Empirical Software Engineering
https://doi.org/10.1007/s10664-019-09775-w

SIEVE: Helping developers sift wheat from chaff via cross-platform analysis

Agus Sulistya[1] · Gede Artha Azriadi Prana[2] · Abhishek Sharma[2] · David Lo[2] · Christoph Treude[3]

Published online: 02 October 2019

Abstract

Software developers have benefited from various sources of knowledge such as forums, question-and-answer sites, and social media platforms to help them in various tasks. Extracting software-related knowledge from different platforms involves many challenges. In this paper, we propose an approach to improve the effectiveness of knowledge extraction tasks by performing cross-platform analysis. Our approach is based on transfer representation learning and word embedding, leveraging information extracted from a source platform which contains rich domain-related content. The information extracted is then used 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 identifying software-related tweets and helpful YouTube comments.

Keywords Word embedding · Transfer representation learning · Software engineering

Communicated by: Denys Poshyvanyk

✉ Agus Sulistya
sulistya@telkom.co.id

Gede Artha Azriadi Prana
arthaprana.2016@smu.edu.sg

Abhishek Sharma
abhisheks@smu.edu.sg

David Lo
davidlo@smu.edu.sg

Christoph Treude
christoph.treude@adelaide.edu.au

[1] PT Telkom Indonesia, Jakarta, Indonesia

[2] Singapore Management University, Singapore, Singapore

[3] University of Adelaide, Adelaide, Australia

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. (2014) 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, Xia et al. (2017) 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 is the microblogging platform Twitter, which is used by a large number of software developers to support their professional activities, e.g., to share and obtain the latest technical news (Singer et al. 2014).

Aside from their activity to seek knowledge in various platforms, many software developers also share their experiences or knowledge through forums and social media. A study by MacLeod et al. (2015) 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 site's 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. (2014) 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 (downloaded from the Twitter accounts of 100 well-known software developers and their followers) found that only 16.7% of the tweets are software related. 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 (Williams and Mahmoud 2017), early detection of software issues (El Mezouar et al. 2018), or topic / trend detection (Sharma et al. 2015b).

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. (2017), 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. (2017) 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 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. We consider two platforms, Software Engineering Stack Exchange and Stack Overflow, as the rich domain-related platforms. We build word embeddings based on the dataset collected from these platforms. We then leverage the word embeddings 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 (Sharma et al. 2015a), and classifying informative comments for software engineering video tutorials on YouTube (Poché et al. 2017). We conducted experiments based on the existing datasets (as described in Table 2) provided by Sharma et al. (2015a) for Twitter, and Poché et al. (2017) 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 embeddings 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 paper are structured as follows. In Section 2, we describe background related to knowledge channels for software developers, and background on representation learning and word embedding. In Section 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 4. Next, we present the second use case on classifying informative comments on YouTube in Section 5. In Section 6, we discuss findings that are relevant to our approach. Threats to validity are discussed in Section 7. We describe related work in Section 8. Finally, we conclude and mention future work in Section 9.

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.

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 (Storey et al. 2014, 2017). 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 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., Begel and Bosch (2013) and Posnett et al. (2012). 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

[1] https://stackexchange.com/

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

[3] https://stackoverflow.com/

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 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 (Barua et al. 2014), generate API call rules (Azad et al. 2017), explore knowledge networks (Ye et al. 2017), build information filtering models (Sharma et al. 2015a) etc. More related work is discussed in detail in Section 8.

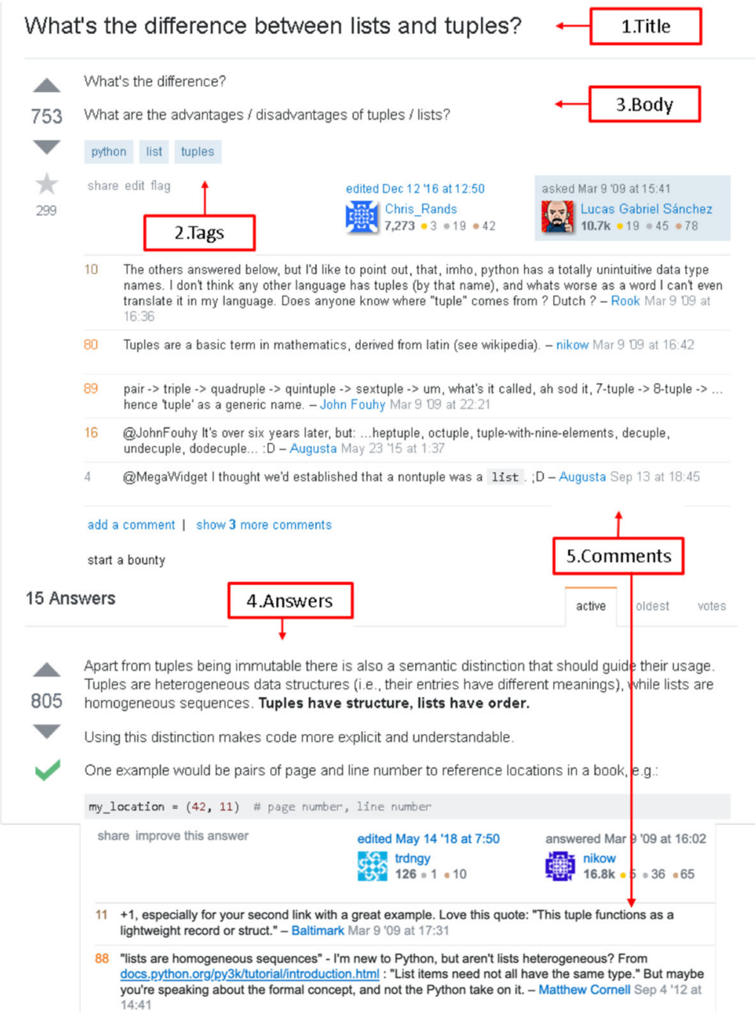
Figure 1 shows a sample question-and-answer thread from Stack Overflow. Each thread generally contains five types of information: title, tags, body, answers, and comments. The title of a thread is a summary of the question asked. The tags represent the metadata related to the question being asked and 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 touch with their family and friends (Kwak et al. 2010). Twitter influences many communities including the software engineering community as highlighted by many prior studies (Singer et al. 2014; Bougie et al. 2011; Wang et al. 2013; Tian et al. 2012). Various techniques have been proposed recently to mine software engineering relevant information from Twitter (Sharma et al. 2015a, c; Williams and Mahmoud 2017; Guzman et al. 2017b).

YouTube and Software Engineering: YouTube is a website where anybody can share videos (Chenail 2008). It has over 1 billion users and generates billions of views daily (YouTube 2017). 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.

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



What's the difference between lists and tuples?

What's the difference?

What are the advantages / disadvantages of tuples / lists?

10 The others answered below, but I'd like to point out, that, imho, python has a totally unintuitive data type names. I don't think any other language has tuples (by that name), and whats worse as a word I can't even translate it in my language. Does anyone know where "tuple" comes from ? Dutch ? – Rook Mar 9 '09 at 16:36

80 Tuples are a basic term in mathematics, derived from latin (see wikipedia). – nikow Mar 9 '09 at 16:42

89 pair -> triple -> quadruple -> quintuple -> sextuple -> um, what's it called, ah sod it, 7-tuple -> 8-tuple -> ... hence 'tuple' as a generic name. – John Fouhy Mar 9 '09 at 22:21

16 @JohnFouhy It's over six years later, but: ...heptuple, octuple, tuple-with-nine-elements, decuple, undecuple, dodecuple... :D – Augusta May 23 '15 at 1:37

4 @MegaWidget I thought we'd established that a nontuple was a `list`. ;D – Augusta Sep 13 at 18:45

15 Answers

Apart from tuples being immutable there is also a semantic distinction that should guide their usage. Tuples are heterogeneous data structures (i.e., their entries have different meanings), while lists are homogeneous sequences. **Tuples have structure, lists have order.**

Using this distinction makes code more explicit and understandable.

One example would be pairs of page and line number to reference locations in a book, e.g.:

```
my_location = (42, 11)  # page number, line number
```

11 +1, especially for your second link with a great example. Love this quote: "This tuple functions as a lightweight record or struct." – Baltimark Mar 9 '09 at 17:31

88 "lists are homogeneous sequences" - I'm new to Python, but aren't lists heterogeneous? From docs.python.org/py3k/tutorial/introduction.html : "List items need not all have the same type." But maybe you're speaking about the formal concept, and not the Python take on it. – Matthew Cornell Sep 4 '12 at 14:41

Fig. 1 A sample question-answer-thread on Stack Overflow with tags (Thread ID 626759)

Software developers also use YouTube for sharing information as well as learning (MacLeod et al. 2015; Ponzanelli et al. 2016a). MacLeod et al. found that developers share videos detailing information they wished they had found earlier (MacLeod et al. 2015). 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 (Ponzanelli et al. 2016b; Yadid and Yahav 2016). Poché et al. analyzed user comments related to software engineering videos posted on YouTube (Poché et al. 2017) and proposed a technique for finding relevant comments.

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 (Pan et al. 2010). 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 (Andrews et al. 2016). 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 (Bengio and Courville 2013). 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 (Mikolov et al. 2013a; Pennington and Socher 2014). One of the most popular word embedding techniques is Word2Vec, which uses a shallow neural network to reconstruct contexts of words. Mikolov et al. (2013a, b) 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 (Semwal et al. 2018). 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).

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.

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

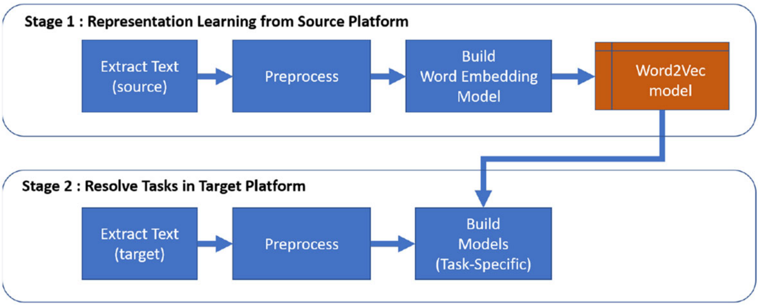

Stage 1 : Representation Learning from Source Platform
Extract Text (source)
Preprocess
Build Word Embedding Model
Word2Vec model
Stage 2 : Resolve Tasks in Target Platform
Extract Text (target)
Preprocess
Build Models (Task-Specific)

Fig. 2 Overall approach

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 2 shows the overall architecture of our proposed framework.

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. Sharma et al. (2015a) and Zhang et al. (2018)), 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" (StackExchange 2019). 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.

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

We used the Stack Overflow data dump provided by previous work by Sharma et al. (2015a). 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 (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 NLTKFLs *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 (Poché et al. 2017; Bacchelli et al. 2012a; Williams and Mahmoud 2017) have shown that it does not boost performance. We chose the Skip-gram Word2Vec model proposed by Mikolov et al. (2013a). 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 computationally costly and produces a larger word embedding matrix. According to the experiment conducted by Pennington and Socher (2014), the best accuracy was achieved with 300 dimensions. In their experiments, performance did not improve dramatically if the number of dimensions is increased further. The 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 1 includes statistics on the generated word embedding learned from the datasets.

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

[6] http://www.nltk.org

[7] https://pypi.org/project/gensim/

Table 1 Statistics of datasets and word embedding extracted from Stack Overflow (SIEVE_SO) and StackExchange-SE (SIEVE_SE)

	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

Vocabulary size refers to the number of unique terms in the Word2Vec model

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.

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. Sharma et al. (2015a) 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. (2015a) 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?

4.1 Dataset

For the Twitter dataset, we use the same dataset created by Sharma et al. (2015a). 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 the majority of them (83.3%) are not software related. Statistics of the tweets dataset are listed in Table 2.

Table 2 Statistics of the Twitter dataset

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

4.2 Approach

Figure 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, 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. (2015a). 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)

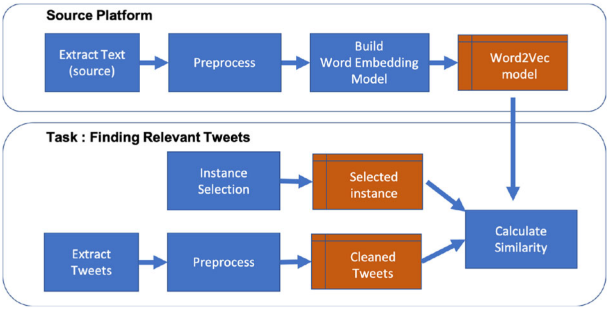


Fig. 3 Our approach for finding software-related tweets

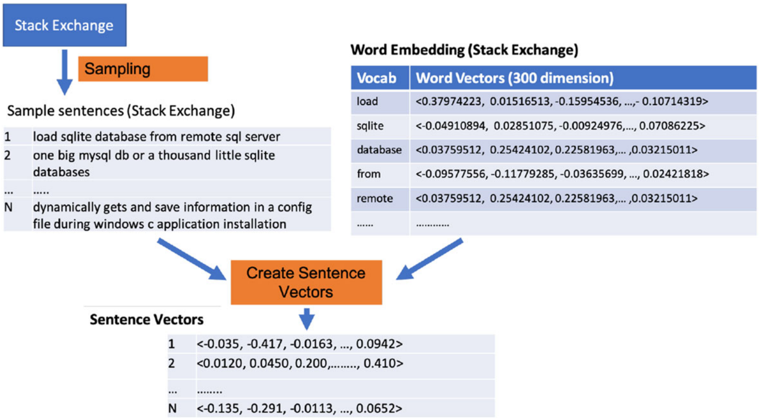
have different lengths, we transform each sentence into fixed-length vector to represent them. To build the vector representation, we leverage the word embedding learned from the source platform. The model consists of word vectors that have 300 dimensions as mentioned in Section 3. We follow these steps:

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 (Socher et al. 2013). In this case, we take the average of the word embedding of all words within the text, following Kenter and De Rijke (2015) At the end, we have a word vector of real values with 300 dimensions for each tweet or sentence.

Figures 4 and 5 illustrate the above mentioned steps. In Fig. 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 ⟨-0.04910894,...,0.07086225⟩. 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 Fig. 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.

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



Sample sentences (Stack Exchange)

1	load sqlite database from remote sql server
2	one big mysql db or a thousand little sqlite databases
...	.....
N	dynamically gets and save information in a config file during windows c application installation

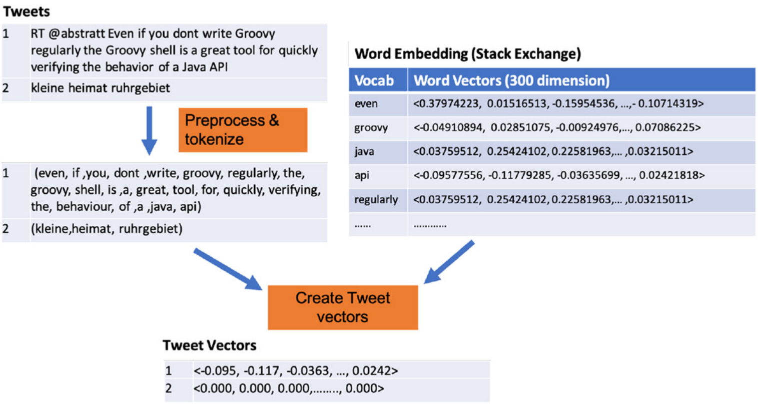
Word Embedding (Stack Exchange)

Vocab	Word Vectors (300 dimension)
load	<0.37974223, 0.01516513, -0.15954536, ...,- 0.10714319>
sqlite	<-0.04910894, 0.02851075, -0.00924976,..., 0.07086225>
database	<0.03759512, 0.25424102, 0.22581963,... ,0.03215011>
from	<-0.09577556, -0.11779285, -0.03635699, ..., 0.02421818>
remote	<0.03759512, 0.25424102, 0.22581963,... ,0.03215011>
......	....

Sentence Vectors

1	<-0.035, -0.417, -0.0163, ..., 0.0942>
2	<0.0120, 0.0450, 0.200,........, 0.410>
...	....
N	<-0.135, -0.291, -0.0113, ..., 0.0652>

Fig. 4 Illustration of sampling process from Stack Exchange, followed by creating sentence vectors



Tweets

1	RT @abstratt Even if you dont write Groovy regularly the Groovy shell is a great tool for quickly verifying the behavior of a Java API
2	kleine heimat ruhrgebiet

1	(even, if ,you, dont ,write, groovy, regularly, the, groovy, shell, is ,a, great, tool, for, quickly, verifying, the, behaviour, of ,a ,java, api)
2	(kleine,heimat, ruhrgebiet)

Word Embedding (Stack Exchange)

Vocab	Word Vectors (300 dimension)
even	<0.37974223, 0.01516513, -0.15954536, ...,- 0.10714319>
groovy	<-0.04910894, 0.02851075, -0.00924976,..., 0.07086225>
java	<0.03759512, 0.25424102, 0.22581963,... ,0.03215011>
api	<-0.09577556, -0.11779285, -0.03635699, ..., 0.02421818>
regularly	<0.03759512, 0.25424102, 0.22581963,... ,0.03215011>
......	....

Tweet Vectors

1	<-0.095, -0.117, -0.0363, ..., 0.0242>
2	<0.000, 0.000, 0.000,........, 0.000>

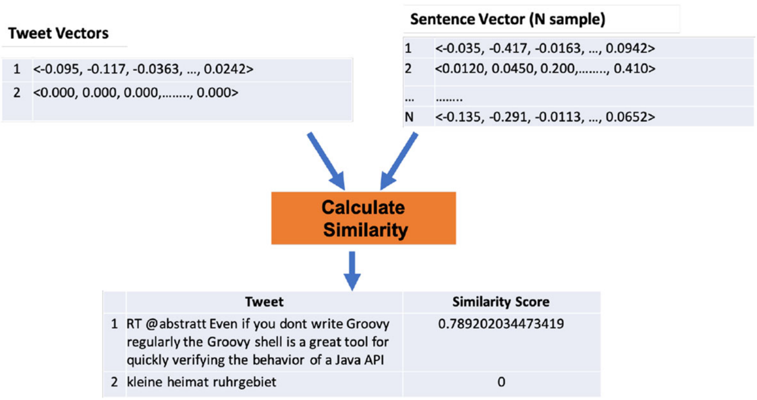
Fig. 5 Illustration of preprocessing, tokenizing and creating tweet vectors

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. Cosine 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 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.



Tweet Vectors

1	<-0.095, -0.117, -0.0363, ..., 0.0242>
2	<0.000, 0.000, 0.000,........, 0.000>

Sentence Vector (N sample)

1	<-0.035, -0.417, -0.0163, ..., 0.0942>
2	<0.0120, 0.0450, 0.200,........, 0.410>
...	....
N	<-0.135, -0.291, -0.0113, ..., 0.0652>

	Tweet	Similarity Score
1	RT @abstratt Even if you dont write Groovy regularly the Groovy shell is a great tool for quickly verifying the behavior of a Java API	0.789202034473419
2	kleine heimat ruhrgebiet	0

Fig. 6 Illustration of calculating similarity score

4.3 Baselines

We used several baselines to show the effectiveness of our approach. First, we compared our proposed approach against NIRMAL (Sharma et al. 2015a), 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., De Lucia et al. (2014) and Palomba et al. (2016). 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 (Stolcke 2002), 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 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 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 (Porter 1980) and removed English stopwords. To remove stopwords, we used stopwords listed in the Python NLTK library[8].

4.4 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 scores between the tweets and selected instances taken from a source platform. We investigated various word embedding

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

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 (Sharma et al. 2017c). As it is impossible for developers to follow everyone who may generate useful content, it is useful to create a content aggregator of tweets (Achananuparp et al. 2012). 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 (Sharma et al. 2015a) 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 (Sharma et al. 2015a), Nirmal's 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 (Xu et al. 2018), duplicate bug report detection (Hindle and Onuczko 2019), and concept location (Rahman and Roy 2017).

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 (Landis and Koch 1977), these values indicate substantial agreement.

Results The results of our experiments are shown in Table 3 and Fig. 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.

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.

Table 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**

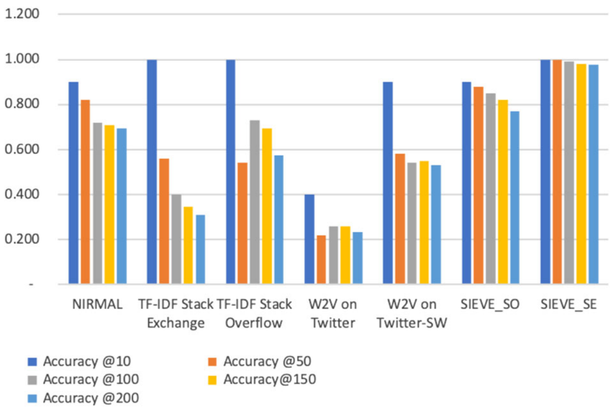


Fig. 7 Comparison of Accuracy@K achieved by different approaches

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. For example, consider the following tweets: *"In New York and off to the pre show cocktail party for From Scotland With Love hosted by Lord Jack M"*, *"In an effort to fit in with real men today I replaced my car stereo by myself and did it right the first time"*. Those two sample tweets achieved high similarity score and appeared in the top-100 list when we used the Word2Vec model trained on Twitter data. On the other hand, when we used the SIEVE_SE model, the tweets were ranked much lower (rank >80,000).

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 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 (Wilcoxon 1945). 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.

RQ1: Two variants of SIEVE (`SIEVE_SE` and `SIEVE_SO`) are able to find software-related tweets with accuracy@K of 0.800 - 1.000.

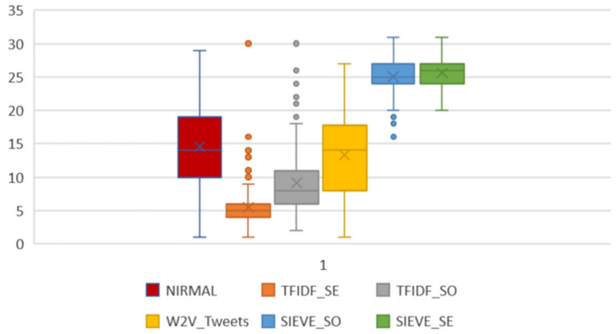


Fig. 8 A box plot diagram representing word count of tweets returned by various approaches

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. (2017) 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?

5.1 Dataset

We used the dataset provided by Poché et al. (2017)[9] They collected a total of 41,773 comments from 12 different coding tutorial 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 6000 comments (500 comments for each video) and labeled them as content concerns (informative) and miscellaneous (uninformative). Based on their manual classification process, they found 1,826 comments (around 30% of the comments) to be informative, meaning that the majority of comments are basically not related to the content.

5.2 Approach

Figure 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

[9] http://seel.cse.lsu.edu/data/icpc17.zip

[10] https://developers.google.com/youtube/v3/

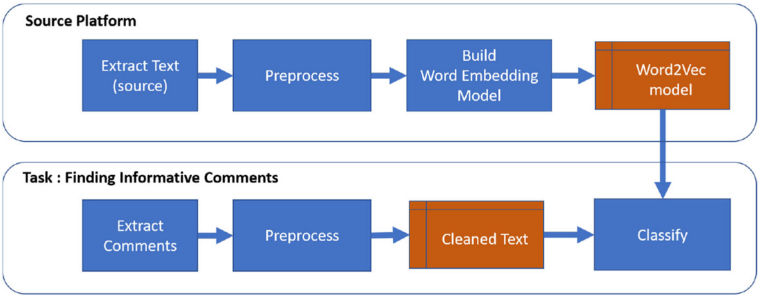


Fig. 9 Approach for finding relevant comments on YouTube

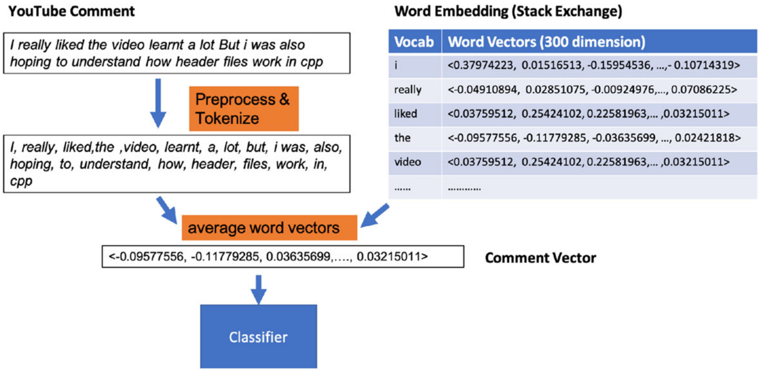
mentioned in Section 3. We build vectors to represent the comments, by following these steps:

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 and De Rijke (2015). At the end, we have a word vector of real values with 300 dimensions for each comment.

Figure 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.

5.3 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



I really liked the video learnt a lot But i was also hoping to understand how header files work in cpp

I, really, liked,the ,video, learnt, a, lot, but, i was, also, hoping, to, understand, how, header, files, work, in, cpp

Vocab	Word Vectors (300 dimension)
i	<0.37974223, 0.01516513, -0.15954536, ...,- 0.10714319>
really	<-0.04910894, 0.02851075, -0.00924976,..., 0.07086225>
liked	<0.03759512, 0.25424102, 0.22581963,... ,0.03215011>
the	<-0.09577556, -0.11779285, -0.03635699, ..., 0.02421818>
video	<0.03759512, 0.25424102, 0.22581963,... ,0.03215011>
......	....

<-0.09577556, -0.11779285, 0.03635699,...., 0.03215011>

Fig. 10 An illustration of preprocessing and creating comment vectors for classifying YouTube comments

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. We briefly describe the baselines as follows:

1. **Normalized Term-Frequency (as proposed by Poché et al. 2017).** 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 stop-words, 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 3 on these datasets. We use these models to predict informative comments following the process that we have described in Section 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 News dataset (about 100 billion words).

5.4 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 (Poché et al. 2017). 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

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

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

Table 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**

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, 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 4 and Fig. 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 of our approach as compared to the baselines are shown in Table 4 and Fig. 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.

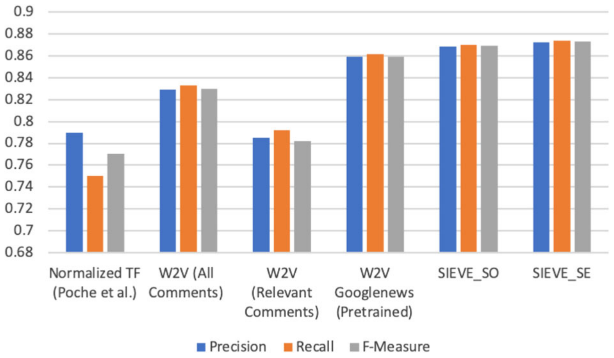


Fig. 11 Comparison of Precision, Recall and F-measure achieved by different approaches

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. (2017), 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 there's 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.

> RQ2: Two variants of SIEVE (`SIEVE_SE` and `SIEVE_SO`) performed better than various baselines with F-measure score of 0.874 and 0.869 respectively.

6 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, (3) learning word embedding in other software-related domains, and (4) finding software-relevant content from sample tweets.

6.1 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 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. 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

Table 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

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.

6.2 Comparing Different Methods to Learn Word Embedding

Skip-gram (Mikolov et al. 2013a) is a popular but not the only method to learn word embedding. There are other methods such as CBOW (Mikolov et al. 2013a), GloVe (Pennington and Socher 2014) and FastText (Bojanowski et al. 2017). CBOW and Skip-gram were proposed by Mikolov et al. (2013a); 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 and Socher (2014), 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. (2017), is an approach built based on the Skip-gram model, where each word is represented as a bag of character n-gram (Bojanowski et al. 2017).

Mikolov et al. have shown that the effectiveness of the word embedding learned via CBOW and Skip-gram differ for different tasks (Mikolov et al. 2013a). Also, Pennington and Socher (2014) and Bojanowski et al. (2017) 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 4) and identifying informative comments for improving software development video tutorials (Section 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 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 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).

To evaluate the effectiveness of the four methods for the second use case, we followed the setting described in Section 5 and repeated the experiment four times using different

Table 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

methods to learn word embedding. The results of our experiments for the second use case are shown in Table 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 (Mikolov et al. 2013a).

6.3 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 Sections 4 and 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. (2018) 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 3. The results of our experiments for the task of finding software-relevant tweets are shown in Table 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.

To evaluate the effectiveness of the HackerNews dataset for the second use case, we followed the settings described in Section 5. The results of our experiment for classifying YouTube comments are shown in Table 9. We can observe from the table that the

Table 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

Table 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

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.

6.4 Finding Software-Relevant Content from Sample Tweets

In Section 4, we experimented with a collection of around 6 million tweets to demonstrate our approach on how to find software relevant tweets. In this section, we want to investigate the performance of our approach on a set of randomly selected tweets. Instead of using all tweets, we randomly selected 1000 tweets (which is a statistically significant sample size considering a confidence level of 95% and a confidence interval of 5) and applied different approaches for finding software-relevant tweets. This dataset consists of 16.7% software-relevant tweets (as described in Table 2). In addition, we use different metrics to evaluate the performance of the approaches. Instead of using the accuracy@k metric, we use Mean Average Precision at k (MAP@k). To calculate MAP@k, we first calculate the average precision across all values between 1 and k. Then, MAP@k is defined as the mean value of all average precision values ranging from 1 to k.

The results of our experiments are shown in Table 10. Overall, the MAP@k results obtained from SIEVE_SE (Word2Vec trained on Software Engineering Stack Exchange) are better than those obtained from other approaches. These findings justify our previous observation that the Stack Overflow and Stack Exchange platforms provide sufficiently rich information that can be leveraged to find software-relevant content across platforms.

7 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

Table 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

Table 10 MAP@K results of different approaches for the task of finding software-relevant tweets on a sample of 1000 tweets

Approach	MAP@10	MAP@50	MAP@100	MAP@150
`SIEVE SE`	1.000	0.889	0.746	0.651
`SIEVE SO`	0.803	0.598	0.514	0.464
`Tf-idf SE`	0.131	0.208	0.238	0.253
`Tf-idf SO`	0.177	0.223	0.241	0.254

the tools we used in our analysis. We used Gensim[13], a popular Python module for machine learning to build word embeddings that has also been used in many previous studies related to word embeddings. 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 (Sharma et al. 2015a; Poché et al. 2017) to enable fair comparisons (i.e., Accuracy@k, Precision, Recall and F-measure). Therefore, we believe that the threat to construct validity is minimal.

8 Related Work

In this section, we describe work related to our study.

8.1 Developer Information Channels

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 (Storey et al. 2014, 2017). They found that a lot of knowledge is embedded in these tools and channels which also encourages a participatory culture in software development. Their work also highlights the challenges faced by developers while using these channels. In the paragraphs below, we discuss work related to channels such as Software Engineering Stack Exchange

[13] https://pypi.org/project/gensim/

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

and Stack Overflow (software question-and-answer sites), Twitter, and YouTube, as these are the domains we have considered in this work.

Stack Overflow has received much attention in recent years in the software engineering research community. Barua et al. analyzed in detail the topics and trends among discussions on Stack Overflow by applying Latent Dirichlet Allocation (LDA) (Barua et al. 2014). They found that the topics which interest developers range from jobs to version control systems to C# syntax etc. Their analysis showed that web development, mobile applications, Git, and MySQL were the topics that were gaining the most popularity over time. Vasilescu et al. explore how the developer's use of Stack Overflow and GitHub relates to each other (Vasilescu et al. 2013). Many empirical studies have focused on understanding and modeling questions and/or answers on Stack Overflow. Asaduzzaman et al. found that some questions go unanswered on Stack Overflow as the questions may be short, unclear, too hard, etc. Asaduzzaman et al. (2013). Rahman et al. developed a prediction model based on behavior, topics, and popularity of a question to determine unresolved questions (Rahman and Roy 2015). Ponzanelli et al. have performed studies to analyze the quality of questions on Stack Overflow (Ponzanelli et al. 2014a) and also proposed an approach to detect low-quality questions (Ponzanelli et al. 2014b). Treude et al. did an analysis of how programmers ask and answer questions on Stack Overflow (Treude et al. 2011). How the crowd generates valuable documentation on Stack Overflow and ways of measuring it have been discussed by Parnin et al. (2011, 2012) A lot of tools have also been proposed which can help developers in their usage of Stack Overflow. Many techniques have been proposed for solving the problem of tag prediction for questions on Stack Overflow (Wang et al. 2014; Zhou et al. 2017; Cai et al. 2016). Seahawk, an Eclipse plugin was proposed to integrate Stack Overflow knowledge within the IDE (Ponzanelli et al. 2013; Bacchelli et al. 2012b). Identification of opinionated sentences from Stack Overflow data and subsequent aspect identification have been proposed recently by Uddin et al. (2017a, b).

Stack Exchange was first explored from a software engineering perspective by Begel and Bosch (2013). This work explored what kinds of service were provided by Stack Exchange, what challenges they face, and how people benefit from the service. Possnet et al. did an empirical analysis of user expertise on Stack Exchange websites and found that the expertise of users does not increase with time spent in the community; experts join the community as experts, and provide good answers from the beginning (Posnett et al. 2012). Vasilescu et al. analyzed how social Q&A sites such as Stack Exchange affect the knowledge sharing practices in open source communities (Vasilescu et al. 2014).

Twitter also has been explored in recent years by the software engineering research community. Singer et al. did a survey involving 271 developers from GitHub and found that Twitter is used by developers to keep themselves up-to-date with the latest happenings in software development (Singer et al. 2014). Bougie et al. did an exploratory study on understanding how Twitter is used in software engineering (Bougie et al. 2011). Wang et al. studied the usage of Twitter in Drupal open source development (Wang et al. 2013). Tian et al. found that Twitter is also used by software developers for coordination of efforts, sharing of knowledge, etc. (Tian et al. 2012, 2014). Sharma et al. have explored the categories of software engineering related tweets and events on Twitter (Sharma et al. 2015b). Methods to filter software-relevant tweets and links have been proposed in Prasetyo et al. (2012) and Sharma et al. (2015a, 2017c). Sharma et al. also proposed an approach to find software experts on Twitter (Sharma et al. 2018). 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 Guzman et al. (2016, 2017a). They also proposed a technique to mine tweets for software requirements and evolution (Guzman et al. 2017b). There has been other work also on mining Twitter feeds for software user requirements such as by Williams and Mahmoud (2017). Mezouar et al. found that tweets generated by users can help in early detection of bugs in software applications, and can help developers know about a bug which may be affecting a large user base (El Mezouar et al. 2018).

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 Macleod et al. (2015, 2017). They found that the main motivation for sharing videos by developers are building an online identity, to give back to community, to promote themselves, etc. Ponzanelli et al. proposed an approach to extract relevant fragments from software development video tutorials Ponzanelli et al. (2016a, b). Their approach splits the video tutorials into coherent fragments, which are then classified into relevant categories. These fragments are then available individually for developers to query, rather than being forced to browse the whole video. Poché et al. proposed an approach to identify relevant user comments on coding video tutorials on YouTube (Poché et al. 2017). Parra et al. proposed a text-mining-based approach to recommend tags for software development videos on YouTube (Parra et al. 2018). Recently there has been work on extracting code and/or code related features also from programming tutorial videos extracted from YouTube by Yadid and Yahav (2016) and Ott et al. (2018).

8.2 Transfer Learning for Neural Models

Transfer learning for neural models has been explored in the literature (Mou et al. 2016; Semwal et al. 2018; Yu et al. 2018; Johnson and Zhang 2015, 2016). We present several recent related work and highlight similarities and differences between their work and ours. As we will discuss below, the major differences between related work and ours is (1) the use of an unlabelled data set from the source platform, (2) the focus on multiple classification tasks instead of the narrower sentiment analysis, and (3) knowledge transfer from a domain-rich platform to a domain-poor platform.

Related to our work is a recent study by Mou et al. that explored the question how transferable are neural networks in Natural Language Processing (NLP) tasks (Mou et al. 2016). They investigated how a labelled dataset created for a specific classification task can be used for the same (or a related) classification task performed on another dataset. In particular, they demonstrated that a large labeled dataset for sentiment analysis of movie reviews can be used to help classify the sentiment of movie reviews (i.e., positive or negative) in another smaller dataset. They also demonstrate that a large dataset for natural language inference – inference whether a hypothesis (e.g., "people ride bikes") is supported by a premise (e.g., "two men on bicycles are competing in a race") – can be used to help in detecting paraphrases. A number of other studies extended Mou et al.'s work for additional NLP tasks, e.g., named-entity-recognition (Lee et al. 2017), retrieval-based question answering (Yu et al. 2018), etc. In our work, different from Mou et al.'s work and its extensions, we use an unlabeled dataset from a platform with rich software engineering related content to help in software engineering related tasks in a different platform. Mou et al.'s study investigates two different methods for transfer learning: parameter initialization and multi-task learning. Our approach can be considered as a parameter initialization method; however, we consider an unsupervised setting.

A study by Semwal et al. provides some practical guidelines for applying transfer learning considering neural models for NLP applications (Semwal et al. 2018). Their work demonstrated that for a sentiment classification task, the content of the embedding layer of a neural network learned from one dataset can be used for the same task performed on another dataset. They also suggest to pick a source dataset with a large vocabulary size that contains content with similar semantics to the target dataset. Our study is different from them in several aspects: (1) we consider different tasks (not sentiment classification); (2) we use an unlabeled dataset from one platform to help in ranking and classification tasks in another platform. Still, we uncover some similar findings; for example, our study demonstrates (at least for the tasks and datasets that we consider) that a source dataset with a large vocabulary size that contains content with similar semantics to a target task can be leveraged to help the target task.

There are prior studies that use unlabelled data to improve neural models (Johnson and Zhang 2015, 2016). For example, Johnson and Zhang made use of unlabelled data from one platform (e.g., IMDB) to boost performance of sentiment classification tasks, done using a Convolutional Neural Network (CNN), for labelled data from the same platform (Johnson and Zhang 2016). In their later work, they considered a Long Short Term Memory (LSTM) recurrent neural network instead of CNN (Johnson and Zhang 2016). Different from their work, in this work, we consider how unlabelled data from one domain-rich platform can be used for domain-related tasks in a different platform that is domain-poor.

Word embedding have been used in software engineering for improving information retrieval tasks (Ye et al. 2016; Xu et al. 2016; Chen et al. 2016c). Chen et al. have used word embedding based methods to mine analogical libraries (Chen et al. 2016a), assist collaborative editing (Chen et al. 2018), recommend tag synonyms (Chen et al. 2017) and recommend similar libraries (Chen and Xing 2016b). In work of Yang et al. (2016), word embedding was combined with information retrieval to recommend similar bug reports. Guo et al. proposed a neural architecture based on word embedding and Recurrent Neural Network (RNN) (Guo et al. 2017), for automatic generation of trace links. Zhao et al. proposed a traceability recovery technique based on combination of word embedding and learning to rank methods (Zhao et al. 2017). Methods similar to or based on word embedding have also been used recently for better code retrieval (Van Nguyen et al. 2017), to find common software weaknesses (Zhenchang et al. 2018), API recommendation (Zhang et al. 2018) and sentiment analysis for software engineering (Calefato et al. 2018). Ye et al. (2016) built a Skip-gram word embedding model by using a corpus extracted from Eclipse repositories (including the Platform API Reference, the JDT API Reference) and evaluated the model for the tasks of bug localization and linking API documents to Java questions posted on Stack Overflow. While SIEVE also uses word embedding technique and computes similarity between texts, our work differs from Ye et al.'s work in the following aspects: (1) We are the first to investigate the power of leveraging data from source platforms which contain rich domain-related content (i.e., Stack Overflow and Software Engineering Stack Exchange), to solve tasks in other platforms with less domain-related content (i.e., Twitter and YouTube) via transfer representation learning; (2) We consider not only information retrieval but also classification tasks relevant to software engineering; (3) We explore the power of word embedding on two use cases that were not considered by Ye et al.(i.e., finding software-related tweets for developer learning and identifying informative comments for improving software development video tutorials); (4) We have explored not only the Skip-gram method but multiple other alternatives to learn word embedding and investigated their impact on the two use cases.

9 Conclusion and Future Work

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.).

Acknowledgments This research is supported by the National Research Foundation, Prime Ministers Office, Singapore under its International Research Centres in Singapore Funding Initiative.

References

Achananuparp P, Lubis IN, Tian Y, Lo D, Lim E-P (2012) Observatory of trends in software related microblogs. In: 2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. IEEE, pp 334–337

Andrews JTA, Tanay T, Morton EJ, Griffin LD (2016) Transfer representation-learning for anomaly detection. In: Anomaly detection workshop. ICML

Aniche M, Treude C, Steinmacher I, Wiese I, Pinto G, Storey M-A, Gerosa MA (2018) How modern news aggregators help development communities shape and share knowledge. In: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE). IEEE, pp 499–510

Asaduzzaman M, Mashiyat AS, Roy CK, Schneider KA (2013) Answering questions about unanswered questions of stack overflow. In: Proceedings of the 10th Working Conference on Mining Software Repositories. IEEE Press, pp 97–100

Azad S, Rigby PC, Guerrouj L (2017) Generating api call rules from version history and stack overflow posts. ACM Transactions on Software Engineering and Methodology (TOSEM)

Bacchelli A, Sasso TD, D'Ambros M, Lanza M (2012a) Content classification of development emails. In: 2012 34Th international conference on software engineering (ICSE). IEEE, pp 375–385

Bacchelli A, Ponzanelli L, Lanza M (2012b) Harnessing stack overflow for the ide. In: Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering. IEEE Press, pp 26–30

Barua A, Thomas SW, Hassan AE (2014) What are developers talking about? an analysis of topics and trends in stack overflow. Empir Softw Eng 19(3):619–654

Begel Andrew, Bosch Jan (2013) Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder. IEEE Softw 30(1):52–66

Bengio Y, Courville A (2013) Representation learning: A review and new perspectives. IEEE Trans Pattern Anal Mach Intell 35(8):1798–1828

Bojanowski P, Grave E, Joulin A, Mikolov T (2017) Enriching word vectors with subword information. Trans Assoc Comput Linguist 5:135–146

Bougie G, Starke J, Storey M-A, German DM (2011) 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, pp 31–36

Cai X, Zhu J, Shen B, Chen Y (2016) Greta: Graph-based tag assignment for github repositories. In: Computer software and applications conference (COMPSAC), 2016 IEEE 40th annual. IEEE, vol 1, pp 63–72

Calefato F, Lanubile F, Maiorano F, Novielli N (2018) Sentiment polarity detection for software development. Empir Softw Eng 23(3):1352–1382

Chen C, Sa G, Xing Z (2016a) Mining analogical libraries in q&a discussions–incorporating relational and categorical knowledge into word embedding. In: 2016 IEEE 23rd international conference on Software analysis, evolution, and reengineering (SANER). IEEE, vol 1, pp 338–348

Chen C, Xing Z (2016b) Similartech: automatically recommend analogical libraries across different programming languages. In: 2016 31st IEEE/ACM international conference on Automated software engineering (ASE). IEEE, pp 834–839

Chen G, Chen C, Xing Z, Xu B (2016c) 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. ACM, pp 744–755

Chen C, Xing Z, Wang X (2017) Unsupervised software-specific morphological forms inference from informal discussions. In: Proceedings of the 39th International Conference on Software Engineering. IEEE Press, pp 450–461

Chen C, Xing Z, Liu Y (2018) 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. ACM, pp 32:1–32:21

Chenail RJ (2008) Youtube as a qualitative research asset: Reviewing user generated videos as learning resources. Q Rep 13(3):18–24

De Lucia A, Di Penta M, Oliveto R, Panichella A, Panichella S (2014) Labeling source code with information retrieval methods: an empirical study. Empir Softw Eng 19(5):1383–1420

El Mezouar M, Zhang F, Zou Y (2018) Are tweets useful in the bug fixing process? an empirical study on firefox and chrome. Empir Softw Eng 23(3):1704–1742

Guo J, Cheng J, Cleland-Huang J (2017) Semantically enhanced software traceability using deep learning techniques. In: 2017 IEEE/ACM 39th international conference on Software engineering (ICSE). IEEE, pp 3–14

Guzman E, Alkadhi R, Seyff N (2016) A needle in a haystack What do twitter users say about software?. In: 2016 IEEE 24th international Requirements engineering conference (RE). IEEE, pp 96–105

Guzman E, Alkadhi R, Seyff N (2017a) An exploratory study of twitter messages about software applications. Requir Eng 22(3):387–412

Guzman E, Ibrahim M, Glinz Martin (2017b) A little bird told me: mining tweets for requirements and software evolution. In: 2017 IEEE 25Th international requirements engineering conference (RE). IEEE, pp 11–20

Hindle A, Onuczko C (2019) Preventing duplicate bug reports by continuously querying bug reports. Empir Softw Eng 24(2):902–936

Johnson R, Zhang T (2015) Semi-supervised convolutional neural networks for text categorization via region embedding. In: Advances in neural information processing systems, pp 919–927

Johnson R, Zhang T (2016) Supervised and semi-supervised text categorization using lstm for region embeddings. arXiv:1602.02373

Kenter T, De Rijke M (2015) Short text similarity with word embeddings. In: Proceedings of the 24th ACM international on conference on information and knowledge management. ACM, pp 1411–1420

Kwak H, Lee C, Park H, Moon SB (2010) 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, pp 591–600

Landis JR, Koch GG (1977) The measurement of observer agreement for categorical data. biometrics pp 159–174

Lee JY, Dernoncourt F, Szolovits P (2017) Transfer learning for named-entity recognition with neural networks. arXiv:1705.06273

Maalej W, Tiarks R, Roehm T, Koschke R (2014) On the comprehension of program comprehension. ACM Trans Softw Eng Methodol (TOSEM) 23(4):31

MacLeod L, Storey M-A, Bergen A (2015) 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. IEEE Press, pp 104–114

MacLeod L, Bergen A, Storey M-A (2017) Documenting and sharing software knowledge using screencasts. Empir Softw Eng 22(3):1478–1507

Mikolov T, Chen K, Corrado G, Dean J (2013a) Efficient estimation of word representations in vector space. arXiv:1301.3781

Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013b) Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems, pp 3111–3119

Mou L, Meng Z, Yan R, Li G, Xu Y, Lu Z, Jin Z (2016) How transferable are neural networks in nlp applications?. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pp 479–489

Ott J, Atchison A, Harnack P, Best N, Anderson H, Firmani C, Linstead E (2018) Learning lexical features of programming languages from imagery using convolutional neural networks. In: Proceedings of the 26th Conference on Program Comprehension, ICPC '18. ACM, New York, pp 336–339

Palomba F, Panichella A, De Lucia A, Oliveto R, Zaidman A (2016) A textual-based technique for smell detection. In: 2016 IEEE 24Th international conference on program comprehension (ICPC). IEEE, pp 1–10

Pan SJ, Yang Q et al (2010) A survey on transfer learning. IEEE Trans Knowl Data Eng 22(10):1345–1359

Parnin C, Treude C (2011) Measuring api documentation on the web. In: Proceedings of the 2nd international workshop on Web 2.0 for software engineering. ACM, pp 25–30

Parnin C, Treude C, Grammel L (2012) Crowd documentation: Exploring the coverage and the dynamics of api discussions on stack overflow. Georgia Institute of Technology, Technical Report

Parra E, Escobar-Avila J, Haiduc S (2018) Automatic tag recommendation for software development video tutorials. In: Proceedings of the 26th Conference on Program Comprehension. ACM, pp 222–232

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

Poché E, Jha N, Williams G, Staten J, Vesper M, Mahmoud A (2017) Analyzing user comments on youtube coding tutorial videos. In: Proceedings of the 25th International Conference on Program Comprehension. IEEE Press, pp 196–206

Ponzanelli L, Bacchelli A, Lanza M (2013) Seahawk: Stack overflow in the ide. In: Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, pp 1295–1298

Ponzanelli L, Mocci A, Bacchelli A, Lanza M (2014a) Understanding and classifying the quality of technical forum questions. In: Quality software (QSIC), 2014 14th international conference on. IEEE, pp 343–352

Ponzanelli L, Mocci A, Bacchelli A, Lanza M, Fullerton D (2014b) Improving low quality stack overflow post detection. In: 2014 IEEE international conference on Software maintenance and evolution (ICSME). IEEE, pp 541–544

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

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

Porter MF (1980) An algorithm for suffix stripping. Program 14(3):130–137

Posnett D, Warburg E, Devanbu P, Filkov V (2012) Mining stack exchange: Expertise is evident from initial contributions. In: 2012 international conference on Social informatics (socialinformatics). IEEE, pp 199–204

Prasetyo PK, Lo D, Achananuparp P, Tian Y, Lim E-P (2012) Automatic classification of software related microblogs. In: 2012 28th IEEE international conference on Software maintenance (ICSM). IEEE, pp 596–599

Rahman MM, Roy CK (2015) An insight into the unresolved questions at stack overflow. In: Proceedings of the 12th Working Conference on Mining Software Repositories. IEEE Press, pp 426–429

Rahman MM, Roy CK (2017) Strict: Information retrieval based search term identification for concept location. In: 2017 IEEE 24Th international conference on software analysis, evolution and reengineering (SANER). IEEE, pp 79–90

Semwal T, Yenigalla P, Mathur G, Nair SB (2018) A practitioners' guide to transfer learning for text classification using convolutional neural networks. In: Proceedings of the 2018 SIAM International Conference on Data Mining. SIAM, pp 513–521

Sharma A, Tian Y, Lo D (2015a) Nirmal: Automatic identification of software relevant tweets leveraging language model. In: 2015 IEEE 22nd international conference on Software analysis, evolution and reengineering (SANER). IEEE, pp 449–458

Sharma A, Tian Y, Lo D (2015b) What's hot in software engineering twitter space?. In: 2015 IEEE international conference on Software maintenance and evolution (ICSME). IEEE, pp 541–545

Sharma A, Tian Y, Sulistya A, Lo D, Yamashita AF (2017c) Harnessing twitter to support serendipitous learning of developers. In: 2017 IEEE 24th international conference on Software analysis, evolution and reengineering (SANER). IEEE, pp 387–391

Sharma A, Tian Y, Sulistya A, Wijedasa D, Lo D (2018) Recommending who to follow in the software engineering twitter space. ACM Trans Softw Eng Methodol 27(4):16:1–16:33

Singer L, Filho FF, Storey M-A (2014) Software engineering at the speed of light: how developers stay current using twitter. In: Proceedings of the 36th International Conference on Software Engineering. ACM, pp 211–221

Socher R, Perelygin A, Wu J, Chuang J, Manning CD, Ng A, Potts C (2013) Recursive deep models for semantic compositionality over a sentiment treebank. In: Proceedings of the 2013 conference on empirical methods in natural language processing, pp 1631–1642

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

Stolcke A (2002) Srilm-an extensible language modeling toolkit. In: Seventh international conference on spoken language processing

Storey M-A, Singer L, Cleary B, Filho FF, Zagalsky A (2014) The (r) evolution of social media in software engineering. In: Proceedings of the on Future of Software Engineering. ACM, pp 100–116

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

Tian Y, Achananuparp P, Lubis IN, Lo D, Lim E-P (2012) What does software engineering community microblog about?. In: 2012 9th IEEE working conference on Mining software repositories (MSR). IEEE, pp 247–250

Tian Y, Lo D (2014) An exploratory study on software microblogger behaviors. In: 2014 IEEE 4Th workshop on mining unstructured data. IEEE, pp 1–5

Treude C, Barzilay O, Storey M-A (2011) How do programmers ask and answer questions on the web?: Nier track. In: 2011 33rd international conference on Software engineering (ICSE). IEEE, pp 804–807

Uddin G, Khomh F (2017a) Automatic summarization of api reviews. In: 2017 32Nd IEEE/ACM international conference on automated software engineering (ASE). IEEE, pp 159–170

Uddin Gx, Khomh F (2017b) Opiner: An opinion search and summarization engine for apis. In: 2017 32Nd IEEE/ACM international conference on automated software engineering (ASE). IEEE, pp 978–983

Van Nguyen T, Nguyen AT, Phan HD, Nguyen TD, Nguyen TN (2017) Combining word2vec with revised vector space model for better code retrieval. In: Proceedings of the 39th International Conference on Software Engineering Companion. IEEE Press, pp 183–185

Vasilescu B, Filkov V, Serebrenik A (2013) Stackoverflow and github: Associations between software development and crowdsourced knowledge. In: 2013 international conference on Social computing (socialcom). IEEE, pp 188–195

Vasilescu B, Serebrenik A, Devanbu P, Filkov V (2014) How social q&a sites are changing knowledge sharing in open source software communities. In: Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing. ACM, pp 342–354

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

Wang S, Lo D, Vasilescu B, Serebrenik A (2014) Entagrec: An enhanced tag recommendation system for software information sites. In: 2014 IEEE international conference on Software maintenance and evolution (ICSME). IEEE, pp 291–300

Wilcoxon F (1945) Individual comparisons by ranking methods. Biom Bullet 1(6):80–83

Williams G, Mahmoud A (2017) Mining twitter feeds for software user requirements. In: 2017 IEEE 25Th international requirements engineering conference (RE). IEEE, pp 1–10

Xia X, Bao L, Lo D, Kochhar PS, Hassan AE, Xing Z (2017) What do developers search for on the web? Empir Softw Eng 22(6):3149–3185

Xu B, Xing Z, Xia X, Lo D, Wang Q, Li S (2016) Domain-specific cross-language relevant question retrieval. In: Proceedings of the 13th International Conference on Mining Software Repositories. ACM, pp 413–424

Xu C, Sun X, Li B, Lu X, Guo H (2018) Mulapi: Improving api method recommendation with api usage location. J Syst Softw 142:195–205

Yadid S, Yahav E (2016) 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. ACM, pp 98–111

Yang X, Lo D, Xia X, Bao L, Sun J (2016) Combining word embedding with information retrieval to recommend similar bug reports. In: 2016 IEEE 27th international symposium on Software reliability engineering (ISSRE). IEEE, pp 127–137

Ye D, Xing Z, Kapre N (2017) The structure and dynamics of knowledge network in domain-specific q&a sites: a case study of stack overflow. Empir Softw Eng 22(1):375–406

Ye X, Shen H, Ma X, Bunescu R, Liu C (2016) From word embeddings to document similarities for improved information retrieval in software engineering. In: Proceedings of the 38th international conference on software engineering. ACM, pp 404–415

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

Yu J, Qiu M, Jiang J, Huang J, Song S, Chu W, Chen H (2018) Modelling domain relationships for transfer learning on retrieval-based question answering systems in e-commerce. In: Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining. ACM, pp 682–690

Zhang J, He J, Ren Z, Chen X (2018) Recommending apis for api related questions in stack overflow. IEEE Access 6:6205–6219

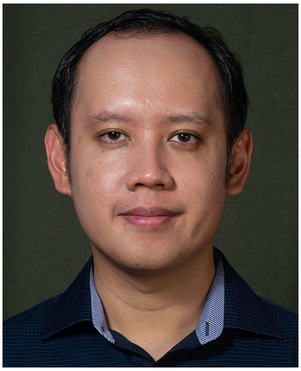
Zhao T, Cao Q, Sun Q (2017) An improved approach to traceability recovery based on word embeddings. In: 2017 24th Asia-pacific software engineering conference (APSEC). IEEE, pp 81–89

Zhou P, Liu J, Yang Z, Zhou G (2017) Scalable tag recommendation for software information sites. In: 2017 IEEE 24th international conference on Software analysis, evolution and reengineering (SANER). IEEE, pp 272–282

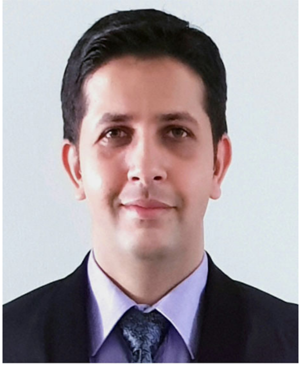
Zhenchang HL, Han XZ, Li X, Feng Z (2018) Reasoning common software weaknesses via knowledge graph embedding. In: 2018 IEEE 25rd international conference on Software analysis, evolution, and reengineering (SANER)

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Agus Sulistya is currently a Senior Expert at PT Telekomunikasi Indonesia. He received his PhD in Information Systems from the School of Information Systems, Singapore Management University (SMU). Earlier, he received his B.Eng in Informatics from Telkom University, Indonesia and Master of Science in Information Systems from Northeastern University. His research interest is in the area of making sense of crowd generated content in domain specific settings. Prior to his doctoral study, he has held several technical and managerial positions at PT Telekomunikasi Indonesia.



Gede Artha Azriadi Prana is a PhD student in the School of Information Systems, Singapore Management University, under supervision of Associate Professor David Lo. He received his B.Eng. in Computer Engineering from Nanyang Technological University and M.Tech. in Knowledge Engineering from National University of Singapore. His research focuses on application of analytics to unstructured data in software engineering domain. Prior to his current study, he has worked in the field of software engineering for about a decade.



Abhishek Sharma is currently a Research Fellow at Singapore Management University, Singapore. Previously, he had obtained his PhD in Information Systems from the School of Information Systems, Singapore Management University, Singapore. Earlier he has completed Bachelor of Technology in Computer Science & Engineering from National Institute of Technology, Hamirpur, India. Prior to his doctoral studies he has also worked as a Software Developer with Accenture and Tata Consultancy Services Ltd.

David Lo is an Associate Professor of Information Systems at Singapore Management University, leading the Software Analytics Research (SOAR) group. His research interest is in the intersection of software engineering and data science, encompassing socio-technical aspects and analysis of different kinds of software artefacts, with the goal of improving software quality and developer productivity. He has served in more than 20 organizing committees and numerous program committees of research conferences including serving as general or program co-chairs of ASE 2020, SANER 2019, ICSME 2018, ICPC 2017, and ASE 2016. He is also serving in the editorial board of Empirical Software Engineering, IEEE Transactions on Reliability, Journal of Software: Evolution and Process, Information and Software Technology, Information Systems, and Neurocomputing (Software Section).

Christoph Treude is a Senior Lecturer and an ARC DECRA Fellow in the School of Computer Science at the University of Adelaide, Australia. He received his Diplom degree from the University of Siegen, Germany, and his PhD degree in Computer Science from the University of Victoria, Canada. The goal of his research is to advance collaborative software engineering through empirical studies and the innovation of processes and tools that explicitly take the wide variety of artefacts available in a software repository into account. He currently serves as a board member on the Editorial Board of the Empirical Software Engineering journal and as General Co-Chair for ICSME 2020.