View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Institutional Knowledge at Singapore Management University

Singapore Management University

Institutional Knowledge at Singapore Management University

Dissertations and Theses Collection (Open Access)

Dissertations and Theses

7-2019

Making sense of crowd-generated content in domain-specific settings

AGUS SULISTYA
Singapore Management University, aguss.2014@phdis.smu.edu.sg

Follow this and additional works at: https://ink.library.smu.edu.sg/etd_coll

 Part of the Numerical Analysis and Scientific Computing Commons, and the Software Engineering Commons

Citation

SULISTYA, AGUS. Making sense of crowd-generated content in domain-specific settings. (2019). 1-108. Dissertations and Theses Collection (Open Access).
Available at: https://ink.library.smu.edu.sg/etd_coll/228

This PhD Dissertation is brought to you for free and open access by the Dissertations and Theses at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Dissertations and Theses Collection (Open Access) by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

MAKING SENSE OF CROWD-GENERATED CONTENT IN DOMAIN-SPECIFIC SETTINGS

AGUS SULISTYA

SINGAPORE MANAGEMENT UNIVERSITY
2019

Making Sense of Crowd-Generated Content in Domain-Specific Settings

by

Agus Sulistya

Submitted to School of Information Systems in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Information Systems

Dissertation Committee:

David LO (Supervisor / Chair)
Associate Professor of Information Systems
Singapore Management University

LIM Ee-Peng (Co-supervisor)
Professor of Information Systems
Singapore Management University

Jing JIANG
Associate Professor of Information Systems
Singapore Management University

Christoph TREUDE
Senior Lecturer and ARC DECRA Fellow
University of Adelaide

Singapore Management University
2019

Copyright (2019) Agus Sulistya

I hereby declare that this PhD dissertation is my original work and it has been written by me in its entirety.

I have duly acknowledged all the sources of information which have been used in this thesis.

This PhD dissertation has also not been submitted for any degree in any university previously.

Agus Sulistya

4 July 2019

Making Sense of Crowd-Generated Content in Domain-Specific Settings

Agus Sulistya

Abstract

The rapid advances of the Web have changed the ways information is distributed and exchanged among individuals and organizations. Various content from different domains are generated daily and contributed by users' daily activities, such as posting messages in a microblog platform, or collaborating in a question and answer site. To deal with such tremendous volume of user generated content, there is a need for approaches that are able to handle the mass amount of available data and to extract knowledge hidden in the user generated content. This dissertation attempts to make sense of the generated content to help in three concrete tasks.

In the first work performed as part of the dissertation, a machine learning approach was proposed to predict a customer's feedback behavior based on her first feedback tweet. First, a few categories of customers were observed based on their feedback frequency and the sentiment of the feedback. Three main categories were identified: spiteful, one-off, and kind. By using the Twitter API, user profile and content features were extracted. Next, a model was built to predict the category of a customer given his or her first feedback. The experiment results show that the prediction model performs better than a baseline approach in terms of precision, recall, and F-measure.

In the second work, a method was proposed to predict readers' emotion distribution affected by a news article. The approach analyzed affective annotations provided by readers of news articles taken from a non-English online news site. A new corpus was created from the annotated articles. A domain-specific emotion lexicon was constructed along with word embedding features. Finally, a multi-target regression model was built from a set of features extracted from online news arti-

cles. By combining lexicon and word embedding features, the regression model is able to predict the emotion distribution with RMSE scores between 0.067 to 0.232.

For the final work of this dissertation, an approach was proposed to improve the effectiveness of knowledge extraction tasks by performing cross-platform analysis. This approach is based on transfer representation learning and word embedding to leverage information extracted from a source platform which contains rich domain-related content to solve tasks in another platform (considered as target platform) with less domain-related content. We first build a word embedding model as a representation learned from the source platform, and use the model to improve the performance of knowledge extraction tasks in the target platform. We experiment with Software Engineering Stack Exchange and Stack Overflow as source platforms, and two different target platforms, i.e., Twitter and YouTube. Our experiments show that our approach improves performance of existing work for the tasks of finding software-related tweets and filtering informative YouTube comments.

Table of Contents

List of Figures

List of Tables

List of Publications

Main Publications

The work performed as a part of this dissertation has been published in the following publications.

Agus Sulistya, Abhishek Sharma, and David Lo. "Spiteful, One-off, and Kind: Predicting customer feedback behavior on Twitter". In *International Conference on Social Informatics*, pages 368–381. Springer, 2016.

Agus Sulistya, Ferdian Thung, and David Lo. "Inferring spread of readers emotion affected by online news". In *International Conference on Social Informatics*, pages 426–439. Springer, 2017.

Agus Sulistya, Gede Artha Azriadi Prana, Abhishek Sharma, David Lo, and Christoph Treude. "Sieve: Helping developers sift wheat from chaff via cross-platform analysis". In *arXiv preprint arXiv*:1810.13144, 2018.

Other

The following are the publications in which the author has contributed, but which are not related to this dissertation.

Abhishek Sharma, Yuan Tian, **Agus Sulistya**, David Lo, and Aiko Fallas Yamashita. Harnessing Twitter to Support Serendipitous Learning of Developers. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering* (SANER), 2017.

Abhishek Sharma, Ferdian Thung, Pavneet Singh Kochhar, **Agus Sulistya** and David Lo. Cataloging GitHub Repositories. In *21st International Conference on Evaluation and Assessment in Software Engineering Conference (EASE)*, 2017.

Abhishek Sharma, Yuan Tian, **Agus Sulistya**, Dinusha Wijedasa, and David Lo. Recommending Who to Follow in the Software Engineering Twitter Space. In *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 27(4), (2018).

Acknowledgments

Undertaking this PhD after working in industry for several years has been a truly life-changing experience for me and it would not have been possible to do without the support and guidance that I received from many people.

Firstly, I would like to say a very big thank you to my supervisor, Dr. David Lo for his continuous support, his patience and encouragement during my PhD journey. Without his guidance and constant feedback, this PhD would not have been achievable. I would also like to thank Dr. Lim Ee-peng for his advice, insight and discussion to widen my research.

Besides my advisors, I would like to thank the committee members: Dr. Jiang Jing and Dr. Christoph Treude for their insightful comments and suggestion which gave me exposure to a different perspective on approaching research problems.

I gratefully acknowledge the funding received towards my PhD from PT Telekomunikasi Indonesia. I am also grateful to the funding received from Living Analytics Research Centre (LARC) for conferences. I am also very grateful to all those at the SIS General office, to Pei Huan, Chew Hong, Caroline, and others who were always so helpful and provided me with their assistance throughout my study.

I also thank to my fellow lab mates Ferdian, Abhishek, Artha and all other members of Software Analytics Research (SOAR) for always being so supportive of my work. A special thank you also for Budi Arief, Dadang and all my fellow Telkom scholarship friends who were always so helpful in numerous ways.

Last but not least, I would like to say a heartfelt thank you to my mother and my father, Wiharti and Utoyo, for supporting me spiritually throughout my life. For my parent in law, Ulung and Tuti, who always support me. Finally, for my wife, Sari, who has been by my side, and also for my children Alif, Hilmi and Faiz. I love you all !

Chapter 1

Introduction

1.1 Motivation

With a fast growth of the Internet and widespread rise of various online platforms nowadays, it is easy for people and organizations to generate and share content [32]. This enables other people or organizations to benefit from the shared content. There are many successful cases where people collaborate and combine their knowledge, creativity, opinions, etc., to help one another and accomplish various tasks. For example, social media such as Twitter provides a convenient way for customers to provide their feedback to companies. These feedback tweets enable companies to improve their products or services. Another example is Stack Overflow, a popular question and answer site, that allows anybody facing a programming issue to post a question on the forum. This platform allows knowledge sharing permitting other users on the forum to post their responses as answers to the posted questions.

Engaging actively with such tremendous volume of user generated content has now become a daily challenge for both organizations and individuals. There is a need for automated approaches that can help one to better manage the mass amount of available data and to extract knowledge hidden in the user generated content [69]. Making sense of this generated content is not a trivial task. There exist several approaches, which make use of user-generated content for various applications such

as in [66, 63, 92, 78, 51, 44]. However, due to tremendous amount and diversity of user generated content and the possible applications that can benefit from them, there are many research opportunities that are still left unexplored and problems that are still open. This dissertation aims to help both organizations and individuals making sense of data from various settings to better perform certain concrete tasks. While this data can be easily collected, there is a need to digest and get insight from the collected data. In this dissertation, an approach based on 5W+1H (*Who, What, Why, Where, When* and *How*) is used to characterize different case studies that are relevant to making sense of crowd generated content. Specifically, the use cases explored should answer each of dimension in 5W+1H framework as follows:

Table 1.1: 5W+1H framework on mining crowd generated content

Dimension	**Description**
Who	Individuals or organizations that get benefit of the research.
What	Research ideas
Why	Objective of the research.
Where	Domains/Platforms used
When	When to process the content
How	Approaches used .

Based on the framework, we explored three different use cases related to making sense of crowd generated content, that cover different aspects/dimensions of 5W+1H framework as described in table 1.2.

1.2 Contribution Summary

In this dissertation, we proposed three machine-learning-based approaches to make sense of user generated content from various online sources to help in concrete tasks. In the first work, a machine learning approach was proposed to predict customer complaint behavior on Twitter based on his or her first tweet. In the second work, an approach was proposed to infer spread of readers' emotion towards certain news. In the final work, an approach was proposed which can help individuals

Table 1.2: Use cases based on 5w+1H framework

Dimension	**Case 1**	**Case 2**	**Case 3**
Who	Organization	Organization	Individuals
What	Profiling customers based on social media content	Inferring users attribute based on textual content	Finding relevant content to a specific domain
Why	The company can benefit from the model to improve customer service strategies to deal with different categories of customers	The organization (i.e., publishers) will have insight on expected public response to a particular textual content (i.e., news article)	Considering the huge volume of data generated today, much of such data is irrelevant to the target domain or task
Where	Short text (microblog/Twitter)	Long text (online news article)	Short text (Twitter, Youtube comment) and long text (Stack Exchange posts)
When	Immediate (after the user posts the content)	Batch processing	Batch processing
How	Extract data from Twitter to identify different types of customer feedback behavior using a machine learning approach	analyze text and build lexicon to estimate readers reaction affected by the textual content (i.e., online news article)	Leverage data extracted from a platform that contains rich domain-related content (source platform) to solve tasks in another platform with less domain-related content (target platform)

to find domain-specific content. We particularly focus on the software engineering domain. We give a brief summary of each of the completed works below.

Predicting Customer Complaint Behavior on Twitter

In this work, a method was proposed to predict customer categories (i.e., kind, one-off, and spiteful) given a customer's profile and first feedback tweet. The approach extracts a set of profile features and content features and uses these to build a prediction model using a classification algorithm. To demonstrate the accuracy of the approach, experiments were conducted to evaluate the approach using real

dataset of labeled tweets mentioning an official account of a large telecommunication company in Indonesia. The effectiveness of the approach was measured by using common evaluation metrics in data mining research (i.e., precision, recall, and F-measure), and was compared to a weighted random picker baseline. The results showed that three variants of the approach that use different underlying classification algorithms can substantially outperform the baseline.

Inferring Spread of Readers' Emotion affected by Online News

In this work, an approach was proposed to use emotion lexicon and word embedding in order to predict readers' emotion scores distribution towards an online news article. A new corpus was built containing around 1.5k Indonesian news articles taken from a popular online news site, namely *detik.com*, along with affective annotations provided by readers of those articles. After conducting experiments, the results show that, by using combined features of domain-specific emotion lexicon together with word embeddings vectors, the proposed approach was able to predict the distribution of readers' emotion scores with a Root Mean Squared Error (RMSE) score ranging from 0.067 to 0.232.

Helping Developers Sift Wheat from Chaff via Cross-Platform Analysis

A challenge in mining crowd data is to find content relevant to a specific target domain or task. Solving this challenge is important considering the huge volume of data generated today. Much of such data is irrelevant to the target domain or task. Needing end users working on a particular domain or performing a particular task to read or browse through such sheer amount of content is likely to result in information overload. Therefore, we propose to design an approach to leverage data extracted from a platform that contains rich domain-related content (source platform) to solve tasks in another platform with less domain-related content (target platform). In this work, we focus on one specific domain, namely software engineering. Software developers have benefited from various sources of knowledge such as forums, question-and-answer sites, and social media platforms to help them in vari-

ous tasks. However, these information channels contain a diverse set of information covering a wide range of topics beyond software engineering. Extracting software-related knowledge from different platforms is non trivial. In this work, an approach was proposed that utilizes content from a rich software-development-specific platform based on transfer representation learning, to help automated knowledge extraction tasks in other less software-development-specific platforms. Two platforms are studied (i.e., Software Engineering Stack Exchange and Stack Overflow), as the rich domain-related platforms. Word embedding models are built based on the dataset collected from the two platforms. These models are used to solve ranking and classification problems in two different target platforms. Experiments were conducted in two different use cases: finding tweets relevant to software development on Twitter [82], and classifying informative comments for software engineering video tutorials on YouTube [70]. The results show the effectiveness of the proposed cross-platform analysis approach which achieves performance improvements of up to 28% and 10.3% for the first and second use case respectively.

1.3 Structure of this Dissertation

The remainder of this dissertation is organized as follows: Chapter 2 is a literature review which examines related research to the use cases discussed in this dissertation. Chapter 3 describes a study which investigates an approach to predict customer complaint behavior on Twitter. Chapter 4 studies a method for inferring the spread of readers' emotion distribution towards online news article. Chapter 5 present an approach to help software developers to identify software-engineering-relevant information. Finally, Chapter 6 summarizes the contributions of this dissertation and presents some future direction.

Chapter 2

Literature Review

In this chapter, we discuss the work in literature related to this dissertation. In the first part, we briefly describe studies related to making sense of crowd generated content. In the second part, we describe studies related to various domain-specific tasks.

2.1 Making Sense of Crowd Generated Content

Due to vast amount of user generated content generated daily, there is a need for automated approaches that can help both organization and individuals to make use of the generated content [69]. Many organizations have explored different data sources to mine user generated content for different purposes. For example, social media can help an organization to better understand their customers through social monitoring [14], develop systems to prioritize relevant posts [3], or examining brand awareness [60]. Other source of data such as email conversation also has been explored in [63].

Crowd generated content can also benefit individuals. As an example, knowledge seekers rely on many sources of knowledge to help them stay up-to-date on the latest technologiess [85], and to accomplish their development tasks. Online resources such as web search engines, public documentation, user forums are ex-

amples of crowd generated content that are useful for knowledge seekers [54]. Automated solutions can be designed to help these individuals benefit from the aforementioned user generated content.

2.2 Domain-Specific Tasks

In this section, we describe work related to three domain-specific tasks described in this dissertation: predicting user's attributes based on social media data, inferring user's emotion towards textual content, and finding informative content related to software engineering domain.

2.2.1 Social Monitoring and User Attribute Prediction

It is important for companies to continuously monitor voices of their customers in social media, which is often referred to as *social monitoring*. A number of works have designed social monitoring systems. For example, Bhatia et al. developed a system that automatically monitors social network platforms, analyzes data from the platforms, and triggers events that lead to corrective actions [14]. Ajmera et al. analyzed posts and messages in social network platforms based on its to an enterprise [3]. They built a system that mines conversations on social platforms to identify and prioritize those relevant posts and messages. The system developed in their work aims to empower an agent in an enterprise to monitor, track and respond to customer communication. Einwiller et al. examined the complaining behavior and complaint management on Social Media, focusing primarily on how companies manage the complaints [33]. Millard et al. conducted a study that investigates what customers are actually engaging with on social media (Twitter, Facebook and forums) with respect to brands. They found that customers engage with brands not only to complain but also to complement [60]. Chen et al. introduced a brand-specific intelligent filters on Twitter which is called *CrowdE* using a common crowd-enabled process [28]. They also evaluated the system, and

found that CrowdE's intelligent filters improved task performance and were generally preferred by users in comparison to keyword-based filters. Chapter 3 of this dissertation highlights another framework that has been implemented and currently used by a large telecommunication company using customized commercial tools. This proposed framework extends the company's social listening framework with a capability to predict customer categories.

User attribute prediction have been studied in the past. For example, Pennacchiotti et al. presented an approach to infer the values of a Twitter user's hidden attributes by analyzing observable information such as the user behavior, network structure and the linguistic content of the user's Twitter feed [66]. They also found that content features taken from linguistic content of user messages are in general highly valuable across different tasks: political affiliation detection, ethnicity identification and detecting affinity for a particular business. Another work by On et al. studied interactions in an email network [63]. They investigate user engagingness and user responsiveness as two interaction behaviors on how users email one another. They developed four types of models to quantify this two behavior. They found that engagingness and responsiveness behavior features are useful in the task of predicting the email reply order. The work presented in Chapter 3 in this dissertation differs from previously mentioned works since we use different sets of features taken from user profile and feedback content, extracted from Twitter. We also focus on a different problem, namely the prediction of customer category based on feedback tweets. For this type of customer feedback, typically we would see more negative feedback rather than positive, which raised an imbalanced dataset problem.

2.2.2 Inferring Users' Reaction to Textual Content

Predicting users' emotions towards particular textual content has been studied in the past. Most existing works focus on building emotion lexicons or devising prediction algorithms. In this section, we briefly summarize research efforts conducted on

these two fronts.

SemEval-2007 [92] is considered the first research effort in predicting readers' emotions by analyzing news article headlines. It used news headlines as its data source. They found that the task of emotion annotation is not trivial, since there are words that act only as an indirect reference to emotions depending on the context. Lin et al. [52] proposed the use of a regression model to estimate readers' emotion towards news article in Chinese. They use Chinese character bi-gram, Chinese words, and news metadata as features, and use Support Vector Regression (SVR) as the regression model. They found that the regression method is more effective at identifying the most popular emotion. Lei et al. [51] proposed an approach that performs document selection, Part-Of-Speech (POS) tagging, and a social emotion lexicon generation system to build a social emotion detection system for online news. Based on experiment conducted, the system performs better with the words and POS combination as features. Hsieh et al. [44] proposed a document modeling method that utilizes embedding of emotion keywords to perform readers' emotion classification. They used two Chinese corpora to build word embeddings, and find a set of keywords for each emotion category using log likelihood ratio (LLR). They found that their approach can achieve best macro average accuracy as compared to several baselines. Different with related work described above, in the work described in Chapter 4 of this dissertation, we focus on inferring spread of readers' emotion distribution. We also explored various sets of features by combining word vectors and emotion lexicon generated from different parts of news articles (headlines, contents and both).

Several methods have been proposed to build emotion lexicon, either manually or automatically. A popular resource for emotion lexicon is WordNetAffect [93], which contains manually assigned affective labels (anger, joy, etc.) to WordNet synsets (i.e., set of synonyms). AffectNet, which is a part of the SenticNet project [19], contains around 10,000 words taken from ConceptNet and aligned with WordNetAffect. AffectNet maps common sense knowledge to af-

fective knowledge (i.e.,WordNetAffect affective labels). Another popular resource is NRC-EmoLex [61], which consists of 10,000 lemmas (i.e., a base word form that is indexed in the lexicon) annotated with an intensity label for each emotion. These data are manually labeled by multiple annotators. Another approach for building lexicons is through automated means. Staiano and Guerini presented DepecheMood [88], an emotion lexicon that is built by harvesting crowd-sourced affective annotation from a social news network. Rao et al. [78] proposed an algorithm and pruning strategies to automatically build a word-level emotion dictionary, in which each word is associated with a distribution of social emotions. They also proposed to use topic modeling for constructing a topic-level emotion dictionary, in which each topic is associated with a distribution of social emotions. A work by Bandhakavi et al. [10] compared General Purpose Emotion Lexicons (GPELs) and Domain-Specific Emotion Lexicons (DSELs) for emotion detection from text. They confirmed the superiority of DSELs for emotion detection. In the domain of non-English lexicon, Abdaoui et al. [1] built a French lexicon by performing semi-automatic translation and synonym expansion for words in NRC-EmoLex. Nguyen et al. [62] proposed an approach to mine public opinions from Vietnamese text using a domain-specific sentiment dictionary that was built incrementally. In the work described in Chapter 4 of this dissertation, we extend Staiano and Guerini work [88]. Specifically, we automatically build emotion lexicon using affective-annotated news articles in an under-resourced language (Indonesian) from a popular online news platform.

2.2.3 Finding Software-Relevant Content for Software Developers

In the past few years, there has been a substantial amount of work which has analyzed tools or channels used by software developers. Storey et al. found that software developers use many communication tools and channels in their software

development work [90, 91]. In the following paragraphs, we discuss work related to identification of relevant content in social media channels (Twitter and YouTube) for software developers, as these are the domains we have considered in the work described in Chapter 5.

Several studies have investigated software related content on Twitter. For example, Bougie et al. did an exploratory study on understanding how Twitter is used in software engineering [17]. Tian et al. found that Twitter is used by software developers for coordination of efforts, sharing of knowledge, etc. [95, 94]. Sharma et al. have explored the categories of software engineering related tweets and events on Twitter [83]. Methods to identify software-relevant tweets and informative links have been proposed by Prasetyo et al. [75] and Sharma et al. [82, 84]. Guzman et al. analyzed tweets on Twitter which talked about software applications and companies, and demonstrated that machine learning techniques have the capacity to identify valuable information for companies and developers of software applications [37, 38]. They also proposed a technique to mine tweets for software requirements and evolution [39]. There has been other work also on mining Twitter feeds for gathering user requirements such as by Williams et al. [101] Mezouar et al. found that tweets generated by users can help in early detection of bugs in software systems, and can help developers know about a bug which may be affecting a large user base [34]. Our work presented in Chapter 5 extends previous work by Sharma et al. [82] to rank software-relevant tweets. Different from their approach, we use word embedding to capture relations between software-related terms. In addition, we sample selected Stack Overflow titles, use them as seed sentences, and calculate similarity scores between the samples and the tweets.

Software development videos on YouTube in recent years have been studied as a repository from which software-related knowledge can be extracted. MacLeod et al. studied the developer's usage of videos (on YouTube) to document software knowledge [56, 55]. They found that the main motivating factors for sharing videos by developers are building an online identity, to give back to the community, to

promote themselvess, etc. Poché et al. proposed an approach to identify informative user comments on coding video tutorials on YouTube [70]. We extend the work by Poché et al. by leveraging word embedding learned from Stack Exchange and Stack Overflow, as described in Chapter 5.

Chapter 3

Predicting Customer Feedback Behavior on Twitter

3.1 Introduction

The use of social media in the customer relationship context has gained popularity nowadays. A report by VB Insight [1] reveals that modern consumers complain about brands 879 million times a year on Facebook, Twitter, and other social media portals. About 10% of those consumers make a complaint on social media every day. With this extensive use of social media by customers, opportunities arise for companies to engage with their customers and be aware of the issues that they face. For example, a customer can complain on social media after experiencing a failure of service; this complaint notifies the company and prompts it to take necessary actions to prevent further damage to the company's reputation and customer base. Therefore, it is important for the company to continuously monitor the voices of their customers, which refer to as an activity called *social listening and monitoring*.

It would be interesting to be able to predict different types of customer feedback behavior. Such prediction can help a company to formulate a suitable strategy to manage and improve customer satisfaction and retention. For example, some users

[1] http://venturebeat.com/2014/12/12/social-media-we-complain-879-million

may complain many times to a company's Twitter account if the users are not given sufficient attention in a short period of time, others may complain only once, and yet others may express their thanks after a good service has been rendered by a company. From the company's side, multiple complaints are considered as something that should be avoided, since it can affect the company's reputation. Having an ability to predict this type of customer would allow the company to take preventive action before the user spreads negative opinion about the company in social media. A company may also want to provide good reasons for the third category of customers to publicize good service and improve the company's reputation.

In this study, we try to address the aforementioned prediction problem by employing a two-stage machine learning algorithms. In the first stage, our approach clusters social media users into several categories based on their feedback frequency and sentiment polarity. We identify three categories of users: spiteful (i.e., the user complains many times in social media), one-off (i.e., the user only provides negative feedback once), and kind (i.e., the user provides positive feedback). In the second stage, our approach builds a prediction model that can assign a user into one of the three categories based on his/her first feedback. We experiment with different supervised machine learning algorithms (i.e., Naive Bayes, Logistic Regression, and Random Forest), to build an automated prediction model.

As a case study, we use internal data from a state-owned telecommunication company in Indonesia to evaluate the effectiveness of our proposed approach. The company named Telkom extensively uses social media such as Facebook and Twitter, to interact with its customers. To facilitate social listening, the company has set up a dedicated unit to actively monitor customer feedback. Our work extends the current social listening platform that the company has by adding some predictive capabilities. Under 10-fold cross validation, our experiments show that our proposed approach can predict customer feedback behavior categories with a weighted precision, recall, and F-measure of up to 0.797 (Random Forest), 0.881 (Naive Bayes), and 0.800 (Random Forest) respectively. Our approach outperforms a baseline that

randomly assign categories to customers based on the distribution of customer feedback behavior categories in the training data.

Extracting knowledge from microblogs has been one of the active research areas. We believe that this study would be important towards the development of techniques that make use of social media data to improve product and service quality. Specifically, our contributions are as follows:

1. We propose a new problem of predicting different types of customer feedback behavior on Twitter.

2. We use a clustering algorithm to identify different types of customer feedback behavior.

3. We propose a set of features, i.e. content features and profile features, that can be used to predict customer feedback behavior by leveraging a supervised machine learning algorithm to build a prediction model.

4. We have evaluated our proposed approaches on a dataset containing 11,809 tweets. Our proposed approaches can achieve reasonable precision, recall and F-measure which are higher than those of a baseline approach.

The structure of the remainder of this chapter is as follows. In Section 3.2 we describe social listening activities in a company used as our case study and data analysis techniques that we leverage for this work. We describe how we cluster customers to create several categories in Section 3.3. In Section 3.4, we explain our approach which extracts features from customer Twitter accounts and their corresponding tweets and uses them to build a prediction model to predict customer categories based on their first feedback tweet. We describe our experiments which evaluate the prediction accuracy of our approach in Section 3.5. We finally conclude and mention future work in Section 3.6.

3.2 Background

3.2.1 Social Listening at Telkom

In this chapter, we experiment with a dataset collected and annotated by a state-owned telecommunication service provider in Indonesia, namely Telkom[2]. The company serves tens of millions of customers throughout Indonesia, offering a wide range of products including broadband internet connections, cable TV, and land line telephone connections.

Telkom has set up a system that actively monitors what customers say on social media, and handles each issue raised by forwarding the problem to a back-room unit. To monitor customer voices, the company uses tools provided by Brand24[3] and BrandFibres[4]. The first tool is used to crawl any content containing keywords related to the company's product from different platforms, including Facebook, Twitter, blog posts, and news media. These crawled records are then filtered by removing irrelevant posts. The filtering process requires manual work performed by several social media analysts. The analysts use a second tool called BrandFibres dashboard. Using this tool, they evaluate each post, and then assign a sentiment score to each post. They give scores ranging from "+5" (very positive feedback) to "-5" (very negative feedback). The analysts also assign a post into one of the 8 different categories shown in Table 3.1. Note that a tweet can be assigned to more than one category, and an analyst will assign a sentiment score for every category that applies to a tweet.

Figure 3.1 shows an example of a customer complaint on Twitter. In the figure, the tweet mentions a company's account (@telkomcare). The tweet also mentions other users (@detikcom and @telkompromo). The first one is an online news media account and the latter is the company's other account that focuses on disseminating the company's promotional events and deals.

[2] http://www.telkom.co.id/en/tentang-telkom
[3] http://www.brand24.com/
[4] http://www.brandfibres.org/

Figure 3.1: A sample of a tweet posted by a customer to Telkom's customer care channel (in Indonesian). The English translation of the tweet is *"When will our home is installed with Indihome? It's been 3 months since we requested it"*.

This study analyzes data consisting of tweets collected and annotated by Telkom for a 3 month period from June-August 2015. In total, there are 12,634 posts. We consider only the posts that have been collected from Twitter, which results in about 11,809 posts (or tweets) constituting about 93.4% of the total posts.These tweets are those that mention the official company's customer care account on Twitter, namely @telkomcare. For the tweets in our dataset, we extract distinct twitter users who posted them, resulting in 6,031 distinct users. We use this set of users as the input to our clustering and prediction tasks described in the next two sections. The data provided by Telkom did not include the profiles of these 6,031 Twitter users. To get these profiles, we call the standard Twitter API using Tweepy[5] Python module.

3.2.2 Handling Imbalanced data

The imbalanced data problem typically refers to a classification problem where the classes are not represented equally. For customer feedback, typically we would see more negative feedback rather than positive. One way to deal with imbalanced data is by using sampling methods, which modifiy the distribution of the original training samples to obtain a relatively balanced data. There are two types of sampling methods: oversampling and undersampling [45]. Oversampling is conducted by adding more samples to the minority class, while undersampling is done by creating a subset of the majority class. One popular oversampling algorithm to handle imbalanced data is SMOTE (Synthetic Minority Over-sampling Technique) [22]. This oversam-

[5] `http://www.tweepy.org/`

Table 3.1: Customer Feedback Categories

Category	**Description**
Quality Evaluation (QUE)	Tweets related to general quality of a product or service (for example, slow or unstable internet connection).
Offer Evaluation (OFE)	Tweets providing feedback to a product offering (such as an ongoing promotion of a certain product)
Activity Disturbance (ACD)	Tweets reporting specific disturbance in a user's activity while using a product (for example, trouble when browsing or downloading).
Invoice Related (INV)	Tweets reporting issues related to product or service invoicing (such as reports of incorrect billing).
Customer Service Quality (CSE)	Tweets reporting issues related to quality of customer service (such as quality of customer service agents, and how the company handles current problems experienced by the customer)
Actions (ACT)	Tweets related to actions taken by the customer (such as comparing product provided by the company with other competitors).
Social Media (SMI)	Tweets related to social media interactions between company and customers.
Others (CIS)	Tweets about other issues related to the company and subsidiaries.

pling algorithm creates synthetic samples from the minority class instead of creating copies. SMOTE works by finding the k nearest neighbors of each sample in the minority class. Next, artificial samples are then generated along the line of some or all of the k nearest neighbors, depending on the amount of oversampling required.

3.3 Clustering Customers

In the first stage of our work, we cluster customers in our dataset (i.e., the 6,031 users described in Section 3.2.1) into several categories based on their feedback frequency and the sentiment polarity of these feedback. Figure 3.2 shows our overall

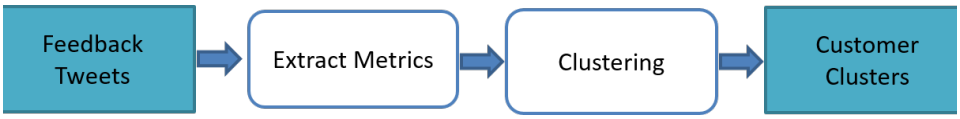

Figure 3.2: Our approach for clustering customers

approach to cluster customers.

We represent each customer as a set of metrics: NumOfFeedback, NumOfPosFeedback and NumOfNegFeedback. These metrics are listed and defined in Table 3.2. Next, based on this representation, we cluster the users together. To cluster the users, we use Expectation-Maximization (E-M) algorithm. E-M algorithm assigns a probability distribution to each instance which indicates the probability of it belonging to each of the clusters. A previous study conducted by Meilă and Heckerman [57] has found that the E-M algorithm often performs better than other clustering methods such as k-means and model-based hierarchical agglomerative clustering.

We use the implementations of E-M Algorithm in Weka [40]. We do not initiate the number of cluster and let the E-M algorithm decide the best number of clusters. All parameters are set to Weka default setting.

Table 3.2: Metrics used for clustering users

Features	**Description**
NumOfFeedback	Number of feedback tweets generated by a user
NumOfPosFeedback	Number of feedback tweets that are of positive sentiment polarity.
NumOfNegFeedback	Number of feedback tweets that are of negative sentiment polarity.

Table 3.3 shows the results of the E-M clustering algorithm. We verify the result by manually investigating the properties of each cluster. Based on this manual investigation, we conclude general properties for each group as shown in the fourth column of the table.

Note that there are similarities among these clusters. Cluster 4 represents the

Table 3.3: Clusters of users based on their tweets mentioning the company

Cluster	Count	Percentage	Observed Properties
0	1235	20.48 %	post one or two times, with at least one positive feedback
1	152	2.52 %	post more than 2 tweets, with more than two possitive feedback
2	481	7.98 %	post at least 4 tweets, with majority of negative feedback
3	82	1.36 %	post at least 9 tweets, with majority of negative feedback
4	2837	47.04 %	post only one tweet with negative feedback
5	1244	20.63 %	post 2 or 3 times with majority of negative feedback

majority of customers who only provide one negative feedback instance, without posting further tweets. Clusters 2, 3 and 5 correspond to customers who post more than one tweet with negative sentiment. These customers are typically the group of customers that may damage a company's reputation if they are not managed well. The other two groups (clusters 0 and 1) are groups of customers that post at least one positive feedback instance such as thanking the company for its good service. These customers can improve the company's reputation. Based on this observation, we decide to group the clusters further into three groups based on how the customers complain or behave. These new groups are shown in Table 3.4. We will use these three groups as class labels for the second stage of our approach that predicts customer feedback behavior.

Table 3.4: Three main categories of customers

Class	Cluster	Percentage
Kind	0,1	23.0%
One-Off	4	47.0%
Spiteful	2,3,5	30.0%

3.4 Predicting Customer Categories

In the second stage, our approach builds a prediction model that can assign a customer into one of the three categories based on their first feedback tweet. With our prediction model, a company would be able to know the category of a customer early and take necessary actions. Our approach first extracts a number of features that characterize a customer and his/her first feedback tweet. Features of customers belonging to the three categories are then used to train a prediction model that can differentiate each category. The following subsections explain features used and our approach to build the prediction model.

3.4.1 Feature Engineering

We use two types of features: profile features (i.e., features that we extract from a customer's Twitter profile) and content features (i.e., features that we extract from a customer's first feedback tweet).

Profile Features

Twitter provides several pieces of information about its users which include the user's number of followers, number of followees, etc. We consider five profile features to infer customer categories. These five features are described in detail below.

- **TweetCount** This feature is the number of tweets or re-tweets generated by a user. This metric represents a user's level of activity on Twitter.
- **FollowerCount** This feature is the number of followers that a user has. If A follows B on Twitter, all of B's tweets would be propagated to A. This feature is a basic measure of a user's popularity on Twitter.
- **FolloweeCount** This feature is the number of people a user follows. It represents the user's level of interest in others and correlates to the number of

tweets that the user would receive daily.

- **FavCount** Twitter users may express their liking of a tweet by marking the tweet as a favorite. This feature is the number of the tweets that a particular user has favorited. A higher value of this metric indicates that this user often gives positive feedback to others and may indicate his/her level of agreeableness.

- **ListCount** A Twitter user can create lists of other Twitter users whom he/she follow. Each of these lists typically contains related Twitter users who belong to a particular topic or interest (e.g., a list of friends, co-workers, celebrities, athletes, etc.). This feature is the number of lists that a user creates. We use this feature to capture another aspect of a user's level of activity on Twitter.

Content Features

Content features characterize a tweet based on its content. In our case, we use sentiment polarity scores assigned to each tweet to represent content features. The tweets in the dataset that we use have been annotated by Telkom's social media analyst. The analysts gave sentiment polarity scores for each tweet. They also categorized the tweet into eight different customer feedback categories. We use a total of eight customer feedback categories as content features. Each of content feature corresponds to one of the eight possible categories of tweets as listed in Table 3.1. The value of each of these eight features is the sentiment polarity score that is assigned manually by Telkom's social media analysts. Figure 3.3 shows sample of tweets with its corresponding values of content features.

TwitID	QUE	OFE	ACD	INV	CSE	ACT	SMI	CIS
http://twitter.com/fakhrululum48/status/638501038488727556	0	0	0	0	-2	0	0	0
http://twitter.com/aridwiatr/status/638499561602838528	0	0	0	0	0	0	-1	0
http://twitter.com/abroanaq/status/638497576438775808	0	0	0	0	0	0	1	0

Figure 3.3: Sample tweets with its corresponding content features

3.4.2 Methodology

In general, our methodology contains two phases: a model building phase and a prediction phase, as shown in Figure 3.4. In the model building phase, our goal is to build a prediction model based on a training set of customers along with their profiles, first feedback tweets and category labels. In the prediction phase, this model is used to predict the category of a new customer based on his/her profile and first feedback tweet.

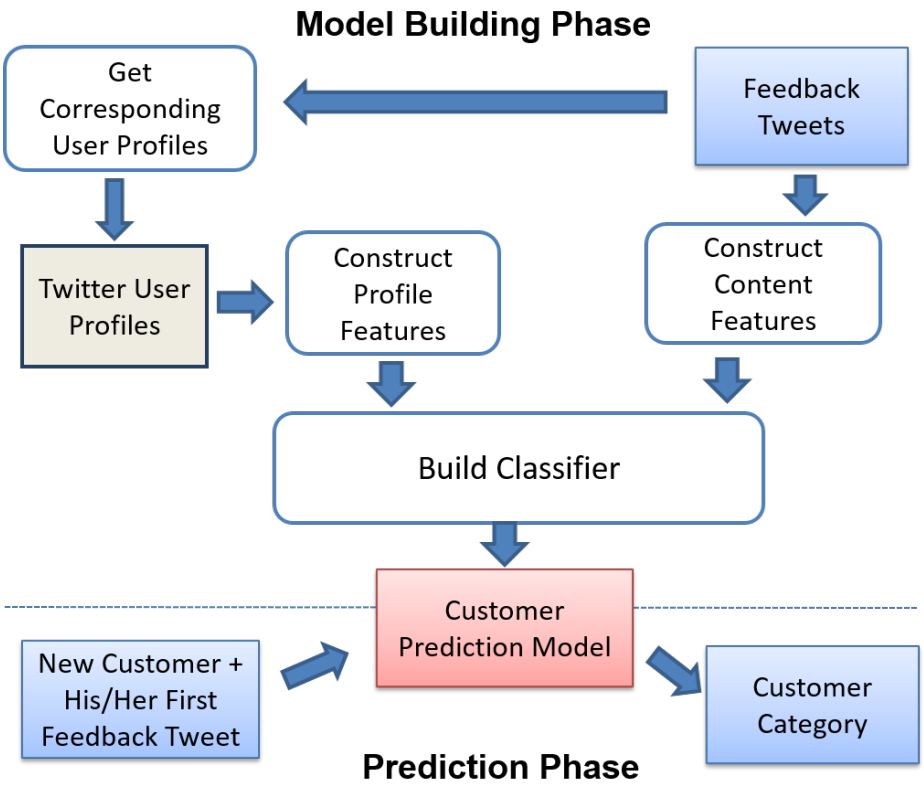

Figure 3.4: Our approach for predicting customer categories

In the model building phase, we first extract profile and content features from customers in the training data. Next, for the profile features (i.e., *TweetCount, FollowerCount, FolloweeCount, FavCount, ListCount*), since the variation of the feature values is high, we normalize them to have values between 0 and 1. However, we do not normalize the content features, since we want to preserve the actual sentiment polarity scores and the variation of these scores is not high. After the features are extracted, we apply SMOTE (described in Section 3.2.2) to handle imbalanced

data. Finally, we use a classification algorithm to build a prediction model.

We explore three classification algorithms, namely Logistic Regression, Naive Bayes and Random Forest. These algorithms are widely used in data mining research such as in [6, 21, 98].

In the prediction phase, we extract values of profile and content features for a new customer whose category is to be inferred. These feature values are extracted from the new customer's Twitter profile and his/her first feedback tweet. Next, we apply the prediction model that we have learned in the model building phase on the new customer's feature values. This model will output a prediction, which is one of the three categories listed in Table 3.4.

3.5 Experiments and Results

3.5.1 Dataset and Experiment Setting

There are 6,031 distinct Twitter users in our Telkom dataset. However, we could not collect profile features for a number of them. This is the case since not all Twitter accounts are public. Among the 6,031 users, we are able to get 5,813 user profiles. This represents 96.39% of distinct users in our dataset. For each of these users, we identify his/her tweet that will be used as input to the prediction task. We consider the earliest feedback tweet that is posted during the observation period (i.e., June-August 2015) as such tweet.

We use the implementations of Logistic Regression, Naive Bayes and Random Forest in Weka [40]. We apply SMOTE filter for all of the three variants. All parameters are set into Weka default settings. We also perform 10-fold cross validation to investigate the effectiveness of our approach.

As a baseline, we use an approach which we refer to as *WeightedRandomPicker*. This baseline picks one of the three categories randomly based on the percentage of customers of each category in our dataset. For example, given a new customer,

WeightedRandomPicker predicts that the customer belongs to Class 1 (Kind) with a probability of 0.23, Class 2 (One-Off) with a probability of 0.47 or Class 3 (Spiteful) with a probability of 0.30.

3.5.2 Evaluation Metrics

As yardsticks to measure the effectiveness of our approach and the baseline, we use precision, recall, and F-measure. These metrics are common metrics that have been widely used in many past studies such as [77, 46, 31, 103, 96].

These metrics are calculated based on four possible outcomes of a Twitter user in an evaluation set: True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). For example, in case of predicting a spiteful customer, TP is when a spiteful customer is correctly predicted as such; FP is when a non spiteful customer is wrongly predicted as a spiteful customer; FN is when a spiteful customer is wrongly predicted as a non spiteful customer; TN is when a non-spiteful customer is correctly predicted as non-spiteful customer.

Since we deal with multi-class classification, we also calculate weighted precision, weighted recall and weighted F-measure. We use the following formula to calculate weighted F-measure (similarly for weighted precision and recall):

$$WeightedFM = \frac{\sum_c (FM(x) \times x)}{n} \tag{3.1}$$

In the above equation, c is total number of classes (in our case: 3), FM(x) is the F-measure score for class x, x is total number of data instances that belong to a particular class, and n is the total number of instances in the dataset.

3.5.3 Research Questions and Results

RQ1: How well does our approach perform in predicting different categories of customers?

Table 3.5: Effectiveness of various variants of our approach which uses different underlying classification algorithms to predict customer categories. C1: Kind customers, C2: One-Off customers, C3: Spiteful customers.

Algorithm	**Metrics**	**C1(Kind)**	**C2(One-Off)**	**C3(Spiteful)**	**Weighted**
Logistic Regression	precision	0.969	0.732	0.574	0.738
Logistic Regression	recall	0.769	0.971	0.372	0.744
Logistic Regression	F-measure	0.858	0.835	0.451	0.724
Random Forest	precision	0.973	0.754	0.697	0.787
Random Forest	recall	0.819	0.926	0.667	0.824
Random Forest	F-measure	0.890	0.832	0.682	0.800
Naive Bayes	precision	0.881	0.733	0.525	0.704
Naive Bayes	recall	0.767	0.925	0.897	0.881
Naive Bayes	F-measure	0.820	0.818	0.663	0.772
Baseline	precision	0.452	0.474	0.296	0.415
Baseline	recall	0.212	0.474	0.366	0.382
Baseline	F-measure	0.288	0.472	0.322	0.385

Approach: In this research question, we investigate three variants of our approach which uses three supervised classification algorithms (Logistic Regression, Naive Bayes and Random-forest), and compare its performance (measured in terms of precision, recall, and F-measure) with that of the *WeightedRandomPicker* baseline.

Results: Table 4.4 shows the results of our experiments. From the table, we can see that the three variants of our approach consistently outperform the *WeightedRandomPicker* baseline. Among the three classes, determining spiteful customers (C3) is the hardest problem. In this case, Logistic Regression performs the worst when compared to the other two supervised algorithms. But still, it outperforms the baseline by 13% in terms of weighted F-measure. Meanwhile, determining kind customers (C1) is the easiest task, and all three variants of our approach outperform the baseline by more than 53% in terms of F-measure.

RQ2: How effective is the oversampling strategy to improve classification accuracy?

Approach: We apply SMOTE to handle the imbalanced class problem. In this research question, we compare results obtained by our approach when SMOTE is used and when it is not used.

Results: Table 4.5 shows results that our approach achieves when SMOTE is turned off and on. We can note that by applying SMOTE the effectiveness of our approach (measured in terms of weighted F-measure) can be improved by 33-44%. This result gives evidence that handling imbalanced data by using minority-class oversampling improves the accuracy of the constructed prediction model.

Table 3.6: Weighted F-Measure of our approach with and without SMOTE

Algorithm	**No SMOTE**	**With SMOTE**	**Improvement**
Logistic Regression	0.542	0.724	33.58 %
Random Forest	0.591	0.800	35.33 %
Naive Bayes	0.534	0.772	44.49 %

3.5.4 Results

Our experiments show that among the three variants of supervised classification algorithm, Random Forest performs the best with F-Measure of 0.890, 0.832 and 0.682 for predicting kind, one-off, and spiteful customers respectively. This finding is consistent with a previous study by Caruana et.al.[21] which observed that random forest tended to perform well across different settings. Even for the variant that uses the most basic machine learning approach among the three (i.e., Naive Bayes), the prediction performance is 30% better than that of *WeightedRandomPicker*. These results highlight a promising potential of applying machine learning techniques to identify different categories of customers based on their first feedback tweets.

Our prediction model relies on sentiment polarity of customer feedback. To ensure the correctness of sentiment polarity of user's tweet, we decided to use labeled/annotated data that Telkom provides and none of the authors are involved in the labeling/annotation process.

A limitation of our study is the sample used in the case study. We have evaluated the effectiveness of our approach to infer customer categories from tweets that mention one company in Indonesia. In the future, we plan to address this limitation

by considering a larger set of tweets collected over a longer period of time. We also plan to experiment with other companies situated in different countries.

3.6 Chapter Conclusion

In this chapter, we propose a method to predict customer categories (i.e., kind, one-off, and spiteful) given a customer's profile and first feedback tweet. Our approach extracts a set of profile features and content features and uses these to build a prediction model using a classification algorithm. To demonstrate the accuracy of our approach, we evaluate our approach using a real dataset of labeled tweets mentioning an official account of a large telecommunication company in Indonesia. We evaluate our approach by using common evaluation metrics in data mining research (i.e., precision, recall, and F-measure), and compare its performance with that of a weighted random picker baseline. Our experiment results show that three variants of our approach that use different underlying classification algorithms can substantially outperform the baseline. Our approach can benefit companies to improve their customer service strategies to deal with different categories of customers. For the company in our case study, our approach extends the current capability of their social media listening system by adding a prediction functionality.

In the future, we plan to evaluate our proposed approach on more datasets. To improve the accuracy of our approach further, we plan to extract more features to better characterize different categories of users. We also plan to investigate more advanced classification algorithms.

Chapter 4

Inferring Spread of Readers' Emotion affected by online news

4.1 Introduction

Nowadays, online news platforms are popular due to their up-to-date content, and have become important sources of information. The platforms provide a convenient way for publishers to share latest news that can quickly reach online readers. The platforms also allow readers to interact by providing comments and votes, and by sharing news articles on social media.

Publishers, writers, and many others can potentially benefit from news readers' responses. The responses can be used to measure degree of user engagement. More comments or feedbacks given by readers indicate higher popularity of a news article. The responses can also be used as a clue for placing advertisement. Moreover, readers' responses can help publishers, writers, individuals, and organizations to learn how a certain issue is viewed by the public in general. Such insight can potentially be used by decision makers to make more informed decisions (e.g., on policy and business strategy). Given the value of readers' responses, it would be beneficial to be able to predict *early* how the public are likely to respond to a particular issue described in a news article.

A news article should be objective as it is intended only to report facts. This means that readers' opinions to a news article are not contained in the article itself. To give an impression of objectivity, the writers often avoid using overly positive or negative vocabulary, or resort to other means to express their opinion, such as embedding statements in a more complex discourse or argument structure, and quoting other persons who said what they feel [9]. Separate responses to the news, when available, contain readers' opinions and emotions toward the content of the news.

Predicting readers' emotion for a particular article is an emerging research area. Most studies on predicting readers' emotion translate the task into a classification problem, either by considering it as a multi-class classification (assign an article into one of the emotion categories) [92, 51, 44, 53] or a multi-label classification (assign to each articles a set of emotion categories) [112] problem. In this study, we formulate the problem as a multi-target regression with the goal of predicting readers' *emotion distribution*. By knowing the predicted emotion distribution, we can get a deeper insight on likely readers' responses, e.g., estimated proportion of readers who are happy with a piece of news.

We explore lexicon-based and word-vector-based features, and input them to a regression model to predict emotion distribution. As a case study, we use a popular Indonesian online news site, namely *detik.com*. Our work complements existing works on readers' emotion analysis and prediction. Specifically, our contributions are as follows:

1. We create a new corpus consisting of Indonesian news articles for predicting readers' emotion distribution affected by news articles.

2. We compare the effectiveness of using different parts of news articles (headlines only, contents only, and both headlines and contents) to predict the spread of readers' emotion.

3. We compare the effectiveness of *domain-specific* emotion lexicon and word

embedding with *general purpose* lexicon and word embedding for the problem of predicting emotion distribution of a news article readers.

The structure of the remainder of this chapter is as follows. In Section 4.2, we describe the methodology of our proposed approach which consists of 5 main steps: corpus creation, word vector construction, emotion lexicon formation, feature extraction, and regression model learning. We describe our experiments and evaluate the effectiveness of our proposed approach in Section 4.3. Threats to validity are discussed in Section 4.4. We finally summarize the findings and mention future work in Section 5.7.

4.2 Methodology

The overall framework of our approach is depicted in Figure 4.1. In the *Construct Corpus* step, we collect a set of online news' links that are mentioned on Twitter, and crawl the corresponding news headlines and contents to build our news article corpus. By analyzing the corpus, we build an emotion lexicon and train word vectors in the *Build Emotion Lexicon* and *Build Word Vector* step, respectively. We extract features based on the emotion lexicon and word vectors in *Extract Features* step. In the *Build Regression Model* step, we use different combinations of extracted features to build regression models that predict reader's emotion distribution. We elaborate the above-mentioned steps in the following subsections.

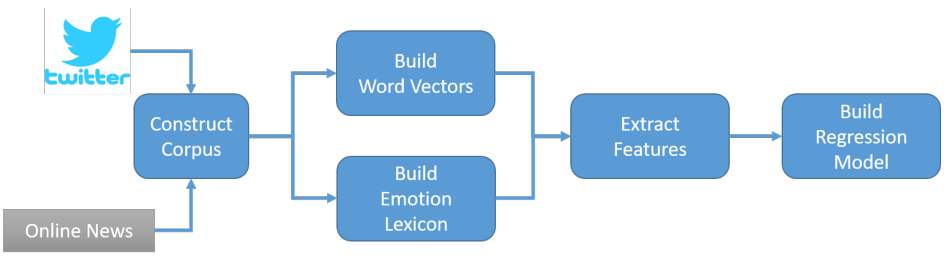

Figure 4.1: Our approach's overall framework to predict readers' emotion distribution

4.2.1 Problem Definition

In this study, we aim to predict readers emotion distribution affected by reading a news article. Given a corpus of documents D, with their emotion scores E, where E_i is the emotion score vector for a news article D_i, we want to predict emotion scores E' for a set of new articles D'. An example of a document-emotion score vector of a particular article is: ⟨ happy:0.4; amused:0.0; inspired:0.0; dont_care:0.6; sad:0.0; afraid:0.0; angry:0.0 ⟩.

4.2.2 Step 1: Construct Corpus

Many news organizations have recognized the potential of social media and have used social media marketing to attract online audiences; for example, by using Twitter to promote certain articles that might interest their readers. Therefore, we are interested in an online news platform that actively tweets news article headlines, and also provides emotion scoring and commenting features for the readers.

We identify an online news platform in Indonesia, namely *detik.com*. According to Alexa web ranking[1], it is ranked as the most popular online news and the fourth most popular website in Indonesia. The news platform provides features that allow users to give an emotion score to a particular article, as shown in Figure 4.2. There are eight different emotion categories, which can be translated in English as: *Happy, Amused, Inspired, Don't Care, Annoyed, Sad, Afraid* and *Angry*. The emotion score for each category will be shown in the same page as the article that the scores correspond to.

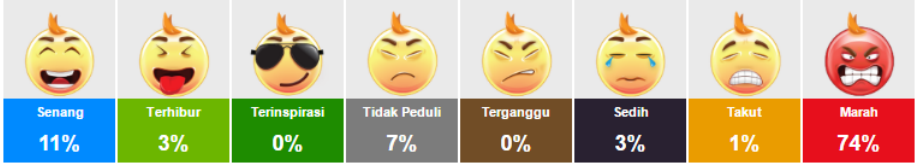

Figure 4.2: a sample of emotion scores of an article published in online news

The online news platform (*detik.com*) also has a Twitter account: *detikcom*. It

[1] http://www.alexa.com/topsites/countries/ID

has a large number of followers (13.7 millions as of April 2017) and ranked number 3 in Indonesia in terms of number of followers[2]. This makes *detik.com* a good source of data for analyzing the sentiment distribution of news articles' readers.

We initially collect news articles that were mentioned in *detik.com*'s Twitter account from November 2016 - February 2017. We use the Python Tweepy[3] module to get its Twitter timeline. We find that there are many duplicate tweets that refer to the same news article. Online media tend to repost the same content in order to get more traffic, hit multiple time zones, and reach new followers. We remove duplicate tweets by keeping the earliest tweet and removing newer ones. After removing duplicate tweets, we have 36,587 distinct tweets.

We then process the tweets, get their contents, numbers of retweets, and favorite counts. To get the corresponding articles from the online news platform, we extract URLs from the tweets. We build a custom webpage scraper and download the articles pointed to by the URLs. For each article, we get its headline, content and emotion scores.

We remove news articles that have no emotion scores; after this step, we have 11,704 news articles. However, some of these articles may only receive very few emotion votes. We further remove articles that are likely to receive few emotion votes. We use number of comments as a proxy to the number of votes[4]. We believe that the number of comments should be less than the number of votes, since it is more difficult and time consuming to write a comment, as compared to providing a vote. Therefore, we further filter our dataset to exclude articles that have less than 10 comments. At the end of this step, we have 1,575 articles remaining as our final dataset. Since we want to predict emotion distribution of unseen documents, we order the dataset based on the article's date, and use 80% from the ordered data as our training corpus. This corpus is used for building an emotion lexicon and word

[2] `https://www.socialbakers.com/statistics/twitter/profiles/indonesia/`

[3] `http://www.tweepy.org/`

[4] The website does not provide the number of votes but only the proportion of votes for the various emotions.

vector features. The remaining 20% of the ordered set is used as our testing corpus. This corpus is used to evaluate the performance of emotion distribution prediction approaches.

Table 4.1: Average emotion scores from our *detik.com dataset*

Sentiment	**Mean Score**
Happy	0.41
Amused	0.05
Inspired	0.05
Don't care	0.18
Annoyed	0.04
Sad	0.09
Afraid	0.02
Angry	0.18

Table 4.1 reports the average proportion of votes for each emotion for articles in our detik.com dataset. Note that the "happy" emotion has a higher score (i.e., most articles receive higher proportion of "happy" votes than other votes) than other emotions. Possible explanations for this observation are due to characteristics of commenters, or our dataset selection process. The predominance of the "happy" emotion has also been found in other datasets used in a related work by Staiano and Guerini [88].

4.2.3 Step 2: Build Word Vector

Word embedding is a technique to represent words in a form of continuous value vectors. These vectors encode meanings of words. One of the most popular word embedding technique is *word2vec*. *word2vec* uses a shallow neural network to reconstruct contexts of words. Two architectures can be used to generate the vectors: continuous bag-of-words (CBOW) or continuous skip-gram (SG) [58]. For CBOW, a neural network is trained to predict a word based on its surrounding words. In this architecture, the continuous value vector for a word is the vector that is input to the last layer in the network after we input its surrounding words to the network. For SG, a neural network is trained to predict surrounding words based on the current

word. In this architecture, the continuous value vector for a word is the vector that is output by the first layer in the network.

Continuous value vectors that are generated by *word2vec* contain semantic meanings of words. Words that appear in similar contexts tend to have similar vector representations. The vectors also have an interesting arithmetic feature. For example, the resultant vector of the following arithmetic operation (vector of brother - vector of man + vector of woman) is pretty similar to the vector of sister. This is related to analogical reasoning where brother is to sister as man is to woman, which is encoded in the vector representation learned by *word2vec*.

Building on top of the success of word embedding, we learn a custom word embedding from our training corpus. In practice, SG tends to be more effective than CBOW when larger datasets are available [49]. However, due to relatively small size of our training corpus, we use a CBOW model to build word vectors. To create word vectors, we first split news articles in the corpus into sentences. Indonesian texts use the same sentence end symbols as those used in English texts (sentences can end with "?", "!", or "."). We use NLTK's punkt tokenizer[5] for sentence splitting. Given the generated sentences, we train *word2vec* model using Python's gensim module [79]. We compute 300-dimensional word embedding with CBOW model on our training corpus, without removing stop words. We have 76,752 word vectors generated from our training set.

4.2.4 Step 3: Build Emotion Lexicon

An Emotion lexicon is a dictionary that associates words with emotion categories, such as anger, fear, surprise, sadness, etc. It is typically constructed via crowdsourcing. In the crowdsourcing process, a group of people is asked to label a set of words by associating each word with one or more basic emotions. Labels from the group of people are then aggregated for each word by summing up votes for each basic

[5] `http://www.nltk.org/`

emotion. The resultant sums are then normalized across basic emotions, which represent emotion distribution for the corresponding word. The resultant collection of words along with their corresponding emotion distribution is the constructed emotion lexicon.

Another approach to create an emotion lexicon is to use a crowd-sourced affective annotation from a social news network, such as the one used in Staiano and Guerini work [88]. Typically, an automated tool such as a web crawler is used to get news articles and related emotion scores tagged by readers. By splitting a news article into words, emotion scores for each word in the article are calculated.

To create an emotion lexicon, we first construct a document-by-emotion matrix containing the eight emotion scores for each document. We follow a previous work to create a word-by-emotion matrix [88]. We also create a word-by-document matrix containing normalized word frequency across documents. We multiply the document-by-emotion matrix and the word-by-document matrix to produce a word-by-emotion matrix. In the end, we have 22,346 words and their corresponding eight emotion scores that we refer to as our generated Emolex (Emotion Lexicon). An excerpt of the matrix is shown in Table 4.2.

Table 4.2: Sample taken from *word-by-emotion* matrix generated by analyzing our detik.com training corpus

Word	**Happy**	**Amused**	**Inspired**	**Don't Care**	**Annoyed**	**Sad**	**Afraid**	**Angry**
Walikota (Mayor)	0.488	0.032	0.087	0.209	0.010	0.028	0.006	0.142
Membunuh (Kill)	0.246	0.038	0.091	0.055	0.017	0.152	0.047	0.354
Pahlawan (Hero)	0.442	0.050	0.051	0.058	0.040	0.080	0.016	0.264

4.2.5 Step 4: Extract Features

A news article contains headline and content. We explore different combinations of news article parts to extract features from, i.e., use only headlines, contents or both. We follow an emotion lexicon construction process (described in Section 4.2.4) and word vectors training process (described in Section 4.2.3).

Lexicon Features

We build around 22,000 lexicons tagged with emotion scores as described in Section 4.2.4. We transform each news article in our corpus into a document-by-emotion feature vector by following these steps:

1. We split the considered portion of a news article into words, and then remove the stop words.

2. For each word, we retrieve its emotion score vector from our word-by-emotion matrix.

3. We calculate the emotion vector for the news article by averaging emotion vectors of the words in the news article.

In the end, we have a document-by-emotion matrix of the following dimension: total number of articles in the corpus (1,575) $\times$ emotion scores (8). Each document-emotion vector in the matrix represents emotion lexicon features for the corresponding news article.

Word Vector Features

Our set of trained word vectors model includes around 76,000 vectors of 300 dimensions. We generate a vector for each news article. To do this, we use a vector-averaging approach, which consists of the following steps:

1. We split the considered portion of a news article into words, and then remove the stop words.

2. For each word, we retrieve its word vector from our trained *word2vec* model.

3. We generate a vector for the news article by averaging word vectors that corresponds to the words in the news article.

As a result, we have a vector for each news article in the corpus. Since we have 1,575 news articles, we get a 1,575 $\times$ 300 matrix.

4.2.6 Step 5: Build Regression Model

We formulate our problem as a multi-target regression task. Multi-target regression is a family of regression techniques where there are multiple output variables. In multi-target regression task, a set of training example E is given, where each example is in the form of $(\mathbf{x}, \mathbf{y})$. $\mathbf{x} = \{x_1, x_2, x_3, ..., x_A\}$ is a vector of A attributes and $\mathbf{y} = \{y_1, y_2, y_3, ..., y_T\}$ is a vector of T target values. Multi-target regression learns a model that, given $\mathbf{x}$, predicts all T target values in $\mathbf{y}$ simultaneously. Multi target regression is generally solved by transforming it to multiple single-target regression or adapting the regression algorithm to directly deal with multiple outputs.

Given features extracted from our training corpus, we build a multi-target regression model to predict spread of readers' emotions. We explore different sets of features, i.e. emotion lexicon features (with 8 independent variables), word vector features (with 300 variables), and combination of both.

4.3 Experiments and Results

In this section, we first describe our experiment setting, baselines used and evaluation metrics. Then, we present our research questions and results of our experiments which answer the questions.

4.3.1 Experiment Setting and Evaluation

Experiment Setting

Our dataset consists of 1,575 articles. We use 80% of this data as training corpus, and the remaining as testing corpus. Before we build the word vector model as described in Section 4.2.3, we preprocess the corpus using Python NLTK and the Scikit module. We remove punctuations and non-word characters, and convert the remaining characters into lowercase.

We use the Scikit-Learn[6] module to build a multi-target regression model. The module supports multi-target regression by transforming it into multiple single-target regression tasks. The single-target regression algorithm is determined by choosing a base regressor. For choosing a good base regressor, we experimented with different regressors such as Linear Model, Random Forest Regressor, Support Vector Regressor, and Gradient Boosted Regressor. We found that Gradient Boosted regressor achieves the best overall performance compared to other regressors. Therefore, we use this regressor in our experiments.

All experiments were done on an Intel Core i7 CPU, 8 GB RAM notebook running Windows 10 64 bit.

Baseline

We compare our model with two general purpose models that can be used for emotion prediction:

1. We use Sentic-API [20] as a general purpose emotion lexicon. Sentic-API supports Indonesian language. It contains denotative (i.e., semantics) and connotative information (sentics) associated with 50,000 common sense concepts. A word-emotion lexicon that associates words with four dimensions of sentics (pleasantness, attention, sensitivity, and aptitude) is also provided. For each news article, we take the sentics values for each word and compute the average value for each sentics dimension. The generated four average values (i.e., each corresponding to a particular sentics dimension) are the news sentics. These average values are converted to a feature vector that is input to a multi-target regression model. An excerpt of the word-sentics matrix is shown in Table 4.3.

2. We use a freely available word vector model trained using FastText [15] as the general purpose word vector. This model is trained on a Wikipedia dataset,

[6]http://scikit-learn.org

Table 4.3: An excerpt of Sentic-API's word-sentics matrix

Word	Aptitude	Attention	Pleasantness	Sensitivity
Gembira (Happy)	0.193	0.156	0.504	-0.176
Sedih (Sad)	-0.051	0.266	-0.826	-0.461
Walikota (Mayor)	0.000	0.152	0.079	-0.061

and it is available for 294 languages including Indonesian. The pre-trained model has word vectors with dimension of 300, and was obtained using the skip-gram model described in Bojanowski et al.'s paper [15]. For each news article, the word vector associated with each word is collected and averaged. The averaged word vector is considered as the news representation and input to a multi-target regression model.

Evaluation

To measure the effectiveness of our approach and the baselines, we use RMSE (Root Mean Squared Error). RMSE is a widely used evaluation metric when estimating continuous values. It is the square root of the average of squared differences between prediction and actual observation. The metric is defined below:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

where N is the number of documents, y_i is the ground truth of emotion score, and $\hat{y}_i$ is the predicted emotion score.

4.3.2 Research Questions and Results

RQ1: How effective is the use of different portions of news article (headlines only, contents only, headlines+contents) in predicting emotion scores?

Approach: In this research question, we investigate the effectiveness of using three different portions of news articles: news headlines only, news contents only and combination of news headlines and contents. For each of them, we trained separate

word2vec models using gensim. We also create different word-by-emotion matrices. Based on the extracted features, we build regression models and calculate RMSE for each emotion category.

Results: Table 4.4 shows the results of our experiments. We can see that generally, using headline combined with contents performs at least as good as using headline or content only. This finding suggests that both headlines and contents contain useful information that can be combined together to create a better model.

Table 4.4: RMSE scores of the emotion lexicon (EM) and word vectors (WV) when considering different portions of news articles: Headlines (H), Contents (C), and Headlines+Contents (H+C)

	Features	**Happy**	**Amused**	**Inspired**	**Don't Care**	**Annoyed**	**Sad**	**Afraid**	**Angry**	**Average**
H	WV	0.299	0.105	0.105	0.230	0.068	0.149	0.039	0.259	0.157
C	WV	0.284	0.107	0.113	0.235	0.063	0.148	0.058	0.243	0.156
H+C	WV	0.278	0.098	0.120	0.213	0.060	0.145	0.077	0.252	0.155
H	EM	0.308	0.071	0.118	0.242	0.071	0.143	0.051	0.239	0.155
C	EM	0.299	0.079	0.111	0.257	0.046	0.142	0.056	0.252	0.155
H+C	EM	0.277	0.103	0.095	0.203	0.056	0.131	0.034	0.216	0.151

RQ2: How effective are the generated emotion lexicon (EM) and word vectors (WV) as compared to the general purpose baselines?

Approach: In this research question, we compare the effectiveness of using an emotion lexicon and word vector generated by our approach against the general purpose baselines (see Section 4.3.1) to predict emotion scores distribution. Our previous experiment shows that using the news headline combined with news content generally produces a better result. Therefore, we use this combination for this experiment.

Results: Table 4.5 shows the results of our experiments. Our generated emotion lexicon features achieve better performance for predicting scores in all emotion categories, when compared to using Sentics. Similarly, our generated word vectors achieves a better performance, as compared to using a general pre-trained word vector from Wikipedia using FastText. Comparing average RMSE over all emotions, EM outperforms Sentics by 14.7 %, while WV outperforms FastText by 84.9

%. These results show the usefulness of building a domain specific emotion lexicon and training domain specific word vectors.

Table 4.5: RMSE scores of our generated emotion lexicon (EM) and word vectors (WV) as compared to a general purpose lexicon (Sentics) and word vectors trained on Wikipedia (FastText) on predicting emotion distribution scores

Features	Happy	Amused	Inspired	Don't Care	Annoyed	Sad	Afraid	Angry	Average
EM	0.277	0.103	0.095	0.203	0.056	0.131	0.034	0.216	0.151
Sentics	0.328	0.105	0.114	0.297	0.059	0.154	0.083	0.272	0.177
WV	0.278	0.098	0.120	0.213	0.060	0.145	0.077	0.252	0.155
FastText	0.373	1.695	1.314	0.494	1.385	1.127	1.293	0.54	1.028

RQ3: Can combining emotion lexicon and word embedding vectors improve the performance of the prediction model?

Approach: To answer this question, we combine the emotion lexicon and word vector features (EM+WV), and compare it with using the emotion lexicon features alone (EM) and word vector features alone (WV). Similar to RQ2, we extract features from both headlines and contents of news articles.

Results: Table 4.6 shows the results of our experiments. By combining emotion lexicon features and word embedding vector features (EM+WV), the average RMSE score is reduced from 0.151 to 0.130 (13.91%) as compared to EM, and from 0.155 to 0.130 (16.13%) as compared to WV. Therefore, by combining emotion lexicon and word vector features, we can improve performance of the regression model.

Table 4.6: RMSE scores of the combination of the emotion lexicon and word vector features (EM+WV) compared to when each set of features is used alone (EM or WV)

Features	Happy	Amused	Inspired	Don't Care	Annoyed	Sad	Afraid	Angry	Average
EM	0.277	0.103	0.095	0.203	0.056	0.131	0.034	0.216	0.151
WV	0.278	0.098	0.120	0.213	0.060	0.145	0.077	0.252	0.155
EM+WV	0.232	0.090	0.102	0.158	0.067	0.129	0.067	0.193	0.130
Sentics+WV	0.304	0.132	0.133	0.207	0.090	0.166	0.083	0.256	0.171

4.4 Threats to Validity

There are a number of threats that may affect the validity of our findings. In this section, we discuss threats to internal validity, external validity, and construct validity.

Internal Validity. Threats to internal validity relate to experimenter bias and errors in our implementation. We have checked our implementation, but there could still be errors that we do not notice.

External Validity. Threats to external validity relate to the generalizability of our findings. We have evaluated the effectiveness of our approach to infer readers' emotion scores in a corpus of 1,575 online news articles. In the future, we plan to reduce this threat further by considering a larger set of articles from various online news platforms.

Construct Validity. Threats to construct validity relate to the suitability of our evaluation metric. In this work, we use RMSE as the evaluation metric. RMSE is a standard metric and it has been used as evaluation metric in past studies such as in Lin and Chen [52]. Thus, we believe that threats to construct validity are minimal.

4.5 Chapter Conclusion

In this chapter, we have presented an approach that uses an emotion lexicon and word embeddings in order to predict readers' emotion scores distribution towards an online news article. We build a new corpus containing around 1.5k Indonesian news articles taken from *detik.com* along with affective annotations provided by readers of those articles. Our experiments show that, by using combined features of a domain-specific emotion lexicon together with word embedding vectors, we are able to predict the distribution of readers' emotion scores with a Root Mean Squared Error (RMSE) score ranging from 0.067 to 0.232. Our approach is generic and can be easily replicated to other online news platforms that allow readers to

provide affective annotations. Our approach can benefit publishers by giving them early insight on expected public response to a particular article, before they actually publish it.

In the future, we plan to evaluate our proposed approach on another corpus. To improve the accuracy of our approach further, we plan to experiment with more features to better characterize different reader emotions. One possibility is by improving accuracy of the emotion lexicon using bag-of-concepts [80] instead of bag-of-words.

Chapter 5

Helping Developers Sift Wheat from Chaff via Cross-Platform Analysis

5.1 Introduction

Software developers rely on many sources of knowledge to help them stay up-to-date on the latest technologies and to accomplish their development tasks. A study conducted by Maalej et al. [54] showed that 70% of developers use online resources (web search engines, public documentation, forums) as channels to access knowledge. Among those channels, some are more popular and contain richer content relevant to software engineering compared to others. For example, Xin et al. [102] observed how developers commonly make use of a web search engine such as Google to find online resources to improve their productivity. They found that 63% of the searches on the Internet ended with a visit to Stack Overflow, a popular question and answer (Q&A) site. Another popular channel are microblogging platforms (e.g., Twitter): a large number of software developers use Twitter frequently to support their professional activities, e.g., to share and obtain the latest technical news [85].

Aside from their activity to seek knowledge in various platforms, many software developers also share their experiences or knowledge through forums and so-

cial media. A study by MacLeod et al. [56] found that video is a useful medium for communicating knowledge between developers, and that developers build their reputation by sharing videos through social channels (e.g., YouTube). On YouTube, developers as content creators will also benefit from comments given by their viewers. Such comments are valuable for content creators since the comments reflect users experience with the video.

However, extracting software-related knowledge from different platforms requires varying levels of effort. For example, on Stack Overflow, almost all of the content is related to software development. The content is also maintained to be of high quality by the collective community effort and the siteFLs moderators. But it is more challenging to extract software-development-related information from Twitter, since Twitter is a social media channel which contains a wide range of content. In order to help software developers to handle these challenges, there is a need for an approach that helps developers filter relevant content from various platforms. Our work is motivated by the following use cases:

Use Case 1: We consider a developer who wants to acquire new knowledge based on software-relevant tweets. Singer et al. [85] found that developers face challenges while using Twitter, which relate to having to deal with a huge amount of irrelevant tweets produced on Twitter, as well as the challenge of maintaining a relevant network. It is impossible to follow all relevant Twitter users since tweeting behaviour constantly changes and new Twitter users enter the service. We can collect tweets generated from tens of thousands of users regularly (which can amount to millions of tweets per day), but the problem is in the identification of relevant tweets in this large collection of tweets. Furthermore, even Twitter users who are considered experts in the software development community may at various times post tweets that are not software-relevant. Indeed, a manual investigation of 1,000 randomly selected tweets (done by an author and an independent annotator) from a collection of more than 6 millions tweets found that only 16.7% of the tweets are software re-

lated. As the developer has limited time to inspect new tweets, a content aggregator for Twitter data related to software development is essential. This will address the problem of following the right people and the large amount of irrelevant content. To create such an aggregator, there is a need for a solution that finds relevant tweets among the tweets that are released in a given time period. More generally, automatic identification of software-relevant tweets will also enable downstream applications such as creation of a specialized Twitter feed for the developer community. It can also be used to aid downstream analytics task related to tweets such as mining of user requirements [101], early detection of software issues [34], or topic / trend detection [83].

Use Case 2: We consider a software developer who acts as a content creator and publishes a coding tutorial on YouTube. Through these tutorials, viewers can visually follow the instructions provided in the videos. Additionally, viewers can leave a comment that expresses their experience with the video. From the content creator point of view, as stated in the work by Poché et al. [70], digesting information taken from the comments will help content creators to be more engaged with their audience and improve their future videos. The identification of software-relevant comments will be useful, in particular given that Poché et al. [70] report that only a small percentage (30%) of comments in the videos they analyzed are content-related. Automatic filtering of relevant comments (referred to as content concerns comments or informative comments by Poché et al.) will enable creators to study feedback provided by viewers more efficiently, and similar to tweets, such filtering can be used to improve downstream analytics tasks, such as detection of common topics among relevant comments.

In both platforms mentioned above (Twitter and YouTube comments), sentences are typically short, contain a lot of noise, and may contain non-standard words. In order to address these challenges, we propose SIEVE, an approach to help automated tasks in a non software-development-specific platform. Our approach uti-

lizes content from a rich software-development-specific platform based on transfer representation learning, to help automated knowledge extraction tasks in other less software-development-specific platforms. We consider two platforms, Software Engineering Stack Exchange and Stack Overflow, as the rich domain-related platforms. We build word embedding based on the dataset collected from these platforms. We then leverage the word embedding models to solve information retrieval and classification problems in two different target platforms. We experiment with two different use cases: finding tweets relevant to software development on Twitter [82], and classifying informative comments for software engineering video tutorials on YouTube [70]. We conducted experiments based on the existing datasets (as described in Table 5.2) provided by Sharma et al. [82] for Twitter, and Poché et al. [70] for YouTube comments. Our experiments show the effectiveness of our proposed cross-platform analysis approach which achieves performance improvements of up to 28% and 10.3% for the first and second use case respectively. Our contributions can be summarized as follows:

1. We propose an approach based on transfer representation learning and word embedding to solve information retrieval problems on how to use data from domain-specific platforms to help tasks in other platforms.

2. We conduct experiments to show the effectiveness of the proposed approach for two different tasks and platforms (i.e., Twitter and YouTube), and use baselines described in existing work.

3. We compare the performance of various word embedding algorithms with regards to our use cases.

The next sections in this chapter are structured as follows. In Section 5.2, we describe background related to knowledge channels for software developers, and background on representation learning and word embedding. In Section 5.3, we describe our approach on learning a knowledge representation from source platforms.

We present our first use case on finding software-related tweets in Section 5.4. Next, we present the second use case on classifying informative comments on YouTube in Section 5.5. In Section 5.7, we discuss findings that are relevant to our approach. Threats to validity are discussed in Section 5.6. Finally, we conclude and mention future work in Section 5.8.

5.2 Background

In this section, we first discuss the knowledge sources used by developers which we have considered in our current work. Next, we discuss background on transfer representation learning and word embedding.

5.2.1 Knowledge Sources for Software Developers

Storey et al. found that software developers use many communication tools and channels in their software development work [90, 91]. In this work, we focus on learning word embedding from software-development-specific channels such as Software Engineering Stack Exchange and Stack Overflow (which are popular software discussion forums), and use the learned embedding to improve the performance of information retrieval and classification tasks related to the extraction of software-development-related knowledge from open domain channels such as Twitter (a microblogging site) and YouTube (video sharing). In the subsequent paragraphs we give background on these channels.

Software Engineering Stack Exchange: Stack Exchange[1] is a network of question and answer (Q&A) websites, where each website focuses on a specific topic. On any of the websites each of which is related to a particular domain, its users can ask questions related to that domain and other users can provide answers to these questions. The motivation for users to answer questions comes from the points that

[1] https://stackexchange.com/

they can gain when other users in the same community upvote or accept their answers. These points help them to build a reputation in the domain (and the related community), which the Stack Exchange website is focused on. The Stack Exchange community has been the focus of many studies e.g. [12, 74]. In this work as we are interested in improving the performance of information retrieval and classification tasks related to software engineering, we focused on Stack Exchange communities related to software engineering and programming, which are Software Engineering Stack Exchange[2] and Stack Overflow[3] respectively. The difference between these two sites is that *Stack Overflow* is focused only on specific programming tasks and problems, whereas *Software Engineering Stack Exchange* allows more general questions related to software development and engineering such as discussions about various libraries, methodologies etc. The latter has about 50,655 questions and 260,361 users. The intuition behind using Software Engineering Stack Exchange is that models trained on the general nature of content may achieve different performance on the task of filtering information from open domain websites such as Twitter and YouTube.

Stack Overflow: Stack Overflow[3] is a programming question and answer website founded in 2008 with a focus on software development. It is an online forum where anybody facing a programming issue can post a question describing the problem they face. The questions posted are public on the forum, so any other user on the forum can post their solutions as answers to the posted questions. The original asker can then mark an answer as accepted if it solved the problem. Other users can also upvote an answer if they think it is the right method to solve the programming challenge being addressed. Thus Stack Overflow helps developers in getting answers to their problems with the help of the crowd. It is one of the most used websites by software developers in the world having more than 9,000,000 registered users, more

[2]https://softwareengineering.stackexchange.com/
[3]https://stackoverflow.com/

than 16,000,000 questions and an Alexa Rank of 70[4]. As Stack Overflow contains rich software development and software engineering content, it has been immensely popular among software engineering researchers in recent years, where it has been used to discover topics and trends [11], generate API call rules [7], explore knowledge networks [108], build information filtering models [82].

Figure 5.1 shows a sample question-and-answer thread from Stack Overflow. Each thread generally contains five types of information: title, tags, body, answers, and comments. The title of a thread is a summary of the question asked. The tags represent the metadata related to the question being asked and are entered by the person who asked the question. Whenever somebody asks a question on Stack Overflow, they receive a recommendation to attach at least three tags to the question. The body part of the thread contains the description of the question. Whenever a question is answered, the answer appears in the answers section of the thread. Other developers can also ask further clarifying questions or comment on the question or answers posted up to that point.

Twitter and Software Engineering: Twitter is currently one of the most popular microblogging sites in the world. On Twitter, a user can post short messages (a.k.a. *tweets*) broadcasted to all other Twitter users who are following the user. Twitter allows a user to *follow* another user, which means the latter subscribes to all the tweets of the user he/she is following. Users also have an option of reposting the tweets posted by others – an activity known as *retweeting*. Twitter also allows users to mark favorite tweets, which conveys their interest in the content of a tweet.

By virtue of its simple design and easy-to-use functionality, Twitter has become a powerful medium for information sharing and dissemination. It started as a social networking medium but has nowadays become one of the important sources of information for people to keep up-to-date with the latest news and information about their domains of interest, to share and promote knowledge, and to keep in

[4]https://en.wikipedia.org/wiki/Stack_Overflow

Figure 5.1: A sample question-answer-thread on Stack Overflow with tags (Thread ID 626759)

touch with their family and friends [48]. Twitter influences many communities including the software engineering community as highlighted by many prior studies [85, 17, 99, 94]. Various techniques have been proposed recently to mine software engineering relevant information from Twitter [82, 84, 101, 39].

YouTube and Software Engineering: YouTube is a website where anybody can share videos [29]. It has over 1 billion users and generates billions of views daily [110]. YouTube has also evolved into a knowledge sharing resource, where people can share informational videos, follow other users and comment on videos. Thus it provides people with resources to share information, learn new knowledge, as well as get and provide feedback.

Software developers also use YouTube for sharing information as well as learning [56, 71]. MacLeod et al. found that developers share videos detailing information they wished they had found earlier [56]. The videos mainly relate to sharing knowledge about development experiences, implementation approaches, design pattern application, etc. Other work focuses on extracting relevant information for developers from YouTube, which is a challenging task given the large size of videos. Tools to help developers find relevant content from software engineering videos have been proposed [72, 106]. Poché et al. analyzed user comments related to software engineering videos posted on YouTube [70] and proposed a technique for finding relevant comments.

5.2.2 Word Embedding and Transfer Representation Learning

In machine learning, many methods perform well under the common assumption that the training and test data are drawn from the same feature space and the same distribution. In many contexts, this assumption may not hold. For example, we attempt to solve a classification problem in a domain that does not have enough training data, but we have sufficient data in other related domains. In this case, knowledge transfer or transfer learning would be useful to solve the classification problem [65]. In the context of representation learning, transfer representation learning is where rich representations are learned in a source platform with the aim of transferring them to different target platforms [4]. Representation learning also can be described as learning representations of data that make it easier to extract useful

information when building classifiers or other predictors [13]. In the field of Natural Language Processing (NLP) applications, distributed word representations, i.e., word embedding, are one of the products of representation learning.

Word embedding represent words in a low dimensional continuous space, to convey semantic and syntactic information [58, 67]. One of the most popular word embedding techniques is Word2Vec, which uses a shallow neural network to reconstruct contexts of words. Mikolov et al. [58, 59] proposed two methods to learn word embedding: Continuous Bag-of-Word (CBOW) and Skip-gram. They have been widely adopted due to their effectiveness and efficiency. For CBOW, a neural network is trained to predict a word based on its surrounding words. In CBOW, the continuous value vector for a word is the vector that is input to the last layer in the network after we input its surrounding words to the network. For Skip-gram, a neural network is trained to predict surrounding words based on the current word. In this architecture, the continuous value vector for a word is the vector that is output by the first layer in the network. It has been shown that the embedding vectors produced by these models preserve the syntactic and semantic relations between words under simple linear operations.

A recent study by Semwal et al. provided some practical guidelines for applying transfer learning to NLP applications [81]. They highlighted that the content of the embedding layer of a neural network learned from one dataset can potentially be used for another dataset. They also suggested to pick a source domain with a large vocabulary size that contains content with similar semantics to the target task. Their findings motivate us in investigating the effectiveness of word embedding learned from platforms that contain a large collection of software engineering content (e.g., Stack Overflow and Stack Exchange) to help software engineering related tasks in other platforms with much less software engineering content (e.g., Twitter and YouTube).

5.3 Approach

In this section, we first describe an overview of SIEVE, our approach to help automate tasks in a non software-development-specific platform. Afterwards, we provide more detailed discussion regarding stages in the proposed approach.

5.3.1 An Overview of SIEVE

Our work is related to transfer representation-learning, where rich representations are learned from a software-development-specific platform, and leveraged in a different target platform. To represent knowledge in the source platform, we build a word embedding model that represents each word as a low-dimensional vector such that words that are similar in meaning are associated with similar vectors.

Our approach consists of two stages: representation learning from a source platform, and model building for a target platform. In the first stage, we learn a word embedding representation from a software-development-specific platform. In the second stage, we leverage the word embedding model built from the platform to resolve tasks in the target platform. Figure 5.2 shows the overall architecture of our proposed framework.

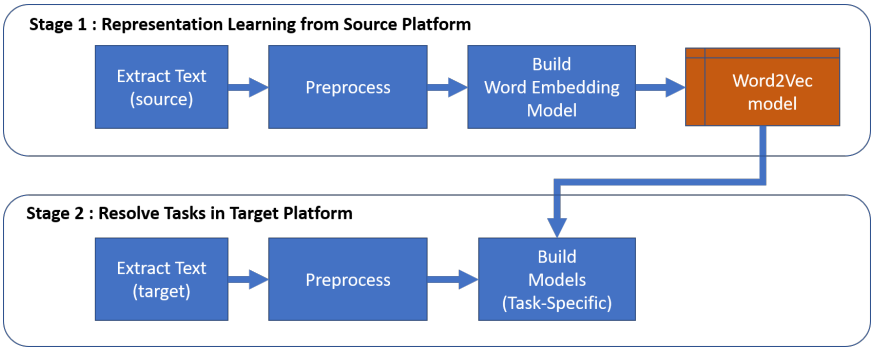

Figure 5.2: Overall approach

5.3.2 Stages of the Proposed Approach

Stage 1: Representation Learning from Source Platform

While most research done on Q&A sites is based on Stack Overflow data (e.g. [82, 111]), we believe that relevant domains of Stack Exchange are also a good source for software engineering related terms. In this work, we focus on one domain: Software Engineering Stack Exchange (Stack Exchange-SE). Stack Exchange-SE is an excellent source as it is a "question and answer site for professionals, academics, and students working within the systems development life cycle" and contains posts related to "software development methods and practices", "requirements, architecture, and design", "quality assurance and testing", and "configuration, build, and release management" [87]. We use text data extracted from the two sites (i.e., Stack Overflow and Stack Exchange-SE), and build two models: `SIEVE_SO` which is based on Stack Overflow and `SIEVE_SE` which is based on Stack Exchange-SE data.

The StackExchange-SE dataset is publicly available on the Stack Exchange data dump site.[5] We use the following two files: Posts.7z and Comments.7z. Posts.7z contains the title and body of posts (i.e., questions and answers) that appear on Stack Exchange. Comments.7z contains comments that users give to the questions and answers on Stack Exchange. Our Stack Exchange dataset contains a total of 149,478 posts, 409,740 comments, and 46,246 titles generated in a time period spanning from September 2010 to August 2017. We combined all of the posts, comments and titles for learning word embedding from this dataset. We do not perform filtering of posts in the Stack Exchange-SE and Stack Overflow datasets (e.g., for answers or questions with negative votes), since we assume that most of the posts will be software related, regardless of their quality.

We used the Stack Overflow data dump provided by previous work by Sharma et al. [82]. The data was also taken from the data dump site.[5] They extracted the questions and answers from the Posts.7z file, and user's comments from Comments.7z. These files contain content posted on Stack Overflow from September 2008 to September 2014. There are a total of 7,990,787 titles, 21,736,594 posts

[5]http://archive.org/download/stackexchange

(questions and answers), and 32,506,636 comments. Since there are too many posts and comments to efficiently process the data, to reduce the time it takes to learn a model, they randomly selected 8 million posts and comments from the data dump. We use this randomly selected data and combine all of the posts, comments and titles.

Before we build the word embedding model, we performed the following text preprocessing for both datasets:

1. Parse the posts into sentences, since we want to train word embedding at sentence-level. We use NLTK *punkt* tokenizer[6] for sentence splitting.

2. Remove all HTML tags since they do not contain useful information for word embedding.

3. Remove all special characters (e.g., symbols, punctuations, etc.) and words that contain only numbers.

4. Change all words to their lower case.

We do not use stemming in the preprocessing steps as many past studies [70, 8, 101] have shown that it does not boost performance. We chose the Skip-gram Word2Vec model proposed by Mikolov et al. [58]. We trained the word embedding models by using the Word2Vec Skip-gram model implemented in Gensim[7]. We use the following hyper-parameter settings: context window size of 5, vector dimension of 300, negative samples of 5, and minimum word frequency of 0. Context window size determines the number of surrounding words to be included as the context of a target word. For example, a window size of 5 takes five words before and after a target word as its context for training. For our dataset, since the sentences are typically short, we chose a window size of 5. The vector dimension is the size of the learned word vector. Training a higher dimension word vector is more

[6]http://www.nltk.org
[7]https://pypi.org/project/gensim/

computationally costly and produces a larger word embedding matrix. According to the experiment conducted by Pennington et al. [67], the best accuracy was achieved with 300 dimensions. In their experiments, performance did not improve dramatically if the number of dimensions is increased further. Negative samples parameter refers to the number of randomly chosen negative words during training process. We use the default value of negative samples in Gensim (negative sample = 5). Minimum word frequency is set to 0, which means that the model will preserve all words in the dataset.

The output of the model is a dictionary of words, each of which is associated with a vector representation. Table 5.1 includes statistics on the generated word embedding learned from the datasets.

Table 5.1: Statistics of datasets and word embedding extracted from Stack Overflow (`SIEVE_SO`) and StackExchange-SE (`SIEVE_SE`). Vocabulary size refers to the number of unique terms in the Word2Vec model.

	StackExchange-SE	**Stack Overflow**
Number of Posts	149,478	21,736,594
Number of Comments	409,740	32,506,636
Number of Titles	46,246	7,990,787
Number of Sentences (sampled)	3,152,950	8,000,000
Vocabulary size in Word2Vec	233,098	275,103

Stage 2: Model Building for Target Platform

Our goal is to leverage knowledge extracted from software-development-specific platforms and apply it to open-domain platforms. In order to examine the learned word embedding representation in stage 1, we utilize the word embedding in two different use cases. In the first use case, we aim to resolve the task of finding tweets related to software engineering. In the second use case, we leverage the word embedding to classify user comments on YouTube coding tutorial videos. We discuss each of the use cases further in the next sections.

5.4 Finding Relevant Tweets Using Word Embedding

In this section, we show how our approach can be used for the task of finding tweets related to software engineering. Researchers have found that developers use Twitter to support their professional activities by sharing and discovering various information from microblogs, e.g., new features of a library, new methodologies to develop a software system, opinions about a new technology or tools, etc. [82] However, due to various topics posted on Twitter, it becomes a challenge to find interesting software-related information on Twitter. To overcome this problem, Sharma et al. [82] proposed a language-model based approach and used the model to rank tweets based on their relevance to software engineering. We will use the proposed model as a baseline, along with other baselines. We aim to answer the following research question:

RQ1. How effective is our approach at the task of finding software related tweets?

5.4.1 Approach

Figure 5.3 shows our proposed approach for the task of finding software development-related tweets, by utilizing word embedding trained from a source platform. In general, we formulate the task of finding software-related tweets as a ranking problem, i.e., ranking the tweets in the order of their similarity scores with selected sentences from the source platforms. We follow these steps:

Step 1: Instance Selection

In our approach, selecting instances (i.e., sentences) from the source platform is an important task, since we will use these selected sentences to rank the tweets based on a similarity measure. Sentences extracted from the source platform (StackExchange-SE/Stack Overflow) are considered as software-related. However, some of the sentences may have different characteristics from Twitter. Therefore,

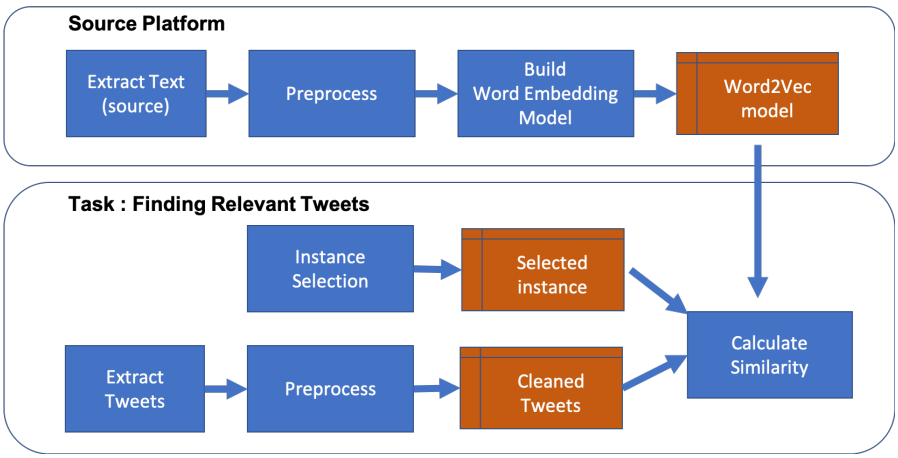

Figure 5.3: Our approach for finding software-related tweets

we use the following heuristic methods to select suitable sentences from a source platform:

1. We select sentences that have a length of no more than 140 characters which corresponds to a tweet's maximum length.

2. Among these selected sentences, we randomly sample sentences. By default, we sample 1000 sentences. We believe that this sampled set should be enough to represent sentences that contain software-related terms.

Step 2: Preprocess Tweets

We use the Twitter dataset provided by Sharma et al. [82]. We also use the same preprocessing steps as the prior work, i.e., we remove punctuation marks and URLs, and convert all words into lowercase.

Step 3: Calculate Similarity

To measure similarity between tweets and selected sentences taken from the source platforms, we need to use the same representation for both texts. Because sentences (or tweets) have different lengths, we transform each sentence into fixed-length vector to represent them. To build the vector representation, we leverage the word embedding learned from the source platform. The model consists of word vectors that have 300 dimensions as mentioned in Section 5.3. We follow these

steps:

1. For each sentence or tweet in the dataset, we tokenize it into words.

2. For each word, we look up its weight from the word embedding model. If a word does not exist in the model, we can either ignore that word, use a vector whose values are all 0 to represent it, or use the average of the embedding from words having the lowest frequency in the model. By default, we ignore the word that does not exist in the model. The result is a 300 dimensions vector of real values taken from the word embedding model.

3. We represent the sentence into a fixed-length vector. There are different ways to obtain text representation from word embedding. The most common methods use the maximum, minimum, or average of the embedding of all words (or just the important words) in a sentence [86]. In this case, we take the average of the word embedding of all words within the text, following Kenter et al. [47] At the end, we have a word vector of real values with 300 dimensions for each tweet or sentence.

Figures 5.4 and 5.5 illustrate the above mentioned steps. In Figure 5.4, N sentences are sampled from Stack Exchange. Each word of the sentence is then converted into a vector of values using the word embedding model. For example, the word 'sqlite' will be converted to a vector $\langle$-0.04910894,...,0.07086225$\rangle$. The vectors of words belonging to the same sentences are then averaged. At the end, there are N vectors representing the N sentences from the source platform.

In Figure 5.5, two sample tweets are first preprocessed into two vectors of words. Each word is then converted into a vector of values using the word embedding learned from the domain-rich platform (i.e., Stack Exchange). Next, the vectors of words belonging to the same tweet are then averaged and the resultant vector is used to represent the tweet. The second tweet is an example where there are no corresponding word vectors in the word embedding model. Thus, the average score for this tweet is zero.

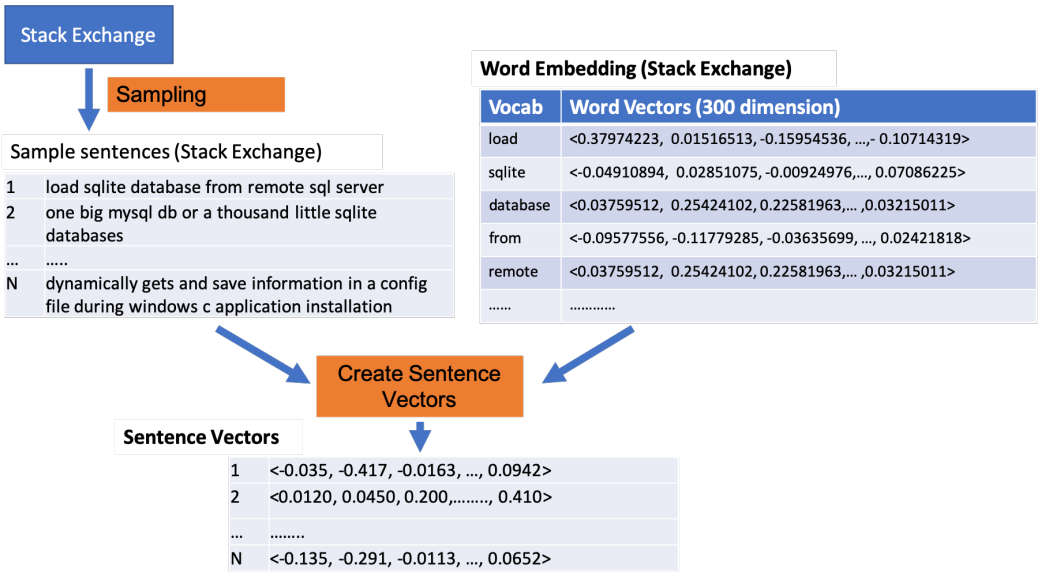

Figure 5.4: Illustration of sampling process from Stack Exchange, followed by creating sentence vectors

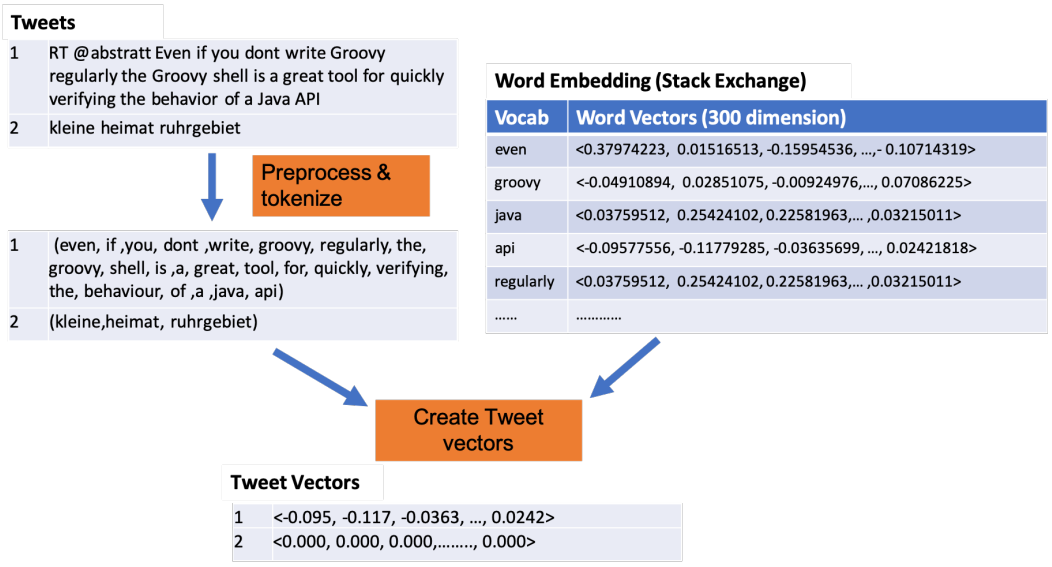

Figure 5.5: Illustration of preprocessing, tokenizing and creating tweet vectors.

For each tweet, we calculate similarities between the vector representation of the tweet with each of the N vector representations of the sample sentences. Considering a sample size of N, for each tweet, there will be N similarity scores. We then calculate the average of the N similarity scores. Next, we rank the tweets based on their average similarity scores. The higher the score, the more likely the corresponding tweet is software-related.

To calculate similarity between two word vectors, we use cosine similarity. Co-

sine similarity is a measure of similarity between two vectors (in this case, vectors of text representation) that measures the cosine of the angle between them. Given a tweet T and a selected sentence S, that are represented by two word vectors wv_{tweet} and wv_{so}, we define their semantic similarity as the cosine similarity between their word vectors:

$$similarity(T, S) = \frac{wv_{tweet}^{T}.wv_{so}}{||wv_{tweet}||||wv_{so}||}$$

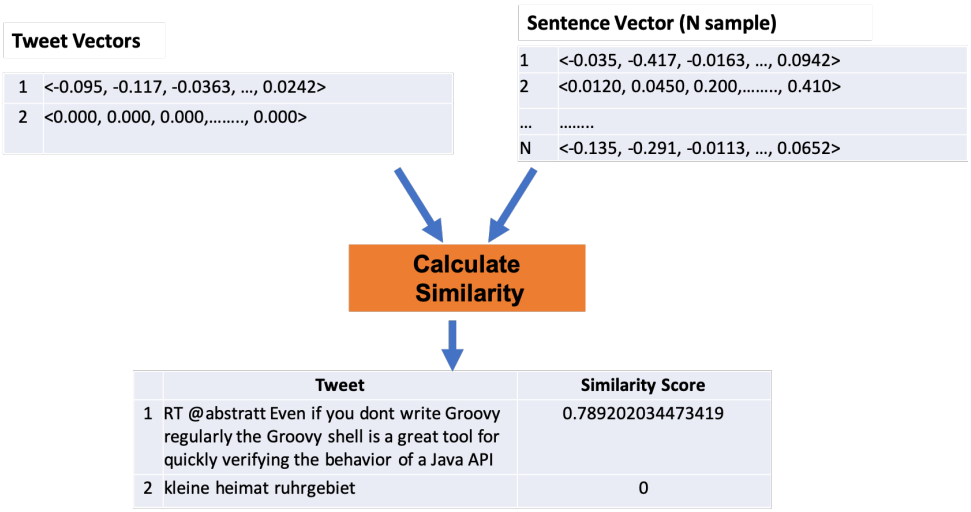

Figure 5.6: Illustration of calculating similarity score.

Figure 5.6 shows an example on how the similarity score is calculated. For each tweet vector, we calculate a similarity score between the tweet and each of the sentence vectors. The final score would be the average similarity score. In this example, the similarity score for the first tweet is 0.7892, while for the second tweet it is zero. We repeat this step for all tweets available in the dataset.

5.4.2 Dataset and Baselines

Dataset. For the Twitter dataset, we use the same dataset created by Sharma et al. [82]. The dataset consists of around 6.2 million tweets downloaded through the Twitter REST API. To collect tweets, they first obtained a set of microbloggers that are likely to generate software-related content. They started with a collection of 100 seed microbloggers who are well-known software developers. Next, they analyzed

the follow links of these software developers to identify other Twitter accounts that follow or are followed by at least 5 seed microbloggers. After they had identified the target microbloggers, they downloaded tweets that were generated by these individuals. The tweets collected were assumed to be mostly software related, since they were collected from potential software developers. However, based on our observation from a random sample of 1000 tweets, we found that only 16.7% are software related, while a majority of them (83.3%) are not software related. Statistics of the tweets dataset are listed in Table 5.2.

Table 5.2: Statistics of the Twitter dataset

Number of Tweets (Raw data)	6,294,015
Software related (based on sample of 1000)	16.7 %

Baselines. We used several baselines to show the effectiveness of our approach. First, we compared our proposed approach against NIRMAL [82], since we used the same dataset as their work. Next, since our models are trained on software-development-specific platforms, we compared the models with a within-platform model trained from the target platform (Twitter). To show the effectiveness of the Word2Vec-based models that we use, we compared the models with a model that uses Term Frequency − Inverse Document Frequency (td-idf) vectors generated from a source platform. The tf-idf technique has been widely used in other software-engineering-related information retrieval tasks, e.g. [30, 64]. We briefly describe the baselines as follows:

1. **NIRMAL by Sharma et al.** We used this approach as the main baseline. This approach builds an N-gram language model by using SRILM [89], a language modeling toolkit. NIRMAL learns a language model from Stack Overflow data. NIRMAL then uses the learned model to compute the perplexity score of each tweet. The lower the perplexity score, the more likely the tweet is software related. NIRMAL then ranks the tweets in ascending order of their perplexity scores and returns a ranked list. NIRMAL differs from SIEVE

in several aspects. First, SIEVE uses word embedding to capture relations between software-related terms. In addition, SIEVE samples selected Stack Overflow titles, uses them as seed sentences, and calculates similarity scores between the samples and the tweets. SIEVE then outputs a ranked list of tweets in ascending order of their similarity scores.

2. **Word2Vec trained on Twitter.** We consider this approach as a within-platform baseline, since we leverage knowledge extracted from Twitter itself as the target platform. We considered two datasets: (1) all tweets, (2) all software engineering (SE) relevant tweets. For the second dataset, we used 1,151 tweets that are labeled by an author and an independent annotator. The dataset as SE-related in our dataset. For each of these two datasets, we trained word embedding models using the approach (i.e., Word2Vec) and parameters described in Section 5.3.

3. **Term Frequency − Inverse Document Frequency (*tf-idf*).** In this approach, instead of using vectors generated by word embedding, we used td-idf vectors generated from a source platform. We built two variants of tf-idf vectors: one from Stack Overflow posts, and one from Stack Exchange posts. Term frequency (tf) is the number of times a word occurs in a given sentence, accompanied with a measure of the term scarcity across all the sentences, known as inverse document frequency (idf). Before constructing the vectors, we performed stemming using Porter Stemmer [73] and removed English stopwords. To remove stopwords, we used stopwords listed in the Python NLTK library[8].

5.4.3 Experiments and Results

Experiments Setting. We conducted experiments to answer RQ1 and evaluated the effectiveness of our approach as compared to the baselines. After following the steps in our proposed approach, we ranked the tweets based on similarity

[8]https://www.nltk.org/

scores between the tweets and selected instances taken from a source platform. We investigated various word embedding models trained from Stack Overflow, StackExchange-SE and Twitter, and one non-word embedding model (tf-idf).

The task of finding relevant tweets helps developers who wants to learn new knowledge based on software-relevant tweets [84]. As it is impossible for developers to follow everyone who may generate useful content, it is useful to create a content aggregator of tweets [2]. Such content aggregator will follow a large number of Twitter users who may potentially generate software engineering relevant tweets and capture all their tweets. Nirmal [82] and the solution that we built here help rank tweets for such content aggregators. As developers have limited time for learning, it is important that the top-K results listed in such aggregator are software related. Thus, to measure the effectiveness of Nirmal [82], Nirmals authors have proposed the use of accuracy@K, which is defined as the proportion of tweets in the top-K positions that are software-related. Here, we use the same evaluation metric. This metric has also been used in several other software analytics works such as API recommendation [105], duplicate bug report detection [42], and concept location [76].

We used accuracy@K, which is defined as the proportion of tweets in the top-K positions that are software-related. We manually evaluated the top-K tweets ranked by their similarity scores. We asked two labelers who have master's degrees in Computer Science to manually label the tweets, either as *"relevant"* or *"not relevant"* to software engineering. For our final ground truth, we labeled a particular tweet as "relevant" only if both labelers agreed that the tweet is software-development-related. We used Cohen's Kappa to measure inter-rater reliability for the labeling task. We obtained a Kappa value of 0.78 for labeling `SIEVE_SE` and a Kappa value of 0.68 for labeling `SIEVE_SO` – following Landis and Koch's interpretation [50], these values indicate substantial agreement.

Results. The results of our experiments are shown in Table 5.3 and Figure 5.7. Overall, the word embedding model trained on Stack Exchange performed best

in terms of all accuracy@K measures. Word embedding models trained on platforms that contain rich software-development-related knowledge (Stack Exchange and Stack Overflow) performed better as compared to the baselines.

Table 5.3: Accuracy@K results of different approaches in our experiments (best results are in bold).

Approach	acc@10	acc@50	acc@100	acc@150	acc@200
Nirmal	0.900	0.820	0.720	0.707	0.695
TF-IDF (Stack Overflow)	**1.000**	0.540	0.730	0.693	0.575
TF-IDF (Stack Exchange)	**1.000**	0.560	0.400	0.347	0.310
Word2Vec (Twitter-All)	0.400	0.220	0.260	0.260	0.235
Word2Vec (Twitter-SWrelated)	0.900	0.580	0.540	0.547	0.530
SIEVE_SO	0.900	0.880	0.870	0.847	0.800
SIEVE_SE	**1.000**	**1.000**	**0.990**	**0.980**	**0.975**

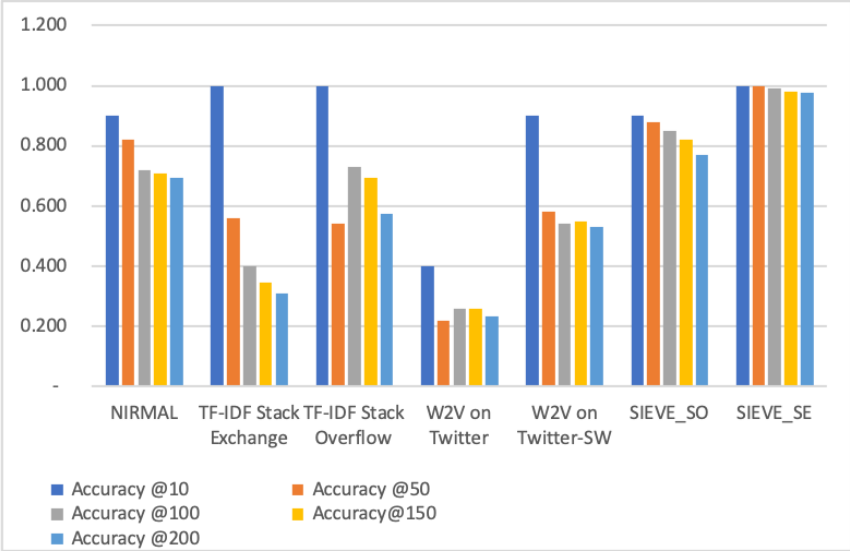

Figure 5.7: Comparison of Accuracy@K achieved by different approaches

Based on our experiments, the performance of the Word2Vec model trained on all tweets was lower than the other Word2Vec models. When we use the Word2Vec model trained on the software-related tweets, the accuracy was improved as compared to the one trained on all tweets (accuracy@10 of Word2Vec trained on all tweets is 0.400, while accuracy@10 of Word2Vec trained on software-related tweets is 0.900). However, the performance achieved was still lower than that of the Word2Vec model trained on the Stack Exchange-SE corpus. The low scores achieved by the word embedding models trained on all Twitter data can be attributed

to the fact that the content of typical tweets is mostly unrelated to software development.

While the tf-idf-based approach performed well when ranking the top-10 most relevant tweets, the performance degrades significantly when ranking top 50, and fluctuates when ranking the top 100, 150 and 200 tweets. Figure 5.8 shows a box-plot diagram, describing the word count of the top 200 tweets returned by various approaches. We found that, in the top-200 tweets ranked by the tf-idf approach, the tweets mostly contain the word stem "use", such as "*What is the use of c*", "*Java MySQL Insert Record using Jquery*", "*@shayman I used to work there"*. On the other hand, the top 200 tweets returned by SIEVE_SE contain more diverse vocabulary and tend to be lengthy, such as "*I still very much admire all the work put into TinyMCE Building a RTE is one of the most gruesome things you can do in a browser*," "*@Youdaman yes angular is very minimal in the amount of code and glue you need to do specially if you use a RESTful service*". This finding highlights the benefit of leveraging word embedding models to learn feature representation from a rich software-related platform.

The results show that SIEVE_SE improves all accuracy@K scores of up to 28% as compared to Nirmal. To measure whether this improvement is significant, we conducted a Wilcoxon signed-rank test [100]. Our null hypothesis is there is no difference in the mean accuracy@K (for K = 1 to 200) of SIEVE_SE and Nirmal. Wilcoxon signed-rank test results show that the p-value is less than 0.00001 which shows that we can reject the null hypothesis. This demonstrates that the improvement that SIEVE_SE achieves over Nirmal is statistically significant.

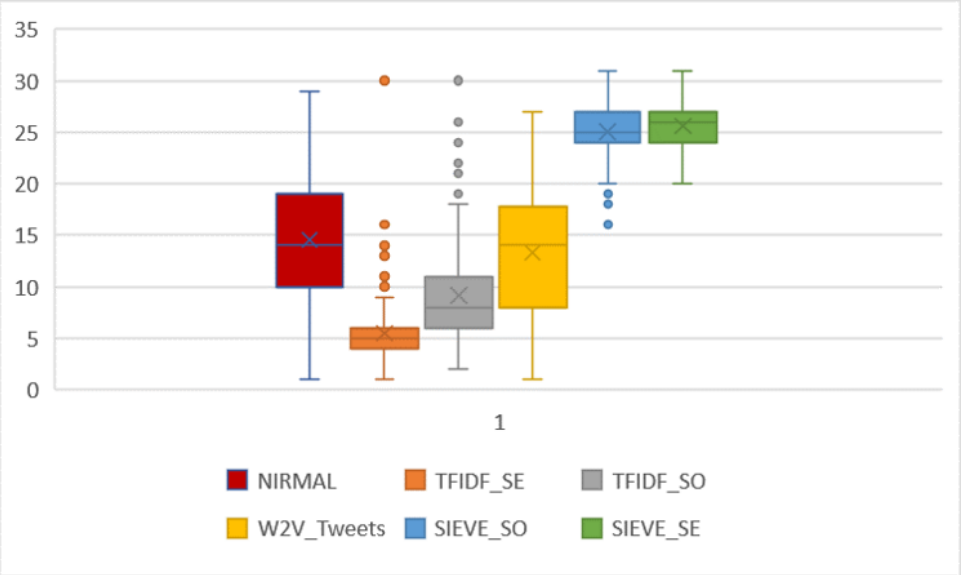

Figure 5.8: A box plot diagram representing word count of tweets returned by various approaches

5.5 Finding Informative Comments on YouTube Using Word Embedding

The objective of this task is to analyze user comments for YouTube coding tutorial videos. Important users' questions and concerns can then be automatically classified in order to help content creators to better understand the needs and concerns of their viewers, as described in work by Poché et al. [70] They categorized the comments into two general categories: informative vs. non-informative (which corresponds to other miscellaneous comments). We aim to answer the following research question:

RQ2. How effective is our approach at the use case of finding informative comments on YouTube?

5.5.1 Approach

Figure 5.9 shows our proposed approach for the use case of finding informative comments on YouTube, by utilizing word embedding models trained on the source platforms (StackExchange-SE and Stack Overflow). We formulate this task as a

binary classification problem, where a comment can be either informative or non-informative with regards to the video content. In order to build a classifier for this use case, we need to represent the YouTube comments into a feature representation. We leverage the word embedding learned from a source platform. The model consists of word vectors that have 300 dimensions as mentioned in Section 5.3. We build vectors to represent the comments, by following these steps:

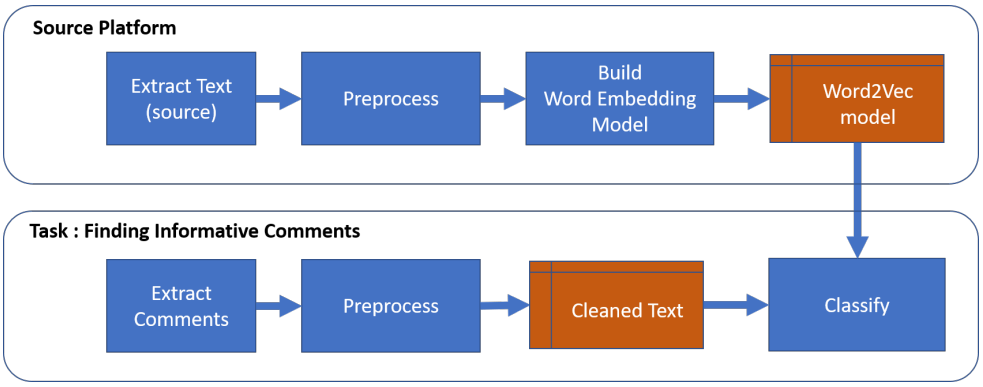

Figure 5.9: Approach for finding relevant comments on YouTube

1. For each comment, we tokenize it into words, remove words that contain only numbers, and change all words into lowercase.

2. Next, for each word in the comment, we look up its vector value taken from the word embedding model. We ignore a word if it does not exist in the word embedding model. We take the average of the word embedding of all words within the text, following Kenter et al. [47]. At the end, we have a word vector of real values with 300 dimensions for each comment.

Figure 5.10 serves as an example that illustrates the above-mentioned steps. Each word of the comment is then converted into a vector of values using the word embedding. The vectors of words belonging to the same comment are then averaged. As a result, the comment is represented as a vector with 300 dimensions. We repeat this step for all comments available. We then use the comment vectors as a set of features to be used by the classifier to distinguish informative from non-informative comments.

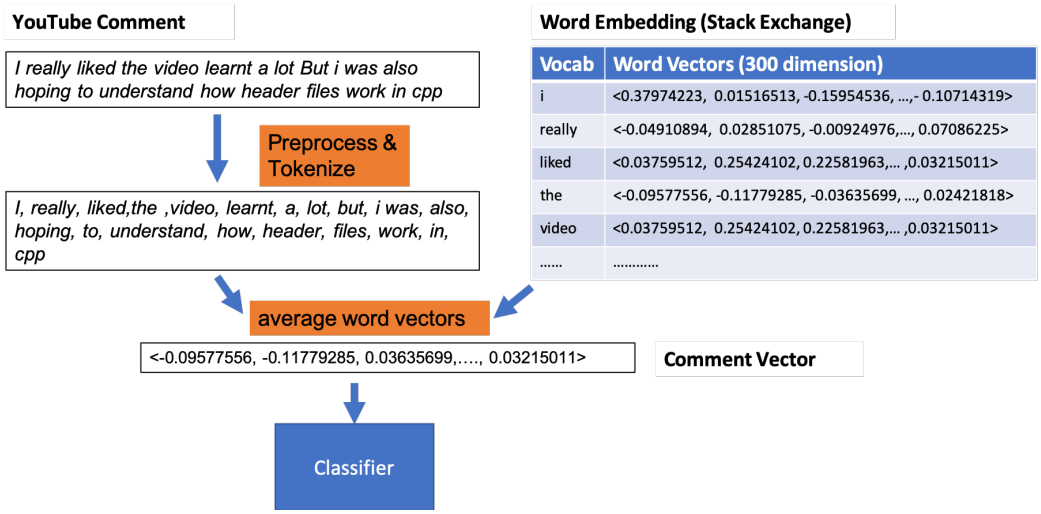

Figure 5.10: An illustration of preprocessing and creating comment vectors for classifying YouTube comments.

5.5.2 Dataset and Baselines

Dataset. We used the dataset provided by Poché et al. [70][9] The dataset consists of 6,000 YouTube comments sampled from 12 different coding tutorial videos. The data was collected on Sep 6, 2016. They collected a total of 41,773 comments from all videos. They used YouTube Data API3[10] to retrieve the comments. This API extracts comments and their metadata, including the author's name, the number of likes, and the number of replies. Finally, Poché et al. sampled 500 comments and labeled them as *content concerns* (*informative*) and *miscellaneous* (*uninformative*). Based on their manual classification process, they found around 30% of the comments to be informative, meaning that the majority of comments are basically not related to the content.

Baselines. Since we used the same dataset and experiment setting as Poché et al.'s work, we used their approach as the first baseline. To show the effectiveness of our models that are trained on software-development-specific platforms, we compared the models with a within-platform model trained from YouTube comments, and another cross-platform pretrained model that was learned from more general content.

[9]http://seel.cse.lsu.edu/data/icpc17.zip
[10]https://developers.google.com/youtube/v3/

We briefly describe the baselines as follows:

1. **Normalized Term-Frequency (as proposed by Poché et al. [70]).** In order to automatically identify content-relevant comments, Poché et al. investigate the performance of two classification algorithms: Naive Bayes (NB) and Support Vector Machines (SVM). They performed text preprocessing on the dataset, by stemming and removing stopwords. They also remove words that appear in one comment only since they are highly unlikely to carry any generalizable information. As feature representation, they use normalized term frequency (tf) of words in their documents. They found that their SVM classifier performs better than Naive Bayes. They also experimented with different combinations of data preprocessing such as stemming and removing stopwords, and found that the best result was achieved without stemming and stop-word removal.

2. **Word2Vec trained from YouTube Comments.** We consider this baseline as a within-platform baseline, since we leverage knowledge extracted from the target platform itself (i.e., YouTube comments). We built word embedding based on two datasets: (1) Use all YouTube comments available in Poché et al.'s dataset (2) Use only comments in Poché et al.'s dataset that are labeled as informative. There are 1,826 comments in the second dataset. We then trained skip-gram word embedding models using the set of parameters that we described in Section 5.3 on these datasets. We use these models to predict informative comments following the process that we have described in Section 5.3.

3. **Pretrained Word2Vec on GoogleNews.** We used a pretrained word embedding model on GoogleNews[11] which is an alternative cross-platform pretrained model as another baseline. The model contains 300-dimensional vectors for 3 million words and phrases, which was trained on part of the Google

[11] https://code.google.com/archive/p/word2vec/

News dataset (about 100 billion words).

5.5.3 Experiments and Results

Experiments Settings. We conducted experiments to answer RQ2 and evaluated the effectiveness of our approach as compared to the baselines. We used Support Vector Machines (SVM) as the classification algorithm, since this algorithm performs better in Poché et al.'s work [70]. To enable a fair comparison, we used the same implementation of SVM (inside Weka[12]) for classification. For the kernel function, in Poché et al.'s work, the best results were obtained using the universal kernel. Therefore, we also used the universal kernel in our experiment. To validate the result, we used 10-fold cross validation. With this technique, the dataset was first partitioned randomly into 10 partitions of equal size. Afterwards, one of the partitions was selected as validation set while the remaining partitions are used for training. The process was repeated 10 times with a different partition being selected as validation set, ensuring that the entire dataset was used for both training and validation, and each entry in the dataset was used for validation exactly once.

To measure the effectiveness of our approach, we used the same metrics as Poché et al.'s study (i.e., Precision, Recall and F-measure). F-measure is the harmonic mean of precision and recall, and it is used as a summary measure to evaluate if an increase in precision (recall) outweighs a reduction in recall (precision). These metrics are calculated based on four possible outcomes of each comment in an evaluation set: True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). TP corresponds to the case when a comment is correctly classified as an informative comment; FP corresponds to the case when a non-informative comment is wrongly classified as an informative comment; FN is when a comment is wrongly classified as a non-informative comment; TN is when a non-informative comment is correctly classfied as such. The formulas to compute precision, recall,

[12]https://www.cs.waikato.ac.nz/ml/weka/

and F-measure are shown below:

$$Precision = \frac{\#TP}{\#TP + \#FP}$$

$$Recall = \frac{\#TP}{\#TP + \#FN}$$

$$FMeasure = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Results. The results of our approach as compared to the baselines are shown in Table 5.4 and Figure 5.11. The results showed that the best performance (in terms of precision, recall, and F-measure) was achieved by using the word embedding model trained on StackExchange-SE data.

Table 5.4: Performance of different approaches for classifying informative comments.

Approach	Precision	Recall	F-Measure
NTF (Poché et al.)	0.790	0.750	0.770
Word2Vec (YouTube Comments - All)	0.829	0.833	0.830
Word2Vec (YouTube Comments - Relevant)	0.785	0.792	0.782
Word2Vec (GoogleNews)	0.859	0.861	0.859
SIEVE_SO	0.868	0.870	0.869
SIEVE_SE	**0.872**	**0.874**	**0.873**

The results of our approach as compared to the baselines are shown in Table 5.4 and Figure 5.11. The results showed that the best performance (in terms of precision, recall, and F-measure) was achieved by using the word embedding model trained on StackExchange-SE data.

The results also showed that by using word embedding as feature representation, the performance of the classifiers can be improved by up to 10.3% in terms of F-measure, as compared to the normalized-tf based approach proposed by Poché et al. Among the five word embedding models used in our experiment, models trained on StackExchange-SE and StackOverflow performed best with F-measure scores of 0.874 and 0.869 respectively. This finding justifies the importance of choosing a source platform that is more relevant to a target task. Even though the corpus' size is less as compared to GoogleNews data, Stack Exchange and Stack Overflow data

contains more software-development-specific content than GoogleNews, and this explains the improved performance.

In the original work by Poché et al. [70], they classified user comments into two groups: *content concerns (informative)* and *miscellaneous*. Comments that fall under the content concerns category include questions or concerns about certain parts of the video content that need further explanation, comments that point out errors within the video, comments that request for certain future content, comments that provide suggestions to improve the quality of the tutorial, and comments that suggest a change in the media settings. Therefore, a praise comments such as *Very well explained, many thanks!* is identified as a miscellaneous comment, while a comment that points out errors within the video (e.g., *At 27:10 theres a small mistake. The first parameter is the starting index, the second parameter is the number of chars*) is categorized as a content concern or informative comment. Based on our observation, most of the informative comments contain software-related terms. Therefore, the use of word embedding trained on a dataset obtained from a software-specific domain can help.

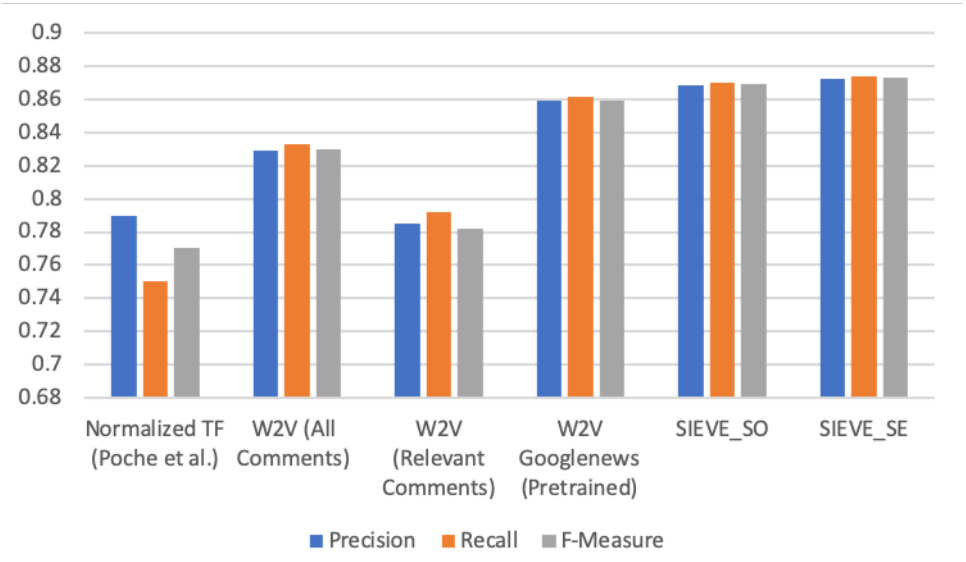

Figure 5.11: Comparison of Precision, Recall and F-measure achieved by different approaches

5.6 Threats to Validity

We present the potential threats to the validity of our findings. The threats include threats to internal, external, and construct validity.

Threats to internal validity. These threats are related to potential errors that may have occurred when performing the experiments and labeling. Internal threats might stem from the tools we used in our analysis. We used Gensim[13], a popular Python module for machine learning to build word embedding that has also been used in many previous studies related to word embedding. For machine learning and classification tools, we used Weka[14] which has been extensively used in the literature and has been shown to generate robust results for various applications. Potential errors might also occur when labelling our dataset. To label tweets as software related or not, we asked two labelers with experience in programming, and with degrees in Computer Science. We believe the labelers have enough expertise to judge if a tweet is software-related or not.

Threats to external validity. These threats refers to the generalizability of our results. To mitigate these threats, we have considered two source domains (Software Engineering Stack Exchange and Stack Overflow), two target domains (Twitter and YouTube), two tasks (relevant tweet identification and informative comment classification), and two settings (ranking and classification).

Threats to construct validity. These threats are related to the suitability of the evaluation metrics that we use for analyzing the result. We use the same evaluation metrics used to evaluate previous studies [82, 70] to enable fair comparisons (i.e., Accuracy@k, Precision, Recall and F-measure). Therefore, we believe that the threat to construct validity is minimal.

[13]https://pypi.org/project/gensim/
[14]https://www.cs.waikato.ac.nz/ml/weka

5.7 Discussion

As shown in previous sections, our approach can improve performance on finding relevant content in less-software-related platforms. However, there are some other observations worth further investigation. In this section, we will discuss these additional observations: (1) determining sample size in the task of finding relevant tweets (2) comparing different methods to learn word embedding, and (3) learning word embedding in other software-related domains.

5.7.1 Determining Sample Size from Source Platform

For the first use case, we sample a number of sentences (i.e., Software Engineering Stack Exchange post titles) to rank tweets based on a similarity measure. By default, we sample 1,000 sentences. In this section, we experiment with different sample sizes (500, 1,000, 1,500, and 2,000) and investigate their impact on the effectiveness of SIEVE. Since Section 5.4 shows that SIEVE_SE performs better than SIEVE_SO, in this experiment, we use word embedding trained on the Stack Exchange dataset.

The results of our experiment are shown in Table 5.5. From the table, we can observe that results achieved using a sample size of 1,000 are better than those achieved using a sample size of 500. This is true for Accuracy@50, Accuracy@100, Accuracy@150, and Accuracy@200. This shows that it is not advisable to reduce the sample size to be below 1,000. Moreover, from the table, we can observe that results achieved using a sample size of 1,000 are comparable with those achieved using a sample size of 1,500 or 2,000. Thus, for efficiency reason, we pick the sample size 1,000 as the default setting.

Table 5.5: Accuracy@K results for different sample sizes

sample size	acc@10	acc@50	acc@100	acc@150	acc@200
500	0.900	0.800	0.830	0.813	0.780
1000	**0**.900	0.980	0.970	0.940	0.925
1500	**0**.900	0.940	0.930	0.933	0.915
2000	0.900	0.940	0.960	0.947	0.925

5.7.2 Comparing Different Methods to Learn Word Embedding

Skip-gram [58] is a popular but not the only method to learn word embedding. There are other methods such as CBOW [58], GloVe [67] and FastText [16]. CBOW and Skip-gram were proposed by Mikolov et al. [58]; CBOW learns word embedding by trying to predict a word from its context, while Skip-gram learns word embedding by trying to predict the surrounding words (aka. context) from a word. GloVe, proposed by Pennington et al. [67], is a method for learning word embedding that leverages global count information aggregated from the entire corpus as word-word occurrence matrix. FastText, proposed by Bojanowski et al. [16], is an approach built based on the Skip-gram model, where each word is represented as a bag of character n-gram [16].

Mikolov et al. have shown that the effectiveness of the word embedding learned via CBOW and Skip-gram differ for different tasks [58]. Also, Pennington et al. [67] and Bojanowski et al. [16] have shown the power of GloVe and FastText over CBOW and Skip-gram. In the previous sections, we have explored the power of the Skip-gram method for two use cases, i.e., finding software-related tweets for developer learning (Section 5.4) and identifying informative comments for improving software development video tutorials (Section 5.5). Here, we want to investigate whether the performance of word embedding learned via the four methods (CBOW, Skip-gram, GloVe and FastText) differ for the two use cases. If they differ, we want to know the best one for each of the use cases.

To evaluate the effectiveness of the four methods for the first use case, we followed the setting described in Section 5.4 and repeated the experiment four times using different methods to learn word embedding. The results of our experiments for the first use case are shown in Table 5.6. From the table, we can observe that all methods perform equally well for acc@10. For acc@50, Skip-gram performed the best but its result is only marginally better than those of the other methods (differences of 0.02-0.04). For acc@100, Skip-gram, CBOW, and FastText achieved the

best performance. For acc@150, Skip-gram performed the best but again its result is only marginally better than those of the other methods (differences of 0.007-0.047). For acc@200, Skip-gram performed the best too, but the differences are again minor (differences of 0.005-0.040).

Table 5.6: Accuracy@K results of different word embedding learning methods for the task of finding software-relevant tweets.

Models	**acc@10**	**acc@50**	**acc@100**	**acc@150**	**acc@200**
GloVe	1.000	0.960	0.920	0.933	0.935
FastText	**1**.000	0.980	0.990	0.973	0.970
CBOW	**1**.000	0.980	0.990	0.973	0.970
Skip-gram	1.000	1.000	0.990	0.980	0.975

Table 5.7: Performance of different word embedding learning methods for classifying YouTube comments.

Models	**Precision**	**Recall**	**F-Measure**
GloVe	0.858	0.861	0.858
FastText	0.868	0.869	0.868
CBOW	0.860	0.862	0.828
Skip-gram	0.872	0.874	0.873

To evaluate the effectiveness of the four methods for the second use case, we followed the setting described in Section 5.5 and repeated the experiment four times using different methods to learn word embedding. The results of our experiments for the second use case are shown in Table 5.7. From the table, we can observe that the F-measures achieved by the four methods are similar (0.828-0.873) with the best results achieved by Skip-gram.

From the above results, we would like to recommend the use of the Skip-gram method for cross-domain tasks where data from source platforms which contain rich domain-related content (e.g., Stack Overflow and Software Engineering Stack Exchange), are used to solve tasks in other platforms with less domain-related content (e.g., Twitter and YouTube). However, the performance differences between Skip-gram and the other methods are not that large and other considerations (e.g., runtime efficiency, etc.) may need to be considered to determine one's choice of the

method to be used. CBOW for example has been shown to be more efficient than Skip-gram [58].

5.7.3 Learning Word Embedding in another Software-Related Domain

We have shown that using Stack Overflow and Software Engineering Stack Exchange as source platforms achieved promising results in the use cases described in Section 5.4 and Section 5.5. Here, we want to investigate whether we can use another software-related platform as source platform. We chose HackerNews, a social news website that focuses on technology news. We used the HackerNews dataset provided by Aniche et al. [5] The dataset consist of 530,446 posts.

First, we learn a Word2Vec model using the Skip-gram method from the dataset, following the settings described in Section 5.3. The results of our experiments for the task of finding software-relevant tweets are shown in Table 5.8. Overall, the accuracy@K results obtained from the HackerNews dataset are comparable to the results obtained from the Stack Exchange dataset, and better than those obtained from the Stack Overflow dataset.

Table 5.8: Accuracy@K results of different source platforms for the task of finding software-relevant tweets.

Platform	**acc@10**	**acc@50**	**acc@100**	**acc@150**	**acc@200**
`HackerNews`	0.900	0.980	0.970	0.980	0.980
`Stack Exchange`	1.000	1.000	0.990	0.980	0.975
`Stack Overflow`	0.900	0.880	0.870	0.847	0.800

Table 5.9: Performance of different source platforms for classifying YouTube comments as Informative/not Informative

Platform	**Precision**	**Recall**	**F-Measure**
`HackerNews`	0.860	0.862	0.860
`Stack Exchange`	0.872	0.874	0.873
`Stack Overflow`	0.868	0.870	0.869

To evaluate the effectiveness of the HackerNews dataset for the second use case,

we followed the settings described in Section 5.5. The results of our experiment for classifying YouTube comments are shown in Table 9. We can observe from the table that the F-measures achieved by all source platforms are comparable (0.860 - 0.873) with the best results achieved by Stack Exchange. These findings show that Stack Overflow and Stack Exchange platforms provide sufficiently rich datasets that are as effective as a dataset that is obtained from HackerNews.

5.8 Chapter Conclusion

We proposed an approach to exploit knowledge from rich software-development-specific platforms, to automate knowledge seeking tasks in other less software-development-specific platforms. We first built word embedding from text extracted from Stack Overflow and Software Engineering Stack Exchange, to represent software-development-related knowledge sources. We then leveraged the word embedding to solve tasks in two different target platforms. In the first use case, we leveraged the word embedding and sampled sentences from source platforms, to find software-related tweets. In the second use case, we used the word embedding to classify informative comments on YouTube video tutorials. Based on our experiments conducted in both use cases, our approach improves performance of existing state-of-the-art work for software-development-specific knowledge extraction tasks in the target platforms.

In the future, we intend to perform additional experiments to evaluate the effectiveness of the approach for additional tasks. We plan to expand the work to other platforms and knowledge sources, such as Wikipedia articles, software development blogs, README files on GitHub, and software documentation. We also plan to apply SIEVE at a finer-grained categories (i.e., mobile, big data etc.).

Chapter 6

Conclusion and Future Work

6.1 Summary

In this dissertation, we demonstrate the usefulness of machine learning-based approaches in order to uncover insight derived from crowd generated content in various settings. Specifically, this dissertation covers the following use cases: predicting customers' complaint patterns on Twitter, inferring spread of readers' emotion on online news articles, finding software-related tweets and identifying informative comments on YouTube. Based on approaches proposed in the use cases explored in this dissertation, we summarize our findings on developing machine-learning based solution on making sense of crowd-generated content, as follows:

1. The approaches mostly depend on the *what, why* and *where* aspects (the tasks, the objective of the tasks, and the domains/platforms used), rather than the *who* (individuals/organization that get benefit of the data), and *when* (immediate or batch processing) aspects.

2. Domain-specific data are beneficial for all cases, either *within-platform* analysis (for the first and the second use cases) or *cross-platform* analysis (for the third use case). Therefore, whenever possible, collecting domain-specific/task-specific data should be accomplished before developing ma-

chine learning based approach.

A summary of contributions from completed works is described below:

Predicting Customer Complaint Behavior on Twitter

This study would be important towards the development of techniques that make use of social media data to improve product and service quality. This work proposes a new problem of predicting different types of customer feedback behavior on Twitter. Different types of customer feedback behavior were identified by using a clustering algorithm. A set of features was proposed, i.e. content features and profile features, that can be used to predict customer feedback behavior by leveraging a supervised machine learning algorithm to build a prediction model. The approach also has been evaluated on a dataset containing 11,809 tweets. The result shows that the proposed approaches can achieve reasonable precision, recall and F-measure which are higher than those of a baseline approach.

Inferring Spread of Readers Emotion

In this work, lexicon-based and word-vector-based features are used to build a regression model which aim to predict readers' *emotion distribution*. A case study has been conducted, by using a popular Indonesian on-line news site, namely *detik.com*. A new corpus consisting of Indonesian news articles was created for predicting readers' emotion distribution affected by news articles. Experiments were conducted to compare the effectiveness of using different parts of news articles (headlines only, content only, and both headlines and content) to predict the spread of readers' emotion. The effectiveness of a domain-specific emotion lexicon and word embedding with general purpose lexicon and word embedding were also compared for the problem of predicting emotion distribution of a news article readers. By knowing the predicted emotion distribution, the publishers can get a deeper insight on likely readers' responses, e.g., estimated proportion of readers who are happy with a piece of news.

Finding Software-relevant Content for Software Developers

This work proposed SIEVE, an approach to leverage knowledge extracted from rich software-development-specific platforms, to automate knowledge seeking tasks in other less software-development-specific platforms. We first built word embedding from text extracted from software-development-related platforms (i.e., Stack Overflow and Software Engineering Stack Exchange). We then leveraged the word embedding to solve tasks in two different use cases. In the first use case, we apply our approach on the task of finding software related tweets. The approach was able to improve performance by up to 28% in terms of accuracy@K, compared to state of the art approach. For the second use case, the task was to identify informative comments on YouTube coding tutorial videos. In this task, SIEVE can improve the performance of an existing baseline of up to 10.3% in terms of F-measure.

6.2 Future Directions

Combining internal data and crowd-generated data

We have demonstrated an approach that makes use of external data (i.e, customer's tweets) to profile customers based on their tweeting behavior. However, companies often also use internal data to better understand their customers. Such data are maintained in the company's customer relationship management systems. In order to get the whole picture of the customers, it is necessary to combine all possible data related to the customers in order to better manage them. Further research can be conducted to combine internal and external data to profile the customers. Another study can be conducted on predicting the customer complaint behavior based on these combined data, such as predicting customer complaint behavior based their past interaction through any kind of social media platforms (i.e, Twitter, Facebook, user forums), or through a company's internal system (email, company's website, or call center).

Enhancing Word Embedding for Domain-Specific Setting

We have shown that word embedding is useful as a knowledge representation in the task of inferring customers' reaction based on a specific article, or finding software-specific information on various platforms. We also experimented with several learning methods to create word embedding (i,.e., Word2Vec, GloVe, and FastText). Based on the use cases studied in Chapter 4 and Chapter 5, we found that word embedding learned from domain-specific content generally performs better as compared to pre-trained word embedding that are learned from Google News, Wikipedia, etc. Several methods have been proposed to improve the quality of the pre-trained word embedding models, so that it will be applicable for other domain-specific tasks. One way to improve the word embedding model is to apply a post-processing step which is called *retrofitting*, proposed by Faruqui et al [35]. This method refines existing pretrained word vectors using relational information from semantic lexicons by encouraging linked words to have similar vector representations. Another work by Howard and Ruder [43] proposed ULMFiT (Universal Language Model Fine-tuning), which explores the benefits of using a pre-trained model on text classification. In the proposed method, a pre-trained word embedding is fine-tuned with task-specific data to make the model parameters adapt to the target task. Recently, Peters et al. proposed ELMo (Embeddings from Language Models) [68], an improved representation of word embedding. Unlike traditional word embedding methods, ELMo is dynamic, meaning that ELMo embeddings change depending on the context even when the word is the same. Further studies can be conducted on whether these methods are able to improve the performance achieved in various tasks that are discussed in this dissertation.

Bibliography

[1] A. Abdaoui, J. Azé, S. Bringay, N. Grabar, and P. Poncelet. Expertise in french health forums. *Health Informatics Journal*, page 1460458216682356, 2016.

[2] P. Achananuparp, I. N. Lubis, Y. Tian, D. Lo, and E.-P. Lim. Observatory of trends in software related microblogs. In *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 334–337. IEEE, 2012.

[3] J. Ajmera, H.-i. Ahn, M. Nagarajan, A. Verma, D. Contractor, S. Dill, and M. Denesuk. A crm system for social media: challenges and experiences. In *Proceedings of the 22nd international conference on World Wide Web*, pages 49–58. International World Wide Web Conferences Steering Committee, 2013.

[4] J. T. Andrews, T. Tanay, E. J. Morton, and L. D. Griffin. Transfer representation-learning for anomaly detection. ICML, 2016.

[5] M. Aniche, C. Treude, I. Steinmacher, I. Wiese, G. Pinto, M.-A. Storey, and M. A. Gerosa. How modern news aggregators help development communities shape and share knowledge. In *ICSE*, 2018.

[6] I. Arapakis, B. B. Cambazoglu, and M. Lalmas. On the feasibility of predicting news popularity at cold start. In *International Conference on Social Informatics*, pages 290–299. Springer, 2014.

[7] S. Azad, P. C. Rigby, and L. Guerrouj. Generating api call rules from version history and stack overflow posts. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 25(4):29, 2017.

[8] A. Bacchelli, T. Dal Sasso, M. D'Ambros, and M. Lanza. Content classification of development emails. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 375–385. IEEE, 2012.

[9] A. Balahur, R. Steinberger, M. Kabadjov, V. Zavarella, E. Van Der Goot, M. Halkia, B. Pouliquen, and J. Belyaeva. Sentiment analysis in the news. *arXiv preprint arXiv:1309.6202*, 2013.

[10] A. Bandhakavi, N. Wiratunga, S. Massie, and D. Padmanabhan. Lexicon generation for emotion detection from text. *IEEE intelligent systems*, 32(1):102–108, 2017.

[11] A. Barua, S. W. Thomas, and A. E. Hassan. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering*, 19(3):619–654, 2014.

[12] A. Begel, J. Bosch, and M.-A. Storey. Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder. *IEEE Software*, 2013.

[13] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[14] S. Bhatia, J. Li, W. Peng, and T. Sun. Monitoring and analyzing customer feedback through social media platforms for identifying and remedying customer problems. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 1147–1154. ACM, 2013.

[15] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[16] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[17] G. Bougie, J. Starke, M.-A. Storey, and D. M. German. Towards understanding twitter use in software engineering: preliminary findings, ongoing challenges and future questions. In *Proceedings of the 2nd international workshop on Web 2.0 for software engineering*, pages 31–36, 2011.

[18] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli. Sentiment polarity detection for software development. *Empirical Software Engineering*, pages 1–31, 2017.

[19] E. Cambria and A. Hussain. *Sentic computing: Techniques, tools, and applications*, volume 2. Springer Science & Business Media, 2012.

[20] E. Cambria, S. Poria, R. Bajpai, and B. W. Schuller. Senticnet 4: A semantic resource for sentiment analysis based on conceptual primitives. In *COLING*, pages 2666–2677, 2016.

[21] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168. ACM, 2006.

[22] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, pages 321–357, 2002.

[23] C. Chen, S. Gao, and Z. Xing. Mining analogical libraries in q&a discussions–incorporating relational and categorical knowledge into word embedding. In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, volume 1, pages 338–348. IEEE, 2016.

[24] C. Chen and Z. Xing. Similartech: automatically recommend analogical libraries across different programming languages. In *Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on*, pages 834–839. IEEE, 2016.

[25] C. Chen, Z. Xing, and Y. Liu. By the community & for the community: A deep learning approach to assist collaborative editing in q&a sites. In *Proceedings of the 21st ACM Conference on Computer-Supported Cooperative Work and Social Computing*, pages 32:1–32:21. ACM, 2018.

[26] C. Chen, Z. Xing, and X. Wang. Unsupervised software-specific morphological forms inference from informal discussions. In *Proceedings of the 39th International Conference on Software Engineering*, pages 450–461. IEEE Press, 2017.

[27] G. Chen, C. Chen, Z. Xing, and B. Xu. Learning a dual-language vector space for domain-specific cross-lingual question retrieval. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 744–755. ACM, 2016.

[28] J. Chen, A. Cypher, C. Drews, and J. Nichols. Crowde: Filtering tweets for direct customer engagements. In *ICWSM*. Citeseer, 2013.

[29] R. J. Chenail. Youtube as a qualitative research asset: Reviewing user generated videos as learning resources. *The Qualitative Report*, 13(3):18–24, 2008.

[30] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella. Labeling source code with information retrieval methods: an empirical study. *Empirical Software Engineering*, 19(5):1383–1420, 2014.

[31] H. Dev, M. E. Ali, J. Mahmud, T. Sen, M. Basak, and R. Paul. A real-time crowd-powered testbed for content assessment of potential social media posts. In *International Conference on Social Informatics*, pages 136–152. Springer, 2015.

[32] S. Diplaris, A. Sonnenbichler, T. Kaczanowski, P. Mylonas, A. Scherp, M. Janik, S. Papadopoulos, M. Ovelgoenne, and Y. Kompatsiaris. Emerging, collective intelligence for personal, organisational and social use. In *Next generation data technologies for collective computational intelligence*, pages 527–573. Springer, 2011.

[33] S. A. Einwiller and S. Steilen. Handling complaints on social network sites–an analysis of complaints and complaint responses on facebook and twitter pages of large us companies. *Public Relations Review*, 41(2):195–204, 2015.

[34] M. El Mezouar, F. Zhang, and Y. Zou. Are tweets useful in the bug fixing process? an empirical study on firefox and chrome. *Empirical Software Engineering*, 23(3):1704–1742, 2018.

[35] M. Faruqui, J. Dodge, S. K. Jauhar, C. Dyer, E. Hovy, and N. A. Smith. Retrofitting word vectors to semantic lexicons. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1606–1615, 2015.

[36] J. Guo, J. Cheng, and J. Cleland-Huang. Semantically enhanced software traceability using deep learning techniques. In *Software Engineering (ICSE), 2017 IEEE/ACM 39th International Conference on*, pages 3–14. IEEE, 2017.

[37] E. Guzman, R. Alkadhi, and N. Seyff. A needle in a haystack: What do twitter users say about software? In *Requirements Engineering Conference (RE), 2016 IEEE 24th International*, pages 96–105. IEEE, 2016.

[38] E. Guzman, R. Alkadhi, and N. Seyff. An exploratory study of twitter messages about software applications. *Requirements Engineering*, 22(3):387–412, 2017.

[39] E. Guzman, M. Ibrahim, and M. Glinz. A little bird told me: mining tweets for requirements and software evolution. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 11–20. IEEE, 2017.

[40] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[41] Z. Han, X. Li, H. Liu, Z. Xing, and Z. Feng. Deepweak: Reasoning common software weaknesses via knowledge graph embedding. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 456–466. IEEE, 2018.

[42] A. Hindle and C. Onuczko. Preventing duplicate bug reports by continuously querying bug reports. *Empirical Software Engineering*, 24(2):902–936, 2019.

[43] J. Howard and S. Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, 2018.

[44] Y.-L. Hsieh, Y.-C. Chang, C.-H. Chu, and W.-L. Hsu. How do i look? publicity mining from distributed keyword representation of socially infused news articles. In *Conference on Empirical Methods in Natural Language Processing*, page 74, 2016.

[45] P. J. Huang. *Classication of Imbalanced Data Using Synthetic Over-Sampling Techniques*. PhD thesis, UCLA, 2015.

[46] D. Kedia and V. Gupta. Identifying suggestions for improvement of product features from online product reviews. In *Social Informatics: 7th International Conference, SocInfo 2015, Beijing, China, December 9-12, 2015, Proceedings*, volume 9471, page 112. Springer, 2015.

[47] T. Kenter and M. De Rijke. Short text similarity with word embeddings. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 1411–1420. ACM, 2015.

[48] H. Kwak, C. Lee, H. Park, and S. B. Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 591–600, 2010.

[49] S. Lai, K. Liu, S. He, and J. Zhao. How to generate a good word embedding. *IEEE Intelligent Systems*, 31(6):5–14, 2016.

[50] J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.

[51] J. Lei, Y. Rao, Q. Li, X. Quan, and L. Wenyin. Towards building a social emotion detection system for online news. *Future Generation Computer Systems*, 37:438–448, 2014.

[52] K. H.-Y. Lin and H.-H. Chen. Ranking reader emotions using pairwise loss minimization and emotional distribution regression. In *Proceedings of the conference on empirical methods in natural language processing*, pages 136–144. Association for Computational Linguistics, 2008.

[53] K. H.-Y. Lin, C. Yang, and H.-H. Chen. Emotion classification of online news articles from the reader's perspective. In *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on*, volume 1, pages 220–226. IEEE, 2008.

[54] W. Maalej, R. Tiarks, T. Roehm, and R. Koschke. On the comprehension of program comprehension. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 23(4):31, 2014.

[55] L. MacLeod, A. Bergen, and M.-A. Storey. Documenting and sharing software knowledge using screencasts. volume 22, pages 1478–1507. Springer, 2017.

[56] L. MacLeod, M.-A. Storey, and A. Bergen. Code, camera, action: how software developers document and share program knowledge using youtube. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension*, pages 104–114. IEEE Press, 2015.

[57] M. Meilă and D. Heckerman. An experimental comparison of several clustering and initialization methods. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 386–395. Morgan Kaufmann Publishers Inc., 1998.

[58] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[59] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[60] N. J. Millard. Serving the social customer: How to look good on the social dance floor. In *HCI in Business*, pages 165–174. Springer, 2015.

[61] S. M. Mohammad and P. D. Turney. Crowdsourcing a word–emotion association lexicon. *Computational Intelligence*, 29(3):436–465, 2013.

[62] H. N. Nguyen, T. Van Le, H. S. Le, and T. V. Pham. Domain specific sentiment dictionary for opinion mining of vietnamese text. In *International Workshop on Multidisciplinary Trends in Artificial Intelligence*, pages 136–148. Springer, 2014.

[63] B.-W. On, E.-P. Lim, J. Jiang, A. Purandare, and L.-N. Teow. Mining interaction behaviors for email reply order prediction. In *Advances in Social Networks Analysis and Mining (ASONAM), 2010 International Conference on*, pages 306–310. IEEE, 2010.

[64] F. Palomba, A. Panichella, A. De Lucia, R. Oliveto, and A. Zaidman. A textual-based technique for smell detection. In *2016 IEEE 24th international conference on program comprehension (ICPC)*, pages 1–10. IEEE, 2016.

[65] S. J. Pan, Q. Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[66] M. Pennacchiotti and A.-M. Popescu. A machine learning approach to twitter user classification. *ICWSM*, 11(1):281–288, 2011.

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

[68] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[69] P. Pirolli. Powers of 10: Modeling complex information-seeking systems at multiple scales. *Computer*, 42(3):33–40, 2009.

[70] E. Poché, N. Jha, G. Williams, J. Staten, M. Vesper, and A. Mahmoud. Analyzing user comments on youtube coding tutorial videos. In *Proceedings of the 25th International Conference on Program Comprehension*, pages 196–206. IEEE Press, 2017.

[71] L. Ponzanelli, G. Bavota, A. Mocci, M. Di Penta, R. Oliveto, M. Hasan, B. Russo, S. Haiduc, and M. Lanza. Too long; didn't watch!: extracting relevant fragments from software development video tutorials. In *Proceedings of the 38th International Conference on Software Engineering*, pages 261–272. ACM, 2016.

[72] L. Ponzanelli, G. Bavota, A. Mocci, M. Di Penta, R. Oliveto, B. Russo, S. Haiduc, and M. Lanza. Codetube: extracting relevant fragments from software development video tutorials. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 645–648. ACM, 2016.

[73] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[74] D. Posnett, E. Warburg, P. Devanbu, and V. Filkov. Mining stack exchange: Expertise is evident from initial contributions. In *Social Informatics (SocialInformatics), 2012 International Conference on*, pages 199–204. IEEE, 2012.

[75] P. K. Prasetyo, D. Lo, P. Achananuparp, Y. Tian, and E.-P. Lim. Automatic classification of software related microblogs. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 596–599. IEEE, 2012.

[76] M. M. Rahman and C. K. Roy. Strict: Information retrieval based search term identification for concept location. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 79–90. IEEE, 2017.

[77] S. Rangnani, V. S. Devi, and M. N. Murty. Autoregressive model for users retweeting profiles. In *International Conference on Social Informatics*, pages 178–193. Springer, 2015.

[78] Y. Rao, J. Lei, L. Wenyin, Q. Li, and M. Chen. Building emotional dictionary for sentiment analysis of online news. *World Wide Web*, 17(4):723–742, 2014.

[79] R. Rehurek and P. Sojka. Software framework for topic modelling with large corpora. In *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Citeseer, 2010.

[80] M. Sahlgren and R. Cöster. Using bag-of-concepts to improve the performance of support vector machines in text categorization. In *Proceedings of the 20th international conference on Computational Linguistics*, page 487. Association for Computational Linguistics, 2004.

[81] T. Semwal, P. Yenigalla, G. Mathur, and S. B. Nair. A practitioners' guide to transfer learning for text classification using convolutional neural networks. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, pages 513–521. SIAM, 2018.

[82] A. Sharma, Y. Tian, and D. Lo. Nirmal: Automatic identification of software relevant tweets leveraging language model. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*, pages 449–458. IEEE, 2015.

[83] A. Sharma, Y. Tian, and D. Lo. What's hot in software engineering twitter space? In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, pages 541–545. IEEE, 2015.

[84] A. Sharma, Y. Tian, A. Sulistya, D. Lo, and A. F. Yamashita. Harnessing twitter to support serendipitous learning of developers. In *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*, pages 387–391. IEEE, 2017.

[85] L. Singer, F. Figueira Filho, and M.-A. Storey. Software engineering at the speed of light: how developers stay current using twitter. In *Proceedings of the 36th International Conference on Software Engineering*, pages 211–221. ACM, 2014.

[86] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[87] StackExchange. About software engineering stack exchange. [Online; accessed 16-April-2019].

[88] J. Staiano and M. Guerini. Depechemood: A lexicon for emotion analysis from crowd-annotated news. *arXiv preprint arXiv:1405.1605*, 2014.

[89] A. Stolcke. Srilm-an extensible language modeling toolkit. In *Seventh international conference on spoken language processing*, 2002.

[90] M.-A. Storey, L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky. The (r) evolution of social media in software engineering. In *Proceedings of the on Future of Software Engineering*, pages 100–116. ACM, 2014.

[91] M.-A. Storey, A. Zagalsky, L. Singer, D. German, et al. How social and communication channels shape and challenge a participatory culture in software development. *IEEE Transactions on Software Engineering*, (1):1–1, 2017.

[92] C. Strapparava and R. Mihalcea. Semeval-2007 task 14: Affective text. In *Proceedings of the 4th International Workshop on Semantic Evaluations*, pages 70–74. Association for Computational Linguistics, 2007.

[93] C. Strapparava, A. Valitutti, et al. Wordnet affect: an affective extension of wordnet. In *LREC*, volume 4, pages 1083–1086, 2004.

[94] Y. Tian, P. Achananuparp, I. N. Lubis, D. Lo, and E.-P. Lim. What does software engineering community microblog about? In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 247–250. IEEE, 2012.

[95] Y. Tian and D. Lo. An exploratory study on software microblogger behaviors. In *2014 IEEE 4th Workshop on Mining Unstructured Data*, pages 1–5. IEEE, 2014.

[96] Y. Tian, D. Lo, X. Xia, and C. Sun. Automated prediction of bug report priority using multi-factor analysis. *Empirical Software Engineering*, 20(5):1354–1383, 2015.

[97] T. Van Nguyen, A. T. Nguyen, H. D. Phan, T. D. Nguyen, and T. N. Nguyen. Combining word2vec with revised vector space model for better code retrieval. In *Proceedings of the 39th International Conference on Software Engineering Companion*, pages 183–185. IEEE Press, 2017.

[98] V. Van Vlasselaer, T. Eliassi-Rad, L. Akoglu, M. Snoeck, and B. Baesens. Afraid: Fraud detection via active inference in time-evolving social networks. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, pages 659–666. ACM, 2015.

[99] X. Wang, I. Kuzmickaja, K.-J. Stol, P. Abrahamsson, and B. Fitzgerald. Microblogging in open source software development: The case of drupal and twitter. *Software, IEEE*, 2013.

[100] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

[101] G. Williams and A. Mahmoud. Mining twitter feeds for software user requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 1–10. IEEE, 2017.

[102] X. Xia, L. Bao, D. Lo, P. S. Kochhar, A. E. Hassan, and Z. Xing. What do developers search for on the web? *Empirical Software Engineering*, 22(6):3149–3185, 2017.

[103] X. Xia, D. Lo, E. Shihab, X. Wang, and X. Yang. Elblocker: Predicting blocking bugs with ensemble imbalance learning. *Information and Software Technology*, 61:93–106, 2015.

[104] B. Xu, Z. Xing, X. Xia, D. Lo, Q. Wang, and S. Li. Domain-specific cross-language relevant question retrieval. In *Proceedings of the 13th International Conference on Mining Software Repositories*, pages 413–424. ACM, 2016.

[105] C. Xu, X. Sun, B. Li, X. Lu, and H. Guo. Mulapi: Improving api method recommendation with api usage location. *Journal of Systems and Software*, 142:195–205, 2018.

[106] S. Yadid and E. Yahav. Extracting code from programming tutorial videos. In *Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pages 98–111. ACM, 2016.

[107] X. Yang, D. Lo, X. Xia, L. Bao, and J. Sun. Combining word embedding with information retrieval to recommend similar bug reports. In *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on*, pages 127–137. IEEE, 2016.

[108] D. Ye, Z. Xing, and N. Kapre. The structure and dynamics of knowledge network in domain-specific q&a sites: a case study of stack overflow. *Empirical Software Engineering*, 22(1):375–406, 2017.

[109] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu. From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th international conference on software engineering*, pages 404–415. ACM, 2016.

[110] YouTube. Youtube. 2017. [Online; accessed 20-AUG-2018].

[111] J. Zhang, H. Jiang, Z. Ren, and X. Chen. Recommending apis for api related questions in stack overflow. *IEEE Access*, 6:6205–6219, 2018.

[112] Y. Zhang, L. Su, Z. Yang, X. Zhao, and X. Yuan. Multi-label emotion tagging for online news by supervised topic model. In *Asia-Pacific Web Conference*, pages 67–79. Springer, 2015.

[113] T. Zhao, Q. Cao, and Q. Sun. An improved approach to traceability recovery based on word embeddings. In *Asia-Pacific Software Engineering Conference (APSEC), 2017 24th*, pages 81–89. IEEE, 2017.