THE COOPER UNION FOR THE ADVANCEMENT OF SCIENCE AND ART

MASTER'S THESIS

On Software Development Product Management: Feature Selection and Model Analysis for Predicting Jira Issue Attributes

Author:
Eugene Sokolov

Supervisor:
Professor Carl Sable

A thesis submitted in fulfillment of the requirements for the degree of Master of Engineering

in the

Albert Nerken School of Engineering
Department of Electrical Engineering

April 2017

THE COOPER UNION FOR THE ADVANCEMENT OF SCIENCE AND ART

ALBERT NERKEN SCHOOL OF ENGINEERING

This thesis was prepared under the direction of the Candidate's Thesis Advisor and has received approval. It was submitted to the Dean of the School of Engineering and the full Faculty, and was approved as partial fulfillment of the requirements for the degree of Master of Engineering.

Dean, School of Engineering - Date

Prof. Carl Sable - Date
Candidate's Thesis Advisor

Acknowledgements

I would like to thank my advisor Professor Carl Sable for the guidance, support, knowledge, and overall help he has provided my classmates and I throughout our time at Cooper Union.

I would also like to thank The Cooper Union, Peter Cooper, and his descendants for giving students like myself the opportunity to study under a full tuition scholarship. I am very grateful toward the dedicated and enthusiastic faculty for their continuous support and for providing this invaluable education.

Finally, I would not have been able to survive through Cooper Union without my spontaneous friends, intelligent classmates, and supportive family. I am grateful to be surrounded by such phenomenal people.

THE COOPER UNION FOR THE ADVANCEMENT OF SCIENCE AND ART

Abstract

Department of Electrical Engineering

Master of Engineering

On Software Development Product Management: Feature Selection and Model Analysis for Predicting Jira Issue Attributes

by Eugene Sokolov

In this paper, we explore feature selection and model analysis for predicting bug attribute labels for issues in a modified Jira issue tracking repository dataset. Analysing different features of the dataset and performing model exploration, we were able to predict issue completion time, issue priority, and issue resolution with high precision and recall. We built on previous research on predicting such issue tracking classes and confirmed the importance of features such as *reporter*, *assignee*, and *watchers*.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Issue tracking systems, such as Bugzilla, Jira, and many others, are used by many software development teams to track developer progress, assign tasks, and, manage products. In our work, we built on previous research [1, 2, 3] in analysing features of a Jira dataset provided by Ortu et. al. [4]. In addition to the Jira issue report features, which include priority, start and end datetimes, reportee, assignee, and other fields related to issue tracking, the dataset also includes emotion and sentiment related features derived from the comments about the issues.

Additionally, we have explored various models and features to predict certain classes of the issues. Using several models including SVM and XGBoost, we obtain high precision and recall scores when predicting issue completion time, issue resolution, and, issue priority. Our results and models can be used by software development product managers and senior engineers to self evaluate software developers and teams and improve software development life cycles.

Our work is motivated by attempting to analyze software developers and gain insight into their habits. We aimed to answer questions such as: What is most important in pushing developers to complete issues? Does sentiment and emotion play a role in helping resolve issues quickly? What is most important in categorizing an issue as important or not (besides the obvious label *priority*)? In answering such questions, we believe that we can gain insight into the workflow of software developers and help to better manage them.

Chapter 2

Background

2.1 Support Vector Machines

Support vector machines (SVMs) are a set of classification and regression machine learning methods applied to a wide set of problems. Invented in 1963 and expanded to tackle non-linear classification problems in 1992 with the application of a kernel, SVMs aim at finding optimal hyperplanes, the boundaries with the maximal margin of separation between two classes. A boundary is found in some N-dimensional space which separates points of type A from points of type B. After the hyperplanes are found, new test data can be classified using the separating hyperplanes [5].

Let $\{x_i, y_i\}$, i = 1, 2,..., L be L training data vectors x_i with class labels y_i, and $y_i \in \{-1, +1\}$ for binary classification. Given an input vector **x**, the training of an SVM constructs a classifier of the form:

$$g(x) = sign(\sum_{i=1}^{L} \alpha_i y_i K(x_i, x) + b) \tag{2.1}$$

where $\{\alpha_i\}$ are non-negative Lagrange multipliers each of which corresponds to an example from the training data, b is a bias constant, and K($\cdot$, $\cdot$) is a kernel function. During the training phase, in order for SVMs to easily map a dataset to a higher dimensional space and search for an optimal linear hyperplane, a kernel is applied to every pair of input vectors. Kernels are functions that take two inputs and output the similarity of the inputs. Frequently used kernel functions are the polynomial kernel $K(x_i, x_j) = (x_i \cdot x_j + 1)^d$ and the Gaussian radial basis function (RBF) $K(x_i, x_j) = e^{\frac{-|x_i - x_j|^2}{2\sigma^2}}$ [5]. Gaussian kernels tend to yield good performance under general smoothness assumptions [6].

Figure 2.1 illustrates an optimal hyperplane for two classes of training data. The Kernel function effectively maps the original data from one space to another space. In $\mathbf{R}^2$, the data is not linearly separable. However, the data is linearly separable after being mapped to a higher dimension, $\mathbf{R}^3$, commonly known as the "kernel trick" [5].

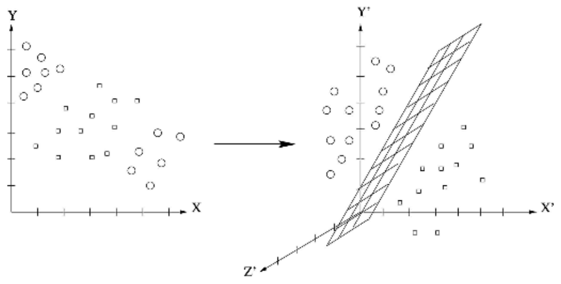


FIGURE 2.1: Projection of data into higher dimension where data is linearly separable [7]

Classification with an SVM is fast because the SVM only uses data points closest to the maximum margin hyperplane, known as support vectors, for classification as opposed to all of the available data. Figure 2.2 helps explain the concept of support vectors.

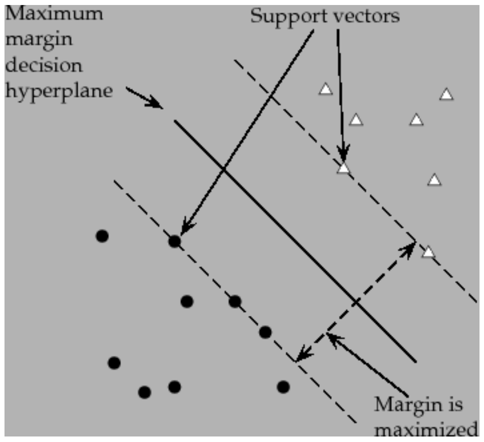


FIGURE 2.2: SVM with maximum marginal supports, or support vectors [8]

SVMs are designed for binary classification. In order to perform multiclass classification, two commonly used methods are one-versus rest and one-versus-one approaches. The one-versus-rest method constructs k classifiers for k classes, each of which separates a class from the rest of the data, and a test data point will be classified as the class with the largest margin. The one-versus-one method constructs a classifier for each pair of classes. The test data point is classified as the class chosen by the most classifiers [5].

2.2 Gradient Boosting

Gradient boosting (GB) is a regression and classification supervised machine learning method in the form of an ensemble of weak prediction models, typically decision trees. The idea was proposed by Leo Breiman in 1997 [9] and further developments were made by Jerome H. Friedman and many others [10, 11]. GB aims to combine outputs of many "weak" classifiers to produce a powerful "committee", in an iterative fashion, often minimizing a loss function. An implementation known as "iterative functional gradient descent algorithms" optimize a loss function by iteratively choosing a function or weak hypothesis that points in the negative gradient direction. As such, each step of the algorithm correctively updates the error of the loss function.

The loss in using $f(x)$ to predict *y* on the training data is:

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i)). \tag{2.2}$$

The goal is to minimize *L(f)* with respect to *f* at each step of the solution tree. Thus, the tree predictions $T(x_i, \theta_m)$ are analogous to the components of the negative gradient [12].

A generic gradient tree-boosting algorithm is described in Algorithm 1 [12]. Specific algorithms can be obtained by using different loss functions $L(y, f(x))$.

Algorithm 1 Gradient Tree Boosting Algorithm.

```
Initialize $f_0(x)$ = $\text{argmin}_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
for $m$ = 1 to M do
    for $i$ = 1, 2, . . . , N do
        $r_{im} = -[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]_{f=f_{m-1}}$.
    end for
    Fit a regression tree to the targets $r_{im}$ giving terminal regions $R_{jm}$, $j = 1, 2, \ldots, J_m$.
    for $j$ = 1, 2, . . . , $J_m$ do
        $\gamma_{jm}$ = $\text{argmin}_\gamma \sum\limits_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$.
    end for
    Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
end for
Output $\hat{f}(x)$ = $f_M(x)$.
```

First, initialize the optimal constant model, which is just a single terminal node tree. Next, iteratively update the model by computing the negative gradient of the loss function, referred to as generalized or *pseudo* residuals, r. This is done an arbitrary amount of times, with each iterative progressively updating the model. Build the base learner to these pseudo residuals using training data to fit a regression tree and solve a one dimensional optimization problem to find γ_{jkm} - to which the model is updated. After this is done iteratively, the output $\hat{f}$ is obtained.

2.3 XGBoost

A modern and popular implementation of gradient tree boosting is known as XGBoost [13]. Short for "eXtreme Gradient Boosting", XGBoost provides a "scalable, portable and distributed gradient boosting (GBM, GBRT, GBDT) Library" which is open source. It is a scalable end-to-end tree boosting system that is widely used by data scientists to achieve state of the art results in many machine learning challenges and competitions. The speed, efficiency, portability, and high performance of the algorithm makes it appealing to many machine learning applications.

A few advantages of using XGBoost include speed and memory efficiency. The most time consuming part of tree learning is to get the data into sorted order. In order to reduce this computation time, XGBoost stores the data in in-memory units, which are called blocks. Data in each block is stored in the compressed column (CSC) format, with each column sorted by the corresponding feature value. This input data layout only needs to be computed once before training, and can be reused in later iterations. This structure also helps with the approximation algorithms using in gradient boosting. Using the stored block structure, the algorithm to approximate tree split points (known as the quantile finding step) becomes a linear scan over the sorted columns. Additionally, collecting statistics for each column can be parallelized, giving us a parallel algorithm for split finding.

To increase memory efficiency, XGBoost implements a cache-aware prefetching algorithm to limit the number of cache-misses. The algorithm allocates an internal buffer in each thread, fetching the gradient statistics into it, and then performs accumulation in a mini-batch manner. This prefetching changes the direct read/write dependency to a longer dependency and helps to reduce the runtime overhead when the number of rows in the block is large, which is quite frequent.

2.4 Bug Attribute Prediction

An issue tracking system is a software repository that hosts all development tasks of a software organization, e.g., new features, bug fixes and other maintenance tasks. For each task, the system provides a description, administrative metadata such as the state of the issue (e.g., opened, resolved, or unable to fix), the priority, comments made by members of the team, attachments and other various information necessary to manage the project. With this information, many have attempted to explore how developers interact [14], as well as how they feel about the project and their peers [15, 16]. Attempts in predicting issue completion time could help product managers better plan their development cycle when using issue tracking systems.

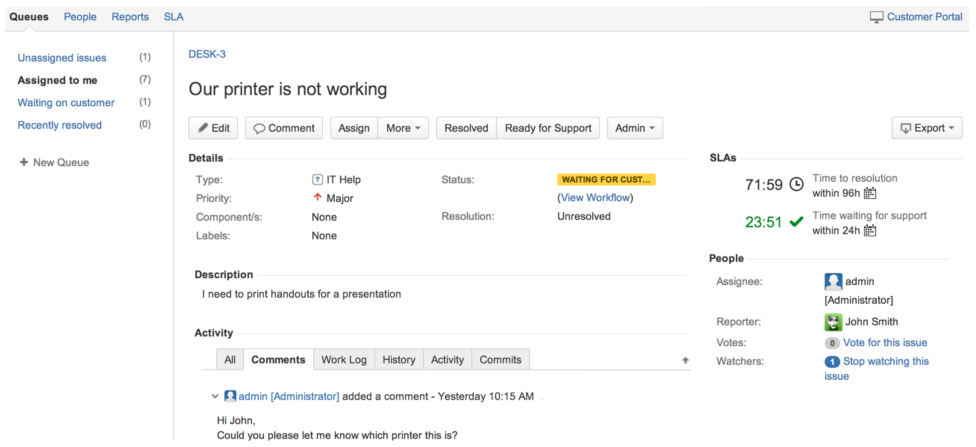
In an issue comments section, developers discuss issues by providing technical details and/or opinions. From this textual information it is possible to extract emotions, sentiments, and politeness. Murgia et al. [15] showed that developers do express emotions such as love, joy, and, sadness towards colleagues. Ortu et al. [14] showed that emotions contained in these issue comments have negligible correlation with each other. More of these related works are discussed in Section 3.

2.5 The Jira Dataset

Jira is one of the most common issue tracking systems. Developed by Atlassian in 2002, Jira is a proprietary issue tracking product providing bug/issue tracking and product management functions. The name derives from a truncation of *Gojira*, the Japanese name for Godzilla, itself a reference to Jira's main competitor Bugzilla. According to Atlassian, Jira is used by over 25,000 customers in 122 countries around the world. Written in Java, Jira supports remote procedure calls, REST, SOAP, XML-RPC and has integration with source control programs such as Git, Mercurial, Subversion and others. It ships with various translations including English, French, German, Spanish, and, Japanese. The main features of Jira for agile software development are the ability to plan development iterations, create iteration reports, and track bug functionality [3].

A public Jira dataset is provided by Ortu et al. [4]. The authors mined the issue repository of the Apache software foundation gathering data from over one thousand projects, seven hundred thousand issue reports and two million comments. In addition to the Jira issue report features, which include priority, start and end datetimes, reportee, assignee, and other fields related to issue tracking, the dataset also includes emotion and sentiment related features derived from the comments about the issues. The dataset is useful for studying productivity [17], projects' attractiveness to new developers [17], building predictive models to analyze social and technical debt in software development [18, 19], estimating bug fixing time and bug life cycles [20], testing hypotheses concerning software maintenance and studying the relationships among software metrics [21], etc.

2.6 Jira Feature Analysis



DESK-3

Our printer is not working

Details

Type:	IT Help	Status:	WAITING FOR CUST... (View Workflow)
Priority:	Major		
Component/s:	None	Resolution:	Unresolved
Labels:	None		

SLAs

71:59 Time to resolution within 96h

23:51 Time waiting for support within 24h

People

Assignee:	admin [Administrator]
Reporter:	John Smith
Votes:	0 Vote for this issue
Watchers:	1 Stop watching this issue

Description

I need to print handouts for a presentation

Activity

All | Comments | Work Log | History | Activity | Commits

admin [Administrator] added a comment - Yesterday 10:15 AM

Hi John,
Could you please let me know which printer this is?

FIGURE 2.3: An example JIRA Service Desk issue [3]

An example Jira issue can be seen in Figure 2.3. Although organizations can make use of Jira in different manners, all Jira issues have a general commonality. Descriptive fields include project name, labels, description, creation date, resolved date, reportee (the person who created and is managing the issue), assignee (the person who is responsible for the task of the issue), priority, status, and resolution. An issue type can be a bug, improvement, new feature, task, or custom issue. A priority can be highest, high, medium, low, or lowest. A status can be open, in progress, resolved, reopened, or closed. A resolution can be fixed, won't fix, duplicate, incomplete, cannot reproduce, won't do. In addition to these descriptive fields, developers are able to comment on an issue and share opinions, thoughts, actions, or advice. An issue can have several developers, known as "watchers", who follow it and receive notifications. Developers are able to "vote" on an issue to voice a preference that it be resolved or completed.

Sentiment scores are measured using the SentiStrength tool [22], which is able to estimate the degree of positive and negative sentiment in short texts, even for informal language. The tool reports two scores, as research has shown that humans positive and negative sentiment in parallel [23] - hence mixed emotions.

Mood has been measured using a tool by De Smedt et al [24]. Pattern for Python is a package with functionality for web mining (Google + Twitter + Wikipedia, web spider, HTML DOM parser), natural language processing (tagger/chunker, n-gram search, sentiment analysis, WordNet), machine learning (vector space model, k-means clustering, Naive Bayes + k-NN + SVM classifiers) and network analysis (graph centrality and visualization). The tool does not specialize in one domain but provides general cross-domain functionality

Politeness has been measured using Danescu et. al.'s tool [25]. They proposed a computational framework for identifying the linguistic aspect of politeness using domain-independent lexical and syntactic features operationalizing key components of politeness theory, such as *indirection, deference, impersonalization*, and, *modality*.

Chapter 3

Related Works

3.1 Sentiment and Emotion Mining in Software Development

A recent trend has emerged to promote positive aspects of happiness and affect in general, e.g. [14, 26, 27, 28, 29, 30, 31, 32]. It has been shown that positive emotions like happiness help people to be more creative [33]. In software development, previous research has shown that emotions effect task quality, productivity, creativity, group rapport and job satisfaction [34]. Developers and managers alike need to be aware of the emotions and feelings of the people working on projects they are involved with to take corrective action where necessary.

Sentiment polarity analysis has been applied in the software engineering context to study commit comments in GitHub [35, 36], GitHub discussions related to security [37], and, the activity of contributors in Gentoo [38]. Guzman et. al. [35] used lexical sentiment analysis to study emotions expressed in commit comments of different open source projects and analyzed their relationships with different factors such as programming language and time of day. They found that projects developed in Java tend to have more negative commit comments, as do commit comments written on Mondays. Ortu et. al. [39] found that team diversity (gender and nationality) is linked with lower issue fixing time.

The Jira dataset provided by Ortu et. Al [4] has been analyzed by many. Murgia et. al. [15] showed that developers do express emotions such as love, joy, and, sadness towards colleagues and software artifacts. Ortu et. al. [14] showed that emotions contained in these issue comments have negligible correlation with each other. They also showed that bug fixing time correlates with the affects expressed by developers in issue comments. Murgia et. al. [40] and Ortu et. Al. [14] showed that issue fixing time is related respectively to the type of maintenance performance and the affects in issue reports. Ortu et. al. showed that the more polite developers were, the more new developers wanted to be a part of a project and the more they were willing to continue working on it over time [41]. In addition, Ortu et. al. [2] found the presence of developers' communities for 7 open source projects hosted in Jira showing that social interaction is a large part of the software development field.

3.2 Uses of SentiStrength

SentiStrength extracts sentiment strength from informal English text, using new methods to exploit the de facto grammars and spelling styles of cyberspace [42].This tool simultaneously assigns both a positive and a negative score to bits of English text, as research shows that humans can express both types of sentiment in parallel. Positive sentiment strength scores range from +1 (not positive) to +5 (extremely positive). Similarly, negative sentiment strength scores range from -1 to -5. The tool works by assigning scores to tokens in a dictionary which includes common emoticons. The final sentiment strength for a bit of text is then computed by taking the maximum score among all individual positive scores, and similarly for the negative sentiment strength. Previous research has shown that SentiStrength has a good accuracy for many applications of short text in Twitter and movie reviews [43], Yahoo! Answers [44], and, Stack Overflow [45]. Bazelli et. al. [46] studied the personality traits of authors of questions on StackOverFlow and showed that top reputed authors are less neurotic, more extroverted and open compared to medium and low reputed authors.

3.3 Predicting Bug Fix Time

Previous research in predicting bug fix time has used various datasets, feature and model analysis. Giger et. al. [1] explored prediction models in a series of empirical studies with bug report data taken from Bugzilla of the three open source projects Eclipse, Mozilla, and, Gnome. Classifying bug reports into *fast* and *slow* using attributes such as *assignee, reportee*, and, *monthOpened* shows precision and recall between 60% and 70%. Post-submission data of bug reports improves the performance of prediction models by 5% to 10%, including information such as *milestone*. Similarly, Zhang et. al. [47] uses Markov-based methods to estimate the total amount of time required to fix a bug, in a similar *fast* and *slow* category and saw similar results to Giger et. al. Using multivariate and uni-variate regression testing on the Mozilla project, Bhattacharya et. al. [48] show results between 30% and 49%.

Chapter 4

Experimentation

4.1 Overview

In this chapter, we explain our approach to predicting completion time, priority and, resolution in Jira and the different steps we took throughout our analysis. In Chapter 5 we further discuss the results of our experiments and potential implications and applications.

4.2 Considerations of Features

The first step in our analysis involved cleaning the data. Entries that had null values for any feature, including assignee, reportee, priority, resolution as well as non resolved issues were discarded in our analysis. Certain features, such as *politeness*, which originally classified a comment as either *polite* or *impolite*, were manipulated and casted to numerical types in order to leverage the models used. Completion time was defined to be the difference in resolved time and created time. Entries whose completion time was less than 5 minutes were also discarded in our analysis. This method was taken from Giger et al. [1] who noticed removing that post-submission data of bug reports improves prediction models. This is due to the fact that developers may find a bug and quickly fix it, after which they open Jira to document the issue and immediately close it. In addition, we discarded the feature *resolution* when predicting completion time as it brought non-causal information to the model. Presumably, when predicting an issue completion time, one would not know what the final resolution of the issue is yet.

We predict three separate categories, issue completion time (how long it took to complete a bug), issue priority (how important an issue is), and, issue resolution (the final outcome of an issue). To predict these categories, our features consisted of generic features found in an issue tracking system as well as sentiment and emotion features found in the Jira dataset provided by Ortu et. al. [4]. Each of these features were described in detail in Chapter 2. Our feature set included all of the features described in Chapter 2 of the Ortu et. al. Jira dataset, but low weighted features were dropped throughout the analysis.

A novel aspect of this work was the binning of the completion time of bug reports. Giger et. al. classifies bug reports as either *fast* or *slow* using the median of completion time [1]. In this paper, completion time is binned as show in Table 4.1. Previous research [1, 20, 47] which focused on classifying *fast* or *slow* completion time used a given a threshold, usually the median of the given data. Although it may be useful to use such an approach, we believe that our binning is better suited for more practical applications. Our binning is also more difficult to predict due to the increased number of classes, five instead of 2. The application of our binning is further discussed in Chapter 5.

Bin	Completion Time (days)
Day	<=1
Week	> 1 and <= 7
Month	> 7 and <= 31
Year	> 31 and <= 365
Long Term	> 365

TABLE 4.1: Issue Completion Time Bins

4.3 Initial Experiments

Most previous research in analyzing issue tracking systems focuses on using naive bayes or linear models. We initially started our analysis by using such methods to predict completion time using a *fast* and *slow* split. We noticed that naive bayes methods did not perform as well as logistic regression in classifying issue completion. Using logistic regression, we saw similar results to previous research, varying dependent on the features used; we found that *reportee, assignee, watchers*, and, *votes* are the most influential features. Regardless of our feature selection, we were unable to improve the results of published papers.

In addition, we used a similar approach to predict issue resolution and priority. We noticed these results, in particular the precision and recall, were not as high as predicting completion time due to a large skew in the data. Most issues were marked with a resolution of *resolved* meaning a developer has completed the task. In addition, most issues were marked with a priority of *medium* as this is the default setting when creating a new Jira issue.

To reiterate Section 2.6, we have provided Table 4.2 to describe the features used in our analysis.

Feature	Description
Assignee	individual responsible for the completion of the task, usually a software developer
Reporter	individual overlooking the completion of the task, usually a manager
Votes	number of "likes" or "upvotes" on the issue, usually used to express an importance of an issue
Watchers	number of individuals who are following the issue
Sentiment	sentiment score assigned to a comment as calculated by SentiStrength
Politeness	politeness score assigned to a comment as calculated by Danescu et. al.'s tool
Politeness Confidence Level	the confidence of the politeness score
Mood	mood score assigned to a comment as calculated by Pattern for Python
Modality	linguistic modality score assigned to a comment as calculated by Pattern for Python, i.e. the presence of auxiliary verbs (e.g., could, would) and adverbs (e.g., definitely, maybe) that express certainty

TABLE 4.2: Description of Features Used in Our Analysis

4.4 Further Analysis

After some initial analysis, we decided to try different prediction models. We attempted to use SVMs, random forests and gradient boosting to predict issue completion time, priority and resolution. These models showed some improvement as compared to previous models used. They also provided feature analysis in the form of weighted features.

Based on initial experiments, we narrowed down our final analysis to include two models, SVM using an RBF kernel and XGBoost. These two models proved to be the most accurate in predicting our classes for issue completion time, priority and resolution. For all of our analysis, we used an 80-20 data split for training and testing. When training data, we used 5-fold cross validation to build our models. To ensure that we would not overfit in our training, the 20% data split was set aside and only used in the final testing to get our results. These results are described in Chapter 5.

Chapter 5

Results and Evaluation

In this chapter we present our final results in predicting Jira issue completion time, issue priority, and issue resolution. We compare our results to previous research, when available, and provide an analysis and evaluation. The features in this analysis are described in Section 4.3.

5.1 Issue Completion Time

We predict Jira issue completion time using two methods of binning: the *fast/slow* approach as taken in previous research, and, our novel binning approach consisting of *day, week, month, year* and, *long-term* classes as described in Table 4.1. We note that the output classes are fairly evenly distributed.

5.1.1 Results with SVM with RBF kernel

Table 5.1 shows our results predicting issue completion time using the *fast/slow* bins using an SVM with an RBF kernel. Our results show extremely high precision and recall. Table 5.2 shows our results predicting issue completion time using fine-grained bins using SVM with an RBF kernel. Increasing the number of output classes degrades the performance of the model, as expected, but it still performs well. Note that the class *long-term* has the highest scores, followed by *year*. Our results score higher precision and recall scores than those presented by Ortu et. al. [14] and Giger et. al. [1] where similar analysis is performed.

Bucket	Precision (%)	Recall (%)	F1 (%)
Fast	95	91	93
Slow	85	92	88

TABLE 5.1: Predicting Fast/Slow Issue Completion Time using SVMs

Bucket	Precision (%)	Recall (%)	F1 (%)
Day	65	78	71
Week	78	60	69
Month	85	66	74
Year	81	92	86
Long-Term	99	93	96

TABLE 5.2: Predicting Fne-grained Binned Issue Completion Time using SVMs

5.1.2 Results with XGBoost

Table 5.3 shows our results predicting issue completion time using the *fast/slow* bins with XGBoost. Again, our results show high precision and recall. Table 5.4 shows our results predicting issue completion time using fine-grained bins with XGBoost. Similar to SVMs, increasing the number of output classes degrades the performance of the model but it still performs fairly well. Also, similar to SVMs, the class *long-term* has the highest scores, followed by *year*. In general, the SVM approach performed better than XGBoost for this task.

Bucket	Precision (%)	Recall (%)	F1 (%)
Fast	95	84	89
Slow	77	92	84

TABLE 5.3: Predicting Fast/Slow Issue Completion Time using XGBoost

Bucket	Precision (%)	Recall (%)	F1 (%)
Day	56	67	61
Week	70	42	54
Month	70	44	54
Year	71	88	78
Long-Term	97	90	93

TABLE 5.4: Predicting Fne-grained Binned Issue Completion Time using XGBoost

Class	Weight (%)
Assignee	23
Reporter	55
Mood	3
Watchers	18

TABLE 5.5: Weights of XGBoost Predicting Jira Issue Completion Time

Table 5.5 shows the weight distribution of the XGBoost model. We note that the most weighted features include *assignee* and *reporter* as well as *watchers*. Notably, the most weighted feature is the *reporter*, the individual who created the issue ticket. These are features which are inherent in the Jira tracking system and are created by users of the issue without writing comments. From the perspective of issue comments, sentiment analysis and emotions do not play a large part in predicting issue completion time.

5.2 Issue Resolution

We predict Jira issue resolution as the output class described in Section 2.6. We note that the output classes are not evenly distributed and as such, combined similar classes. Table 5.6 shows the approximate distribution per combined class. *Fixed* includes only the Jira *fixed* class. *Unfixed* includes the Jira classes *cannot reproduce, rejected,* and, *won't fix*. *Non-actionable issue* includes the Jira classes *duplicate, out of date,* and, *incomplete* and is meant to describe issues which are faulty due to an error in writing the issue request.

Class	Distribution (%)
Fixed	79
Unfixed	12
Non-Actionable Issue	9

TABLE 5.6: Distribution of Combined Jira Issue Resolution Classes

5.2.1 Results with SVM with RBF kernel

Table 5.7 shows our results predicting issue resolution using an SVM with an RBF kernel. We note very high scores, specifically for the *fixed* class.

Bucket	Precision (%)	Recall (%)	F1 (%)
Fixed	93	99	96
Unfixed	95	77	85
Major	98	73	83

TABLE 5.7: Predicting Issue Resolution using SVMs

5.2.2 Results with XGBoost

Table 5.8 shows our results predicting issue resolution using XGBoost. These results slightly mimic the results in Section 5.2.1. Table 5.9 shows the weights of the features of the XGBoost model. These weights are similar to those described in Section 5.1.2.

Bucket	Precision (%)	Recall (%)	F1 (%)
Fixed	90	99	94
Unfixed	89	52	66
Major	83	63	72

TABLE 5.8: Predicting Issue Resolution using XGBoost

Class	Weight (%)
Assignee	16
Reporter	42
Politeness Confidence level	15
Votes	13
Watchers	14

TABLE 5.9: Weights of XGBoost Predicting Jira Issue Resolution

5.3 Issue Priority

We predict Jira issue priority with the output classes as described in Section 2.6. We note that the output classes are not evenly distributed. Table 5.10 shows the approximate distribution per priority class. Note that the majority of the support is the *major* class, the default class when creating a Jira issue.

Class	Distribution (%)
Blocker	2.5
Critical	5
Major	80
Minor	10
Trivial	2.5

TABLE 5.10: Distribution of Jira Issue Priority Classes

5.3.1 Results with SVM with RBF kernel

Table 5.11 shows our results predicting issue priority using using SVM with an RBF kernel. Our results show varied results dependent on each class. The classes *critical* and *major* scored nearly perfectly. These are classes which are more common among Jira issues as they are more neutral priority assignments. On the other hand, the classes *blocker* and *trivial* did not score so well. These are classes which lie on opposite ends of the priority spectrum and hence have the lowest number of instances.

Bucket	Precision (%)	Recall (%)	F1 (%)
Blocker	79	56	66
Critical	100	90	95
Major	93	99	96
Minor	92	69	79
Trivial	100	52	69

TABLE 5.11: Predicting Issue Priority using SVMs

5.3.2 Results with XGBoost

Table 5.12 shows our results predicting issue priority using XGBoost. This model has very high precision, but recall and F-1 scores are very low compared to the SVM model. Table 5.13 shows the weights of the features of the XGBoost model. These weights are similar to those described in Section 5.1.2.

Bucket	Precision (%)	Recall (%)	F1 (%)
Blocker	95	38	54
Critical	100	60	75
Major	88	99	94
Minor	92	45	61
Trivial	93	28	43

TABLE 5.12: Predicting Issue Priority using XGBoost

Class	Weight (%)
Assignee	15
Reporter	39
Modality	7
Politeness Confidence Level	12
Votes	12
Watchers	14

TABLE 5.13: Weights of XGBoost Predicting Jira Issue Priority

5.4 Evaluation of Results

Our feature analysis coincides with that of Ortu et. al. [14]. For the most part, sentiment and emotion analysis of Jira comments did not play a large role in predicting issue completion time, priority or resolution. It seems as such these features are not indicative of the time it takes to complete an issue, the priority an issue has, or the final outcome of an issue.

The most influential feature in predicting issue completion time, resolution and priority was the *reporter* field, as well as the *assignee* and *watchers* fields. These results coincide with that of Giger et. al. [1]. It is logical to assume that an assignee will complete different issues in a similar manner, as their ability to do their work probably does not change much. It is interesting to see that the number of watchers on an issue is correlated to issue completion time, resolution, and priority. This could imply that the higher number of individuals are linked on an issue, the harder the assignee will work on that project to finish it.

The most influential feature, *reporter*, was weighted much higher than any other feature. This shows that the individual who opens an issue has a strong influence on the issue. This individual is usually a project manager, senior developer, or someone with more authoritative power in a company. This individual is directly responsible for setting an issue *priority*, and is able to push developers/assignees to complete certain tasks. Thus his influence over Jira issues are verified in our analysis.

Our analysis can be used by software development product managers and senior software engineers to analyze the performance of software development teams. Our models can be used internally at companies to show time efficiency of teams, to perform self evaluations, and to compare results to other software engineering teams. Such analysis can show if there is room for improvement for individuals or groups of individuals in a software development team.

Chapter 6

Future Work

6.1 Additional Features

One area of future work includes adding additional features to analyze in parallel to an issue tracking system. Graziotin et. al. [36] have taken this approach by conducting surveys of 181 software developer participants. Such an approach could be taken to construct a different set of features to explore. In addition to analyzing sentiment and emotions, other important attributes such as productivity can be explored. These additional features can be obtained by surveying software developers, to which questions such as "When are you most happy at work?", "What sort of projects do you enjoy working on the most?", or, "What sort of activities do you enjoy doing with your colleagues?" can be asked. Although a bit unlikely, it could be possible that these new features are able to predict issue completion time, priority and resolution. It could also open different areas of research, in particular, the interaction between software engineers.

Another path of feature analysis includes attempting to group issue tracking system projects by industry and analyzing how software developers across different fields compare. In addition, one may attempt to analyze how financial aspects of software development roles play into happiness and productivity, as well as various office "perks" such as the inclusion of an office gym, cafeteria, health benefits, number of happy hours, etc. It would be interesting to quantify each of these individual benefits, which can help new companies evaluate which perks they choose to provide.

Further exploring the impact of assignee and reportee to issue tracking systems, one might analyze how specific assignees and reportees impact projects. What sort of features are important in predicting specific reportee and assignee issue completion time? Why is it that certain assignees and reportees have better predictions of issue completion time, if that is true? This type of analysis may require additional features which are not originally found in any issue tracking system.

6.2 Cross Dataset Evaluation

Previous research focuses on analyzing an individual issue tracking system. Another field of exploration includes linking user Jira and GitHub profiles, which have different software development attributes, with social profiles such as Facebook, Twitter, Quora, etc. Such an analysis could provide insight on how software developers differ in their interactions at work and outside of work. Do they express positive and negative emotion differently? Do they use similar lexicons to express themselves? Are they truly happy at work or are they striving for something else in their lives? Such questions can be answered by linking software developer profiles with social datasets.

6.3 General Improvement

A final recommendation for future work involves improving current features in Jira and other issue tracking systems such as the sentiment and emotion features. This would require improving the natural language processing techniques used in applied tools, such as SentiStrength or Power for Python, for short sentence text. This is an area that is commonly explored in research today [49, 50, 51, 52, 53, 54]. Developing a domain specific tool would be useful for such analysis, as it is likely that software engineering and people in technical occupations are likely to express emotions and sentiment differently than people of other fields.

Chapter 7

Conclusion

Analysing different features of the Jira issue tracking repository and performing model exploration, we were able to predict issue completion time, issue priority, and issue resolution with high precision and recall. Building on previous research in predicting such issue tracking classes, we have confirmed the importance of features such as *reportee*, *assignee*, and *watchers*. We also used a novel approach in binning issue completion time, utilizing more fine-grained categories for completion time.

Using SVMs and XGBoost we were able to create models that can be used by product managers and senior software developers to perform self evaluations of software development teams. Analyzing issue completion time and resolution can show how efficient a team is working and comparisons can be made to external teams. Such an analysis can help improve individual developers and teams and make the software development life cycle more efficient.

Bibliography

[1] E. Giger, M. Pinzger, and H. Gall. "Predicting the Fix Time of Bugs". In: *Proceedings of the 2Nd International Workshop on Recommendation Systems for Software Engineering*. RSSE '10. Cape Town, South Africa: ACM, 2010, pp. 52–56. ISBN: 978-1-60558-974-9.

[2] M. Ortu, G. Destefanis, M. Kassab, and M. Marchesi. "Measuring and Understanding the Effectiveness of JIRA Developers Communities". In: *Proceedings of the Sixth International Workshop on Emerging Trends in Software Metrics*. WETSoM '15. Florence, Italy: IEEE Press, 2015, pp. 3–10.

[3] *Atlassion Jira Software*. Web. 2017.

[4] M. Ortu, G. Destefanis, B. Adams, A. Murgia, M. Marchesi, and R. Tonelli. "The JIRA Repository Dataset: Understanding Social Aspects of Software Development". In: *Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering*. PROMISE '15. Beijing, China: ACM, 2015, 1:1–1:4. ISBN: 978-1-4503-3715-1.

[5] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 0387310738.

[6] A. J. Smola and B. Schölkopf. “From Regularization Operators to Support Vector Kernels”. In: *Proceedings of the 10th International Conference on Neural Information Processing Systems*. NIPS’97. Denver, CO: MIT Press, 1997, pp. 343–349.

[7] C. Li, L. Khan, and B. Prabhakaran. “Real-time Classification of Variable Length Multi-attribute Motions”. In: *Knowl. Inf. Syst.* 10.2 (Aug. 2006), pp. 163–183. ISSN: 0219-1377.

[8] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008, pp. 319–346. ISBN: 0521865719, 9780521865715.

[9] L. Breiman. *Arcing the edge*. Tech. rep. 1997.

[10] J. H. Friedman. “Greedy Function Approximation: A Gradient Boosting Machine”. In: *Annals of Statistics* 29 (2000), pp. 1189–1232.

[11] J. H. Friedman. “Stochastic Gradient Boosting”. In: *Comput. Stat. Data Anal.* 38.4 (Feb. 2002), pp. 367–378. ISSN: 0167-9473.

[12] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[13] *XGBoost ReadTheDocs*. Web. 2015-2016.

[14] M. Ortu, B. Adams, G. Destefanis, P. Tourani, M. Marchesi, and R. Tonelli. “Are Bullies More Productive?: Empirical Study of Affectiveness vs. Issue Fixing Time”. In: *Proceedings of the 12th Working Conference on Mining Software Repositories*. MSR ’15. Florence, Italy: IEEE Press, 2015, pp. 303–313. ISBN: 978-0-7695-5594-2.

[15] A. Murgia, P. Tourani, B. Adams, and M. Ortu. “Do Developers Feel Emotions? An Exploratory Analysis of Emotions in Software Artifacts”. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. Hyderabad, India: ACM, 2014, pp. 262–271. ISBN: 978-1-4503-2863-0.

[16] M. Ortu, A. Murgia, G. Destefanis, P. Tourani, R. Tonelli, M. Marchesi, and B. Adams. “The Emotional Side of Software Developers in JIRA”. In: *Proceedings of the 13th International Conference on Mining Software Repositories*. MSR ’16. Austin, Texas: ACM, 2016, pp. 480–483. ISBN: 978-1-4503-4186-8.

[17] K. Yamashita, S. McIntosh, Y. Kamei, and N. Ubayashi. “Magnet or Sticky? An OSS Project-by-project Typology”. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. Hyderabad, India: ACM, 2014, pp. 344–347. ISBN: 978-1-4503-2863-0.

[18] J. Yli-Huumo, A. Maglyas, and K. Smolander. “How Do Software Development Teams Manage Technical Debt? - An Empirical Study”. In: *J. Syst. Softw.* 120.C (Oct. 2016), pp. 195–218. ISSN: 0164-1212.

[19] A. Potdar and E. Shihab. “An Exploratory Study on Self-Admitted Technical Debt”. In: *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution*. ICSME ’14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 91–100. ISBN: 978-1-4799-6146-7.

[20] S. Kim and E. J. Whitehead Jr. “How Long Did It Take to Fix Bugs?” In: *Proceedings of the 2006 International Workshop on Mining Software Repositories*. MSR ’06. Shanghai, China: ACM, 2006, pp. 173–174. ISBN: 1-59593-397-2.

[21] A. Murgia, G. Concas, R. Tonelli, M. Ortu, S. Demeyer, and M. Marchesi. “On the Influence of Maintenance Activity Types on the Issue Resolution Time”. In: *Proceedings of the 10th International Conference on Predictive Models in Software Engineering*. PROMISE ’14. Turin, Italy: ACM, 2014, pp. 12–21. ISBN: 978-1-4503-2898-2.

[22] *SentiStrength*. Web. ?

[23] R. Berrios, P. Totterdell, and S. Kellett. “Eliciting mixed emotions: a meta-analysis comparing models, types, and measures”. In: *Frontiers in psychology* 6 (2015).

[24] T. De Smedt and W. Daelemans. “Pattern for Python”. In: *J. Mach. Learn. Res.* 13 (June 2012), pp. 2063–2067. ISSN: 1532-4435.

[25] C. Danescu-Niculescu-Mizil, M. Sudhof, D. Jurafsky, J. Leskovec, and C. Potts. “A Computational Approach to Politeness with Application to Social Factors”. In: *CoRR* abs/1306.6078 (2013).

[26] D. Graziotin, X. Wang, and P. Abrahamsson. “How Do You Feel, Developer? An Explanatory Theory of the Impact of Affects on Programming Performance”. In: *CoRR* abs/1505.07240 (2015).

[27] D. Graziotin, X. Wang, and P. Abrahamsson. “Happy software developers solve problems better: psychological measurements in empirical software engineering”. In: *CoRR* abs/1505.00922 (2015).

[28] D. Graziotin, X. Wang, and P. Abrahamsson. “Do feelings matter? On the correlation of affects and the self-assessed productivity in software engineering”. In: *CoRR* abs/1408.1293 (2014).

[29] F. Fagerholm, M. Ikonen, P. Kettunen, J. Münch, V. Roto, and P. Abrahamsson. “How Do Software Developers Experience Team Performance in Lean and Agile Environments?” In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. EASE ’14. London, England, United Kingdom: ACM, 2014, 7:1–7:10. ISBN: 978-1-4503-2476-2.

[30] I. A. Khan, W.-P. Brinkman, and R. M. Hierons. “Do Moods Affect Programmers&#x2019; Debug Performance?” In: *Cogn. Technol. Work* 13.4 (Nov. 2011), pp. 245–258. ISSN: 1435-5558.

[31] G. Destefanis, M. Ortu, S. Counsell, S. Swift, M. Marchesi, and R. Tonelli. “Software development: do good manners matter?” In: *PeerJ Computer Science* 2 (2016), e73.

[32] S. C. Müller and T. Fritz. “Stuck and Frustrated or in Flow and Happy: Sensing Developers’ Emotions and Progress”. In: *Proceedings of the 37th International Conference on Software Engineering - Volume 1*. ICSE ’15. Florence, Italy: IEEE Press, 2015, pp. 688–699. ISBN: 978-1-4799-1934-5.

[33] B. Fredrickson. “The role of positive emotions in positive psychology: The broaden-and-build theory of positive emotions”. In: *American Psychologist* 56.3 (Mar. 2001), pp. 218–226. ISSN: 0003-066X.

[34] M. De Choudhury and S. Counts. “Understanding Affect in the Workplace via Social Media”. In: *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*. CSCW ’13. San Antonio, Texas, USA: ACM, 2013, pp. 303–316. ISBN: 978-1-4503-1331-5.

[35] E. Guzman, D. Azócar, and Y. Li. “Sentiment Analysis of Commit Comments in GitHub: An Empirical Study”. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. Hyderabad, India: ACM, 2014, pp. 352–355. ISBN: 978-1-4503-2863-0.

[36] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson. “Unhappy Developers: Bad for Themselves, Bad for Process, and Bad for Software Product”. In: *CoRR* abs/1701.02952 (2017).

[37] D. Pletea, B. Vasilescu, and A. Serebrenik. “Security and Emotion: Sentiment Analysis of Security Discussions on GitHub”. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. Hyderabad, India: ACM, 2014, pp. 348–351. ISBN: 978-1-4503-2863-0.

[38] D. Garcia, M. S. Zanetti, and F. Schweitzer. “The Role of Emotions in Contributors Activity: A Case Study on the GENTOO Community”. In: *CoRR* abs/1306.3612 (2013).

[39] M. Ortu, G. Destefanis, S. Counsell, S. Swift, M. Marchesi, and R. Tonelli. "How diverse is your team? Investigating gender and nationality diversity in GitHub teams". In: *PeerJ PrePrints* 4 (2016), e2285.

[40] A. Murgia, G. Concas, R. Tonelli, M. Ortu, S. Demeyer, and M. Marchesi. "On the Influence of Maintenance Activity Types on the Issue Resolution Time". In: *Proceedings of the 10th International Conference on Predictive Models in Software Engineering*. PROMISE '14. Turin, Italy: ACM, 2014, pp. 12–21. ISBN: 978-1-4503-2898-2.

[41] M. Ortu, G. Destefanis, M. Kassab, S. Counsell, M. Marchesi, and R. Tonelli. "Would you mind fixing this issue? - An Empirical Analysis of Politeness and Attractiveness in Software Developed Using Agile Boards". In: *Agile Processes, in Software Engineering, and Extreme Programming - 16th International Conference, XP 2015, Helsinki, Finland, May 25-29, 2015, Proceedings*. 2015, pp. 129–140.

[42] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas. "Sentiment in Short Strength Detection Informal Text". In: *J. Am. Soc. Inf. Sci. Technol.* 61.12 (Dec. 2010), pp. 2544–2558. ISSN: 1532-2882.

[43] M. Thelwall, K. Buckley, and G. Paltoglou. "Sentiment Strength Detection for the Social Web". In: *J. Am. Soc. Inf. Sci. Technol.* 63.1 (Jan. 2012), pp. 163–173. ISSN: 1532-2882.

[44] O. Kucuktunc, B. B. Cambazoglu, I. Weber, and H. Ferhatosmanoglu. "A Large-scale Sentiment Analysis for Yahoo! Answers". In: *Proceedings of the Fifth ACM*

International Conference on Web Search and Data Mining. WSDM '12. Seattle, Washington, USA: ACM, 2012, pp. 633–642. ISBN: 978-1-4503-0747-5.

[45] R. Jongeling, S. Datta, and A. Serebrenik. "Choosing Your Weapons: On Sentiment Analysis Tools for Software Engineering Research". In: *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. ICSME '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 531–535. ISBN: 978-1-4673-7532-0.

[46] B. Bazelli, A. Hindle, and E. Stroulia. "On the Personality Traits of StackOverflow Users". In: *Proceedings of the 2013 IEEE International Conference on Software Maintenance*. ICSM '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 460–463. ISBN: 978-0-7695-4981-1.

[47] H. Zhang, L. Gong, and S. Versteeg. "Predicting Bug-fixing Time: An Empirical Study of Commercial Software Projects". In: *Proceedings of the 2013 International Conference on Software Engineering*. ICSE '13. San Francisco, CA, USA: IEEE Press, 2013, pp. 1042–1051. ISBN: 978-1-4673-3076-3.

[48] P. Bhattacharya and I. Neamtiu. "Bug-fix Time Prediction Models: Can We Do Better?" In: *Proceedings of the 8th Working Conference on Mining Software Repositories*. MSR '11. Waikiki, Honolulu, HI, USA: ACM, 2011, pp. 207–210. ISBN: 978-1-4503-0574-7.

[49] S. Kiritchenko, X. Zhu, and S. M. Mohammad. "Sentiment Analysis of Short Informal Texts". In: *J. Artif. Int. Res.* 50.1 (May 2014), pp. 723–762. ISSN: 1076-9757.

[50] R. Narayanan, B. Liu, and A. Choudhary. “Sentiment Analysis of Conditional Sentences”. In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1*. EMNLP ’09. Singapore: Association for Computational Linguistics, 2009, pp. 180–189. ISBN: 978-1-932432-59-6.

[51] A. Agrawal and A. An. “Kea: Sentiment Analysis of Phrases Within Short Texts”. In: *Proceedings of the 8th International Workshop on Semantic Evaluation, SemEval@COLING 2014, Dublin, Ireland, August 23-24, 2014.* 2014, pp. 380–384.

[52] T. Wilson, J. Wiebe, and P. Hoffmann. “Recognizing Contextual Polarity in Phrase-level Sentiment Analysis”. In: *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*. HLT ’05. Vancouver, British Columbia, Canada: Association for Computational Linguistics, 2005, pp. 347–354.

[53] T. Wilson, J. Wiebe, and P. Hoffmann. “Recognizing Contextual Polarity: An Exploration of Features for Phrase-level Sentiment Analysis”. In: *Comput. Linguist.* 35.3 (Sept. 2009), pp. 399–433. ISSN: 0891-2017.

[54] S. Bai, C.-L. Huang, Y.-K. Tan, and B. Ma. “Language Models Learning for Domain-specific Natural Language User Interaction”. In: *Proceedings of the 2009 International Conference on Robotics and Biomimetics*. ROBIO’09. Guilin, China: IEEE Press, 2009, pp. 2480–2485. ISBN: 978-1-4244-4774-9.