determine the time locality of the training data, i.e., how much back in time one should go to prepare the training data to train the underlying classification models. We finally deploy the system and analyze its effects in a multifaceted manner.

In particular, we assign issue reports to development teams, rather than to individual developers – a decision we made based on our discussions with the IT-HD and AST teams. Note that this design decision is not necessarily desirable (or practical) for all the scenarios. For example, in the presence of a single development team, assigning issue reports to individual developers makes more sense.

At Softtech, however, there were several reasons as to why the issue reports were preferred to be assigned to the development teams. First, the sole purpose of an AST team member is to resolve as many issue reports as possible on behalf of the development team, in which he/she is embedded, before the reports are attended by

the individual developers in the team. This necessities the assignment of the reports to the teams in order to utilize the AST team as much as possible. Second, assigning issue reports to individual developers does not take into account 1) the current workloads owned by the individual developers, 2) the changes in the team structures, such as the developers leaving or joining the teams, and 3) the current status of developers, such as the developers who are currently on leave of absence. Therefore, especially in the presence of close-knit development teams, which is the case with Softtech, assigning issue reports to the development teams help the teams make more educated decisions as to which team member the report should be addressed by.

Furthermore, rather than carrying out the issue assignments in the context of a single product, such as Eclipse, Mozilla, and Firefox, where the incoming issues are assigned to individual software engineers working on the product, we do the assignments at the level of an entire company (Softtech), which has 489 software products comprised of around 100 millions of lines of code. More specifically, we assign issue reports filed for any product owned by Softtech to the development teams responsible for resolving the reported issues. This is challenging because with the collection of software products maintained by Softtech, which heavily interact with each other in a business-critical environment by sharing many resources, such as databases, file systems, and GUI screens, the boundaries of the products from the perspective of issue reporting and management are not clear at all.

Deploying IssueTAG presented us with an unprecedented opportunity to observe the practical effects of automated issue assignment in practice as well as to carry out user studies, which (to the best of our knowledge) have not been done before in this context. First, we observed that it is not just about deploying a data mining-based system for automated issue assignment, but also about designing/changing the assignment process around the system to get the most out of it. We, in particular, made simple, yet effective changes in the manual issue assignment process employed at IsBank and Softtech (Section 5.4.1).

Second, the accuracy of the assignments does not have to be higher than that of manual assignments in order for the system to be useful, which is further validated by the user studies we carried out on actual stakeholders in the field (Section 5.5). In a nutshell, although the daily assignment accuracy of IssueTAG was slightly lower than that of manual assignments (0.831 vs. 0.864), it reduced the manual effort required for the assignments by about 5 person-months per year and improved the turnaround time for resolving the reported issues by about 20% (Section 5.4.2). Furthermore, about 79% of the stakeholders participated in our user study "agreed"

or "strongly agreed" that the system was useful (Section 5.5).

Last but not least, we observed that stakeholders do not necessarily resist change. In particular, we did not receive any objection at all to the deployment of IssueTAG. We believe that this was because all the stakeholders believed that they would benefit from the new system and none of them felt threatened by it (Section 5.6). We, furthermore, observed that gradual transition helped stakeholders build confidence in IssueTAG, which, in turn, facilitated the acceptance of the system (Section 5.6).

This chapter is organized as follows: Section 5.1 describes the issue assignment process employed at IsBank and Softtech before the deployment of IssueTAG; Section 5.2 describes the solution approaches for deciding on the data mining approach to be used and the time locality of the data; Section 5.3 first decides on the data mining approach to be used by evaluating the existing approaches for automated issue assignment on the collection of issue reports maintained by IsBank and Softtech and then, presents the empirical studies we carried out to determine the amount and time locality of the training data required for training/recalibrating the underlying data mining models; Section 5.4 deploys IssueTAG and evaluates its effects in practice; Section 5.5 carries out a user study on the end users of IssueTAG to evaluate whether the deployed system is perceived as useful; Section 5.6 presents the lessons learnt regarding the deployment of the automation in a large scale industrial setting; and Section 5.7 concludes the chapter.

5.1 Manual Issue Assignment Process

Before the deployment of IssueTAG, IT-HD clerks, after creating an issue report, was assigning it to a development team. To this end, they were maintaining a knowledge base, which was simply comprised of spreadsheets mapping certain keywords with development teams. In the presence of an incorrect assignment, although the AST member(s) or the software engineers in the respective development team could reassign the issue to a different team, the incorrectly assigned reports were often returned back to IT-HD for reassignment. Figure 5.1 summarizes the assignment process. The issue reports are managed by using Maximo[1] at IsBank and by using

[1] https://www.ibm.com/products/maximo

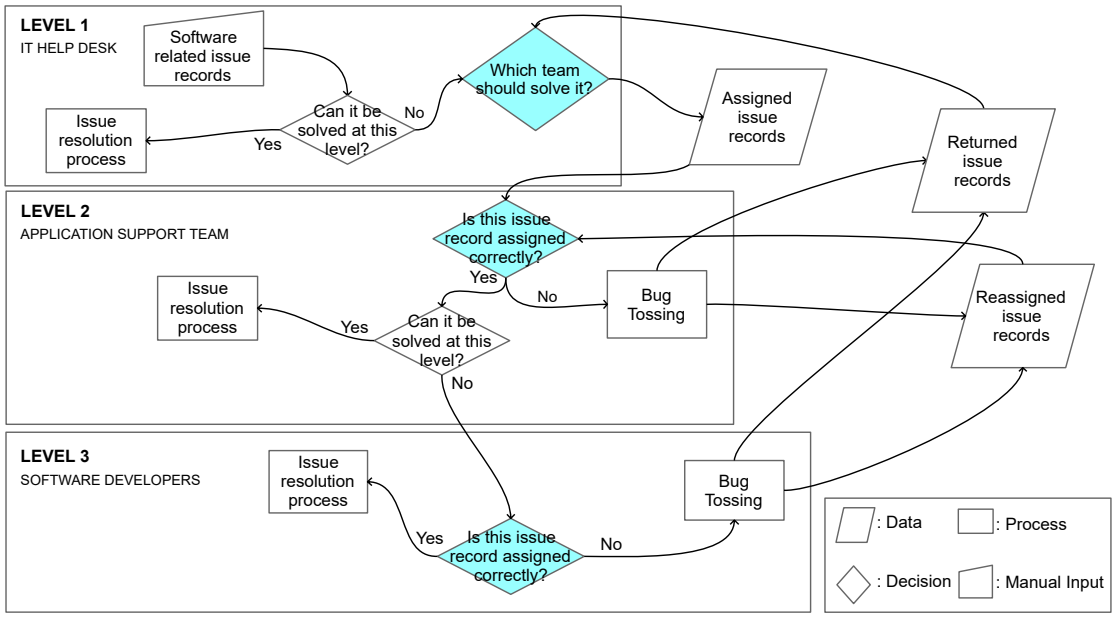


Figure 5.1 Issue report assignment process before the deployment of IssueTAG.

Jira[2] at Softtech.

There were a number of issues with the aforementioned process. First, the learning curve for the IT-HD clerks (especially for the new hires) for excelling in team assignments was generally steep due to the large number of issue reports received on a daily basis (an average of about 350 issue reports) and the relatively large number of products and development teams present (more than 450 products and between 47 and 57 teams at any given point in time). Second, although IT-HD clerks were using a knowledge base to help with the assignments, it was maintained in an ad hoc manner, which was error prone, cumbersome, and time consuming. Last but not least, incorrect assignments were not only causing frictions between the IT-HD clerks and the AST members, but also increasing the turn around time for resolutions as the incorrectly-assigned issue reports would typically bounce back and forth between the IT-HD clerks and AST members (*bug tossing*) until the correct development team was located, which was causing a great deal of wasted time.

[2]https://www.atlassian.com/software/jira

5.2 Approach

IsBank and Softtech wanted to improve their current practices. In order to automate the issue assignments, we start with investigating how the existing classification approaches for automated issue assignment compare with each other. This is important as the results will determine the baseline classification technique to be used in the deployed system. Note that our goal in this work is neither to propose yet another approach for automated issue assignment nor to evaluate all existing approaches to determine the best possible approach, but to identify an existing approach that can produce similar or better assignment accuracies with the manual assignment process employed at IsBank/Softtech and that can be developed and deployed with as little risk as possible. After all, most of the issue reports the system will process, concern business-critical software systems. Therefore, neither IsBank nor Softtech was willing to take too much risk. As such, we opted to refrain ourselves from using other sources of information not included in the issue reports, such as source code files, commit history, developer social networks, skill profiles or tossing graphs. We observed that delivering a much more involved system, which needs to be integrated with many other systems, may both complicate the acceptance of the system by various stakeholders and make the system hard to maintain.

Since this is a live system, the dataset that we can use is evolving continuously. Therefore we need to recalibrate the models as needed, and when recalibrating, we have to consider the time locality of the data, in other words we have to determine how much we should go back in time in the issue report repository to decide on the training set that we need to use.

Text Classification

To carry out the study, we have determined a number of approaches, which had been shown to be effective for automated issue assignment (Anvik et al., 2006; Anvik & Murphy, 2011; Bhattacharya et al., 2012; Jonsson et al., 2016; Murphy & Cubranic, 2004). We then empirically evaluated them by using the issue database, which has been maintained by Softtech since December 2016. In particular, we cast the problem of issue assignment to a classification problem where the natural language descriptions in issue reports are analyzed by using various classification algorithms.

Given an issue report, we first combine the “description” and “summary” parts of the report, then tokenize the combined text into terms, and finally remove the non-letter characters, such as punctuation marks, as well as the stop words, which are

extremely common words of little value in classifying issue reports (Schütze et al., 2008), such as "the", "a", and "is." We opt not to apply stemming in this work as an earlier work suggests that stemming has a little effect (if any at all) in issue assignments (Murphy & Cubranic, 2004), which is also consistent with the results of our initial studies where stemming slightly reduced the assignment accuracies.

We then represent an issue report as an n-dimensional vector. Each element in this vector corresponds to a term and the value of the element depicts the weight (i.e., "importance") of the term for the report. The weights are computed by using the well-known *tf-idf* method (Schütze et al., 2008). The more a term t appears in an issue report r and the less it appears in other issue reports, the more important t becomes for r, i.e., the larger tf-idf$_{t,r}$ is.

Once an issue report is represented as an ordered vector of tf-idf scores, the problem of assignment is cast to a classification problem where the development team, to which the issue report should be assigned, becomes the class to be predicted and the tf-idf scores of the report become the attributes, on which the classification will be based on.

We train two types of classifiers: *level-0 classifiers*, each of which is comprised of an individual classifier, and *level-1 classifiers*, which were obtained by combining multiple level-0 classifiers using stacked generalization – an ensemble technique to combine multiple individual classifiers (Wolpert, 1992). All the classifiers we experiment with in this study have been shown to be effective for automated issue assignment (Anvik et al., 2006; Anvik & Murphy, 2011; Bhattacharya et al., 2012; Jonsson et al., 2016; Murphy & Cubranic, 2004).

For the level-0 classifiers, we use multinomial naive bayesian (Schütze et al., 2008), decision tree (Breiman, 2017), k-nearest neighbor (Schütze et al., 2008), logistic regression (Bishop, 2006), random forest (Breiman, 2001), and linear support vector classifiers (SVCs) (Joachims, 1998).

For the level-1 classifiers, we first train and evaluate our level-0 classifiers by using the same training and test sets for each classifier. We then use the prediction results obtained from these level-0 classifiers to train a level-1 classifier, which combines the probabilistic predictions of the level-0 classifiers using linear logistic regression (Wolpert, 1992).

Inspired from (Jonsson et al., 2016), we, in particular, train two types of level-1 classifiers: *BEST* and *SELECTED*. The BEST ensemble is comprised of k (in our case, $k = \{3, 5\}$) level-0 classifiers with the highest assignment accuracies, where as the SELECTED ensemble is comprised of a diversified set of k (in our case,

$k = \{3, 5\}$) level-0 classifiers, i.e., the ones with different representation and classification approaches, which are selected regardless of their classification accuracies, so that errors of individual classifiers can be averaged out by better spanning the learning space (Wolpert, 1992). Note that the BEST and SELECTED ensembles are not necessarily the same because the best performing level-0 classifiers may not be the most diversified set of classifiers. More information on how these ensembles are created can be found in Section 5.3.1.

Furthermore, for the baseline classifier, which we use to estimate the baseline classification accuracy for our classifiers, we assign all issue reports to the team that have been assigned with the highest number of issue reports. That is, our baseline classifier always returns the class with the highest number of instances as the prediction.

Deep Learning Based Text Classification

Recently, deep learning based issue triaging techniques have been proposed Lee et al. (2017) as well. We experimented with deep learning based text classification techniques as well, so as to compare with our baseline model (5.3.2).

With the Word2Vec technique developed by Mikolov et al. (2013), words can be represented with low-dimensional vectors, and semantically similar words are clustered in close coordinates. Joulin et al. (2017) proposed fastText, a fast and efficient technique, using n-grams in representation of the words. With this technique, words that are not in the training data but in the test data, can be represented as well. Erdinc & Guran (2019) conducted experiments using Word2Vec, Doc2Vec and FastText techniques with word representations from 5 million Turkish news documents and they shared the vectors they obtained as a result[3]. They achieved the most successful result with the fastText technique. In our study, we used the fastText vectors they obtained after removing punctuation marks and stop words to represent the words, where the size of the related word representation vectors is 100.

Lee et al. (2017) reported that they achieved high accuracy with deep learning techniques in bug report classification. They used convolutional neural networks (CNN) in their work. Kim (2014) showed that successful results can be obtained in text classification with a simple CNN model using pre-trained word vectors. The architecture he proposes includes filtering the relevant word vectors with a 1-dimensional sliding window in the convolution layer, and extracting the most important features by max-pooling, followed by the classifier layer where the classification is realized. He compared 4 different variations of the architecture in his study: In the *CNN-*

[3]https://github.com/hakkiyagiz/SIU2019

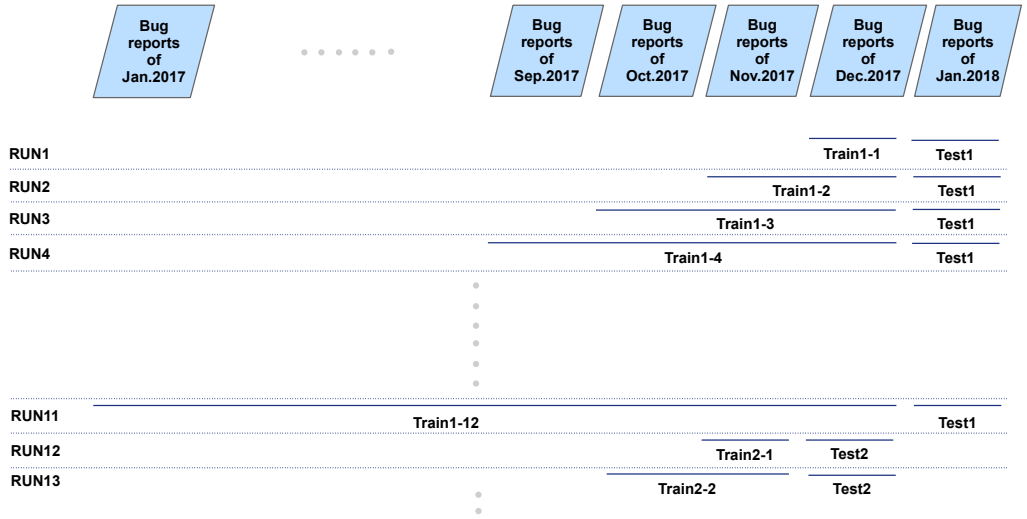


Figure 5.2 Overview of the cumulative window approach to study the effect of the amount of training data on assignment accuracies.

random method, word vectors are randomly initialized and their values are updated during training; in the *CNN-static* method, pre-trained word vectors are not updated during training, while with the *CNN-non-static* method word vectors are updated; The *CNN-multichannel* method combines the vectors passing through different convolution and max-pooling layers, where word vectors are kept static in one channel and trainable in the other.

Another deep learning technique we used in text classification is *Recursive Neural Networks (RNN)*, which can capture the long-term dependencies between words. *Long Short-Term Memory (LSTM)* technique has been developed to deal with the vanishing gradient problem that can be encountered while training traditional RNNs (Hochreiter & Schmidhuber, 1997).

Time Locality and Amount of Training Data

Another important issue to consider before automation is how the amount and time locality of training data affect the assignment accuracies. The results will be used to determine the amount of training data (e.g., how many issue reports we should use) as well as the time locality of this data (i.e., how much back in time we should go) required for preparing the training data every time the underlying classification model needs to be retrained.

To answer these questions, we use the *sliding window* and *cumulative window* ap-

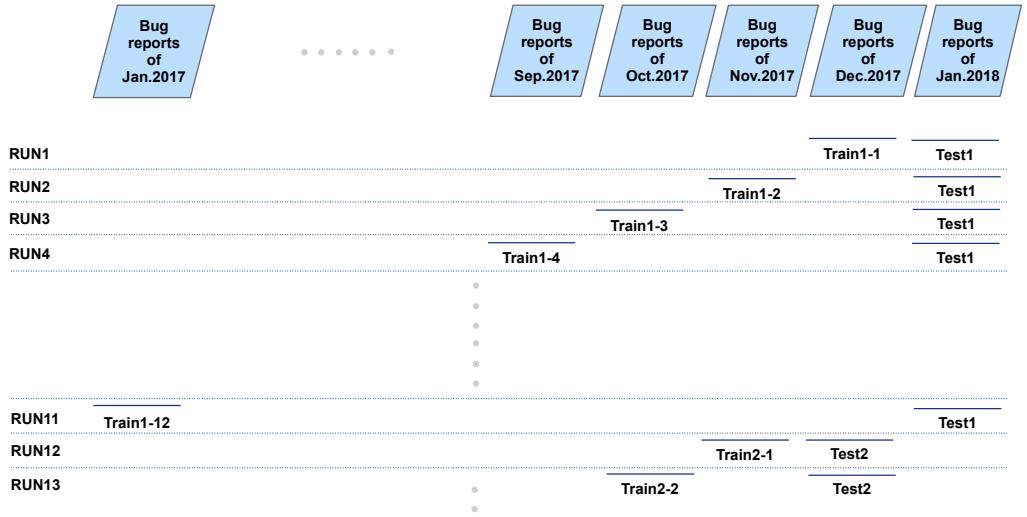


Figure 5.3 Overview of the sliding window approach to study the effect of the time locality of training data on assignment accuracies.

proaches introduced in (Jonsson et al., 2016). More specifically, we conjecture that using issue reports from "recent past" to train the prediction models, as opposed to using the ones from "distant past", can provide better assignment accuracies since organizations, products, teams, and issues may change overtime.

To evaluate this hypothesis, we take a long period of time T (in our case, 13 months) and divide it into a consecutive list of calendar months $T = [m_1, m_2, \ldots]$. For every month $m_i \in T$, we train and evaluate a linear SVC model. To this end, we use all the issue reports submitted in the month of m_i as the test set and all the issue reports submitted in the month of m_j as the training set, where $i - j = \Delta$, i.e., the sliding window approach in (Jonsson et al., 2016). Note that given m_i and Δ, m_j is the month, which is Δ months away from m_i going back in time. For every month $m_i \in T$, we repeat this process for each possible value of Δ (in our case, $\Delta \in \{1, \ldots, 12\}$). By fixing the test set and varying the training sets, such that they come from different historical periods, we aim to measure the effect of time locality of the training data on the assignment accuracies.

Figure 5.3 illustrates the sliding window approach using the period of time from Jan 1, 2017 to Jan 31, 2018. For example, for the month of Jan 2018, we train a total of 12 classification models, each of which was trained by using all the issue reports submitted in a distinct month of 2017 (marked as Train1-1, Train1-2, ..., Train1-12) and separately test these models using all the issue reports submitted in the month

of Jan, 2018 as the test set (marked as Test1). We then repeat this process for every month in the time period of interest, except for Jan 2017 as it does not have any preceding months. That is, for Dec 2017 (marked as Test2), we train and evaluate 11 models (marked as Train2-1, Train2-2, ...), for Nov 2017, we train and evaluate 10 models, etc.

To evaluate the effect of the amount of training data on the assignment accuracies, we use a related approach, called the cumulative window approach (Jonsson et al., 2016). This approach, as is the case with the sliding window approach, divides a period of interest T in to a consecutive list of months $T = [m_1, m_2, \ldots]$. Then, for every possible pair of $m_i \in T$ and Δ, we train and evaluate a classification model, where all the issue reports submitted in the month of m_i are used as the test set and all the issue reports submitted in the preceding Δ months, i.e., $\{m_j \in T \mid 1 \leq i - j \leq \Delta\}$, are used as the training set.

Figure 5.2 illustrates the approach. For example, for the month of Jan 2018, we train a total of 12 classification models. The first model is created by using the previous month's data (marked as Train1-1), the second model is created by using the previous two months' data (marked as Train1-2), and the last model is created by using the previous year's data (marked as Train1-12). The same process is repeated for every possible month in the period of interest.

5.3 Experiments

5.3.1 Deciding on the Baseline Model

We have conducted a series of experiments to evaluate the assignment accuracies of the level-0 and level-1 classifiers.

Experimental Setup

In these experiments, we used the issue reports submitted to Softtech between June 1, 2017 and November 30, 2017 as the training set and the issue reports submitted in the month of December 2017 as the test set. We picked this time frame because it provided us with a representative data set in terms of the number of issue reports

submitted, the number of teams present, and the distribution of the reported issues to these teams.

For the aforementioned time frame, we had a total number of 51,041 issue reports submitted to 65 different teams. Among all the issue reports of interest in this section as well as in the remainder of the chapter, we only used the ones that were marked as "closed," indicating that the reported issues had been validated and resolved. Furthermore, as the correct assignment for an issue report, we used the development team that had closed the report. The remainder of the issue reports were ignored as it was not yet certain whether these reports were valid or whether the development teams, to which they were currently assigned, were correct. After this filtering, a total of 47,123 issue reports submitted to 64 different development teams remained for analysis in this study.

To create the level-1 classifiers, we combined 3 or 5 individual classifiers, i.e., $k = 3$ or $k = 5$. We used the latter setting as it was also the setting used in a recent work (Jonsson et al., 2016). We used the former setting as it was the best setting we could empirically determine for ensemble learning, i.e., the one that produced the best assignment accuracies. In the remainder of the chapter, these models are referred to as *BEST-3*, *SELECTED-3*, *BEST-5*, and *SELECTED-5*.

The BEST-3 and BEST-5 models were obtained by combining Linear SVC-Calibrated, Logistic Regression, and K-Neighbours; and Linear SVC-Calibrated, Logistic Regression, K-Neighbours, Random Forest, and Decision Tree classifiers, respectively, as these were the classifiers providing the best assignment accuracies. The SELECTED-3 and SELECTED-5 models, on the other hand, were obtained by combining Linear SVC-Calibrated, K-Neighbours, and Multinomial Naive Bayesian; and Linear SVC-Calibrated, Logistic Regression, K-Neighbours, Random Forest, and Multinomial Naive Bayesian classifiers, respectively, with the goal of better spanning the learning space by increasing the diversity of the classification algorithms ensembled. Note further that to include SVCs in level-1 classifiers, we used calibrated linear SVCs instead of linear SVCs as we needed to have class probabilities to ensemble individual classifiers (Ting & Witten, 1999), which are not supported by the latter.

The classifiers were trained and evaluated by using the `scikit-learn` Python library (for level-0 classifiers) (Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg & others, 2011) and `mlxtend` (for level-1 classifiers) (Raschka, 2018a) packages. All of the classifiers (unless otherwise stated) were configured with the default settings and the experiments were carried out on a dual-core Intel(R) Xeon(R) E5-2695 v4 2.10 GHz machine with 32 GB of RAM

running Windows Server 2012 R2 as the operating system.

Evaluation Framework

To evaluate the quality of the assignments obtained from different classifiers, we used well-known metrics, namely *accuracy* and weighted *precision*, *recall*, and *F-measure* (Schütze et al., 2008). Accuracy, which is also referred to as *assignment accuracy* in the remainder of the chapter, is computed as the ratio of correct issue assignments. Precision for a particular development team (i.e., class) is the ratio of the issue reports that are correctly assigned to the team to the total number of issue reports assigned to the team. Recall for a team is the ratio of the issue reports that are correctly assigned to the team to the total number of issue reports that should have been assigned to the team. F-measure is then computed as the harmonic mean of precision and recall, giving equal importance to both metrics. Note that each of these metrics takes on a value between 0 and 1 inclusive. The larger the value, the better the assignments are. Furthermore, we report the results obtained by both carrying out 10-fold cross validation on the training data and carrying out the analysis on the test set.

To evaluate the cost of creating the classification models, we measured the time it took to train the models. The smaller the training time, the better the approach is.

Data and Analysis

Table 5.1 summarizes the results we obtained. We first observed that all the classifiers we trained performed better than the baseline classifier. While the baseline classifier provided an accuracy of 0.10 on the training set and 0.12 on the test set, those of the worst-performing classifier were 0.47 and 0.52, respectively.

We then observed that the SELECTED ensembles generally performed similar or better than the BEST ensembles, supporting the conjecture that using diversified set of classifiers in an ensemble can help improve the accuracies by better spanning the learning space. For example, while the accuracy of the BEST-5 ensemble was 0.67 on the training set and 0.64 on the test set, those of the SELECTED-5 ensemble were 0.80 and 0.78, respectively. Furthermore, the ensembles created by using 3 level-0 classifiers, rather than 5 level-0 classifiers, performed slightly better on our data set. For example, while the accuracy of the SELECTED-5 ensemble was 0.80 on the training set and 0.78 on the test set, those of the SELECTED-3 ensemble were 0.81 and 0.79, respectively.

Last but not least, among all the classifiers, the one that provided the best assignment accuracy (as well as the best F-measure) and did so at a fraction of the cost,

Table 5.1 Accuracy (A) and weighted precision (P), recall (R), and F-measure (F) values obtained from different classification models as well as the training times of these models.

	using training set with 10-fold cross validation		using test set			
classifier	A	training time	P	R	F	A
Baseline	0.10	-	0.01	0.12	0.03	0.12
Multinomial NB	0.47 (+/- 0.01)	31 s	0.70	0.52	0.50	0.52
Decision Tree	0.66 (+/- 0.02)	50 m 11 s	0.64	0.63	0.63	0.63
K-Neighbours	0.73 (+/- 0.02)	1 m 4 s	0.71	0.72	0.71	0.72
Logistic Regression	0.74 (+/- 0.01)	18 m 37 s	0.76	0.74	0.74	0.74
Random Forest	0.66 (+/- 0.02)	51 m 43 s	0.64	0.65	0.63	0.65
Linear SVC	**0.82 (+/- 0.01)**	3 m 32 s	**0.80**	**0.80**	**0.80**	**0.80**
Linear SVC-Calibrated	0.81 (+/- 0.01)	7 m 50 s	**0.80**	0.79	0.79	0.79
BEST-5	0.67 (+/- 0.02)	2 h 20 m 38 s	0.65	0.64	0.64	0.64
SELECTED-5	0.80 (+/- 0.01)	1 h 49 m 11 s	0.79	0.78	0.78	0.78
BEST-3	0.81 (+/- 0.01)	56 m 7 s	**0.80**	0.79	0.79	0.79
SELECTED-3	0.81 (+/- 0.01)	32 m 57 s	**0.80**	0.79	0.79	0.79

was the linear SVC classifier (Table 5.1). While the linear SVC classifier provided an accuracy of 0.82 on the training data set and 0.80 on the test set with a training time of about three minutes, the runner-up classifiers, namely the SELECTED-3 and BEST-3 ensembles, provided the accuracies of 0.81 and 0.79, respectively, with a training time of about half an hour or more.

Based on both the assignment accuracies and the costs of training obtained from various classifiers using our data set, we have decided to employ linear SVC in IssueTAG. Consequently, all of the results presented in the remainder of the chapter were obtained by using linear SVC classifiers.

5.3.2 Deep Learning Based Text Classification

We have conducted a series of experiments to compare the CNN and LSTM techniques with our baseline model, Linear SVC (Aktas, Yeniterzi & Yilmaz, 2020).

Experimental Setup

We included the issue reports that were opened and closed during 2017, that is, the solution process was completed and the class was determined. After removing the reports belonging to teams with less than 150 reports from the collection, we had a total of 78,363 issue reports assigned to 46 different software teams that we used in

the experiments. We observed that the issue reports belonging to the first 8 teams with the most number of reports constitute approximately 50% of all reports. The team with the highest number of reports has 5,977 reports, while the team with the lowest number has 174. Although the distribution was not balanced, we preferred to conduct the experiments with the final collection in accordance with the real situation in the production environment.

The summary and description attributes were combined and the text obtained was used as input. All non-alphanumeric characters and numbers longer than two characters are omitted from the relevant text. The corresponding numeric data has been omitted from the collection because it contains numeric data such as customer numbers, card numbers, etc. The remaining letters were converted to lowercase letters. Turkish stop words and words consisting of a single letter were also removed. The stop words used were from the work of Erdinc & Guran (2019). After pre-processing, we saw that the combined field contains 30.51 words on the average and 338 words at most.

Evaluation Framework

Accuracy, Precision, Recall, and F-measure metrics are used to compare the models. In the study, $top-1$, $top-2$ and $top-3$ Accuracy values and Precision, Recall and F-measure values are reported. If the correct class for a record is within the first n classes predicted, the corresponding record is considered as the correct predicted record when calculating the $top-n$ accuracy value. Note that, in order to obtain $top-n$ accuracies for the Linear SVC algorithm, it had to be calibrated to obtain probabilities for the predicted classes.

In the study, the keras toolbox was used to create the deep learning layers (Chollet & others, 2015). For all the deep learning models, training was run with 32 batches during 2 epochs.

For the CNN models, CNN-random and CNN-multichannel techniques are used in our experiments. In addition, a variation of the CNN-multichannel is used, which we call *CNN-triage*, similar to the work of Lee et al. (2017): One of the channels use the summary attribute as input, while the other uses the description attribute. A non-static model is applied for word vectors in both channels. In our work, while filtering with a 1-dimensional sliding window, the window sizes we used are 2, 3 and 4; and "ReLU" is used as the activation function.

For the LSTM model, we used bi-directional LSTM (Bi-LSTM) technique, where the number of hidden nodes was 128, and the words were randomly initialized.

Table 5.2 Performance comparison with deep learning techniques

Classifier	Results on Test Set					
	Top-1 Accuracy	Top-2 Accuracy	Top-3 Accuracy	Precision	Recall	F-measure
Linear SVC	**0.82**	**0.92**	**0.95**	**0.82**	**0.82**	**0.82**
CNN-random	0.81	0.91	**0.95**	0.81	0.81	0.80
CNN-multichannel	0.80	0.91	0.94	0.80	0.80	0.79
CNN-triage	0.80	0.91	**0.95**	0.80	0.80	0.79
Bi-LSTM	0.79	0.90	0.94	0.79	0.79	0.79

Data and Analysis

Since the corpus we used contains various domain specific words, we checked the most similar words to a set of selected words. For example, the words "batcheft", "bkyu", "tyu" are special to the field, where some are abbreviations of some applications. When the vectors created by Erdinc & Guran (2019) with the fastText method are used, the most similar words are listed as "batch", "sangyo", "tyurin", respectively. Although it is an advantage that the relevant words can be represented even though they are outside the vocabulary, we think that the representation is not good enough, especially when considering the most similar words suggested for "bkyu" and "tyu". We also saw that the out-of-vocabulary rate is $35\%(25,917/74,299)$. The rate seems to be high as the word roots are not addressed and the misspellings are not corrected.

The results we obtained are reported in Table 5.2. Although the results are very close to each other, it was observed that Linear SVC, the baseline method, is successful both in terms of accuracy and training time.

While the top-1 accuracy values are around 0.8, a top-3 accuracy of more than 0.94 for each algorithm is an indicator that there are similar subjects that the software teams are working on, hence the boundaries between some teams are not clearly distinguishable.

For Linear-SVC and CNN-random techniques, we also reviewed the confusion matrix. Since our collection shows an uneven distribution amonf teams, recall values were compared for teams with a small number of issue reports and precision values for teams with a large number of issue reports. The precision for the first 3 teams with the highest number of reports is 0.71, 0.99, 0.92 for CNN-random respectively; while the results are 0.71, 0.98, 0.93 for Linear SVC. The recall results for the first 3 teams with the lowest number of reports is 0.75, 0.39, 0.93 for CNN-random respectively; while the results are 0.75, 0.25, 0.89 for Linear SVC. Thus, for the teams with few issue reports, CNN-random seems to be relatively more successful.

Table 5.3 Number of issue reports submitted.

month	# of issue reports submitted	# of teams assigned
Jan 2017	6364	57
Feb 2017	5038	56
Mar 2017	7188	57
Apr 2017	6623	55
May 2017	6601	56
Jun 2017	6145	56
Jul 2017	6341	53
Aug 2017	6025	54
Sep 2017	5961	54
Oct 2017	6774	52
Nov 2017	7996	54
Dec 2017	7881	49
Jan 2018	7426	51
Total	86363	69

5.3.3 Time Locality and Amount of Training Data

We conducted a series of experiments to evaluate the effect of the amount and time locality of training data on assignment accuracies.

Experimental Setup

In these experiments, we used all the issue reports that were submitted during the period from Jan 1, 2017 to Jan 31, 2018. The summary statistics for this data set can be found in Table 5.3. All told, we have trained and evaluated a total of 144 linear SVC models for this study. All the experiments were carried out on the same platform with the previous study (Section 5.3.1).

Evaluation Framework

We used the assignment accuracies (Section 5.3.1) for evaluations.

Data and Analysis

Figures 5.4 and 5.5 represent the results we obtained from the sliding window and cumulative window approach, respectively. In these figures, the vertical and horizontal axes depict the assignment accuracies obtained and the Δ values used in the experiments, respectively. The accuracies associated with a Δ value were obtained from the classification models, each of which was created for a distinct month in the period of interest by using the same Δ value. Furthermore, the polynomials in the figures are the second degree polynomials fitted to the data.

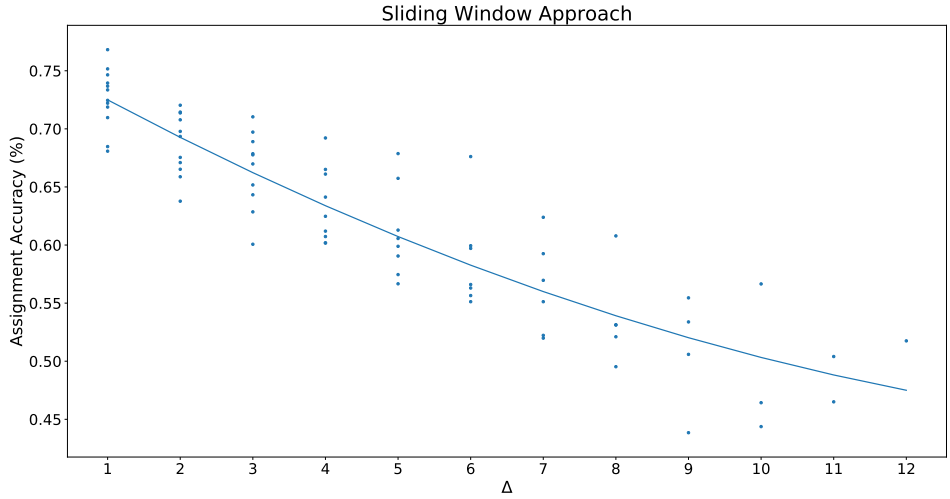


Figure 5.4 Assignment accuracies obtained from the sliding window approach.

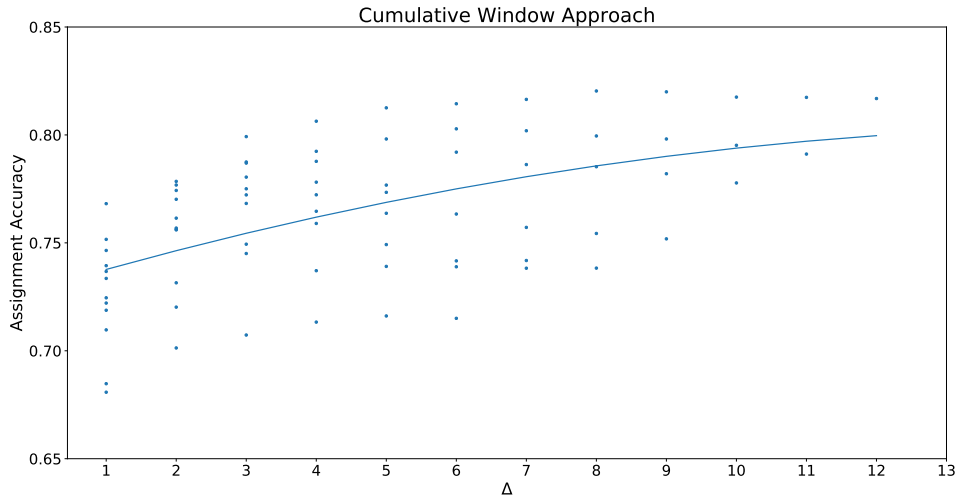


Figure 5.5 Assignment accuracies obtained from the cumulative window approach.

Looking at Figure 5.4, we first observed that using issue reports from recent past to train classification models, rather than the ones from distant past, provided better assignment accuracies; the accuracies tended to decrease as Δ increased. For example, while the average assignment accuracy obtained when $\Delta = 1$, i.e., when the issue reports submitted in the immediate preceding months were used as the training sets, was 0.73, that obtained when $\Delta = 12$, i.e., when the issue reports submitted in Jan 2017 were used as the training set for the issue reports submitted in Jan 2018, was 0.52%.

Looking at Figure 5.5, we then observed that as we went back in time to collect the training data starting from the immediate preceding months (i.e., as Δ increased

in the cumulative window approach), the assignment accuracies tended to increase first and then stabilized around a year of training data. For example, while the average accuracy obtained when $\Delta = 1$, i.e., when the issue reports submitted only in the immediate preceding months were used as the training sets, was 0.73%, that obtained when $\Delta = 12$, i.e., when all the issue reports submitted in the preceding 12 months were used as the training data set, was 0.82%.

Based on the results of these studies, to train a prediction model at a given point in time, we decided to use all the issue reports that have been submitted in the last 12 months as the training set. Clearly, among all the issue reports of interest, we filter out the ones that have not yet been closed (Section 5.3.1).

5.4 Automated Issue Assignments in Practice

In this section, we present the practical effects of automating issue assignment compared to manual assignment. Note that the results of this study will help evaluate the pros and cons of automated issue assignments in the field.

5.4.1 Changing the Process

To deploy IssueTAG at IsBank and Softtech, we carried out a number of meetings with the IT-HD, AST, and software development teams. In these meetings, the problems with the manual issue assignment process were discussed, IssueTAG was presented, and the effect of automating the assignment process was demonstrated by using the results of a number of preliminary studies conducted on historical data collected from the field.

One commonly accepted observation, which was made numerous times in these meetings, was that automating the issue assignment process (i.e., deploying IssueTAG) would also require to modify the other parts of the process around the deployed system to improve the efficiency and effectiveness of the entire process to the extent possible.

One refinement suggestion came from us (Process Improvement Team at Softtech).

In our preliminary studies, we observed that wrong assignments made by IssueTAG were often caused due to the difficulty of distinguishing related, but different development teams from each other, such as the teams working on related products or working on different components of the same product. That is, when an issue report was assigned to a wrong team, the assignee and the correct team (i.e., the one, to which the report should have been assigned) were often related to each other, e.g., they were aware of each other's works. Consequently, we suggested that in the presence of an incorrect assignment, rather than returning the issue report to IT-HD for reassignment, which was typically the case in the manual assignment process, letting the assignee (e.g., the AST member embedded in the incorrectly assigned team) do the reassignment, could profoundly speed up the process.

Another refinement suggestion came from the IT-HD management. They simply suggested to prevent IT-HD clerks from modifying the issue assignments made by IssueTAG. On one hand, this was a natural consequence of the design decision discussed above in the sense that when the reassignments are made by the current assignee, IT-HD clerks will not necessarily be aware of these modifications, thus may not learn from them to improve their assignment accuracies. On another hand, we observed that IT-HD was actually looking forward to deferring the responsibility of issue assignments. One reason was that, especially for the new IT-HD clerks, the learning curve for excelling in assignments was generally steep due to the large number of issue reports received on a daily basis and the relatively large number of development teams present. In fact, IT-HD was maintaining a knowledge base (comprised mostly of spreadsheets) to help the clerks with the assignments. However, it was cumbersome and costly for them to keep this knowledge base up to date. Nevertheless, incorrect assignments were often causing friction between the IT-HD clerks and AST members as well as the development teams.

5.4.2 Deployment and Results

We deployed IssueTAG on Jan 12, 2018. The system has been fully operational since then, making automated assignments for all the issue reports submitted, which is more than 250,000 automated assignments so far. Figure 5.6 presents the overall system architecture. Furthermore, Table 5.4 reports some summary statistics regarding the operations of the deployed system.

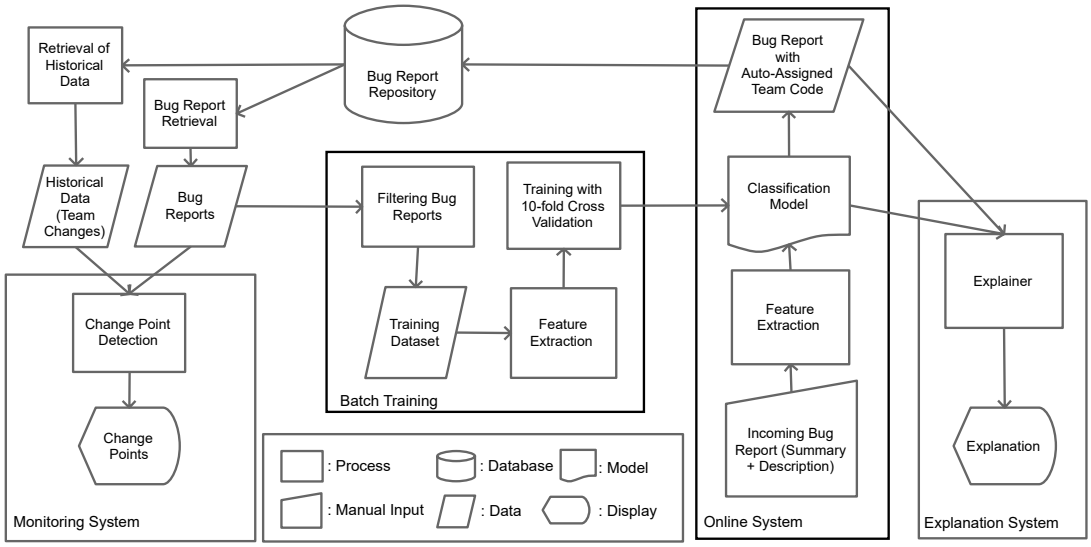


Figure 5.6 High level architecture of IssueTAG.

Table 5.4 Summary statistics regarding the operations of IssueTAG, starting from its deployment on Jan 12, 2018 till June 30, 2019.

Item	Value
Total number of issue reports assigned	134,622
Average number of issue reports per day	380
Total number of distinct teams	62
Average time it takes to train the model (with one year of data)	7m 42s
Average response time of the system	746 msec
Size of the trained model (trained with one year of data)	588 MB

Deployment Setup

Based on the results of our empirical studies in Section 5.3.1, IssueTAG was configured to use Linear SVC to train the classification models. And, based on the results obtained in Section 5.3.3, the models have been trained by using the issue reports submitted in the last 12-month time frame. Furthermore, as all the process improvement suggestions discussed in Section 5.4.1 were accepted by all the stakeholders involved, we configured IssueTAG such that once an issue report was created by an IT-HD clerk for the first time, the report was automatically assigned to a development team by the deployed system and the IT-HD clerk did not have any means of interfering with the assignment process and/or modifying the assignment.

The system is deployed on a Dual-Core Intel(R) Xeon(R) E5-2695 v4 2.10 GHz machine with 32 GB of RAM running Windows Server 2012 R2 as the operating system.

Evaluation Framework

To evaluate the quality of the assignments over a period of time, we compute the assignment accuracy on a daily basis, which we refer to as *daily assignment accuracy*. More specifically, the daily assignment accuracy achieved on a day d, is the ratio of the assignments that are correctly made for the issue reports opened on the day d. Note that we compute the daily accuracies based on the dates, on which the issue reports are opened, rather than they are closed. This is because the automated assignments are made as soon as the issue reports are created (i.e., opened) by using the underlying classification model, which was available at the time of the creation.

To evaluate the reduction in the amount of manual effort required for the issue assignments, we measure the person-months saved by automating the process. To this end, a survey we conducted on the IT-HD clerks revealed that, given an issue report, it takes about 30 seconds on average for an IT-HD clerk to assign the report to a development team, which is mostly spent for reasoning about the issue report and (if needed) performing a keyword-based search in the knowledge base. Note that this effort does not include the effort needed to maintain the knowledge base. Therefore, the actual amortized manual effort is expected to be higher than 30 seconds. IssueTAG, on the other hand, requires no human intervention to make an assignment once an issue report has been created.

To evaluate the effect of the deployed system as well as the improvements made in the issue assignment process, we compute and compare the *solution times* before and after the deployment of IssueTAG. In particular, we define the solution time for

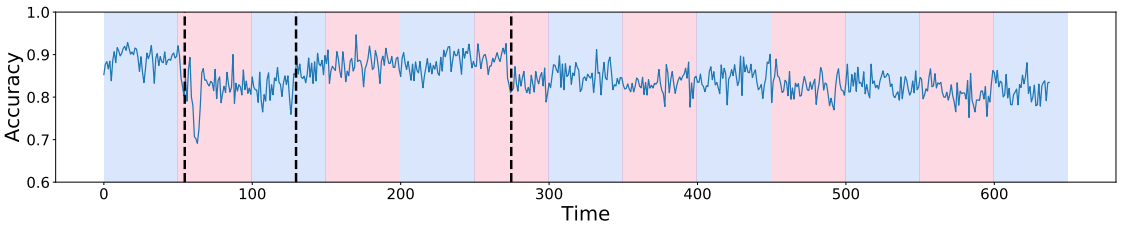


Figure 5.7 Daily assignment accuracies achieved between December 2016 and June 2019. The time point 0 represents the date, on which the manual issue assignment process started. The vertical dashed lines represent the points in time where a shift in daily accuracies was automatically detected by our change point detection approach (Section 7). IssueTAG was deployed at the third dashed line, i.e., all the accuracies before this line were obtained by manual assignments, whereas those after were obtained by automated assignments. The first dashed line represents the date, on which significant changes in team responsibilities occurred due to migrating certain functionalities from mainframes to state-of-the-art platforms. Therefore, the time gap between the first and second dashed lines (i.e., about 2.5 months) represent the amount of time it took for the IT-HD clerks to adapt to these changes.

an issue report as the time passed between the report is opened and it is closed. The shorter the solution times, the better the proposed approach is. Furthermore, as the characteristics of the reported issues, thus the solution times, can change over time, we, in the evaluations, compute and compare the solution times for the issue reports that were opened within two months before and after the deployment of IssueTAG.

Data and Analysis

Figure 5.7 presents the daily assignment accuracies achieved between December 2016 and June 2019. The time point 0 in this figure represents the date, on which the manual issue assignment process as it is described in Chapter 5.1, was started. Furthermore, the vertical dashed lines in the figure represent the points in time where a shift in daily accuracies was automatically detected by the change point detection approach we had developed (Section 7). IssueTAG was, indeed, deployed exactly at the 273^{th} time point where the third vertical dashed line resides. That is, all the accuracies before this dashed line were obtained by manual assignments, whereas those after were obtained by automatic assignments. The other vertical dashed lines will be discussed below in this section.

We first observed that after IssueTAG was deployed, the daily assignment accuracies dropped slightly (Figure 5.7). More specifically, the average daily accuracies before and after the deployment were 0.864 ($min = 0.691$, $max = 0.947$, $stddev = 0.040$) and 0.831 ($min = 0.752$, $max = 0.912$, $stddev = 0.027$), respectively.

We, however, observed that the accuracy of an automated issue assignment system does not have to be higher than that of manual assignments in order for the system to be useful. First, we observed that IssueTAG reduced the manual effort required for the assignments. In particular, given that it takes an average of 30 seconds for an IT-HD clerk to assign an issue report to a development team and an average of $8,000$ issue reports are received on a monthly basis, IssueTAG has been saving 5 person-months yearly, on average ($8,000$ issue reports * 30 seconds = $240,000$ seconds per month = 5 person-months per year).

Second, we observed that the deployed system together with the process improvements we implemented, profoundly reduced the turnaround time for closing the issue reports. More specifically, the average solution times before and after the deployment were 3.26 days and 2.61 days, respectively.

Third, we observed that it can take quite a while for a human stakeholder to excel in the issue assignment task, which is, in deed, a problem, especially in the presence of high employee turn over rates. For example, the first vertical dashed line in Figure 5.7, represents the date on which an integral part of the core banking system was migrated from mainframes to state-of-the-art hardware and software platforms. As a result of this migration, the structure and the responsibilities of the related development teams changed significantly. In particular, the responsibilities of one development team working on mainframes were migrated to 3 development teams working on state-of-the-art platforms, which consisted of completely different software engineers. Evidently, the assignment accuracies were affected by this change; the daily accuracies dropped at the first vertical dashed line (i.e., 55th time point) and stayed low until the second vertical dashed line (i.e., the 130th time point). More specifically, the average daily accuracies obtained from the manual assignments before the first dashed line, in between the first and second dashed lines, and after the second dashed line until IssueTAG was deployed at the third dashed line were, 0.889 ($min = 0.825$, $max = 0.929$, $stddev = 0.024$), 0.819 ($min = 0.691$, $max = 0.900$, $stddev = 0.039$), and 0.879 ($min = 0.822$, $max = 0.947$, $stddev = 0.024$), respectively. That is, it took the IT-HD clerks about 2.5 months to adapt to the new development teams. Therefore, this time frame can be considered to be a lower bound on the amount of time a new hire would require to learn to make accurate assignments. It is a lower bound in the sense that only 19% of the issue reports were affected by the changes in the team responsibilities during the aforementioned period of time and that the IT-HD clerks already had a great deal of experience; for a new hire, everything will be new.

Note further that the 0th time point in Figure 5.7 represents the date, on which

Table 5.5 Survey questions used for evaluating IssueTAG.

No	Question	Type	Category
Q1	I know the business requirements that the system is supposed to meet.	Likert scale	requirements satisfaction
Q2	The system (as a software product) is reliable.	Likert scale	requirements satisfaction
Q3	The system is useful.	Likert scale	requirements satisfaction
Q4	The system reduces the solution times for issue reports.	Likert scale	product quality
Q5	The issue assignments made by the system are trustworthy.	Likert scale	product quality
Q6	The system is robust.	Likert scale	product quality
Q7	I recommend the system to other companies.	Likert scale	product quality
Q8	What do you like and don't like about the system? Do you have any suggestions for improvement?	open-ended	product quality

Jira was started to be used for storing and managing the issue reports. That is, IT-HD clerks had been making manual assignments before this date, but had different means of managing the reports, which explains the high daily assignment accuracies event at the 0th time point in the figure. As we didn't have any access to the issue databases maintained before the 0th time point, we used only the issue reports managed by Jira in this research.

5.5 User Study

In this section, we investigate whether the automation is perceived useful by the end-users.

To carry out the study, we created a survey by following a survey template frequently used at Softtech. It had a total of 8 questions from two categories: *requirement satisfaction* and *product quality*. The former category aims to evaluate the extent to which the deployed system meets its requirements, whereas the latter category aims to evaluate the quality of the final product. All questions, except for the last one, were Likert scale questions each with answer options: *no opinion*, *1 - strongly disagree*, *2 - disagree*, *3 - agree*, and *4 - strongly agree*. The last question was an open-ended question. Furthermore, for the Likert scale questions, we asked the participants to elaborate on their responses, if they had "disagreed" or "strongly disagreed." Table 5.5 presents the questions we used in the survey.

We conducted the survey on the AST members. We chose this group of stakeholders as the recipients of the survey because, being embedded in the development teams, they were the direct end-users of IssueTAG. That is, they, as the first recipients of

the issue reports, were the ones to validate whether the assignments were correct or not and to reassign them as needed. The IT-HD clerks, on the other hand, could not participate in the survey because they were not considered to be the end-users of the deployed system in the sense that they neither made use of the assignments automatically made by the deployed system nor had a control over them.

5.5.1 Evaluation

Experimental Setup

About half of the AST members (more specifically, 14 out of 30) voluntarily agreed to participate in the study. The participants filled out the survey online at their spare time.

Evaluation Framework

For the Likert scale questions, we use the frequencies and the average scores obtained to quantitatively analyze the results. The average scores were computed as the arithmetic average of the scores with the "no opinion" responses excluded. For the open-ended question, we present the answers we received (Table 5.7) and qualitatively discuss them.

Data and Analysis

The results of the survey strongly suggest that IssueTAG meets its business needs with high quality. Regarding the questions in the category of "requirements satisfaction," we observed that the majority of the participants thought IssueTAG was useful and reliable (Figure 5.8). More specifically, all of the participants "strongly agreed" or "agreed" to Q1, indicating that they knew the business requirements that IssueTAG was supposed to meet. And, 92.86% (13 out of 14) of the participants for Q2 and 78.57% (11 out of 14) of the participants for Q3, responded "agree" or higher. The average scores were 3.71, 3.21, and 3.69 (out of 4) for these questions, respectively.

Only 1 participant for Q2 and 2 participants for Q3 "disagreed." The comments that they provided as to why they disagreed are given in Table 5.6. Evidently, part of the reason was that these participants were unrealistically expecting to have perfect assignments (with no incorrect assignments) from the deployed system.

Regarding the other quality aspects of the system, 100% (14 out of 14) of the

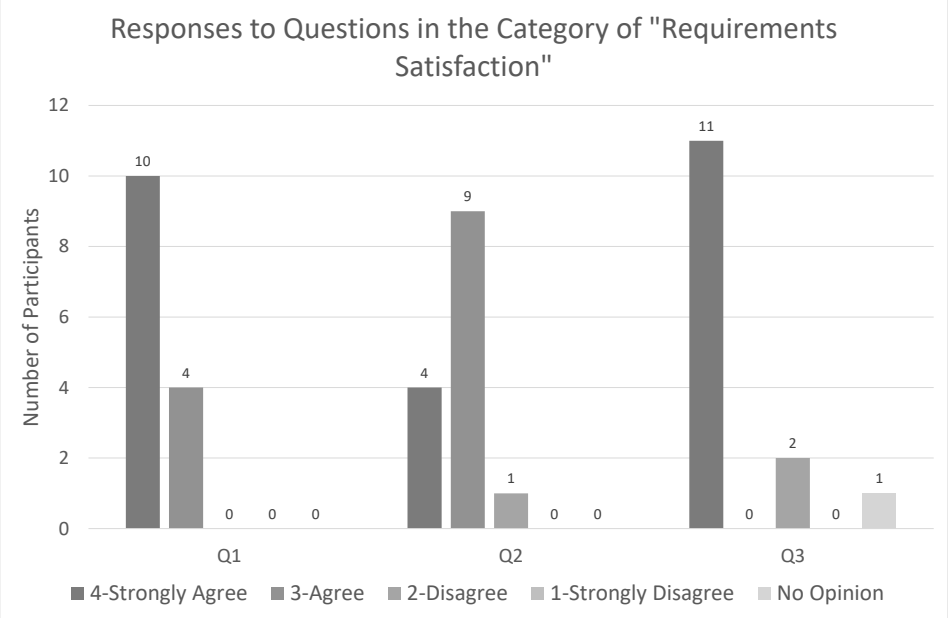


Figure 5.8 Responses to questions in the category of "requirements satisfaction."

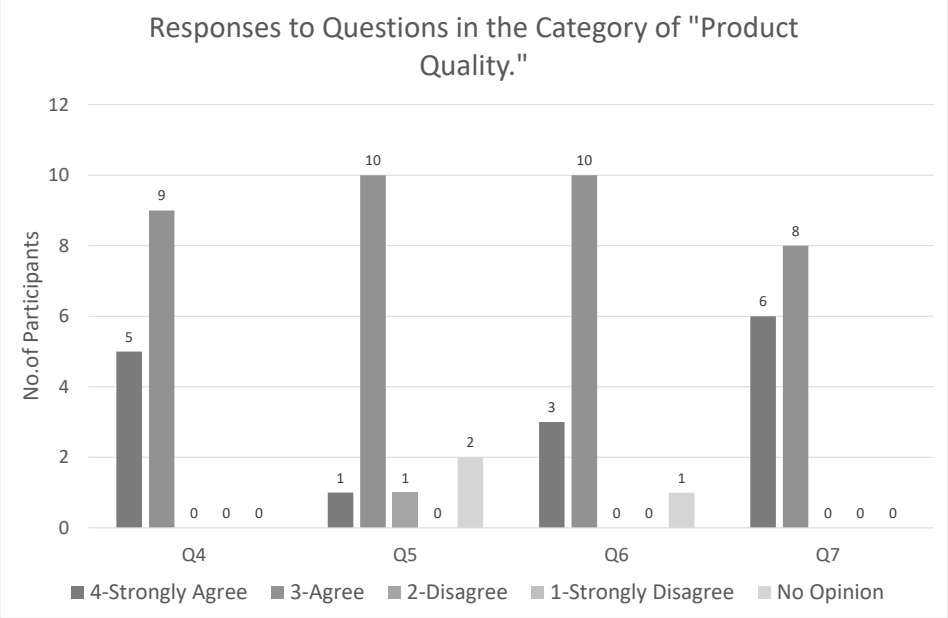


Figure 5.9 Responses to questions in the category of "product quality."

Table 5.6 Comments that the participants provided as to why they "disagreed."

Question No	Comment No	Comment
Q2	1	Due to some keywords [appearing in issue reports], the system sometimes make incorrect assignments to my team.
Q3	1	In fact, the application is useful. However, in the presence of a wrong assignment made by the system, reassigning the bug report to the correct team, especially when we don't know which team it should really be assigned to or when the other team refuses to take the responsibility for the issue report, causes delays in the process.
Q3	2	The system sometimes makes incorrect assignments.
Q5	1	I can't say "I agree" because I sometimes encounter incorrect assignments.

Table 5.7 Responses given to the open-ended question Q8.

Comment No	Comment
1	I think that the assignments are made rapidly and accurately. I have not been having any issues with the system. We [as a team] rarely receive incorrect assignments. I, however, believe that this is normal because the same words [terms] can be related with multiple development teams. It is quite normal for the system not being able to distinguish between teams in such situations.
2	Most of the time, the system works for us. Sometimes, however, it assigns irrelevant issue reports to my team.
3	General words, such as "problem", should not be used by the system when assigning issue reports to development teams.
4	The system profoundly reduced the loss of time by rapidly assigning issue reports to development teams with high accuracy.
5	I think that it takes a while for an AI algorithm to learn about new issue reports. Improvements can be made in this area.
6	I believe that the system had a profound effect on assigning the issue reports to the right development teams.
7	It is a nice and practical system, better results can be obtained with further development. One area for improvement could be to explain the keywords [terms] used for the assignments.

participants for Q4, 78.57% (11 out of 14) for Q5, 92.86% (13 out of 14) for Q6, and 100% (14 out of 14) for Q7 responded "agree" or higher (Figure 5.9). The average scores were 3.36, 3.0, and 3.0, and 3.43 (out of 4) for these questions, respectively. Only 1 participant disagreed with Q5, the comment of whose can be found in Table 5.6. We, furthermore, observed that all the participants would recommend the system to other companies; all responded "agree" or higher to Q7 (Figure 5.9).

Last but not least, the responses given to the open-ended question Q8 can be found in Table 5.7. All of these comments can be considered as generally positive. A couple of them actually make some suggestions for future improvements. For example, the last comment basically suggests that the system should provide an explanation as to why a given issue report is assigned to the selected development team. As a matter of fact, this request turned out to be a common one, for which we have developed an automated approach (Section 6).

5.6 Lessons Learnt

Stakeholders do not necessarily resist change. To deploy IssueTAG, we carried out a number of meetings with the IT-HD, AST, and software development teams. One thing we repeatedly observed in these meetings was that all the stakeholders, although they had some rightful concerns, such as what if the proposed approach adversely affects the issue-resolution process – a major concern for a company developing business-critical software systems – were actually willing to automate the process of issue assignments as much as possible.

We, indeed, had no objection at all. The AST members and the development teams were looking forward to reducing the turnaround time for issue resolutions; the incorrectly assigned issue reports were bouncing back and forth between the IT-HD clerks and the AST members, causing a great deal of wasted time. The IT-HD clerks were looking forward to 1) avoiding the costly and cumbersome process of maintaining a knowledge base about the development teams and their responsibilities and 2) deferring the responsibility of making assignments as much as possible since incorrect assignments were often causing friction with the AST members and the development teams.

Another reason behind the absence of any resistance was that none of the stakeholders felt threatened by the new system. The IT-HD clerks were still needed as they were the ones communicating with both the bank costumers and employees to collect the issues, resolving the ones that they could, and creating issue reports for the remaining ones. The AST members were still needed as they were the ones helping the development teams manage the issue reports. The development teams were still needed as they were the ones developing the software products.

Gradual transition helps stakeholders build confidence, facilitating the acceptance of the system. To address the rightful concerns of the stakeholders regarding the accuracy of the proposed system, we followed a gradual transition strategy. First, we simply added a single button to the screen, which the IT-HD clerks used to create the issue reports. We initially did not modify the assignment process at all in the sense that the use of this button was optional. If the IT-HD clerk chose to arm the button after creating an issue report, it would simply display the assignment made by IssueTAG. The clerk could then accept the assignment as it was or modify it. We observed that 3 months after the deployment of this button, enough confidence was built among the stakeholders to fully deploy the system.

It is not just about automating the issue assignments, but also about changing the process around it. One observation we made numerous times during the meetings with the stakeholders was that automating the issue assignments also requires to modify the other parts of the assignment process to improve the efficiency and effectiveness of the entire process to the extent possible. This was because most of the steps in the assignment process were dependent on the fact that issue assignments were made by the IT-HD clerks. Changing this, therefore, necessitated other changes. In particular, we prevented the IT-HD clerks from modifying the issue assignments made by IssueTAG and the incorrectly assigned issue reports from being returned back to the IT-HD clerks for a reassignment. All of these changes were based on the discussions we had with the stakeholders as well as the analysis of the results we obtained from a number of feasibility studies (Section 5.4.1).

The accuracy of the deployed system does not have to be higher than that of manual assignments in order for the system to be useful. Although the assignment accuracy of IssueTAG was slightly lower than that of manual assignments, it reduced the manual effort required for the assignments and improved the turnaround time for closing the issue reports. All of these helped improve the usability of IssueTAG, which was also evident from the survey we conducted on the stakeholders in the field (Section 5.5).

5.7 Concluding Remarks

In this chapter, we have developed and deployed a system to automate the process of issue assignments at Softtech/IsBank. To this end, we first cast the problem to a classification problem and determined the classifier to be used in the deployed system by empirically evaluating a number of existing classifiers, which are known to perform well for the problem at hand, on the actual database of issues maintained by the company. We then carried out further studies to determine both the amount and time locality of the historical data required for training the underlying classification models. We finally deployed the proposed system on Jan 12, 2018 by configuring it based on the results we obtained from these studies.

The system is fully operational since then, making automated assignments for all the issue reports submitted, which is about more than 250,000 automated assignments so

far. We observed that after IssueTAG was deployed, the daily assignment accuracies dropped slightly from 0.86 to 0.83. We, however, observed that the accuracy of an automated issue assignment system does not have to be higher than that of manual assignments in order for the system to be useful. First of all, IssueTAG reduced the manual effort required for the assignments by about 5 person-months per year. Second, it improved the turnaround time for resolutions by about 20%. More specifically, the average solution times before and after the deployment were 3.26 days and 2.61 days, respectively. And third, it slightly reduced the average number of issue tosses from 1.50 to 1.49.

The lessons we learnt are: 1) It is not just about deploying a data mining-based system for automated issue assignment, but also about designing/changing the assignment process around the system to get the most out of it; 2) The accuracy of the system does not have to be higher than that of manual assignments in order for the system to be useful, which was further validated by the user studies we carried out on actual stakeholders in the field; 3) Stakeholders do not necessarily resist change; and 4) Gradual transitions can help stakeholders build confidence, which, in turn, facilitates the acceptance of the system.

6. EXPLAINING ASSIGNMENTS (RQ2.1)

We observed that deploying a data mining-based approach for automated issue assignments, requires the development of additional functionalities. One functionality we needed, which we also did not foresee before the deployment of IssueTAG, was to create human-readable, non-technical explanations for the assignments made by the system. We address RQ2.1 in this chapter: "How can the assignments made by the system be explained to even non-technical staff?", to improve the usability of the automation.

Explaining the assignments made, was indeed a need we came to realize when we received several phone calls from the stakeholders shortly after the deployment of IssueTAG, demanding explanations as to why certain issue reports (especially, the incorrect-assigned ones) were assigned to them. Note that this is not a trivial task at all, especially when the underlying data mining models are not human readable. To this end, we have generated model-agnostic explanations (Ribeiro et al., 2016) and carried out a user study to evaluate the quality of these explanations (Section 6.2). To the best of our knowledge, explaining the predictions made have not been evaluated before in the context of issue assignment.

The chapter is organized as follows: Section 6.1 presents an approach for automatically generating explanations for the assignments; Section 6.2 evaluates our approach for explaining team assignments by conducting a user study; and Section 6.3 concludes the chapter.

6.1 Approach

One interesting observation we made after IssueTAG had been deployed was that, occasionally, especially for incorrect assignments, the stakeholders demanded some

explanations as to why and how certain issue reports had been assigned to their teams. This was an issue we didn't expect to face before deploying the system. As a matter of fact, based on the informal discussions we had with the stakeholders, we quickly realized that explaining the assignments could further improve the trust in IssueTAG.

In this section, we describe our approach for automatically generating explanations for the issue assignments made by the underlying classification model to answer if the issue assignments made by the underlying data mining model can be explained in a non-technical manner.

Note that since the classification models we use, namely the linear SVC models, are not human-readable, providing such explanations is a non-trivial task. To the best of our knowledge, there is, indeed, no work in the literature of automated issue assignment, addressing this problem.

One requirement we have is that the explanations should easily be interpreted and understood even by non-technical stakeholders as the recipients of these explanations are not necessarily technical stakeholders. Another requirement is that they should be given in terms of the natural language descriptions present in the issue reports, so that stakeholders can relate to them.

With all these in mind, we conjecture that providing a list of most influential (positive or negative) words for an issue assignment together with their relative impact scores as an explanation for the assignment, could help stakeholders understand the rationale behind the assignments.

Interestingly enough, we observe that such explanations could also be used in an interactive manner to enable the stakeholder creating the issue report to provide feedback to the classification model. Although such human-in-the-loop assignments are out of the scope of this work, we, nevertheless, added additional questions to our survey to evaluate the plausibility of the idea.

6.1.1 Local Interpretable Model-Agnostic Explanations

Our initial approach to explain the predictions made by IssueTAG was to find the most similar issue reports solved by the team in the past and explain the predictions by checking these similar reports and their classifications manually. Our manual explanation relied on using the words we find as most important. In other words,

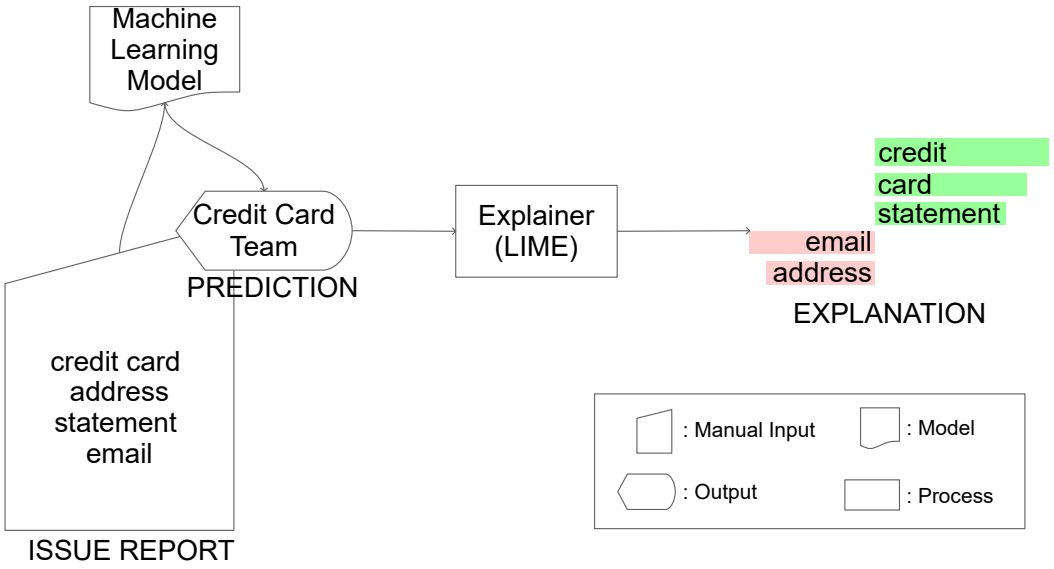


Figure 6.1 An example of explaining individual predictions. A machine learning model predicts that an issue report related to credit card domain is to be solved by the Credit Card Team and the explainer highlights the words leading to the prediction with their positive or negative effects on the prediction. With this output an expert on the domain can decide whether to trust the prediction or not.

we found samples similar to the given case and explained using the words in these samples. As these questions continued we realized that providing end users with a list of influential words (either in a positive or a negative manner) could help the users understand the rationale behind the predictions. An example of such a case is depicted in Figure 6.1. Using the input textual data, the machine learning based system assigns the issue report to the "Credit Card Team" and the explainer explains the prediction by providing a set of most influential terms mentioned in the textual data with their relative impact scores that either support the prediction (words with green labels) or oppose it (words with red labels).

We use Local Interpretable Model-Agnostic Explanations (LIME) to automatically produce explanations for the issue assignments made by IssueTAG. LIME is a model-agnostic algorithm for explaining the predictions of a classification or regression model (Ribeiro et al., 2016). In this work, we, (to the best of our knowledge) for the first time, use LIME in the context of automated issue assignment and evaluate it by carrying out a survey on actual stakeholders in the field. Next, we briefly describe the LIME algorithm without any intention to provide all the mathematics behind it. The interested reader can refer to (Ribeiro et al., 2016) for further details.

LIME, in our context, aims to identify a human-interpretable, locally faithful model,

which provides qualitative understanding between the terms used in issue reports and the development teams, to which they are assigned. In a nutshell, given an issue report, the assignment made for this report, and the underlying classification model, LIME first represents the report as a bag of words and samples instances around the report by drawing subsets of the words in the bag uniformly at random. Then, the samples are weighted by their proximities to the original issue report and fed to the classification model to label them. Next, all the samples together with their associated labels are used to learn a linear model comprised of K terms (in our case, $K = 6$), which distinguishes the labels. Finally, the linear model learnt is reported as an explanation for the assignment.

The explanation generated for an assignment is, indeed, a set of K terms selected from the original issue report together with their relative weights, indicating the influential terms that either contribute to the assignment or are evidence against it. Figure 6.5a presents an example explanation created for an assignment made by IssueTAG in the field. The vertical axis reports the most influential terms selected, whereas the horizontal axis denotes their relative weights. The terms with positive weights depict the terms that contribute to the assignment, wheres as those with negative weights depict the ones that are evidence against it. That is, in a sense, the former set of terms vote for the assignment, whereas the latter ones vote against it in an attempt to change the assignment.

LIME uses an interpretable representation of the original input that is understandable to humans. For example, an issue report may be represented with word embeddings as input for the classification algorithm, but LIME uses a binary vector indicating the absence or presence of a word. In other words, the original issue report may be represented as $x \in \mathbb{R}^d$, LIME represents it as $x\prime \in \{0,1\}^{d'}$.

The model to be explained is denoted by $f : \mathbb{R}^d \rightarrow \mathbb{R}$. Each prediction is explained separately and $f(x)$ is the probability that x belongs to a specific class. The approach is model agnostic since it only uses the outputs provided by f. $g \in G$ is an explanation model, where G is a set of models that are interpretable, such as linear models or decision trees. So, g is a model that can be presented to users with visual and textual methods. The domain of g is $\{0,1\}^{d'}$, in other words it uses the interpretable inputs.

To define locality around x, $\pi_x(z)$ is defined as the proximity measure between issue report z to x. $\Omega(g)$ is defined as a measure of complexity of g as opposed to interpretability. $\mathcal{L}(f, g, \pi_x)$ is a measure of faithfulness of g in approximating f around the locality defined by π_x, the greater $\mathcal{L}$ the more unfaithful g. So as to ensure that the model is locally faithful and interpretable, $\mathcal{L}$ and Ω must be

minimized and the following formula is obtained to get an explanation ξ for issue report x:

$$\xi(x) = \underset{g \in G}{\arg\min} \ \ \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{6.1}$$

The intention is to have a model-agnostic explainer. In order to learn the behaviour of f around x, samples are drawn weighted by π_x to approximate $\mathcal{L}(f, g, \pi_x)$. Instances around $x\prime$ are sampled by drawing non-zero elements of $x\prime$ uniformly at random. Then, $f(z)$ is obtained for each sample and used as a label for the explanation model. Given the set $\mathcal{Z}$ and the related labels, the above equation is optimized to get an explanation $\xi(x)$.

We define $\pi_x(z)$ as an exponential kernel defined on some distance function $\mathcal{D}$ with width σ:

$$\pi_x(z) = \exp\left(\frac{-\mathcal{D}(x, z)^2}{\sigma^2}\right) \tag{6.2}$$

And $\mathcal{L}$ is defined as a weighted square loss:

$$\mathcal{L}(f, g, \pi_x) = \sum_{z, z\prime \in \mathcal{Z}} \pi_x(z)(f(z) - g(z\prime))^2 \tag{6.3}$$

Note that a limit K is set on the number of words used for an interpretable representation which is constant.

As a result, given the textual features of an issue report, we obtain a local explanation for the prediction. The explanation is a list of the most influential words for the prediction made, as shown in Figure 6.1. The weights of the words indicate their additional effect on the prediction probability of that class for that issue report.

6.2 Experiments

We could not simply ask the stakeholders to evaluate each and every explanation created for the issue assignments, which were of interest to them. The reason was that there was a large number of issue reports submitted on a daily basis (Section 5.4)

Table 6.1 Survey questions related to selected issue reports and their explanations.

No	Question	Type
Q1	Is the explanation helpful in understanding the assignment?	Yes/No
Q2	Given the issue report, the assignment, and the explanation for the assignment, how would you rate the trustworthiness of the assignment?	Likert scale
Q3	Which terms in the explanation make you think that the assignment is not trustworthy?	Open ended
Q4	What are the, additional terms that you would like to see in the explanation; or the terms, the impact factor of which you would like to increase in the explanation, before you can trust the assignment?	Open ended

and that checking out the explanations was optional, i.e., the stakeholders were not required to have a look at the explanations. Therefore, forcing the stakeholders to evaluate the explanations as the issue assignments were made, could have adversely affected their performance.

6.2.1 Experimental Setup

We, therefore, carried out an empirical study with the same participants involved in our survey in Section 5.5, after having their consensus to voluntarily participate in this additional study, which were accepted by all of them.

For each participant, we randomly picked 10 issue assignments, which were handled by the participant in the last week before the study, such that the ratio of correctly and incorrectly assigned issue reports roughly resembled the average daily assignment accuracy. When there were less than 10 issue assignments for a participant, we selected all of the available ones. All told, we picked a total of 130 issue assignments (10 for each participant, except for two, for whom we could have only 5 assignments each). Out of all the selected assignments, 13 (10%) were incorrect.

We then created a questionary for each participant by using the issue assignments selected for the participant. For each assignment in the questionary, we included 1) the issue report, 2) the assignment made by IssueTAG, 3) the explanation automatically created by the proposed approach, using the 6 most influential terms involved in the assignment, and 4) four questions (Table 6.1).

The first two questions, namely Q1 and Q2, were directly concerned with our main research question in this study, i.e., whether or not the automatically generated explanations could help stakeholders understand the rationale behind the assignments. Q1 was a "yes" or "no" question, whereas Q2 was a Likert scale question with answer options: *2 - very trustworthy*, *1 - trustworthy*, *0 - not sure*, *-1 - untrustworthy*,

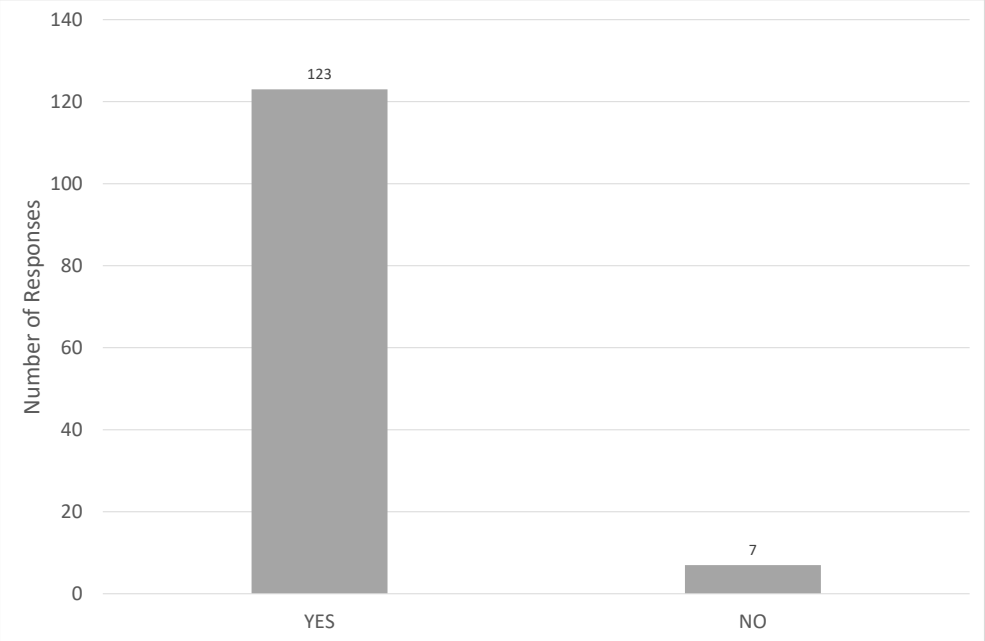


Figure 6.2 Responses to Q1: "Is the explanation helpful in understanding the assignment?"

-2 - very untrustworthy. The last two questions, namely Q3 and Q4, on the other hand, aimed to evaluate the plausibility of using the explanations to get feedback from the stakeholders in an attempt to further improve the assignment accuracies. These questions were open-ended questions, which were conditional on Q2; the participants were asked to answer these questions only when the response to Q2 was either "untrustworthy" or "very untrustworthy."

All the explanations were created by using the `LIME` Python library (Ribeiro et al., 2016) with $K = 6$ – a decision we made based on the maximum number of terms that we thought a stakeholder could efficiently and effectively reason about.

For Q1 and Q2, we use the frequencies of responses to quantitatively analyze the results. For Q3 and Q4 (when answered), we manually investigate how the feedbacks can be used to further improve the accuracies.

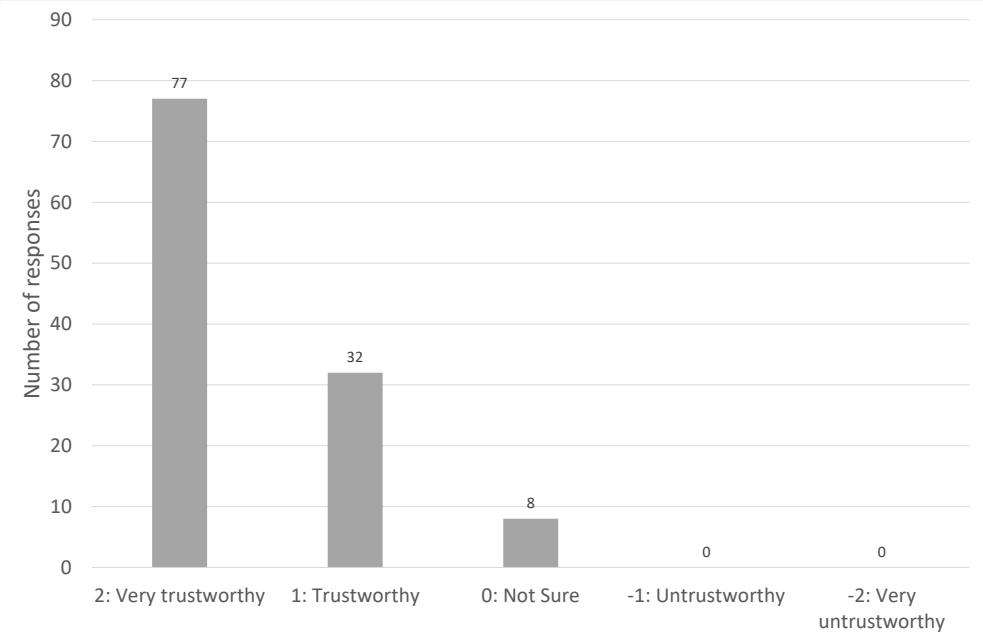


Figure 6.3 Responses to Q2 (for the correct assignments): "Given the issue report, the assignment, and the explanation for the assignment, how would you rate the trustworthiness of the assignment?"

6.2.2 Data and Analysis

Regarding Q1, we observed that participants found 95% (123 out of 130) of the explanations, each of which was created for a distinct issue assignment, helpful in understanding the rationale behind the assignments (Figure 6.2).

Regarding Q2, based on the explanations created for the correct assignments, the participants found 93% of the assignments (109 out of 117) "trustworthy" or "very trustworthy" (Figure 6.3). And, for the remaining 7% of the assignments (8 out of 117), they were "not sure" whether the explanations helped them decide if the assignments were reliable or not. None of the assignments was found "untrustworthy" or "very untrustworthy."

Interestingly enough, based on the explanations created for the incorrect assignments, we observed that the participants found 77% of the assignments (10 out of 13) "trustworthy" or "very trustworthy," suggesting that given the same issue reports, these participants would have made the same or similar mistakes in assigning the reports. We believe that this was because of some missing information in these issue reports, which was required for accurate assignments (Figure 6.4). Further-

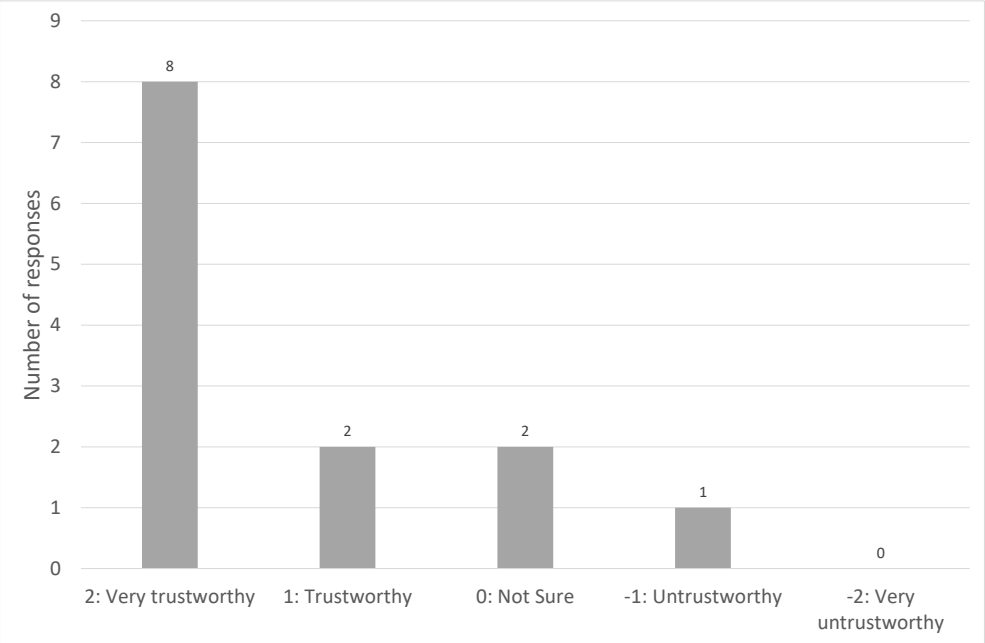
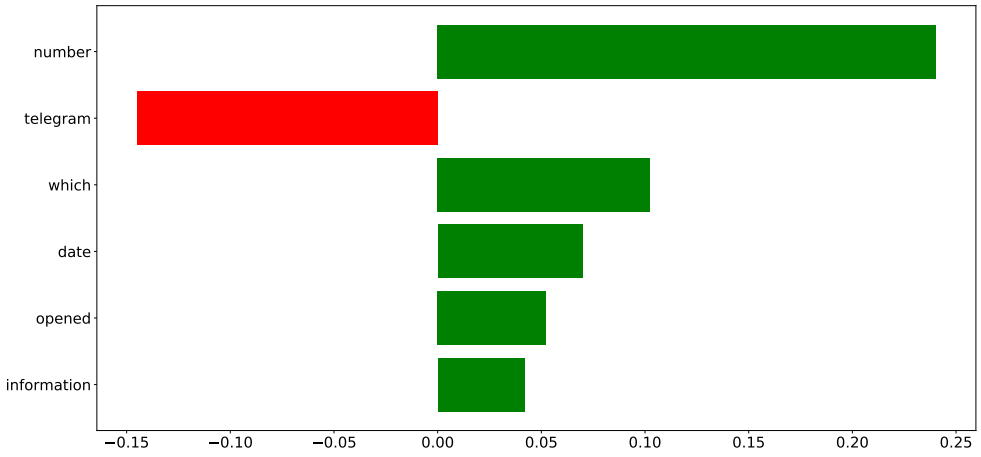
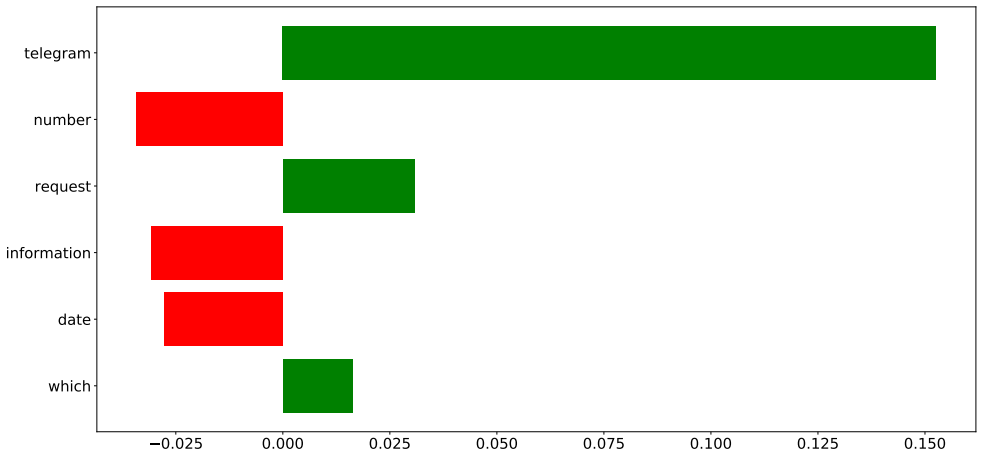


Figure 6.4 Responses to Q2 (for the incorrect assignments): "Given the issue report, the assignment, and the explanation for the assignment, how would you rate the trustworthiness of the assignment?"

more, the participants were "not sure" about the trustworthiness of the 15% of the assignments (2 out of 13).



(a)



(b)

Figure 6.5 The explanations created for the assignment marked as "untrustworthy" by a participant: a) the explanation created for the original assignment, which was incorrect and b) the explanation created for the second likely assignment, which was correct.

Regarding Q3 and Q4, among all the responses given to Q2, only one was scored below 0. That is, based on the explanations created for the assignments, only one of the assignments was found "untrustworthy." And, this assignment was, indeed, an incorrect assignment made by IssueTAG.

The explanation created for the aforementioned assignment is given in Figure 6.5a. Given this explanation, the participant argued in her response that the term "tele-

gram," which is a domain specific term used when creating a credit account, was an important term for the issue report at question. Therefore, it should have positively, rather than negatively, affected the assignment. As a matter of fact, this argument was also well-aligned with the automatically generated explanation given in Figure 6.5a in the sense that "telegram," being a term with a large negative impact, voted against the assignment in an attempt to change it. It was, however, not strong enough to modify the outcome.

Interestingly enough, Figure 6.5b presents the explanation created for the second likely assignment made by the underlying classification model, which turned out to be the correct assignment. Note that in this assignment, the term "telegram" had the largest positive impact on selecting the correct team, which was also suggested by the stakeholder. Therefore, had the participant presented with the explanations created for the top two most likely assignments, she could have selected the second assignment, thus increased the assignment accuracy. Note that the aforementioned type of approaches are beyond the scope of this work. However, as the results of this study are promising, we, as a future work, plan to develop "human-in-the-loop" approaches, which leverage the automatically created explanations to further improve the assignment accuracies.

6.3 Concluding Remarks

Stakeholders may demand some explanations as to why certain issue reports (especially the incorrectly assigned ones) have been assigned to their teams. Note that since the data mining models used for predicting the assignments are not necessarily readable and interpretable by human beings (as was the case in this work), generating such explanations can be a non-trivial task. To this end, we have developed a LIME-based (Ribeiro et al., 2016) approach for automatically generating explanations that can easily be interpreted even by non-technical stakeholders.

To evaluate the explanations we conducted a survey with the AST members at Softtech. We selected 130 issue report assignments such that each participant was responsible in the past and presented the issue reports with their explanations to the participants. We asked them if the explanations were helpful and assignments were trustworthy. It turned out that the explanations were helpful in understanding the assignments. Based on the explanations created for the correct assignments,

the participants found 93% of the assignments "trustworthy", while for the incorrect assignments, we observed that the participants found 77% of the assignments "trustworthy", suggesting that given the same issue reports, these participants would have made the same or similar mistakes in assigning the reports.

In our survey, based on the explanations created for the assignments, only one of the assignments was found "untrustworthy." and, this assignment was, indeed, an incorrect assignment made by IssueTAG. We observed that, if the participant was presented with the explanations created for the top two most likely assignments, she could have selected the correct assignment, and thus increased the assignment accuracy.

7. MONITORING DETERIORATIONS (RQ2.2)

Another functionality we needed was to monitor the assignment accuracy of the system and detect deteriorations in an online manner, so that corrective actions can be taken in time. We address RQ2.2 in this chapter: "How can the deteriorations in assignment accuracies be automatically detected?", related with improving the usability of the automation.

As mentioned earlier, IssueTAG is a live system and the dataset that we use is evolving continuously since the development process is evolving continuously. Therefore, the deteriorations in accuracies have to be monitored so as to recalibrate the models as needed. To this end, we have developed a *change point detection*-based approach.

The chapter is organized as follows: Section 7.1 describes the change point detection-based approach for detecting deteriorations in assignment accuracies; Section 7.2 experiments on whether the approach would detect the deteriorations or not by varying the nature of them; and Section 7.3 concludes the chapter.

7.1 Approach

Whether the deteriorations in the assignment accuracies can be automatically detected in an online manner, was indeed another issue we faced after the deployment of IssueTAG. It is important because such a mechanism not only increases the confidence of the stakeholders in the system, but also helps determine when the underlying classification model needs to be recalibrated by, for example, retraining the model.

One observation we make is that every issue report at Softtech is closed by the development team, who has fixed the reported issue. Therefore, in the presence of an incorrect assignment made by IssueTAG, the report is reassigned and the history

of the reassignments is stored in the issue tracking system. Consequently, at any point in time, the assignment accuracy of IssueTAG can automatically be computed using the history of the issue reports that have been closed. Therefore, deteriorations in the accuracy can be analyzed in an online manner.

7.1.1 Pruned Exact Linear Time (PELT)

To this end, we use an online change point detection approach, called Pruned Exact Linear Time (PELT) (Killick et al., 2012). In a nutshell, PELT is a statistical analysis technique to identify when the underlying model of a signal changes (Truong et al., 2020). In our context, we feed PELT with a sequence of daily assignment accuracies as the signal. The output is a set of points in time (if any) where mean shifts. PELT, being an approach based on dynamic programming, detects both the number of change points and their locations with a linear computational cost under certain conditions (Killick et al., 2012). Further information can be found in (Killick et al., 2012) and (Truong et al., 2020).

PELT has been used for change point detection in many application domains, including DNA sequence data, financial time series, and oceanographic data (Hocking et al., 2013; Killick et al., 2012; Lavielle & Teyssière, 2007). In this work, we, on the other hand, use it (to the best of our knowledge) for the first time in the context of automated issue assignment to detect the deteriorations in the assignments made by a data mining model.

For each day i, we retrieve the issue reports created on that day and their related historical data, such as updates on the responsible software team assigned to solve the report. Using the historical data, if we find that there has been a class change for an issue report, we take that report as wrongly classified. If there has been no change in its class, we take that report as correctly classified. We calculate the daily accuracy Υ_i for the day i, by dividing the number of correctly classified issue reports created that day to the total number of issue reports created that day.

PELT aims to identify when the underlying model of a signal changes (Truong, Oudre & Vayatis, 2018). We use Υ_i as the input signal for which we want to detect mean shifts and select *PELT (Pruned Exact Linear Time)* as the algorithm to detect these shifts. The main reasons to select PELT is, it estimates the number and locations of the change points when their number is unknown and it searches for the exact solution with a linear computation cost under mild conditions. Killick et al.

(2012) introduced PELT, which is based on the dynamic programming approach of Jackson et al. (2005), and it involves a pruning step within the dynamic program. Since our aim is to find the change points without information about their number K, our problem is defined as follows (Truong et al., 2020):

$$\min_{\tau} V(\tau, \Upsilon) + pen(\tau) \tag{7.1}$$

Here, τ denotes the set of indexes defining the segmentation of Υ regarding the change points: $\tau = \{t_0, t_1, t_2, \ldots\ t_{K+1}\}$. Number of change points is penalized with *pen(τ)* and the function *V(τ, Υ)* is the sum of costs of all the segments that define the segmentation as given below:

$$V(\tau, \Upsilon) = \sum_{k=0}^{K} c(\Upsilon_{t_k \cdots t_{k+1}}) \tag{7.2}$$

The cost function $c(\cdot)$ is used to measure the homogeneity of the sub-signal $\Upsilon_{t_k \cdots t_{k+1}}$. Intuitively, $c(\cdot)$ is expected to be low if the sub-signal doesn't contain any change points. We use least squared deviation as the cost function, c_{L_2}, which detects mean-shifts in a signal:

$$c_{L_2}(\Upsilon_{a..b}) = \sum_{t=a+1}^{b} \| \Upsilon_t - \bar{\Upsilon}_{a..b} \|_2^2 \tag{7.3}$$

Here, $\bar{\Upsilon}_{a..b}$ is the empirical mean of the sub-signal $\Upsilon_{a..b}$.

Optimal Partitioning method solves the problem with fixed number of change points, $K \geq 1$, and relies on the additive nature of $V(\cdot)$ to recursively solve sub-problems (Truong et al., 2020). The following recursion is solved, which implies that the first change point is easily computed if the optimal partitions with $K-1$ elements of all sub-signals $\Upsilon_{t..T}$ are known:

$$\min_{|\tau|=K} V(\tau, \Upsilon = \Upsilon_{0..T}) = \min_{t<T-K} [c(\Upsilon_{0..T}) + \min_{|\tau|=K-1} V(\tau, \Upsilon_{t..T})] \tag{7.4}$$

The problem when the number of change points is not known can be solved with a naive approach: Apply the *Optimal Partitioning* method for $K = 1, ..., K_{max}$ for a large K_{max}, then choose the segmentation that minimizes the penalized problem. However, this solution has a quadratic complexity. PELT algorithm solves the problem much faster, given that the penalty is linear: $pen(\tau) = \beta|\tau|$, where $\beta > 0$ is a smoothing parameter (Truong et al., 2020). PELT algorithm removes those values of τ that can never be the minima from the set of potential change points. Formally, if

$$[\min_{\tau} V(\tau, \Upsilon_{0..T}) + \beta|\tau|] + c(\Upsilon_{t..s}) \geq [\min_{\tau} V(\tau, \Upsilon_{0..s}) + \beta|\tau|] \tag{7.5}$$

holds, then t cannot be the last change point prior to T (Truong et al., 2020).

When using PELT, the only parameter to calibrate is the penalty level. Small penalty levels encourage the detection of more change points, while large penalty levels cause the detection of only significant changes (Truong et al., 2020).

7.2 Experiments

We applied the PELT approach to the daily assignment accuracies collected from the field. PELT detected three change points, each of which was depicted by a vertical dashed line in Figure 5.7. It turned out that these change points, indeed, coincided with some important events that affected the assignment accuracies, validating the results obtained from the proposed approach. The first dashed line represents the date, on which significant changes in the team responsibilities occurred due to migrating certain functionalities from mainframes to state-of-the-art platforms. The time gap between the first and second dashed lines (i.e., about 2.5 months) represent the amount of time it took for the IT-HD clerks to adapt to these changes. And, the third dashed line represents the date on which IssueTAG was deployed. Further discussion on these change points can be found in Section 5.4.

We observed that PELT did not detect any other change point after IssueTAG was deployed. We believe that this was because the underlying classification model had been regularly retrained at every month as a part of Softtech's policy by using the issue reports submitted in the last 12 months before the calibration (Section 5.3.3).

To further evaluate the proposed approach, we, therefore, carried out additional experiments where we systematically varied the nature of the deteriorations and evaluated whether the proposed approach detected them or not. Note that controlling the nature of the deteriorations in this study allows us to reliably evaluate the results, because when the true nature of a deterioration, such as the exact point in time at which the deterioration occurred, is not known, which is typically the case with the data collected from the field, the analysis may suffer from the lack of ground truth. Note further that even if the underlying classification model is regularly trained, monitoring for deteriorations is still relevant as the assignment accuracies can still deteriorate in between the calibrations.

7.2.1 Experimental Setup

In each experimental setup, we used an ordered sequence of 200 daily assignment accuracies. The first 100 of these accuracies came from a normal distribution representing the accuracies expected from IssueTAG, whereas the remaining 100 accuracies came from a distribution (or a number of distributions) representing a deterioration. That is, the change point in each experiment was the 100th time point as the deterioration was introduced after this point in time.

For each experimental setup, we then mimicked the real-life operations of IssueTAG. More specifically, given a sequence of 200 daily assignment accuracies, we fed them to the proposed approach one daily accuracy after another in the order they appeared in the sequence. After every daily accuracy, a decision was made whether a deterioration had occurred, and if so, when. We finally determined how long it took for the proposed approach to detect the deterioration. For each experimental setup, we repeated the experiments 1000 times.

As an implementation of the PELT approach, we used the `ruptures` Python library (Truong et al., 2018). As the *penalty level*, i.e., the only parameter to calibrate in PELT, we used the empirically determined value of 0.05. The penalty level is a mechanism used for guarding against overfitting, determining to which extent a shift in the accuracies should be considered as a change point. The larger the penalty level, the fewer (and more significant) change points are detected.

To model the daily accuracies expected from the system, we used a normal distribution with mean of 0.85 and standard deviation of 0.025 (i.e., $\mu = 0.85$ and $\sigma = 0.025$), mimicking the daily accuracies of the deployed system observed in the field (Sec-

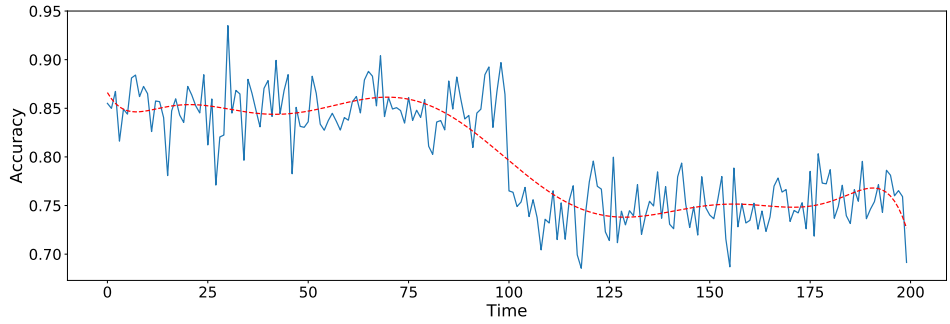


Figure 7.1 An example sequence of daily assignment accuracies showing a sudden 10-point deterioration at the 100th time point.

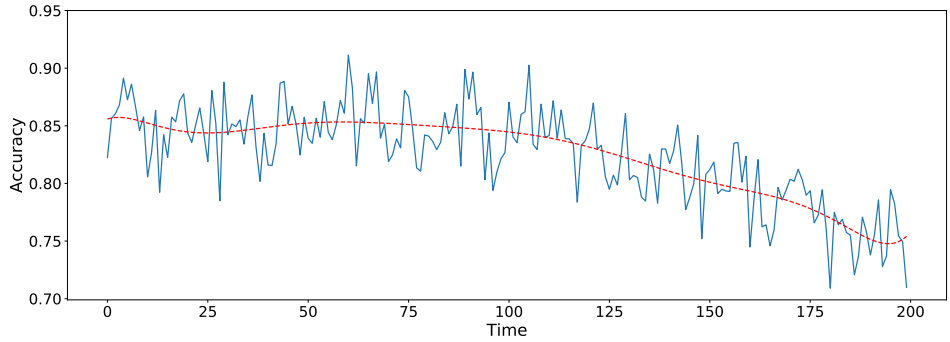


Figure 7.2 An example sequence of daily assignment accuracies showing a gradual 10-point deterioration starting from the 100th time point.

tion 5.4). To model the deteriorations, we experimented with two types of changes: *sudden deteriorations* and *gradual deteriorations*. In either case, we used 5-, 10-, 15-, and 20-point drops in daily accuracies, such that the mean accuracy (i.e., the mean of the distribution, from which the accuracies were drawn) eventually became 0.80, 0.75, 0.70, and 0.65, respectively.

For the sudden deteriorations, we abruptly dropped the mean accuracy from 0.85 to the requested level (i.e., 0.80, 0.75, 0.70, or 0.65, depending on the choice) right after the change point at the 100th time point and kept it intact until and including the 200th time point (i.e., until the end of the experiment). Figure 7.1 presents an example sequence of daily assignment accuracies showing a sudden 10-point deterioration.

For the gradual deteriorations, on the other hand, the changes were obtained by linearly dropping the mean accuracy starting from right after the change point at the 100th time point until and including the 200th time point, such that the mean accuracy at end of the experiment became 0.80, 0.75, 0.70, or 0.65, depending on the choice. For example, if the requested level of accuracy was 0.80, then starting from the mean accuracy of 0.85, the mean accuracy would be dropped by 0.05-point each

day (5-point drop/100 days) until it would become 0.80 at the 200th time point. Figure 7.2 presents an example sequence of daily assignment accuracies showing a gradual 10-point deterioration starting from the 100th time point.

To evaluate the proposed approach, we first determine whether the deteriorations are detected or not. If so, we measure *detection time* as the number of days past after the change point (i.e., after the 100th time point) until the deterioration is detected. The smaller the detection time, the better the proposed approach is.

7.2.2 Data and Analysis

Table 7.1 presents the data we obtained on the sudden deteriorations used in the study. We first observed that the proposed approach detected all the deteriorations. We then observed that as the deterioration amount increased, the detection time tended to decrease, i.e., the proposed approach tended to detect the deteriorations faster. On average, the deteriorations were detected in 1.33, 1.60, 1.84, 2.67 days after there was a 20-, 15-, 10-, and 5-point sudden drop in the mean assignment accuracies, respectively.

Table 7.2 presents the data we obtained on the gradual deteriorations. As was the case with the sudden deteriorations, the proposed approach detected all the deteriorations and as the deterioration amount (thus, the deterioration rate) increased, the detection time tended to decrease. Compared to the sudden deteriorations, however, the detection times for gradual deteriorations increased, which is to be expected. To better evaluate the quality of the detections, we, therefore, analyzed the mean accuracies that were present when the deteriorations were detected. We observed that throughout all the experiments, the proposed approach detected the deteriorations before the mean accuracy dropped more than 5-points (the last column in Table 7.2).

Table 7.1 Results obtained on sudden deteriorations. The experiments were repeated 1000 times.

	Detection Time			
Deterioration	min	avg	max	stddev
5-point	1	2.67	6	1.13
10-point	1	1.84	3	0.38
15-point	1	1.60	2	0.49
20-point	1	1.33	2	0.47

Table 7.2 Results obtained on gradual deteriorations. The experiments were repeated 1000 times.

	Detection Time				Minimum Mean Accuracy
Deterioration	min	avg	max	stddev	at the Point of Detection
5-point	1	31.03	55	13.64	0.8125
10-point	1	20.14	35	8.41	0.8070
15-point	1	15.70	26	6.49	0.8035
20-point	1	13.36	21	4.95	0.8000

7.3 Concluding Remarks

After deployment, we observed that deploying a data mining-based automated issue assignment system requires the development of additional functionalities. When the issue assignments are automatically made by using a data mining model, the accuracy of the assignments needs to be monitored and deteriorations need to be detected in an online manner, so that corrective actions, such as recalibrating the underlying model, can be taken in time. To this end, we have developed a change point detection-based approach using PELT (Killick et al., 2012).

By varying the nature of the deteriorations, we experimented on whether PELT would detect them or not. We used an ordered sequence of daily assignment accuracies as input. We fed them to PELT one daily accuracy after another in the order they appear in the sequence. To model the deteriorations, we experimented with two types of changes: sudden deteriorations and gradual deteriorations. In either case, we used 5-, 10-, 15-, and 20-point drops out of 100 points in daily accuracies. We observed that throughout all the experiments, the proposed approach detected the deteriorations before the mean accuracy dropped more than 5-points.

8. DETECTING MISSING INFORMATION IN ISSUE REPORTS (RQ3.1)

We use one-line summary and description attributes of the issue report for automated assignment in production. So, we expect that especially the description attribute to be high quality for good classification. One observation we made in our studies to further improve the issue triage process was that automated detection of reports that are not describing actual bugs (Non-Bug), and automated detection of incomplete information (missing OB, EB and S2R) could improve the resolution process by guiding the reporters at the time of submission. To improve the effectiveness of IssueTAG, we address RQ3.1 in this chapter: "How can the missing information in manually written reports be identified?"

Timely resolution of issue reports is important for both improving the quality of software products and customer satisfaction. Developers devote a significant amount of time in diagnosing and resolving these reports. However, most of the time the textual descriptions of issue reports lack enough information. Developers then ask questions in comments to better understand the expected behaviour, to obtain a screenshot or the error message observed, when the problem was observed, etc. to clarify any ambiguity. In fact, such information is expected to be already provided in detail in the descriptions of the issue reports in an unstructured form as *Observed Behaviour (OB)*, *Expected Behaviour (EB)* and *Steps to Reproduce (S2R)*.

In fact, developers report *incomplete information* as the most common problem in issue report descriptions (Zimmermann et al., 2010). Steps to reproduce, version numbers, observed and expected behaviour are other common problems, which can be considered as other types of incomplete information. Checking the existence of OB, EB and S2R in issue report descriptions automatically is not a new idea. Chaparro et al. (2017a) developed an automated approach to detect the missing information in issue report descriptions (Note that in this work we prefer to use the term *issue reports* since not all the reports we use are actual bugs). They first manually analyzed 2,912 issue reports, which are actual bugs, from nine software systems, namely Docker, Eclipse, Facebook, Firefox, Hibernate, Httpd, LibreOffice,

OpenMRS, Wordpress-A, and found out that 93.5% of them contain OB, 35.2% contain EB and 51.4% contain S2R information in descriptions. They further cataloged 154 discourse patterns that reporters describe the OB, EB and S2R, which are either sentence-level or paragraph-level. The machine learning based approach they proposed to detect the missing information was the most accurate and detected missing EB with 0.89 F-measure and missing S2R with 0.75 F-measure. Our work improves the work of Chaparro et al. (2017a) by introducing the use of morphological analysis of the words in issue report descriptions and carrying out the study in an industrial setting. They use part-of-speech tags to detect the missing information in descriptions, while we use morphological analysis to extract the root words and suffixes to be used as input for the detection of the missing information.

We first manually analyze 1,200 issue reports written in Turkish, an agglutinative language, in banking domain. Other than actual bugs, where the OB, EB and S2R patterns are observed, we occasionally come across reports where we would classify them as Non-Bug (NB). Note that we categorize these reports as NB, only by checking the textual description where no sign of a bug can be found in the description. In fact, the reporters either ask a question or convey an operational request in such reports most of the time. As a result, after the manual analysis of the reports, we extract 40 discourse patterns that are signs of NB, OB, EB or S2R.

Some of these patterns contain words that can be used as keywords for automatic detection of NB, OB, EB or S2R, but some of them need to be analyzed morphologically, since Turkish is an agglutinative language. The equivalent of whole clauses and sentences is encoded in a single word bu adding multiple suffixes to root words in agglutinative languages such as Turkish, Finnish, Korean (Oflazer, 1994, 2014). Morphological analysis of a word is to represent it as a sequence of tags that correspond to morphemes, or the word's smallest components (Oflazer, 2014).

To understand the need for the morphological analysis of the words, we provide some examples in Table 8.1. These example sentences are provided in the work of Chaparro et al. (2017a) with the keyword that describes the pattern written in bold. We also provide their Turkish translations and the key term (in bold) corresponding to the keyword in English. In the last column of the table, we provide the morphological analysis of the verbs in these sentences and emphasize the key term that describes the pattern again in bold.

Table 8.1 Some example patterns and morphological analysis

Example sentence	Pattern	Turkish translation	Morphological analysis of the verb
The icon **did not** change to an hour glass.	OB: Negative auxiliary verb	Simge kum saati olarak değiş**me**di.	değiş-**me**-di -> Verb Root + **Negative** + Past Tense
Apache **should** make an attempt to print the date in the language requested by the client.	EB: Should	Apache, tarihi istemcinin istediği dilde yazdırmaya çalış**malı**dır.	çalış-**malı**-dır -> Verb root + **Obligative** + Generalizing Modality Marker
When saving a new (transient) entity ..., Hibernate will generate [at least] two INSERT statements...	S2R: conditional sentence with non-negative OB predicate	Yeni (geçici) bir varlığı kaydeder**ken** ..., Hybernate [en az] iki INSERT ifadesi oluşturur...	kaydet-er-**ken** -> Verb root + Present Tense + **Converbial Marker**

In the first example, the term *did not* describes the OB behaviour. However, to automatically detect this negativity in Turkish, one has to morphologically analyze the word *değişmedi*. The root of the word is *değiş*, which means *to change* in Turkish. The suffix *-me* adds the negativity, while *-di* adds the past tense to its meaning. In the second example, the use of the term *should* is a sign of the expression of the reporter for the expected behaviour (EB) of the software. In Turkish, this expression is hidden in the word *çalışmalıdır*, where *çalış* is the root corresponding to *make an attempt*, *-malı* is the suffix expressing the obligation that Apache should behave as, and *-dır* suffix is the generalizing modality marker that appears with 3^{rd} person singular. Finally, the last one is an example of S2R, which corresponds to conditional sentences containing non-negative OB predicates. For the example, we detect this pattern in the word *kaydederken*, where the root *kaydet* corresponds to *save*, the suffix *-er* expresses the present tense, and the suffix *-ken* adds the meaning *when*. Göksel & Kerslake (2004) provide a detailed grammatical description of the terms used here. As it can be seen from the table, the key term we are looking for, describing the pattern, is not a separate word in our specific case, and in order to develop a tool to detect the missing information in these issue reports, we need a morphological analysis first and extract the roots and suffixes to find the describing patterns.

Our aim in this work is to determine the discourse patterns in our industrial setting for describing NB, OB, EB and S2R to automatically detect NB issue reports and (if not NB) then label the sentences describing OB, EB and S2R in these reports. To address these goals, we first analyze the issue reports from the industrial case so as to find out what percentage of issue reports are (NB), and what percentage of them contain the OB, EB and S2R information. Our aim is to better understand the need for automatically validating an issue report (whether it is an actual bug or not) and checking its quality (existence of OB, EB and S2R) in our setting. We then decide on the discourse patterns, and label the issue reports and sentences in these reports whether they possess these patterns. If discourse patterns exist in our setting, then they will be used to automatically detect such reports or sentences. Thus, we also aim to get a list of the patterns to be used in automation. We then evaluate our morphological analysis based approach whether it improves automated validity and quality assessment of issue reports in our industrial setting. In other words, we empirically evaluate our approach based on morphological analysis for automated detection of NB, OB, EB and S2R.

The rest of this chapter is organized as follows: Section 8.1 describes the approach we use to classify the reports as bug or non-bug (NB) and detect the information present/missing (OB, EB, S2R); Section 8.2 presents the experimental set-up and

the results; Section 8.3 discusses the results and Section 8.4 concludes the chapter.

8.1 Approach

To address our goals, we conduct a manual analysis first. First we select the top ten products with the most number of issue reports in our domain, namely Credit Cards, Customer Information Management, Personal Loans Allocation, Commercial Loans Allocation, Commercial Loans Disbursement, Collateral Management, Consumer Loan Services, Deposits, Foreign Currency Transfers and Bancassurance. All the issues are reported through the Jira tool (Atlassian, 2021). We extract the reports using the Jira Api and select randomly among the ones created in October 2020 belonging to the above mentioned products. Note that since we first want to label them as NB or not, we do not eliminate any non-bug reports such as enhancements, feature requests, operational requests or questions. We expect to find discourse patterns in sentences which are the rules for identifying NB, OB, EB and S2R (Chaparro et al., 2017a). In total, we collected 1,200 issue reports, where we used all of them for labeling the NB, OB, EB and S2R information and discovering the discourse patterns.

8.1.1 Manual Labeling and Determining the Discourse Patterns

Before any analysis, we carefully examined the discourse patterns described in Chaparro et al.'s paper (Chaparro et al., 2017a; Chaparro, Lu, Zampetti, Moreno, Di Penta, Marcus, Bavota & Ng, 2017b). The most occurring patterns are provided in their paper as follows: (1) The three most observed OB patterns are NEG_AUX_VERBS (negative sentences with auxiliary verbs), $VERB\_ERROR$ (verb phrases with error related nouns) and NEG_VERB (non-auxiliary negative verbs). (2) The most frequent EB pattern is $SHOULD$, sentences using the modal terms "should" or "shall". (3) The most frequent S2R pattern is $LABELED\_LIST$, which are labeled list of actions. Taking into account these most frequent patterns, three project members (including the writer of this thesis) analyzed the first 100 reports out of 1,200 together, through a virtual meeting. The goal of this meeting was to decide on the labels for the reports to be set as NB or not, define the patterns

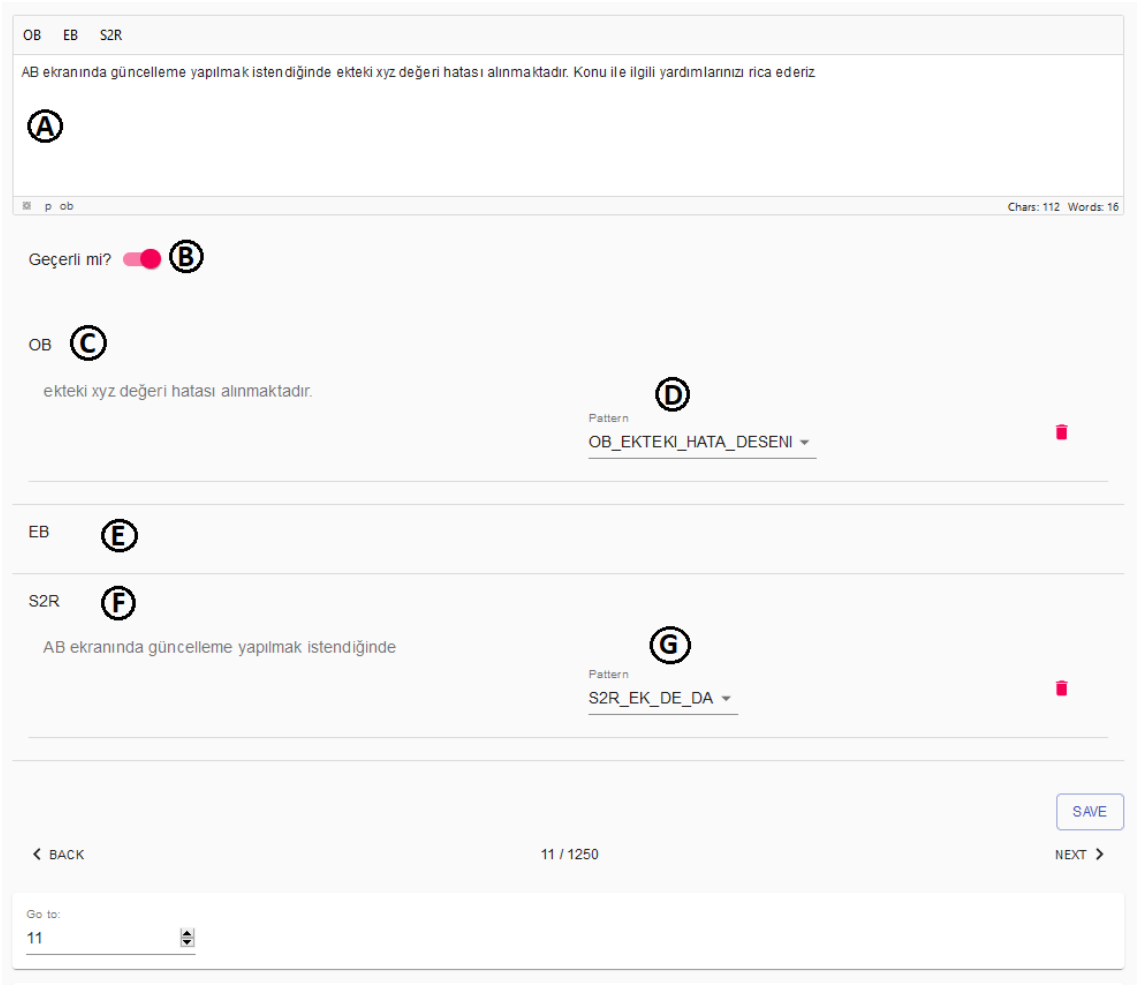


Figure 8.1 A screenshot of the tool we developed to label the issue reports.

for NB, if the reports are labeled as bug, decide on the labels to be assigned to the sentences in descriptions, and also the patterns describing OB, EB and S2R.

After this initial study, we developed a tool to label the issue reports and sentences, and select the patterns found in the sentences in these issue reports. We developed this tool since it would be harder and erroneous to use spreadsheets in the labeling process. A screenshot of the tool is provided in Figure 8.1. The tool first displays the issue report to label (A). The user then labels the report as NB or not (B), and if it is labeled as non NB, then selects the sentences to be labeled as OB (C), EB (E) or S2R (F). S/he further selects the pattern among the list that is displayed with a combobox. For example in Figure 8.1, the user selects a sentence as OB (C), and selects the preferred OB pattern (D) as well. S/he does not label any sentence as EB (E), however labels a sentence as S2R (F) and selects the related S2R pattern (G). Note that if the user labels the reports as NB, s/he can only select among the NB patterns; if not, s/he can only select among the OB, EB and S2R patterns. Note also that one sentence (or part of the sentence) can be labeled with more than one label as OB, EB or S2R. When the labeling for the issue report is finished, s/he saves and passes to the next item.

8.1.2 Classification

In this work, our aim is to develop a tool that provides feedback to reporters at the time of reporting, for our specific industrial case. We want the tool to automatically detect:

- If the report describes a bug (NB or not),
- OB, EB and S2R in the bug report by highlighting the sentences which possess the relevant information so that the reporter can understand whether any of OB, EB or S2R information is missing.

We define these two tasks as supervised classification tasks. Chaparro et al. (2017a) report their machine learning based approach as the best in terms of F-measure, yet the other versions based on regular expressions, heuristics and NLP achieve comparable accuracy. In our work we use a machine learning based approach as well, where we use LinearSVC from scikit-learn (Pedregosa et al., 2011) for each of the classification tasks. Linear SVM (Joachims, 1998) based on *n-grams* is the state of the art text classification algorithm, which proved to perform best on automated

issue report assignment as well (Chapter 5).

For the input textual features, we use *n-grams*, morphological analysis *(MA)* and patterns. *N-grams* capture the vocabulary in our issue report dataset. After preprocessing (tokenization, removing special characters such as punctuation and stopword removal), words in the form of unigrams and bigrams are represented with *tf-idf* (Schütze et al., 2008). For *MA*, we use the Zemberek tool (Akın & Akın, 2007) to extract the roots of all the words and the descriptions for the suffixes (such as passive, ability, negative, etc.). When the root of any word is in our list of roots (which signal patterns), we use it as an input feature. When the word is a verb, we use all the suffix descriptions as textual input. These inputs in the form of unigrams and bi-grams are again represented with *tf-idf*. For *patterns*, if we detect any root or suffix description that is a sign for the specific classification task (NB, OB, EB or S2R) we use that pattern as an input textual feature in the form of unigrams and bigrams and represent it with *tf-idf* in the same manner.

8.2 Experiments

We first labeled the 1,200 issue reports in our dataset. Two project members labeled the issue reports in a two weeks period. The reports appeared randomly in front of the labeling project members such that they check reports from all of the ten products mentioned previously. Throughout the labeling process, three project members (including the ones who label the reports and the writer of this thesis) held a meeting every evening to discuss the labeling process that day. In these meetings, they discussed any ambiguous report and refined the catalog if there is need. If there is need to add a new discourse pattern to the catalog, they discussed first, added the new pattern to the database and labeled the sentence or part of the related sentence using the tool. Note that adding the newly discovered patterns continued until all of the 1,200 issue reports were covered.

After manual labelling, we first analyzed the results and then used the dataset for classification tasks.

8.2.1 Experimental Setup

We evaluate the classification approaches in two settings. In the first setting, we do not take into account the projects that the issue reports belong to, and use all the reports in evaluating performance. We prefer this setting since all the issue reports actually are Softtech products originating from the same customer, IsBank. We expect some variances in the way that teams manage issue reports and in the way that NB, OB, EB or S2R are expressed. In the second setting, we take into account the projects, and check if a model trained and tuned on selected nine projects perform well on the remaining test project. We call the first setting as *within-project* and the second setting *cross-project*.

In our first setting, we perform a nested cross validation (Cawley & Talbot, 2010; Raschka, 2018b) in each experiment. In nested cross validation, the k-fold cross-validation procedure for model hyperparameter optimization is nested inside the k-fold cross-validation procedure for model selection (Brownlee, 2020). In other words, we first divide the dataset into 10 folds and separate one fold for test. The remaining 9 folds are divided into 5-folds in order to tune the C penalty parameter of LinearSVC. Larger C values result in higher penalty on errors. We use a grid search method to decide on the best value of C among $0.01, 0.1, 1, 10, 100$. After deciding on the value for the C parameter, we use it to predict the classes on the seperate test dataset. Following the 10-fold cross validation approach, we repeat this process 10 times and report the average values for the evaluation metrics.

In the second setting, we select all the issue reports in a project for testing, and select the remaining reports for training and hyperparameter tuning. The reports selected for hyperparameter tuning are divided into 5-folds to tune the penalty parameter, C. We again use a grid search method to decide on the best value of C among $0.01, 0.1, 1, 10, 100$. After deciding on the C parameter the model is evaluated on the test dataset. We repeat this process 10 times for each project. Note that in this setting not all the test sets will have the same number of reports belonging to a specific class, so we report the weighted average for performance evaluation.

The metrics we use to evaluate the classification models are *precision*, *recall* and $F-measure$ (Dougherty, 2012). For a specific class, precision is the number of correct predictions divided by number of all predictions for that class, while recall is the number of correct predictions divided by all records actually belonging to that class, and $F-measure$ is the harmonic mean of precision and recall.

For each classification task (as NB, OB, EB and S2R) we evaluate separate models,

where each of them are binary classifiers. We report the *precision*, *recall* and $F-measure$ for the prediction of that specific class. The class_weight parameter of Linear SVC is set to *balanced* in our evaluations, where the class frequencies in the input data are also taken into account in setting the parameter C of Linear SVC.

8.2.2 Data and Analysis

In this section, we present the manual labeling and experimental results. *What percentage of issue reports are (NB), and what percentage of them contain the OB, EB and S2R information?*

The manual labelling results are presented in Table 8.2. In overall, out of 1,200 issue reports, 159 of them are selected as NB (13.25%) and out of the 1,041 issue reports which are selected as bugs, 1,034 of them have the OB information (99.33%), 43 of them have the EB information (4.13%), and 610 of them have the S2R information (58.60%). Credit Cards project have the highest percentage of NB's (25.48%), while the Consumer Loan Services project has no NB. Most of the reports selected as bug have the OB information, 7 projects have OB in all the bug reports and the Commercial Loans Allocation project has the lowest percentage of OB (97.63%), which is still very high. EB information is reported in 8.33% of the bug reports at its highest for the Collateral Management project, and 0.79% at its lowest for the Personal Loans Allocation project. Finally, S2R information is reported in 79.75% of the Customer Information Management project bug reports which is the highest and in 15.79% of the Consumer Loan Services project bug reports which is the lowest.

Table 8.2 Number of non-bugs (NB), and bugs that contain OB, EB and S2R. The percentages for OB, EB and S2R are found by dividing by the number of records selected as bug.

Project	**#NB**	**#OB**	**#EB**	**#S2R**	**Total Bugs**	**Total**
Credit Cards	40 (25.48%)	115 (98.29%)	11 (9.4%)	68 (58.12%)	117	**157**
Customer Information Management	22 (21.78%)	79 (100.00%)	3 (3.80%)	63 (79.75%)	79	**101**
Personal Loans Allocation	19 (13.10%)	126 (100.00%)	1 (0.79%)	76 (60.32%)	126	**145**
Commercial Loans Allocation	9 (5.06%)	165 (97.63%)	4 (2.37%)	104 (61.54%)	169	**178**
Commercial Loans Disbursement	1 (1.02%)	97 (100.00%)	2 (2.06%)	46 (47.42%)	97	**98**
Collateral Management	13 (11.93%)	95 (98.96%)	8 (8.33%)	61 (63.54%)	96	**109**
Consumer Loan Services	0 (0%)	76 (100.00%)	1 (1.32%)	12 (15.79%)	76	**76**
Deposits	5 (7.04%)	66 (100.00%)	1 (1.52%)	24 (36.36%)	66	**71**
Foreign Currency Transfers	32 (21.19%)	119 (100.00%)	9 (7.56%)	81 (68.07%)	119	**151**
Bancassurance	18 (15.79%)	96 (100.00%)	3 (3.13%)	75 (78.13%)	96	**114**
Total	**159 (13.25%)**	**1034 (99.33%)**	**43 (4.13%)**	**610 (58.60%)**	**1041**	**1,200**

Most of the reports selected as bug have the OB information. We frequently observed that users attach screenshots of the errors and write in the description, *"The error is attached"* (or similar statements). We took such textual descriptions as OB patterns without checking the attached screenshot. EB was not observed in most of the issue reports explicitly. Note that the reporters of the issue reports are not software experts, but non-technical clerks, and our intuition is that if the reporter describes the OB behaviour, s/he generally implies the OB should not have happened and does not see that it is necessary to indicate the expectation for the system behaviour.

The distribution of NB is not uniform among projects. While one project has about 1/4 of its reports marked as NB, for another project no report is marked as NB. We believe this is the result of the way the software teams use the issue reporting channel as a way to communicate with their customers. Some teams may not be accepting reports which are NB to be solved through this channel, while other teams may be using this channel in a more general way to communicate their issues. Also, for some teams the systems they work on are so complicated (such as Credit Cards infrastructure) that they may be using work-around solutions for some tasks rather than developing permanent solutions for the incoming NB issue reports. The details of such usage is presented in more detail with the second research question.

What discourse patterns exist in these reports to detect NB, OB, EB and S2R?

We labeled 159 NB issue reports and 2,271 OB, EB or S2R sentences (or parts of sentences). Among the 2,271 sentences, 1,427 (62.84%) of them were OB patterns, 43 (1.89%) of them were EB patterns, and 801 (35.27%) of them were S2R patterns. As a result, we found out 40 discourse patterns, where 7 are NB, 24 are OB, 3 are EB and 6 are S2R patterns. All the patterns required morphological analysis for automation. While some of the patterns could be found in verb suffixes, others could be found as the roots of the words.

The most frequent NB pattern was NB_REQUEST, where the reporter asks for an operational request. We observed this pattern 93.29% of the reports labeled as NB and some example sentences are: *"We require to know if there were any address updates between the dates ...", "I request the following credit application to be archived.", "We kindly request the correction of the fields as ...", "We accidentally updated ... as ... and kindly request that ... is corrected as ...".* These types of requests mean that the product users do not have screens to perform the related operations, or to view the information they need. The developers (or the application support team) may be applying workaround solutions for the immediate solution of the customer's problem. Other types of NB patterns were questions (6.71%), where the user asks why something happened in a specific way. An example is: *"We have*

corrected the card status as ... Why was it ... before?". Some information requests are also reported in question form such as *"Can we learn the policy number that was collected and returned on ... from the credit card ... ?* For the automation of detecting NB patterns related to operational requests, the suffix to be found in verbs for the discovery of NB_REQUEST are the verbal nouns (suffix: "-me", "-ma"), used with a possessive (suffix: "-si", "-sı"). An example is provided in Turkish to show the related suffix in the verb: *"İlgili alanın ... olarak güncellen<u>mesi</u>ni rica ederiz. (We kindly request the update of the relevant field as ...)"* In Turkish, the update word in this example is expressed as a noun with the addition of the suffixes "-me" and "-si". For other cases in request type NB detection, roots of the words need to be extracted such as "archive", "update", "assign", "delete", "accidentally" etc for detecting NB patterns. For detection of question type NB, we need to detect the suffix "-mı, -mi" which are added at the end of the sentences in Turkish to ask a question. An example is *"Bu belgeyi aşağıda fatura numarası belirtilen havale için rica edebilir <u>mi</u>yim? (Can I request this document for a bank transfer with an invoice number below?)"* Also, the question words we came across in this analysis are "why" and "based on what".

We observed that when the issue report is selected as bug, most of them have the OB information. We discovered 24 OB patterns in our repository, and among the patterns the most frequent ones are OB_ATTACH (32.18%), where the reporter expresses that s/he attached a screenshot to to the issue report; OB_NEGATIVE_VERB (18.29%), where the reporter expresses s/he cannot perform an operation; OB_BUT (8.39%), where sentences are combined with the conjunction word "but" ("ancak" or synonyms in Turkish); OB_ALTHOUGH (8.03%) where sentences are combined with the conjunction word "although" ("rağmen" in Turkish). Examples for these patterns are: *"The attached error is received in the allocation process ... for the customer ..."*; *"... cannot be converted into savings account."*; *"The customer has an automatic payment order, but for the statement issued on ..., this order did not work and the debt was not collected."*; *"Although it was approved by our department on ..., it is still displayed as "to be approved"."*, respectively.

For the EB patterns, we observed only 3 patterns explicitly describing the expectation of the user. EB_SHOULD pattern (65.71%) describes how the reporter expects the system should behave. It can be automatically found with morphological analysis either with the suffix added to verbs ("-meli, -malı"), or with the root word "gerek" (need). An example is *"İptal sonrası ... bakiyesi ... olması <u>gerek</u>irken, ... olarak gözükmektedir. (While the ... balance after cancellation should be ..., it appears as ...)"*. Another EB pattern is EB_CAN, which implies that the user

expects to be able to perform an operation (but cannot). The "-ebil, -abil" suffix is added to the verbs to express the expected behaviour and an example is *"Müşteriye kredi teklifi yapabilmemiz için yardımlarınız rica olunur. (Your assistance is requested so that we can make a loan offer to the customer.)"* The final EB pattern is EB_EXPECT, where the user explicitly uses the root "bekle" (expect). An example is *"Vadesi olan ...'de transfer sürecinin başlatılması beklenmekteydi. (The transfer process was expected to start on ..., which was due.)"* We observed that EB is described very rarely in our case. We believe the main reason for this is that the reporters do not find it necessary to write the EB explicitly, since the OB description already implies the EB, which is the opposite of the OB (or not getting the error).

The most frequent S2R pattern we observed was S2R_WHEN (50.3%), which is the verb with the suffix "-de, -da" in Turkish similar to the usage of "when" in English. An example is: *"Teminat numarası, sorgulama sekmesinden arama yapıldığında, sistem teminatı bulamamaktadır. (When the collateral number is searched from the query tab, the system cannot find the collateral.)".* The next frequent S2R pattern is S2R_PERFECTIVE, which are verbs with suffix "-miş, -mış, -muş, -müş" expressing completed actions in Turkish. An example is *"... firmasının grup risk tavanı ... iken ... ye dönüştürülmesi talep edilmiş ve uygun bulunmuştur. (It has been requested that the group risk ceiling of the company ... to be transformed from ... to ... and it has been found appropriate.)".* Another frequent S2R pattern is S2R_IMPERFECTIVE, which is again expressed by adding "-mekte, -makta" suffixes to verbs. They help in expressing actions in Turkish without reference to its completion: *"... Müşterimizin sistemde kayıtlı olan ... mail adresinin silinmesi yönünde talebi bulunmaktadır. (Our customer has a request to delete his/her email address ....)".* In addition, S2R_WHILE pattern is used similar to the S2R_WHEN pattern, but has a slight difference in meaning and expressed in Turkish with the suffix "-ken" added to verbs. We have observed a very few usage of bullets, numbers in expressing the S2R pattern in our case. All of the S2R patterns we observed can be found by extracting the suffixes in verbs.

In summary, we are able to define the discourse patterns for each of the NB, OB, EB and S2R information in our industrial case. Our initial conjecture is confirmed by the existence of these patterns, supporting our research to automatically identify the relevant information.

Table 8.3 Average (P)recision, (R)ecall and (F)-measure for issue report level detection of NB and for sentence level detection of OB, EB and S2R in *within-project* evaluation.

	Issue Report Classification			**Sentence Classification**								
	NB			**OB**			**EB**			**S2R**		
Features	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
patterns	0.33	0.60	0.42	0.90	0.89	0.90	0.74	0.68	0.66	0.55	0.78	0.61
MA	0.67	0.78	0.71	0.95	0.89	0.93	0.78	0.71	0.75	0.81	0.92	0.86
n-grams	0.70	0.54	0.59	0.9	0.88	0.89	0.57	0.28	0.36	0.79	0.80	0.79
patterns + MA	0.64	0.73	0.73	0.95	0.89	0.92	0.78	0.77	0.74	0.81	0.90	0.85
patterns + n-grams	0.74	0.56	0.62	0.93	0.91	0.92	0.67	0.49	0.55	0.81	0.83	0.82
MA + n-grams	0.80	0.73	0.76	0.94	0.92	0.93	0.75	0.70	0.69	0.87	0.88	0.87
patterns + MA + n-grams	0.78	0.76	0.77	0.95	0.92	0.94	0.82	0.83	0.82	0.87	0.88	0.88

Can we verify our conjecture that morphological analysis improves automated validity and quality assessment of issue reports in our industrial setting?

Our aim in this work is to reveal the possibility of developing a tool to automatically label the issue report as NB or not; and if the report is selected as bug (non NB), to display the sentences where the OB, EB and S2R behaviour are described. Thus, we use all the sentences in issue reports as input for the NB classification task, while we use labeled sentences (or parts of sentences) as input for the OB, EB and S2R classification tasks. For each classification task (as NB, OB, EB and S2R) we develop separate models, where each of them are binary classifiers.

The results of *within-project* evaluation are presented in Table 8.3. For issue report classification as NB, any method using outputs of morphological analysis as input features performs higher than 0.71 F-measure. Use of only patterns results in an F-measure of 0.42, which is a sign that some of the patterns we selected may exist in non NB issue reports as well. The best results are obtained when we use the combination of patterns, MA and n-grams as input, where precision, recall and F-measure are all above 0.76.

For sentence level prediction of OB, EB and S2R in *within-project* evaluation, best F-measures are obtained when we combine n-grams, patterns and MA results as input, where 0.94, 0.82 and 0.88 F-measures are obtained respectively. For OB and S2R, by only using outputs of morphological analysis, results were very close to the best results, which are 0.92 and 0.86 respectively. For EB, using all the input features improved F-measure by 0.6 compared to using only MA. For OB and EB, use of all the input features improved recall at the cost of precision when we compare the results to using only MA as input. For S2R, it improved precision at the cost of recall. So we can say that the results are better when we use n-grams, MA and patterns, when we consider precision and recall as well. Although we could label few sentences for EB, morphological analysis highly improved the results, where an increase from 0.36 to 0.75 was achieved compared to using n-grams only.

The results of *cross-project* evaluation are presented in Table 8.4. Use of only patterns in NB classification is very poor. As stated in *within-project* evaluation, it is a sign that these patterns may exist in non NB patterns as well. Furthermore, since the results with only patterns in *cross-project* evaluation is poorer than that of *within-project* evaluation, we conjecture that the NB patterns for a project may be specific to that project. For NB classification, use of MA, and MA with patterns resulted in an F-measure of more than 0.7. However, adding n-grams to the input features decreased performance and introduced noise.

Table 8.4 Average (Weighted) (P)recision, (R)ecall and (F)-measure for issue report level detection of NB and for sentence level detection of OB, EB and S2R in *cross-project* evaluation.

	Issue Report Classification			**Sentence Classification**								
	NB			**OB**			**EB**			**S2R**		
Features	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
patterns	0.25	0.15	0.13	0.90	0.90	0.90	0.46	0.79	0.57	0.57	0.84	0.68
MA	0.72	0.72	0.71	0.95	0.89	0.92	0.82	0.64	0.67	0.82	0.91	0.86
n-grams	0.58	0.15	0.22	0.89	0.84	0.86	0.61	0.12	0.18	0.76	0.81	0.78
patterns + MA	0.69	0.75	0.71	0.95	0.89	0.92	0.83	0.61	0.66	0.82	0.92	0.86
patterns + n-grams	0.70	0.21	0.31	0.90	0.92	0.91	0.75	0.61	0.63	0.79	0.82	0.80
MA + n-grams	0.82	0.42	0.55	0.91	0.93	0.92	0.88	0.76	0.73	0.87	0.82	0.85
patterns + MA + n-grams	0.84	0.38	0.51	0.92	0.94	0.93	0.63	0.73	0.65	0.87	0.82	0.84

In *cross-project* evaluation, best F-measure is obtained with using patterns, MA and n-grams for OB; MA and n-grams for EB; only MA for S2R. Best F-measures are 0.93, 0.73 and 0.86 for OB, EB and S2R respectively. For OB, by only using outputs of morphological analysis, results are very close to the best results, which is 0.93. For EB, using only MA is about 0.7 below the best results. For OB, use of all the input features improved recall at the cost of precision when we compare the results to using only MA as input. For EB, the results are better by using MA and n-grams with respect to all of the performance measures. For S2R, using only MA as input features improves recall at the cost of precision.

In summary, we can say that as in *within-project* evaluation, all the input features can be used for the prediction of OB and S2R sentences, while MA and grams are best in prediction of EB sentences. We can say that morphological analysis highly improved the results, where an increase from 0.18 to 0.67 was achieved compared to using n-grams only, although we have few examples for EB.

As a result, we conclude that use of morphological analysis to extract the roots and suffixes and using them as input features improved the results compared to using only n-grams and best results were obtained when we used all of the input features.

8.3 Usage Scenario

The results showed that it is possible to develop a tool that performs well in predicting an issue report as NB and if not, highlights the sentences as OB, EB and S2R, is possible at Softtech. The result of such a classification with our prototype application is provided in Figure 8.2. The user inputs an issue report description (A) and the tool provides feedback to the user in terms of validity (B) and quality (C), (D), (E).

The description of the issue report provided in Figure 8.2 can be translated as: *The process for the customer ... in branch ... is stuck. "Close all" and "exit" buttons from the upper right corner of the proposal screen were pressed, "ctrl + shift + delete" keys were pressed, the process was re-opened from the inbox and the same warning was received again. We request your help on the subject..*

The tool labeled the issue report as valid (B), reported that the issue report does not describe the EB behaviour (C), reported that the first and second sentences

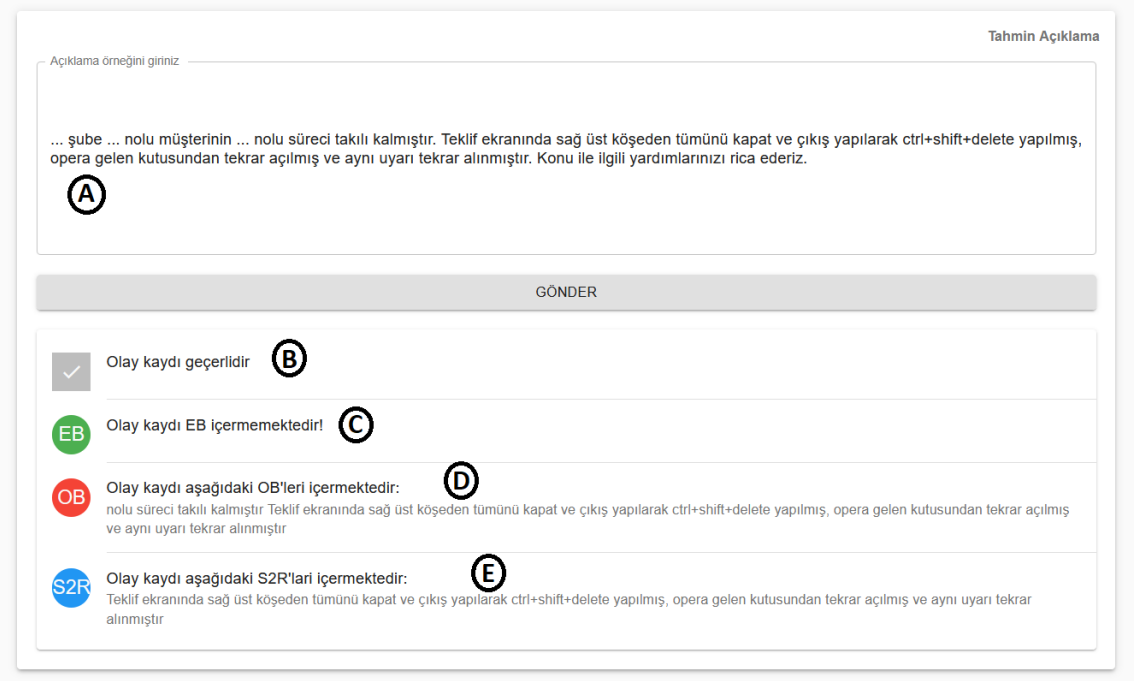


Figure 8.2 A screenshot of the feedback that the prototype tool provides for a selected issue report.

describe the OB behaviour (D), and reported that the second sentence describes the S2R behaviour (E).

8.4 Concluding Remarks

In this chapter, we first analyzed 1,200 software issue reports in banking domain in an industrial setting in terms of their validity and quality. We labeled 13.25% of them as Non-Bug (NB), and out of the 1,041 we labeled 99.33% having described the Observed Behaviour (OB), 4.13% having Expected Behaviour (EB) and 58.60% having Steps to Reproduce (S2R) information. As a result, we found 40 discourse patterns, where 7 are NB, 24 are OB, 3 are EB and 6 are S2R patterns. We observed that few of these patterns appear in most of the issue reports and provided examples for them. For most of these patterns we observed that a morphological analysis was necessary as we had proposed, since the input issue reports are written in an agglutinative language.

We evaluated using n-grams, patterns, morphological analysis and their combinations as input features with a machine learning based approach, in two settings. In

the first setting, we did not take into account different projects since all the reports belonged to the same banking domain, which we called *within-project* setting. In the second setting we took into account different projects, since different teams might have variations in handling and managing issue reports.

As a result we observed that morphological analysis improved validity and quality assessment in both settings. Best results in terms of F-measure were 0.77, 0.94, 0.82 and 0.88 for NB, OB, EB, and S2R prediction respectively, where all the models included morphological analysis of the terms.

9. USING SCREENSHOT ATTACHMENTS FOR ISSUE ASSIGNMENT (RQ3.2)

After the deployment of IssueTAG, we observed a slight decrease in assignment accuracies. It is expected that the issue reports have enough information in their summary and description so that they are assigned correctly, and developers easily understand and reproduce them. For the incorrectly assigned issue reports, one observation we made was they had very short descriptions and the reporters attached screenshots to the reports expecting that the developers checked them to better understand the problem faced.

In this chapter, we aim to exploit the useful information in attached screenshots to improve the accuracy of the deployed issue assignment system. In other words, we investigate RQ3.2: "How can the attached screenshots in issue reports be used for further improving the accuracy of the system?"

As described in Chapter 5, we have developed and deployed an automated issue assignment system, called *IssueTAG* (Aktas & Yilmaz, 2020a). It has been making all the issue assignments in a fully automated manner since its deployment on Jan 12, 2018. Although the assignment accuracy of the aforementioned system has been slightly lower than that of the human triagers (0.83 vs. 0.86), this did not prevent the stakeholders at IsBank from perceiving the deployed system as useful.

In our early work (Aktas & Yilmaz, 2020b), we briefly discussed the plausibility of the idea of using attached screenshots for issue triaging and presented some preliminary results. In this chapter, we provide details about the general approach, analyze the significance of attachments in triaging in an industrial setting, present a better approach for issue assignment, compare it to some alternative approaches, and rigorously evaluate the proposed approach by using real issue reports.

The chapter is organized as follows: Section 9.1 presents the motivating examples to conduct the study; Section 9.2 demonstrates the significance of using attachments in triaging; Section 9.3 presents the approaches we propose to use the attached screenshots as a new source of information; Section 9.4 describes how we evaluated the

Table 9.1 Sample Issue Reports with Attachments

Bug-ID	Summary	Description
1	*ScreenCode*	The screen *ScreenCode* does not open at all terminals in branch *BranchCode* with the error given in the attachment.
2	*ErrorCode*	Although the limits have been discounted, we receive the attached error during *CustomerCode* customer's proposal confirmation.
3	*ScreenCode*/*TransactionCode*	We receive the attached error on the screen *ScreenCode* and trx *TransactionCode*.

proposed methods and presents the results of the experiments; Section 9.5 discusses the results; and Section 9.6 presents concluding remarks.

9.1 Motivating Examples

After automation of issue assignment process at IsBank and Softtech, we realized that for some of the issue reports, the information given within "summary" and "description" attributes of the issue reports was not enough to categorize for even experienced human triagers. The submitters of such issue reports preferred to provide short descriptions of the problem and commented in the "description" that they got the error reported with the attached file. The attached files are mostly screenshots of the errors they got.

Table 9.1 presents some real reports with attachments, which were assigned to wrong development teams by the deployed system. Note that we use placeholders, such as *ScreenCode* and *CustomerCode*, in this table due to legitimate security and privacy concerns. The actual screenshots (i.e., attachments) are not presented either for the same reason.

For the first issue report (Table 9.1), one would expect that the *screen code* indicated both in the summary and the description fields of the report, should help the prediction model to assign the report to the correct team. It, however, turns out that this screen code never occurred in any of the historical issue reports in the training set. Therefore, the prediction model was not aware of it, thus assigned the report to the wrong team. When we manually analyzed the attachment, we, to our surprise, observed that the error message emitted by the system was a generic "HTTP 404 - Web page cannot be found" error, which was, indeed, not useful at all for making any kind of predictions. We, however, quickly realized that the open tabs on the screen contained contextual information, which clearly indicated that the error message was indeed emitted by the retail loan management module. Had

the prediction model used the tab labels extracted from this screenshot, it could have determined the correct team for the assignment.

Regarding the second issue report (Table 9.1), although, at a first glance, this report looks quite similar to the first report in the sense that both reports have a screenshot of the error message emitted by the system as an attachment, a manual analysis of the attached images revealed interesting differences. More specifically, the image attached to the first report was the screenshot of a screen created by the software module responsible for the failure. The image attached to the second report, on the other hand, was a screenshot obtained from a general-purpose workflow engine, visualizing the business process, in which the failure was observed. That is, the workflow engine was not responsible for the failure, it was simply used to create an issue report where the error message mentioned in Table 9.1 was associated with a particular task in the workflow.

The third issue report given in Table 9.1 is also interesting because its attachment is a screenshot of a screen created by a software module installed on a mainframe and written in COBOL for handling certain aspects of chequing accounts. One observation we make is that these screens have a certain look, which distinguishes them from the rest of the screenshots. Furthermore, their layouts are pretty much standardized. For example, the screen codes are displayed at the top left corner of the screen and the error codes appear in a status bar located at the bottom of the screen. However, in our industrial case, the teams and the infrastructures they use are so intertwined that a team may be using both these types of screens (hence the mainframe infrastructure) and other open system infrastructures at the same time.

Our past experience related to incorrect assignment of issue reports is that there exist more cases similar to these ones and including the relevant textual information within the attached screenshots may increase IssueTAG's accuracy.

9.2 Significance of Attachments in Issue Report Assignment

We first carried out a feasibility study to evaluate the plausibility of the proposed idea. To this end, we used a total $41,042$ real issue reports that were submitted during the months of March ($8,322$ reports), April($7,598$ reports), May ($7,876$ reports), June ($5,334$ reports), July ($7,159$ reports), and August ($4,753$ reports) in 2019 and that were closed with the "resolved" status, indicating that the issues reported were

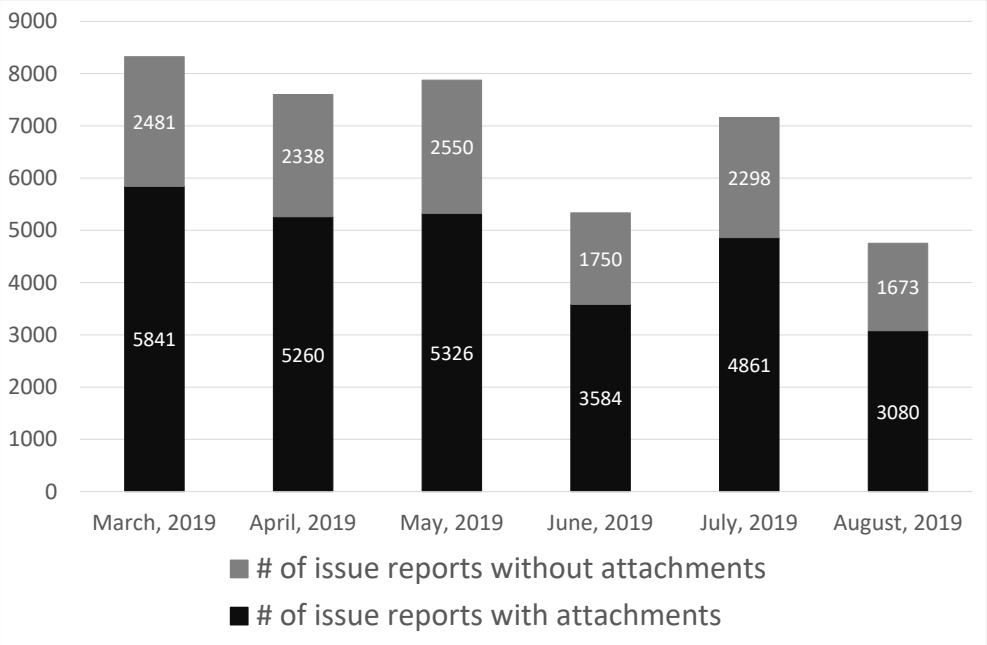


Figure 9.1 Monthly distribution of issue reports with and without attachments, note that less records in June and August stems from holidays, that is, fewer work days.

validated and "fixed" in one way or another. For all of these issue reports, the decision as to which teams they should be assigned to, were automatically made by our deployed system. In the remainder of the chapter, these reports will be referred to as *data set*.

We first observed that about 68.11% (27,952 out of 41,042) of all the issue reports in our data set had at least one attachment. Figure 9.1 presents the distribution of the reports with and without attachments. More specifically, 70.19%, 69.23%, 67.62%, 67.19%, 67.90%, and 64.80% of the issue reports submitted in the months of March, April, May, June, July, and August, respectively, had attachments.

We then observed that these issue reports had a large number of image attachments. Note that some of the issue reports had multiple attachments, in which case we counted each attachment separately. Furthermore, as many images turned out to be embedded in .doc/docx files, we chose to extract them and count them separately. In particular, we had a total of 79,529 attachments in 27,952 issue reports with attachments.

We next observed that these attachments were of different types, including .png, .jpeg, .doc/docx, .xls/xlsx, .msg, .txt, .tif/.tiff, .pdf, .htm/.html, .xml, and

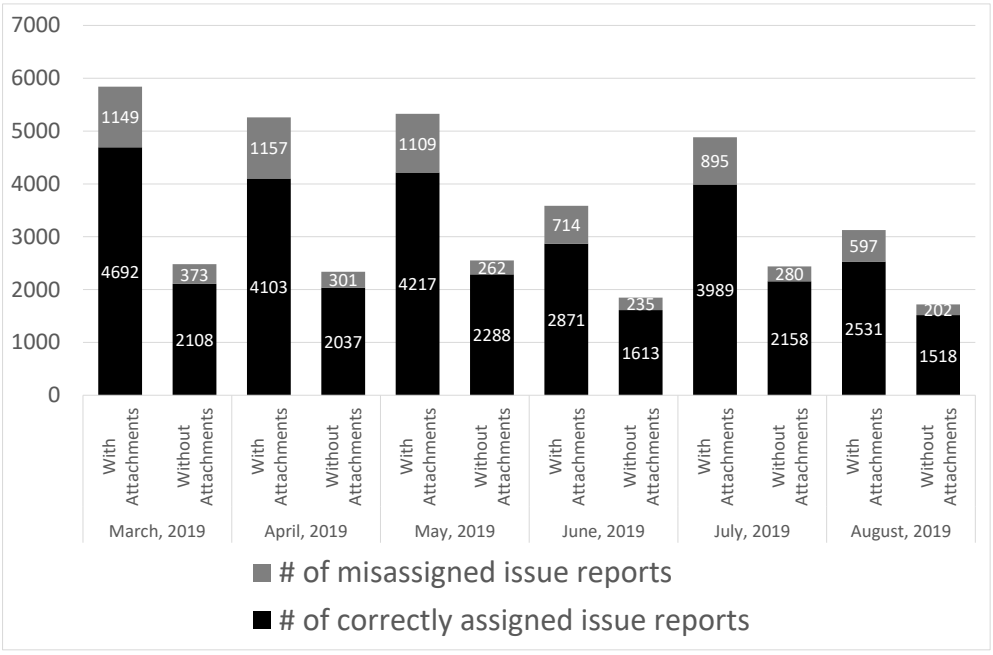


Figure 9.2 Comparison of reassignments for the issue reports with and without attachments. The average accuracy obtained for the former cases was 79.94%, that obtained for the latter cases was 87.64%.

.sql files. And, the most frequently occurring type was screenshots attached in the form of images. In particular, among all the issue reports with attachments, 84.3% of them (83.70%, 83.25%, 84.77%, 84.12%, 85.89%, and 84.06% of them for the months of March, April, May, June, July, and August, respectively) had at least one screenshot attached.

Last but not least, comparing the accuracy of our deployed system on the issue reports with and without attachments (Figure 9.2), we observed that the team assignments automatically made for the reports without attachments were significantly better than those made for reports with attachments. More specifically, while the average accuracy obtained for the former cases was 79.94% (80.33%, 78.00%, 79.18%, 80.08%, 81.67% and 80.91%, respectively for each month), that obtained for the latter cases was 87.64% (84.97%, 87.13%, 89.73%, 87.28%, 88.52% and 88.26%, respectively for each month). An in-depth analysis revealed that this is due to the fact that in presence of attachments, the description field of the issue reports tend to contain less information. In fact, the issue reports with image attachments have 31 words in their descriptions on average, while the remaining have 42.

All of these observations strongly suggest that a significant portion of the issue reports submitted to IsBank tend to have attachments and that using the information contained in these attachments can improve the accuracy of assignments.

9.3 Approach

Our objective in this study is to find a simple and effective technique to improve automated issue triage with the information we extract from attached screenshots in an industrial context. With all the insights we gained from our manual analysis of issue reports in mind, our main approach is to extract textual information in screenshot attachments and use it as an additional source of information for issue report assignment. To this end, we use a well-known OCR technology (Smith, 2007). However, in order to justify our conjecture that image features are not helpful in increasing IssueTAG's performance, we also conduct experiments where we use image features from screenshots.

Note that, when there exists more than one type of information, such as text, image or video, the research problem is categorized as multimodal. In such a context, the aim of multimodal machine learning (Baltrušaitis, Ahuja & Morency, 2018) is to

build models which process and relate information from these multiple modalities. Our problem fits into such a category. However, our case is finding the software team to solve a specific issue report. Most of the time these teams use the same infrastructures for the user interfaces, therefore, the layouts/appearances of the images attached to the issue reports are similar as well. For example, an error that a user observes while using the internet bank branch may belong to many different teams, however the GUI is similar for all of them as they use the same infrastructure for the user interface. They differ in terms of the textual data in error messages, selected tabs, screen names etc. So, our expectation is textual features to be more useful in improving IssueTAG's performance, however we experiment with a multimodal approach as well, where image features and textual features are used together.

The textual data extracted with OCR has a different structure compared to the subject and description attributes of issue reports, which are also textual. OCR data is much more noisy and erroneous, not written as sentences having a semantic relation, but contains words extracted from the image of a web form in the order OCR extracts them. Therefore, preprocessing and representing this new piece of information separately from subject and description features makes sense.

In production, we use Linear SVM as the classification algorithm, which is a simple, fast and successful technique for text classification. In this study, we compare Linear SVM applied on tf-idf, with other techniques so as to check if they better fit this new type of information. Note that we do not want to introduce sophisticated techniques since we are pleased with the baseline algorithm in terms of speed and accuracy, although accuracy being slightly lower than the human triagers' performance.

We first retrieve the "ID, summary, description and attachment URL" features of the issue reports from the issue repository. If the issue report has any attachment, we then check its extension. If the attachment is an image file, or if it is a document sheet that includes an image file in it, we get the image (or images) and utilize Py-tesseract (Smith, 2007) as the OCR engine to get the textual data in it. We also save the image files in a separate folder. In this way, we extract 6 months of data (41,042 issue reports), created in March, April, May, June, July and August of 2019 the details of which we described in Section 9.1. A summary of the data is also provided in Table 9.2.

Table 9.2 Data Summary

Month of Creation	Issue Reports with Screenshots	Issue Reports with Attachments	Issue Reports without Attachments	Total	Distinct Teams
March, 2019	4,889	5,841	2,481	8,322	52
April, 2019	4,379	5,260	2,338	7,598	58
May, 2019	4,515	5,326	2,550	7,876	53
June, 2019	3,015	3,584	1,750	5,334	51
July, 2019	4,175	4,861	2,298	7,159	53
August, 2019	2,589	3,080	1,673	4,753	49
Total	23,562	27,952	13,090	41,042	63

9.3.1 Text classification

Machine learning based issue triage techniques use previously solved issue reports for training and regard the problem as a supervised classification problem. First, words are tokenized, unwanted characters such as punctuation are removed and finally, stop-words which are words that do not have a contribution in classification are removed. We apply the same procedure before any textual representation.

We use the Linear SVC as the baseline classifier in this work, and we represent the documents with the popular and successful technique, $tf - idf$ representation, for the Linear SVC algorithm.

Note that how to combine the text from attached screenshots and text from subject and description is a matter to consider. Text from attached screenshots (or OCR data) is very noisy and not written in a semantic order, while the subject and description contain cleaner text data written in spoken language. Processing them separately makes sense, however, we suggest two methods so as to observe the results: In the first method, we first concatenate subject, description and text from screenshots, pre-process it, and then obtain the tf-idf vectors applied on the combined feature. In the second method, we pre-process text from screenshots, and the combined subject and description separately. We obtain two tf-idf vector representations, where we use different vectorization parameters such as n-grams, minimum and maximum document frequencies of the terms for each. Finally, we combine these vectors before using as input for the selected algorithms.

With BOW model, the context that the words are used, are not taken into account and each word is represented in one dimension. A word embedding is a better representation where words used in similar ways have similar representations.

It has been shown that good results can be obtained for text classification by using pre-trained word embeddings with simple deep learning architectures (Kim, 2014). However, our data set is one having many domain specific terms. For this reason,

we use the keras embedding layer[1] in our work so that the embeddings are targeted to our specific context. The word embeddings are initialized with random weights and learnt for all of the words in the filtered training data set.

We use Support Vector Machines (SVM) with tf-idf vectors, which is a commonly used technique in issue triage. In SVM, classes are separated with hyperplanes such that the margin, which is the perpendicular distance from the hyperplane to the closest points, called support vectors, is maximized. In practice SVM's are implemented using kernels, where we use a linear kernel in this study, or the Linear SVC implementation of Scikit-learn tool (Pedregosa et al., 2011).

Lee et al. (2017) propose to use a multi-channel Convolutional Neural Network (CNN) for issue triage. They first represent the textual data with word embeddings. Thus, they convert each word in summary and description to a vector conveying the semantics. Then, a convolution process with multiple filters and a sub-sampling process *max-pooling* is applied on the two inputs, separately (in other words, CNN with two-channels). Finally, the results are concatenated, and through a fully-connected layer and softmax regression, class probabilities are obtained. To prevent overfitting, dropout and l2 regularization are applied as well. To evaluate the added value of attached screenshots, we use their approach, however, we add a third channel to their architecture, where we feed the textual data from screenshots to the new channel.

9.3.2 Image Classification

We conjecture that image features are not helpful in classifying software teams in our context since the teams use the same infrastructures and the appearances of the screenshots are similar. To test our hypothesis, we use an image classification technique to classify the screenshots according to the software teams that solved the issue reports they are attached to. In a setting where the software teams solve issue reports belonging to completely independent software products and having different layouts for the attached screenshots, image classification would be successful to identify the software team.

Deep learning techniques take a long time to train images, so we use transfer learning, which is simply using a model trained on a problem to solve another related problem.

[1] https://github.com/fchollet/keras

The model we use is the VGG-16 architecture (Simonyan & Zisserman, 2015). Since the model is used to train 1,000 classes, we do not include the final fully connected layers and add a new output layer to classify according to the software teams. We re-use the model weights in the convolutional layers in the VGG-16 model and do not train them on our dataset to get a fast result. We use the Keras tool in our experiments which provides access to the VGG-16 model.

9.3.3 Multimodal Classification

To use textual modalities, one-line summary and description, together with the visual modalities, attached screenshots, one of the new techniques to combine these features is *Multimodal Machine Learning*. We used the *multimodal framework (MMF)* tool provided by Facebook AI (Singh et al., 2020) for this purpose. The architecture we used is the basic architecture provided by the tool. The fastText embeddings (Bojanowski et al., 2017; Joulin et al., 2017) were used to transform the input textual data, and the ResNet image recognition model (He et al., 2016) was used to transform the input images. The resulting features are concatenated and used as input for the fully connected layer for classification.

9.4 Experiments

9.4.1 Experimental Setup

We use the data set introduced in Section 9.2, with 41,042 real issue reports that were submitted between March and August 2019 and were closed with the "resolved" status. The issue reports are assigned to 63 different teams in the given period. It is observed that the records belonging to the first 8 teams with the most number of records constitute approximately 50% of all records. Although the records did not show a balanced distribution, we preferred to conduct the experiments with the current data set which is in accordance with the real situation in the production

environment.

For finding out the best approach, depending on the vectorization technique and the algorithm to be used, we propose five models to compare:

- SVM_{sd} is our baseline model where subject and description are used as input, concatenated, transformed to tf-idf representation and Linear SVC is applied on the representation to predict the team to assign the issue report;
- SVM_{sda1} is the method where subject, description and text from screenshots are concatenated and then processed to obtain only one tf-idf vector to be used by Linear SVC;
- SVM_{sda2} is the method where subject and description, and text from screenshots are processed separately with different parameters to obtain two tf-idf vector representations, which are then combined to be used by Linear SVC;
- CNN_{sd} is a two channel CNN using summary and description as input, represented with word embeddings that are initialized randomly and learnt jointly with the CNN model;
- CNN_{sda} is the same architecture as in CNN_{sd}, the only difference is having three channels that use summary, description and text from screenshots as input.
- $VGG-16$ is the model we used to classify the issue reports using only the image features. So, we train and test this model with only the reports with attached screenshots.
- MMF is the model where the results of the text encoder and image encoder is concatenated to be given as input to the fully connected layer for classification.

We only use the best prediction in our approach, in other words $top-1$ recommendation of the classifiers, since in our case we fully automate the assignment of issue reports.

We further experiment using the dataset from the industrial case to show that we obtain better results with the proposed model (that uses the text from screenshots as another piece of information), statistically significantly, compared to the baseline model (SVM_{sd}). For this purpose, we set the test set fixed with issue reports selected from a specific month and vary the training set with a fixed amount of data with issue reports created before that specific month. We repeat the experiment 30 times with this fixed test set comparing the results of the baseline model and the proposed model. For comparison, we use box-whisker plots for the accuracy levels

of the both models. We then repeat the experiments by forming new test sets using issue reports created in other months and by creating training sets as explained.

Training Linear SVC on textual data is very fast and the stakeholders are satisfied with the current performance of the automated assignment system in terms of speed. We do not want to be in need of extra computer resources for introducing attached screenshots as a new source of information. For this purpose, we conduct experiments such that: 1) We calculate the average time to obtain the OCR data for the related images, 2) We compare the average time to train the best model and the baseline model, 3) We compare the average response time to predict the software team with the best model and the baseline model.

We used accuracy and F-measure as the evaluation metrics, for our multi-class classification problem.

9.4.2 Data and Analysis

We split the data set such that the issue reports created in the first four months (March-June 2019) are used for training (29130 reports, 16798 having attached images and 12332 having none), the ones created in July 2019 for parameter tuning (7159 reports, 4175 having attached images and 2984 having none) and the ones created in August 2019 for testing (4753 reports, 2589 having attached images and 2164 having none). The testing data set was used to measure the accuracy of the baseline and the proposed models. The classification results on the test data set are reported in Table 9.3. Note that we report the results on all of the test data, and on test data with and without attachments.

Can leveraging the information conveyed in screenshot attachments improve the accuracy of the assignments?

We can see in Table 9.3 that the best results are obtained with SVM_{sda2}, where we pre-process text from attachments, and concatenated subject and description features separately, obtain two tf-idf vectors and combine these vectors before training and testing. When we consider the issue reports having attached screenshots, we achieved 1.86% increase in accuracy and 1.89% increase in F-measure compared to the baseline model SVM_{sd}. Note that the daily assignment accuracies dropped about 3% after the deployment of IssueTAG, from 86% to 83% (Aktas & Yilmaz, 2020a), compared to human triagers, meaning that the results are satisfactory considering our target. The tuned parameters for tf-idf vectorization obtained after

Table 9.3 Classification results for (A)ccuracy and (F)-measure on test issue reports created in August 2019

	Test data with screenshots		Test data w/o screenshots		All test data	
Classifier	A	F	A	F	A	F
SVM_{sd}	0.84	0.83	0.85	0.85	0.85	0.84
SVM_{sda1}	0.82	0.81	0.84	0.84	0.83	0.82
SVM_{sda2}	**0.86**	**0.85**	**0.86**	**0.84**	**0.86**	**0.85**
CNN_{sd}	0.82	0.81	0.84	0.83	0.82	0.81
CNN_{sda}	0.83	0.82	0.83	0.82	0.83	0.82
$VGG-16$	0.02	0.02	-	-	-	-
MMF	0.41	0.53	-	-	-	-

detailed experiments are as follows: - N-grams used for subject and description are 1 and 2, while for the OCR data it is 1, - Minimum document frequency is 2 for the former and 100 for the latter and the maximum document frequency is 0.5 for the latter, which means that only words that occur in half of the documents are retained. We did not set a maximum document frequency parameter for tf-idf vectorization of subject and description, only used our stop-word list specific to Turkish language to eliminate common words.

Concatenating all the three textual input before processing (SVM_{sda1}) caused a decrease in both accuracy and F-measure for all types of data. It means that textual OCR data, and subject and description have a different structure as explained before and they have to be converted to vector representations separately for better results.

CNN model results could not surpass the baseline SVM model, which may be a result of the fact that the we have many domain specific terms. Successful CNN models in previous studies for sentence classification use word embeddings pre-trained on millions of textual data (Kim, 2014), while we do not have such pre-trained word embeddings for our domain. Furthermore, in previous work by Lee et al. (2017), CNN model was proposed for issue triage, however the results were not compared to a baseline machine learning model. We believe that CNN models and other deep learning models should be baselined against simpler alternatives for issue triaging.

As expected, the image classification models were unsuccessful in improving the performance. The layout of the GUIs that the software teams use do not change most of the time since they use the same infrastructures. However, for the MMF model, we used the basic architecture, so there is possibility that better results can be obtained compared to the current result, however, we also expect that the result can be improved with better utilizing the information from the textual features.

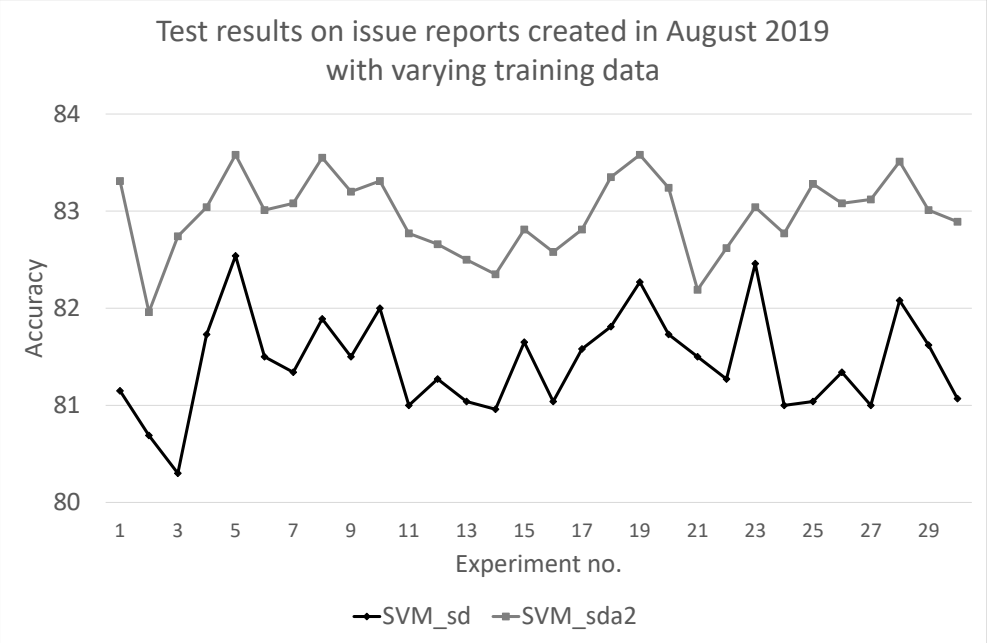


Figure 9.3 Test results with varying training data on the test issue reports created in August 2019 with attachments

Also note that it took 2.5 weeks to train and test this model, which is not feasible when we compare it to the performance of SVM. In fact, it takes minutes to train and test with SVM.

Are the improvements obtained statistically significant?

To evaluate if the results are statistically significant, we first fixed the test data set such that the issue reports are created in August 2019 and they have attached screenshots (2,589 issue reports). Then, by selecting 10,195 random issue reports created between March 2019 and June 2019 (so that we have about 80% training and 20% test data) we repeated the experiments 30 times. For each run, we compared our baseline model SVM_{sd} and the best model SVM_{sda2} on the test data set in terms of accuracy. The results are shown in Figure 9.3. The dark line shows the prediction results with SVM_{sd}, and the other one shows the results when we add the textual data from attached images to our classification pipeline.

It is observed that we obtained better results with the model using textual data from images in each run. The average performance improvement in these experiments were 1.52% with a standart deviation of 0.41%, the maximum being 2.44%, the minimum being 0.58% and the medium being 1.49%. Figure 9.4 presents the box-

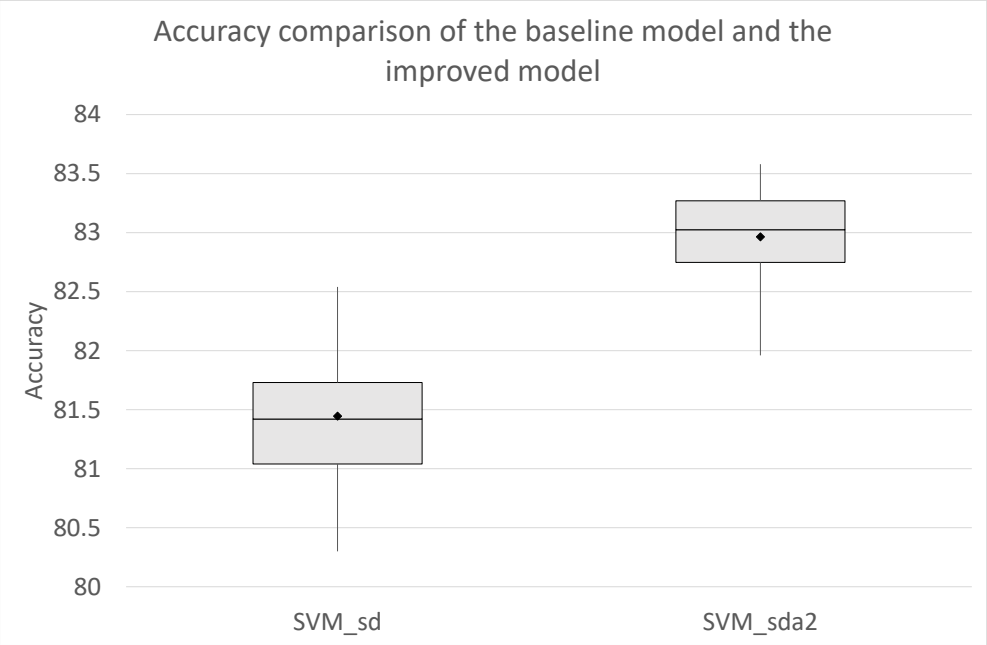


Figure 9.4 Accuracy comparison of the baseline model SVM_{sd} and the improved model SVM_{sda2}. For each box, the bottom and the top bars indicate the first and third quartiles, respectively, whereas the middle bar and the diamond shape represent the median and the mean numbers of the accuracies, respectively, obtained with randomly selected training sets with a fixed size.

whisker plot of the accuracies obtained with the two models, SVM_{sd} and SVM_{sda2}.

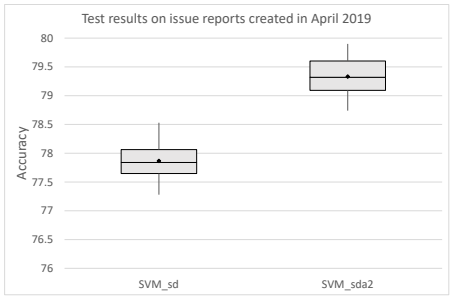
To test the statistical significance of the results, we used the Student's paired t-test (Hsu & Lachenbruch, 2014). The typical use of this statistical tool is to compare the means when the observations have been obtained in pairs, as in this study, thus, to test whether there is a real difference between the two classifiers, based on accuracy. We obtained a very small p-value (< 0.001) on the data shown in Figure 9.3, which indicates a statistically significant difference. Thus, the result reveals that the performance increase with the proposed model cannot be attributed to the random behaviour of the algorithms.

Next, we used other portions of the data set for testing as well and repeated the experiment. We first created four separate test data sets consisting of issue reports created in April, May, June and July of 2019, respectively, and that contain attached image files. Then, we created varying training sets with a fixed size to make predictions on these test sets. To do so, we used the issue reports created in previous months from the relevant test set, since it is more appropriate to use issue reports created back in time for training. For the test sets of June and July, we selected reports to the training set such that we have an 80% to 20% distribution for training and testing. For the test set consisting of the issue reports created in April and May 2019, we had a distribution of 60% to 40% for training and testing, since we only use previous months' data for training, and the number of issue reports that could be used in the training data set consisted of fewer issue reports. We repeated each experiment (on the related test set) 30 times and the results are depicted in Figure 9.5. Again, for each of the experiment in the figure we obtained a very small p-value (< 0.001), which indicates a statistically significant difference. Thus, the results depict that SVM_{sda2} method performs better than the baseline model SVM_{sd} which cannot be attributed to randomness.

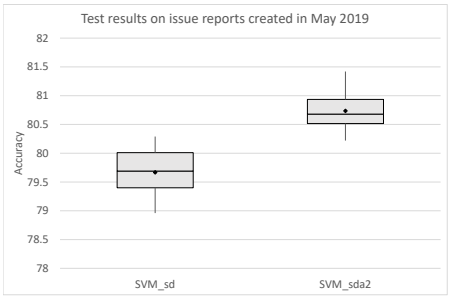
How does the proposed approach affect the overall performance of the system, including the training times and predictions times?

We conducted the following experiments:

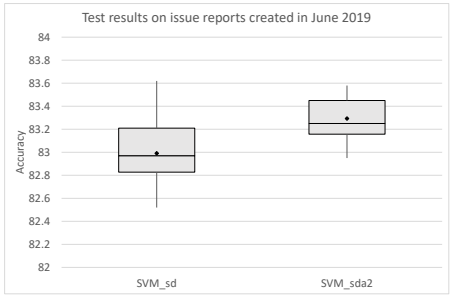
- We used 1000 images to obtain the average time to process them including time to open the image and get the textual data with OCR.
- We trained with six months of issue report data 10 times and reported the average training times for SVM_{sd} and SVM_{sda2}. Note that images are processed before training, and textual data from OCR is ready before this step.
- For 30 issue reports, we compared the response times of prediction with Is-



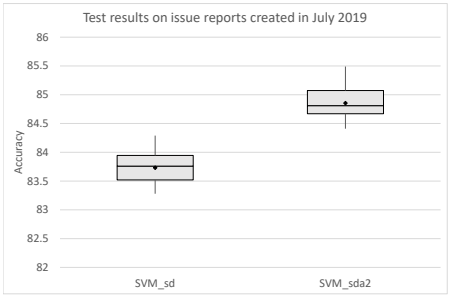
(a)



(b)



(c)



(d)

Figure 9.5 Box-whisker plots of the accuracies obtained with the two models, SVM_{sd} and SVM_{sda2}. Note that each experiment is run 30 times. Results on test data created in a) April 2019, b) May 2019, c) June 2019, d) July 2019.

Table 9.4 Experiment results on the time performance (in seconds) of using attached images for issue triage

	average	std	max	min	median	no.of cases
Processing images	2.11	1.05	8.51	0.57	1.85	1000
Training time for SVM_{sd}	190.4	6.69	202	180	190.5	10
Training time for SVM_{sda2}	317.2	17.08	348	291	314.5	10
Response time for SVM_{sd}	0.9	0.03	1.01	0.86	0.9	30
Response time for SVM_{sda2}	2.17	0.09	2.54	2.07	2.16	30

sueTAG (in other words with only subject and description features), and the response time of prediction using subject, description and images. Note that in the second case opening the image and obtaining the OCR data from the image is included in the response time. We tested the two API's with Postman[2] tool and report the response times that Postman returns.

The experiments are conducted on a Dual-Core Intel(R) Core(TM) i7-6600U CPU @2.60 GHz computer with 16GB of RAM running Windows 10 Enterprise 2017 as the operating system. The results are shown in Table 9.4. As expected, adding the image feature for prediction slightly increases the processing times. However, the increase is negligible and not important when compared to high costs of deep learning models.

9.5 Discussion

Users of a software product may attach screenshots for the errors they get, rather than explicitly describing the issue in the description part. We conducted a series of experiments and consistently obtained better results when we added textual data from attached screenshots to our baseline classifier, while not degrading its performance in terms of speed. Our results show that attached screenshots are an important source of knowledge in issue triage in our industrial case.

The best results are obtained when we pre-processed subject and description, and text from the image attachments seperately, and the two tf-idf vectors are combined to be used for classification by Linear SVC algorithm. The overall approach is shown in Figure 9.6. As a result of this approach, we could obtain around a 2%

[2]https://www.postman.com

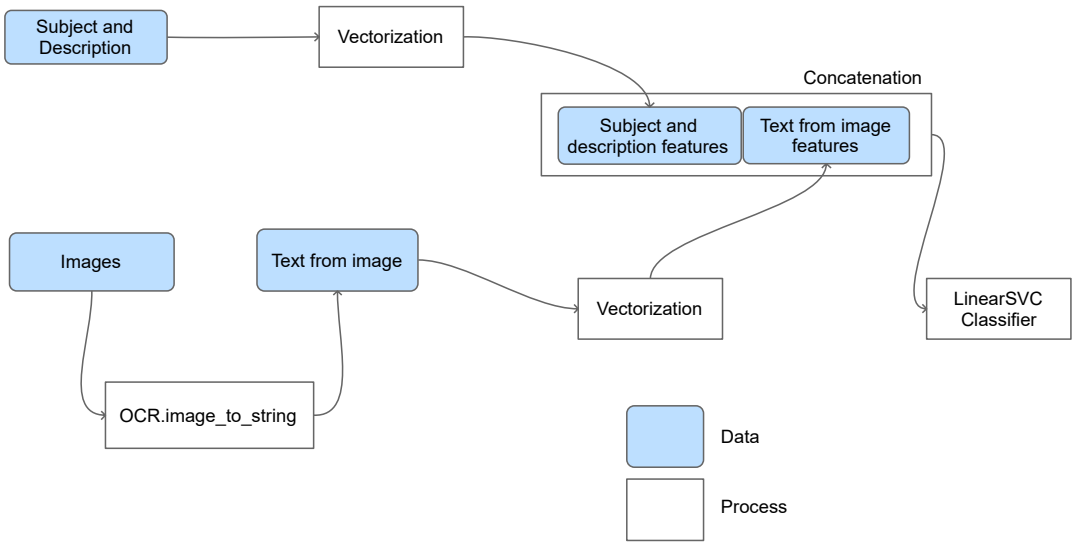


Figure 9.6 Overall approach

improvement in accuracy for the incoming issue reports having attached screenshots. Note that the performance of IssueTAG is 3% below that of human triagers' in terms of accuracy and we are not after a big performance increase.

Fu & Menzies (2017) discuss optimizing hyperparameters of classifiers better, rather than directly implementing complex solutions for software engineering problems. We believe our case has a similar implication such that the processes about software engineering problems have to be studied and understood better before applying these complex techniques. Dealing with the incorrectly assigned issue reports and focusing on the reasons for incorrect classification resulted in using another source of information for our case. We further agree that deep learning models should be baselined against simpler alternatives because of their high computational costs. In case of issue report classification using the textual data as input, SVMs are most probably a better solution than the more complex ones such as CNNs, since the sentences in these reports are relatively short, meaning that their vectors are highly sparse.

We believe using the attached files of issue reports can be also be relevant in automating tasks other than issue report assignment as well, such as finding duplicate or similar issue/bug reports, predicting severity, validity, time to solve issue/bug reports.

9.6 Concluding Remarks

In this chapter, we argued that there exists useful information in attached screenshots that could improve automated issue report triaging. The deployed automated system for issue report assignment at Softtech Inc., $IssueTAG$, that uses only the input textual information is about 3% below the human triagers' accuracy before deployment. When we analyzed the selected incorrectly assigned issue reports, we observed that they did not have much information in their descriptions and they had attached screenshots which the submitter expected the developers to check.

We first obtained six months of data from the industrial case and analyzed the data. We observed that a significant portion of the attachments are screenshots and it is more likely that the issue report with an attached screenshot will be reassigned since IssueTAG only uses the "summary" and "description" features of the issue report.

We proposed to use OCR technique to extract the textual information in these screenshots. We did not use the image features of the screenshots since most of the time different development teams used same or similar GUI's in our case. Processing the manually written "summary" and "description" features, and the OCR text from the screenshots separately, and concatenating the two vectors afterwards, and predicting the teams with Linear SVC gave the best results. The proposed approach increased the assignment accuracy of the issue reports with attachments by 2 points (i.e., from 0.84% to 0.86%) compared to the baseline model that uses "summary" and "description" features only. We furthermore showed that the results are statistically significant and the proposed approach does not degrade the time to train and predict the development teams notably.

As a result, in this study, we extract the textual data written on the image with OCR, and use this extra textual information for improving issue report assignment. However, in other domains, where teams have products that use different infrastructures for the user interfaces and consequently, the images attached have separable appearances, image features may be relevant and may contribute more to issue report assignment.

10. THREATS TO VALIDITY

10.1 Construct Validity

To circumvent the construct threats, we used the well-known *accuracy* metric (Schütze et al., 2008) throughout the thesis to evaluate the quality of the issue assignments. We have also complemented the accuracy results with other well-known metrics, namely precision, recall, and F-measure, as we see fit. We mainly focused on the accuracies because 1) it was the choice of a recent related work in the literature (Jonsson et al., 2016) and 2) the assignment accuracies and F-measures (computed by giving equal importance to both precision and recall) we obtained in the experiments were comparable.

To measure the amount of effort saved by automating the issue assignments (Section 5.4), we used the person-month metric, which is also a well-known metric to quantify effort in software engineering projects (Pressman, 2005).

To measure the effect of the proposed approach on the issue-resolution process, we compared the average times required to close the issue reports and number of issue tosses before and after the deployment of IssueTAG (Section 5.4). To this end, we used the dates and times, and historical data recorded by the issue report management tool (namely, Jira). Furthermore, since the characteristics of the reported issues, thus the times it takes to resolve them, can change over time, we used the issue reports submitted within two months before and after the deployment of the system for this purpose.

To further evaluate the usefulness of the deployed system, we carried out a survey on the actual users of the system (Sections 5.5-6.2). The surveys had both Likert scale and open-ended questions and about half of the actual users of the deployed system voluntarily participated in the surveys.

Throughout the thesis, we used the actual database of issue reports maintained by Softtech. Furthermore, all the survey results were obtained from the actual users on the field. We followed the same approach to evaluate our PELT-based technique to detect deteriorations in assignment accuracies, which, indeed, successfully detected three deteriorations each with a different cause (Section 5.4). To further evaluate the proposed approach, we also carried out controlled experiments, each of which was repeated 1000 times (Section 7.2). We did this because in the data collected from the field, it was not always possible to determine whether there really were some deteriorations or not, and if so, what the nature of these deteriorations were. Therefore, the controlled experiments helped us further evaluate the proposed approach, as in these experiments, we knew both the nature of the deteriorations (e.g., sudden or gradual) and the exact point in time where they occurred.

10.2 Internal Validity

To circumvent the internal threats that may be caused by implementation errors, we used well-known and frequently used tools. In particular, we used the Python scikit-learn (Pedregosa et al., 2011) tool for preprocessing the issue reports and extracting the features; the scikit-learn (Pedregosa et al., 2011) and mlxtend (Raschka, 2018a) tools for training the classification models; the lime (Ribeiro et al., 2016) tool for creating the LIME-based explanations for the assignments; the ruptures (Truong et al., 2018) tool for PELT-based change point detection; Py-tesseract (Smith, 2007) for applying OCR; and keras (Cholletet al. , 2015) for applying the deep learning models.

In Chapter 5, we performed the same preprocessing steps and extracted the same set of features for all the classification algorithms used in the study. However, the performances of these classifiers might have been dependent on the preprocessing steps used and the features extracted. On the other hand, we used well-known preprocessing steps, such as tokenization and removal of non-letter characters as well as stop words and extracted frequently used features, such as the bag-of-words model.

A related concern is that we used the default configurations of the aforementioned classifiers, except for the k-nearest neighbor and the stacked generalization classifiers. For the former, we used cosine similarity and empirically tuned k. For the

latter, we used logistic regression as the level-1 algorithm together with the probabilities emitted by the level-0 classifiers. On the other hand, the performance of these classifiers might have been dependent on the underlying configurations. Note, however, that optimizing the configurations for these classifiers could have only generated better accuracies.

In the evaluations, as the correct team for a given issue report (i.e., as the ground truth), we used the team who actually closed the report. Some reports, however, might have needed to be processed by multiple teams before the reported issues could be fixed. Since in these situations, typically the last team in the chain closed the report, even if the initial assignment of the report was considered to be correct, it was counted as incorrect when computing the assignment accuracies. Note, however, that counting such assignments as correct could have only increased the accuracies.

When computing the amount of manual effort required for issue assignments, we did not take the amount of effort required for maintaining the knowledge base used by the IT-HD clerks into account. Therefore, the actual savings in person-months can be larger than the ones reported.

10.3 External Validity

One external threat is that IssueTAG was deployed at Softtech/IsBank only. Softtech, however, being a subsidiary of IsBank – the largest private bank in Turkey – is the largest software company of Turkey owned by domestic capital, maintaining around 100 millions of lines of code with 1.200 employees. Consequently, it shares many characteristics of large software development houses, especially the ones producing custom, business-critical software systems, such as having a large, continuously evolving code base maintained by dozens of development teams with hundreds of issue reports filed daily, each of which needs to be addressed with utmost importance and urgency.

Another possible threat is that issue reports at IsBank (thus, the ones used in this work) are created by the IT-HD clerks (Section 4.1). Although, this team is a non-technical team, they are specialized in creating issue reports by listening to the bank customers and employees. Therefore, the quality of the issue reports used in this study may differ from the ones directly created by, for example, the end-users of a system. However, many companies, especially the ones that produce business-

critical software systems and that need to deal with a large number of issue reports, employ similar call centers. Furthermore, all the issue reports used in this work were written in Turkish. However, we used simple text processing steps, such as tokenization and removal of non-letter characters and stop words. Therefore, the proposed approach can also be used with issue reports written in other languages.

10.4 Conclusion Validity

All the issue reports we used in the experiments were the real issue reports collected from the field. After the deployment of IssueTAG, once an issue report was created by an IT-HD clerk, the assignment was automatically made by the system. There was no means that the deployed system could be bypassed or that the assignments made by the system could be changed by an IT-HD clerk. Note that the AST members could then reassign the issue reports if needed, in which case the initial assignments made by the system were considered as incorrect. The number of issue reports closed was an important performance metric for the AST members as well as for the development teams at Softtech. Consequently, as a part of the company's policy, the issue reports were required to be closed by the development teams, who actually resolved the reported issues. The stakeholders payed utmost attention to this matter. Therefore, the assignment accuracies reported in this work, reflect the actual accuracies obtained by IssueTAG on the field.

To further evaluate the deployed system, we carried out two surveys (Sections 5.5-6.2). Although 14 participants were involved in these surveys, they constituted about half (14 out of 30) of the AST members, who are the direct end-users of IssueTAG.

11. CONCLUSION AND FUTURE WORK

In this work, we have developed and deployed a system, called IssueTAG, to automate the process of issue assignments in a business-critical environment. To this end, we have first cast the problem to a classification problem and determined the classifier to be used in the deployed system by empirically evaluating a number of existing classifiers. We have also carried out further studies to determine both the amount and time locality of the historical data required for training the underlying classification models whenever needed.

We have then deployed IssueTAG by configuring it based on the results we obtained from these studies. Since its deployment on Jan 12, 2018, IssueTAG has been assigning all the issue reports submitted to Softtech in a fully automated manner, which totals up to more than 250.000 automated assignments so far.

We observed that 1) it is not just about deploying a data mining-based system for automated issue assignment, but also about designing/changing the assignment process around the system to get the most out of it, 2) the accuracy of the system does not have to be higher than that of manual assignments in order for the system to be useful; although the daily assignment accuracies dropped slightly from 0.86 to 0.83 after the deployment of IssueTAG, the manual effort required for the assignments was reduced by about 5 person-months per year and the turnaround time for resolutions was improved by about 20%, 3) stakeholders do not necessarily resist change, and 4) gradual transitions can help stakeholders build confidence, which, in turn, facilitates the acceptance of the system.

We have, furthermore, carried out a survey in the field to evaluate the perceived usability of IssueTAG. We observed that a majority of the participants found the automated assignments reliable (93%) and useful (79%). Furthermore, all of the participants indicated that they would recommend the automated assignment system to other companies.

Deploying IssueTAG also enabled us identify two novel problems, which have been overlooked for before in this context.

One of these problems was how to explain the assignments to the stakeholders. Based on the informal discussions we had with the stakeholders, we quickly realized that explaining the assignments could further improve their trust in the system. Consequently, we have developed an approach for creating human-readable, non-technical explanations for the automatically made assignments. We have also evaluated the proposed approach by conducting user studies. In these studies, the participants found 93% of the explanations created for the correct assignments trustworthy. And, for the incorrect assignments, the participants found 77% of the explanations trustworthy, suggesting that given the same issue reports, these participants would have made the same or similar mistakes in assigning the reports.

The other problem was how to monitor and detect the deteriorations in assignment accuracies, such that corrective actions, including re-calibration of the underlying data mining models, can be taken in time. To this end, we have developed and empirically evaluated an online, change point detection-based approach. In the experiments carried out by using the data collected from the field, the proposed approach identified some deteriorations that aligned with the important historical events. In the simulation studies carried out by using both sudden and gradual deteriorations, the proposed approach detected all the deteriorations before the mean accuracy dropped more than 5 (out of 100) points.

To further improve the quality of the issue reports submitted, we have developed a morphological analysis-based approach to identify missing information in the issue reports, namely steps required to reproduce the reported issue (S2R), expected behavior (EB), and observed behavior (OB). In the experiments conducted by using real issue reports, the proposed approach correctly identified the S2R, EB, and OB information present in the reports with an F-measure of more than 0.82.

Last but not least, to further improve the assignment accuracy, we have used screenshot attachments present in the issue reports, which have been overlooked for in the literature, as an additional source of information for automated assignments. In the experiments, the proposed approach increased the assignment accuracy for the issue reports with screenshot attachments by 2 points (0.84% vs. 0.86%).

As future work, one possible avenue for future research is to use other sources of information for issue assignments, including source code repositories and formal/informal communications between developers/teams. Another interesting avenue is not only assigning the issue reports to the stakeholders, but also identifying the potential root causes by analyzing the similarities between the reports. Furthermore, existing approaches for some related problems in the context of bug triaging, including identifying duplicate issue reports, predicting the severity and the priority of the

issue reports as well as the effort required to resolve them, can be evaluated in our industrial context.

BIBLIOGRAPHY

Ahsan, S. N., Ferzund, J., & Wotawa, F. (2009). Automatic software bug triage system (bts) based on latent semantic indexing and support vector machine. In *2009 Fourth International Conference on Software Engineering Advances*, (pp. 216–221). IEEE.

Akın, A. A. & Akın, M. D. (2007). Zemberek, an open source nlp framework for turkic languages. *Structure*, *10*, 1–5.

Aktas, E. U., Yeniterzi, R., & Yilmaz, C. (2020). Turkish issue report classification in banking domain. In *2020 28th Signal Processing and Communications Applications Conference (SIU)*, (pp. 1–4). IEEE.

Aktas, E. U. & Yilmaz, C. (2020a). Automated issue assignment: results and insights from an industrial case. *Empirical Software Engineering*, *25*(5), 3544–3589.

Aktas, E. U. & Yilmaz, C. (2020b). An exploratory study on improving automated issue triage with attached screenshots. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*, (pp. 292–293).

Alenezi, M., Magel, K., & Banitaan, S. (2013). Efficient bug triaging using text mining. *JSW*, *8*(9), 2185–2190.

Alpaydin, E. (2010). *Introduction to Machine Learning* (2nd ed.). The MIT Press.

Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., & Guéhéneuc, Y.-G. (2008). Is it a bug or an enhancement?: a text-based approach to classify change requests. In *CASCON*, volume 8, (pp. 304–318).

Anvik, J. (2007). *Assisting bug report triage through recommendation*. PhD thesis, University of British Columbia.

Anvik, J., Hiew, L., & Murphy, G. C. (2006). Who should fix this bug? In *Proceedings of the 28th international conference on Software engineering*, (pp. 361–370). ACM.

Anvik, J. & Murphy, G. C. (2011). Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *20*(3), 10.

Atlassian (2021). Jira | issue & project tracking tool | atlassian.

Azodi, C. B., Tang, J., & Shiu, S.-H. (2020). Opening the black box: Interpretable machine learning for geneticists. *Trends in genetics*, *36*(6), 442–455.

Baltrušaitis, T., Ahuja, C., & Morency, L.-P. (2018). Multimodal machine learning: A survey and taxonomy. *IEEE transactions on pattern analysis and machine intelligence*, *41*(2), 423–443.

Baysal, O., Godfrey, M. W., & Cohen, R. (2009). A bug you like: A framework for automated assignment of bugs. In *2009 IEEE 17th International Conference on Program Comprehension*, (pp. 297–298). IEEE.

Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., & Zimmermann, T. (2008). What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, (pp. 308–318). ACM.

Bettenburg, N., Premraj, R., Zimmermann, T., & Kim, S. (2008). Duplicate bug reports considered harmful... really? In *2008 IEEE International Conference*

on Software Maintenance, (pp. 337–345). IEEE.

Bhattacharya, P., Neamtiu, I., & Shelton, C. R. (2012). Automated, highly-accurate, bug assignment using machine learning and tossing graphs. *Journal of Systems and Software*, *85*(10), 2275–2292.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, *5*, 135–146.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

Breiman, L. (2017). *Classification and regression trees*. Routledge.

Brownlee, J. (2020). Nested cross-validation for machine learning with python.

Canfora, G. & Cerulo, L. (2006). Supporting change request assignment in open source development. In *Proceedings of the 2006 ACM symposium on Applied computing*, (pp. 1767–1772). ACM.

Cawley, G. C. & Talbot, N. L. (2010). On over-fitting in model selection and subsequent selection bias in performance evaluation. *The Journal of Machine Learning Research*, *11*, 2079–2107.

Chaparro, O., Florez, J. M., & Marcus, A. (2017). Using observed behavior to reformulate queries during text retrieval-based bug localization. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, (pp. 376–387). IEEE.

Chaparro, O., Florez, J. M., & Marcus, A. (2019). Using bug descriptions to reformulate queries during text-retrieval-based bug localization. *Empirical Software Engineering*, *24*(5), 2947–3007.

Chaparro, O., Florez, J. M., Singh, U., & Marcus, A. (2019). Reformulating queries for duplicate bug report detection. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, (pp. 218–229). IEEE.

Chaparro, O., Lu, J., Zampetti, F., Moreno, L., Di Penta, M., Marcus, A., Bavota, G., & Ng, V. (2017a). Detecting missing information in bug descriptions. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, (pp. 396–407).

Chaparro, O., Lu, J., Zampetti, F., Moreno, L., Di Penta, M., Marcus, A., Bavota, G., & Ng, V. (2017b). Online replication package.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, *16*, 321–357.

Chen, L., Wang, X., & Liu, C. (2011). An approach to improving bug assignment with bug tossing graphs and bug similarities. *JSW*, *6*(3), 421–427.

Chollet, F. et al. (2015). Keras.

Cöltekin, C. (2010). A freely available morphological analyzer for turkish. In *LREC*, volume 2, (pp. 19–28).

Dedík, V. & Rossi, B. (2016). Automated bug triaging in an industrial context. In *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, (pp. 363–367). IEEE.

Dougherty, G. (2012). *Pattern recognition and classification: an introduction*. Springer Science & Business Media.

Erdinc, H. Y. & Guran, A. (2019). Semi-supervised turkish text categorization with

word2vec, doc2vec and fasttext algorithms. In *2019 27th Signal Processing and Communications Applications Conference (SIU)*, (pp. 1–4). IEEE.

Fu, W. & Menzies, T. (2017). Easy over hard: A case study on deep learning. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, (pp. 49–60).

Giger, E., Pinzger, M., & Gall, H. (2010). Predicting the fix time of bugs. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*, (pp. 52–56). ACM.

Göksel, A. & Kerslake, C. (2004). *Turkish: A comprehensive grammar*. Routledge.

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

Harris, Z. S. (1954). Distributional structure. *Word*, *10*(2-3), 146–162.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, (pp. 770–778).

Helming, J., Arndt, H., Hodaie, Z., Koegel, M., & Narayan, N. (2010). Automatic assignment of work items. In *International Conference on Evaluation of Novel Approaches to Software Engineering*, (pp. 236–250). Springer.

Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hocking, T. D., Schleiermacher, G., Janoueix-Lerosey, I., Delattre, O., Bach, F., & Vert, J.-P. (2013). Learning smoothing models of copy number profiles using breakpoint annotations. *BMC bioinformatics*, *14*, 164.

Hsu, H. & Lachenbruch, P. A. (2014). Paired t test. *Wiley StatsRef: statistics reference online*.

Jackson, B., Scargle, J. D., Barnes, D., Arabhi, S., Alt, A., Gioumousis, P., Gwin, E., Sangtrakulcharoen, P., Tan, L., & Tsai, T. T. (2005). An algorithm for optimal partitioning of data on an interval. *IEEE Signal Processing Letters*, *12*(2), 105–108.

Jalbert, N. & Weimer, W. (2008). Automated duplicate detection for bug tracking systems. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, (pp. 52–61). IEEE.

Jeong, G., Kim, S., & Zimmermann, T. (2009). Improving bug triage with bug tossing graphs. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, (pp. 111–120). ACM.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the 10th European Conference on Machine Learning*, ECML'98, (pp. 137–142). Springer-Verlag.

Jonsson, L., Borg, M., Broman, D., Sandahl, K., Eldh, S., & Runeson, P. (2016). Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts. *Empirical Software Engineering*, *21*(4), 1533–1578.

Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2017). Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, (pp. 427–431)., Valencia, Spain. Association for Computational Linguistics.

Kagdi, H., Gethers, M., Poshyvanyk, D., & Hammad, M. (2012). Assigning change

requests to software developers. *Journal of Software: Evolution and Process*, *24*(1), 3–33.

Kallis, R., Di Sorbo, A., Canfora, G., & Panichella, S. (2019). Ticket tagger: Machine learning driven issue classification. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, (pp. 406–409). IEEE.

Karim, M. R., Ruhe, G., Rahman, M. M., Garousi, V., & Zimmermann, T. (2016). An empirical investigation of single-objective and multiobjective evolutionary algorithms for developer's assignment to bugs. *Journal of Software: Evolution and Process*, *28*(12), 1025–1060.

Killick, R., Fearnhead, P., & Eckley, I. A. (2012). Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association*, *107*(500), 1590–1598.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (pp. 1746–1751)., Doha, Qatar. Association for Computational Linguistics.

Lamkanfi, A., Demeyer, S., Giger, E., & Goethals, B. (2010). Predicting the severity of a reported bug. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, (pp. 1–10). IEEE.

Lavielle, M. & Teyssière, G. (2007). *Adaptive Detection of Multiple Change-Points in Asset Price Volatility*, (pp. 129–156). Berlin, Heidelberg: Springer Berlin Heidelberg.

Lee, S.-R., Heo, M.-J., Lee, C.-G., Kim, M., & Jeong, G. (2017). Applying deep learning based automatic bug triager to industrial projects. In *Proceedings of the 2017 11th Joint Meeting on foundations of software engineering*, (pp. 926–931).

Lin, Z., Shu, F., Yang, Y., Hu, C., & Wang, Q. (2009). An empirical study on bug assignment automation using chinese bug data. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, (pp. 451–455). IEEE.

Linares-Vásquez, M., Hossen, K., Dang, H., Kagdi, H., Gethers, M., & Poshyvanyk, D. (2012). Triaging incoming change requests: Bug or commit history, or code authorship? In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, (pp. 451–460). IEEE.

Matter, D., Kuhn, A., & Nierstrasz, O. (2009). Assigning bug reports using a vocabulary-based expertise model of developers. In *2009 6th IEEE international working conference on mining software repositories*, (pp. 131–140). IEEE.

Menzies, T. & Marcus, A. (2008). Automated severity assessment of software defect reports. In *2008 IEEE International Conference on Software Maintenance*, (pp. 346–355). IEEE.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. In Bengio, Y. & LeCun, Y. (Eds.), *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., & Weinberger, K. Q. (Eds.),

Advances in Neural Information Processing Systems, volume 26. Curran Associates, Inc.

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

Nagwani, N. K. & Verma, S. (2012). Predicting expert developers for newly reported bugs using frequent terms similarities of bug attributes. In *2011 Ninth International Conference on ICT and Knowledge Engineering*, (pp. 113–117). IEEE.

Nanayakkara, S., Fogarty, S., Tremeer, M., Ross, K., Richards, B., Bergmeir, C., Xu, S., Stub, D., Smith, K., Tacey, M., et al. (2018). Characterising risk of in-hospital mortality following cardiac arrest using machine learning: A retrospective international registry study. *PLoS medicine*, *15*(11), e1002709.

Oflazer, K. (1994). Two-level description of turkish morphology. *Literary and linguistic computing*, *9*(2), 137–148.

Oflazer, K. (2014). Turkish and its challenges for language processing. *Language resources and evaluation*, *48*(4), 639–653.

Pandey, N., Sanyal, D. K., Hudait, A., & Sen, A. (2017). Automated classification of software issue reports using machine learning techniques: an empirical study. *Innovations in Systems and Software Engineering*, *13*(4), 279–297.

Park, J.-w., Lee, M.-W., Kim, J., Hwang, S.-w., & Kim, S. (2011). Costriage: A cost-aware triage algorithm for bug reporting systems. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning*, (pp. 1310–1318). PMLR.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *Journal of machine learning research*, *12*(Oct), 2825–2830.

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, (pp. 1532–1543).

Podgurski, A., Leon, D., Francis, P., Masri, W., Minch, M., Jiayang Sun, & Bin Wang (2003). Automated support for classifying software failure reports. In *25th International Conference on Software Engineering, 2003. Proceedings.*, (pp. 465–475).

Pressman, R. S. (2005). *Software engineering: a practitioner's approach*. Palgrave Macmillan.

Raschka, S. (2018a). Mlxtend: Providing machine learning and data science utilities and extensions to python's scientific computing stack. *J. Open Source Software*, *3*(24), 638.

Raschka, S. (2018b). Model evaluation, model selection, and algorithm selection in machine learning. *CoRR*, *abs/1811.12808*.

Reyes, M., Meier, R., Pereira, S., Silva, C. A., Dahlweid, F.-M., Tengg-Kobligk, H. v., Summers, R. M., & Wiest, R. (2020). On the interpretability of artificial intelligence in radiology: challenges and opportunities. *Radiology: Artificial Intelligence*, *2*(3), e190043.

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, (pp. 1135–1144). ACM.

Runeson, P., Alexandersson, M., & Nyholm, O. (2007). Detection of duplicate defect reports using natural language processing. In *Proceedings of the 29th international conference on Software Engineering*, (pp. 499–510). IEEE Computer Society.

Schütze, H., Manning, C. D., & Raghavan, P. (2008). *Introduction to information retrieval*, volume 39. Cambridge University Press Cambridge.

Scott, A. J. & Knott, M. (1974). A cluster analysis method for grouping means in the analysis of variance. *Biometrics*, *30*(3), 507–512.

Shokripour, R., Kasirun, Z. M., Zamani, S., & Anvik, J. (2012). Automatic bug assignment using information extraction methods. In *2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, (pp. 144–149). IEEE.

Simonyan, K. & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In Bengio, Y. & LeCun, Y. (Eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Singh, A., Goswami, V., Natarajan, V., Jiang, Y., Chen, X., Shah, M., Rohrbach, M., Batra, D., & Parikh, D. (2020). Mmf: A multimodal framework for vision and language research.

Smith, R. (2007). An overview of the tesseract ocr engine. In *Ninth international conference on document analysis and recognition (ICDAR 2007)*, volume 2, (pp. 629–633). IEEE.

Song, Y. & Chaparro, O. (2020). Bee: a tool for structuring and analyzing bug reports. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, (pp. 1551–1555).

Tamrawi, A., Nguyen, T. T., Al-Kofahi, J. M., & Nguyen, T. N. (2011). 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*, (pp. 365–375). ACM.

Ting, K. M. & Witten, I. H. (1999). Issues in stacked generalization. *Journal of artificial intelligence research*, *10*, 271–289.

Truong, C., Oudre, L., & Vayatis, N. (2018). ruptures: change point detection in python.

Truong, C., Oudre, L., & Vayatis, N. (2020). Selective review of offline change point detection methods. *Signal Processing*, *167*, 107299.

Wang, X., Zhang, L., Xie, T., Anvik, J., & Sun, J. (2008). An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th international conference on Software engineering*, (pp. 461–470). ACM.

Weiss, C., Premraj, R., Zimmermann, T., & Zeller, A. (2007). How long will it take to fix this bug? In *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*, (pp. 1–1). IEEE.

Wolpert, D. H. (1992). Stacked generalization. *Neural networks*, *5*(2), 241–259.

Wu, W., Zhang, W., Yang, Y., & Wang, Q. (2011). Drex: Developer recommendation with k-nearest-neighbor search and expertise ranking. In *2011 18th Asia-Pacific Software Engineering Conference*, (pp. 389–396). IEEE.

Xia, X., Lo, D., Wang, X., & Zhou, B. (2013). Accurate developer recommendation for bug resolution. In *2013 20th Working Conference on Reverse Engineering (WCRE)*, (pp. 72–81). IEEE.

Xie, X., Zhang, W., Yang, Y., & Wang, Q. (2012). Dretom: Developer recommendation based on topic models for bug resolution. In *Proceedings of the 8th international conference on predictive models in software engineering*, (pp. 19–28). ACM.

Zeller, A. (2009). *Why programs fail: a guide to systematic debugging*. Elsevier.

Zhang, H., Gong, L., & Versteeg, S. (2013). Predicting bug-fixing time: an empirical study of commercial software projects. In *Proceedings of the 2013 international conference on software engineering*, (pp. 1042–1051). IEEE Press.

Zimmermann, T., Premraj, R., Bettenburg, N., Just, S., Schroter, A., & Weiss, C. (2010). What makes a good bug report? *IEEE Transactions on Software Engineering*, *36*(5), 618–643.