LEARNING TO INTERPRET KNOWLEDGE FROM SOFTWARE QUESTION AND ANSWER SITES

Bowen Xu

SINGAPORE MANAGEMENT UNIVERSITY

2021

Learning to interpret knowledge from software Q&A sites

Bowen Xu

Submitted to School of Computing and Information Systems
in partial fulfillment of the requirements for
the Degree of Doctor of Philosophy in Computer Science

Dissertation Committee:

Singapore Management University
2021

I hereby declare that this dissertation is my original work and it has been written by me in its entirety.
I have duly acknowledged all the sources of information which have been used in this dissertation.

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

Bowen Xu

23 August 2021

Learning to interpret knowledge from software Q&A sites

Bowen Xu

Abstract

Software question and answer (SQA) data has become a treasure for software engineering as it contains a huge volume of programming knowledge. That knowledge can be interpreted in many different ways to support various software activities, like code recommendation and program repair. In this dissertation, we interpret SQA data by addressing three novel research problems.

The first research problem is about linkable knowledge unit prediction. In this problem, a question and its answers within a post in Stack Overflow are considered as a *knowledge unit* (KU). KUs often contain semantically relevant knowledge, and thus linkable for different purposes. Being able to classify different classes of linkable knowledge units would support more targeted information needs when users search or explore the linkable knowledge. Compare with the prior approaches, we design a relatively simpler but more effective machine learning model to address the problem. Moreover, we discover the limitation of the dataset used in the previous works and construct a new one with a larger size and higher diversity. Our experimental result shows that our model outperforms the state-of-the-art approaches significantly.

The second research problem is about distributed representation for Stack Overflow posts. Past studies have proposed solutions that analyze Stack Overflow content to help users find desired information or aid various downstream software engineering tasks. We find that a common step performed by those solutions is to extract suitable representations of posts. Intuitively, the quality of the representations of posts determines the effectiveness of the solutions in performing the respective tasks. In this dissertation, to aid existing studies that analyze Stack Overflow posts, we propose a specialized deep learning

architecture Post2Vec which extracts distributed representations of Stack Overflow posts. Post2Vec is aware of different types of content present in Stack Overflow posts, i.e., title, description, and code snippets, and integrates them seamlessly to learn post representations. To evaluate the quality of Post2Vec's deep learning architecture, we first investigate its end-to-end effectiveness in tag recommendation task. We observe that Post2Vec achieves significant improvement on the effectiveness at a lower computational cost than the state-of-the-art approaches. Moreover, to evaluate the value of representations learned by Post2Vec, we use them for three other tasks, i.e., relatedness prediction, post classification, and API recommendation. We demonstrate that the representations can be used to boost the effectiveness of state-of-the-art solutions for the three tasks by substantial margins.

The third research problem is about answer summary generation for technical questions. Although search engines can return a list of questions relevant to a user query of some technical question, the abundance of relevant posts and the sheer amount of information in them makes it difficult for developers to digest them and find the most needed answers to their questions. Thus, we aim to help developers who want to quickly capture the key points of several answer posts relevant to a technical question before they read the details of the posts. We formulate the task as a query-focused multi-answer-posts summarization task for a given technical question. Our proposed approach AnswerBot contains three main steps : 1) relevant question retrieval, 2) useful answer paragraph selection, 3) diverse answer summary generation. We conduct user studies to evaluate the quality of the answer summaries generated by our approach, and the effectiveness of its relevant question retrieval and answer paragraph selection components. The results demonstrate those answer summaries generated by AnswerBot are relevant, useful, and diverse; moreover, the two components can effectively retrieve relevant questions and select salient answer paragraphs for summarization.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Contribution 5

1.3 Dissertation Structure 6

2 Related Work **7**

2.1 Overview of Interpreting Knowledge in SQA 7

2.2 Knowledge Linking 9

2.3 Knowledge Representation 10

2.4 Knowledge Summarization 12

3 Linkable Knowledge Prediction **14**

3.1 Introduction 14

3.2 Task and Evaluation Metrics 16

3.2.1 Predicting Relatedness in Stack Overflow 16

3.2.2 Evaluation Metrics 19

3.3 Replication 21

3.3.1 Xu et al.'s Study (CNN MODEL) 21

3.3.2 Fu and Menzies' Study (TUNING SVM) 24

3.3.3 SimBow: A Lightweight Alternative (SOFTSVM) 25

3.4 Data 28

3.4.1 Creating LARGEDATASET 28

3.4.2 Characteristics of the new dataset 29

3.5 Research Questions and Results 30
3.5.1 Research Questions 30
3.5.2 Results 31
3.6 Discussion 35
3.6.1 Shortcomings of the original dataset 35
3.6.2 Performance of techniques 36
3.7 Threats to Validity 37
3.8 Conclusion and Future Work 37

4 Distributed Knowledge Representation 39

4.1 Introduction 39
4.2 Proposed Approach 45
4.2.1 Framework Overview 45
4.2.2 Preprocessing 46
4.2.3 Input Layer 47
4.2.4 Feature Extraction Layers 49
4.2.5 Feature Fusion and Tag Prediction Layers 49
4.2.6 Parameters Learning 51
4.3 Post2Vec for tag recommendation 51
4.3.1 Baseline Approaches 53
4.3.2 Dataset 54
4.3.3 NNs and Hyperparameter Setting 56
4.3.4 Evaluation Metrics 57
4.3.5 Experimental Result 58
4.3.6 Research Questions and Analysis 60
4.4 Post2Vec for Downstream Tasks 62
4.4.1 Baseline Approaches 62
4.4.2 Task 1: Relatedness Prediction 63
4.4.2.1 Problem Formulation 63
4.4.2.2 State-of-the-art Approach 63

4.4.2.3 Dataset 64
4.4.2.4 Experimental Setting 64
4.4.2.5 Evaluation Metrics 65
4.4.2.6 Experimental Result 65
4.4.3 Task 2: Post Classification 66
4.4.3.1 Problem Formulation 66
4.4.3.2 State-of-the-art Approach 67
4.4.3.3 Dataset 67
4.4.3.4 Experimental Setting 67
4.4.3.5 Evaluation Metrics 68
4.4.3.6 Experimental Result 68
4.4.4 Task 3: API Recommendation 70
4.4.4.1 Problem Formulation 70
4.4.4.2 State-of-the-art Approach 70
4.4.4.3 Experimental Setting 70
4.4.4.4 User Study 72
4.5 Discussion 73
4.5.1 Qualitative Analysis 73
4.5.2 Quantitative Analysis 75
4.5.2.1 Stability of Post2Vec Representation 75
4.5.2.2 Robustness of Post2Vec Representation . . . 76
4.5.3 Vector Space Visualization 77
4.5.4 Threats to Validity 80
4.6 Conclusion and Future Work 81

5 Automated Knowledge Summarization **83**

5.1 Introduction 83
5.2 Formative Study of Answer Summary 86
5.3 Proposed Approach 89
5.3.1 Relevant Question Retrieval 89

5.3.2 Useful Answer Paragraphs Selection 91
5.3.3 Diverse Answer Summary Generation 94
5.4 Experiments & Results 95
5.4.1 Question Repository and Tool Implementation 95
5.4.2 Experimental Queries 96
5.4.3 Participant Selection and Task Allocation 96
5.4.4 Research Questions 97
5.5 Discussion 103
5.5.1 Comparison with Community Question Answering 103
5.5.2 Error Analysis 105
5.5.3 Threats to Validity 107
5.6 Conclusion and Future Work 108

6 Conclusion, Impact, and Future Work 109

6.1 Conclusion 109
6.2 Impact 110
6.3 Future Work 112

Bibliography 114

List of Figures

1.1 Three Angles to Interpret Knowledge Units in Software Q&A Sites 2

3.1 Linked Knowledge Units by URL Sharing 17
3.2 CNN Architecture in CNN Model 22
3.3 Dataset Building Process 30
3.4 Topic Distribution of OriginalDataset 31
3.5 Topic Distribution of LargeDataset 32

4.1 Example of a Stack Overflow Post ID=12309269 40
4.2 The overall framework of Post2Vec. 45
4.3 The details of the red dashed box in Figure 4.2. $\mathbf{z_T}$, $\mathbf{z_D}$, and $\mathbf{z_C}$ are the embedding vectors of the title, description, and code snippet, respectively. $\mathbf{z}$ is the post vector and is fed to a fully-connected layer (FC) to produce the tag vector (i.e., the probability distribution over tags). 50
4.4 Vector Space of Post2Vec on Relatedness Prediction 75
4.5 Vector Space 78
4.6 An *Jackson*-related Cluster (#1) of Posts in Figure 4.5 78
4.7 An *Apache POI*-related Cluster (#2) of Posts in Figure 4.5 . . . 79

5.1 The Framework of AnswerBot 90

List of Tables

3.1 Classes of Knowledge Unit Pairs 18
3.2 Example of Duplicate, Direct, Indirect Knowledge Units Pairs . 20
3.3 Confusion Matrix 21
3.4 Precision, Recall and F1-Score of CNN Model and Tuning SVM on Original Dataset 28
3.5 Performance of Three Systems on Large Dataset 32
3.6 Training Time in Different Techniques 33
3.7 Feature Scale and Performance of the Considered Approaches . 36

4.1 Differences between Variants of Post2Vec 53
4.2 Effectiveness of Post2Vec and its baselines 59
4.3 Training time cost comparison 60
4.4 Performance Comparison for Relatedness Prediction 65
4.5 Performance Comparison for Post Classification. pre_T, rec_T, f_T = precision, recall and F1-score for $class_T$. 69
4.6 Performance Comparison for API Recommendation 73
4.7 Recommendation returned by BIKER and BIKER+Post2Vec for an experimental query 74
4.8 Pairs in Figure 4.4a, 4.4b, 4.4c, and 4.4d 76
4.9 Related Importance of Different Combinations of Features for Relatedness Prediction 77

5.1 The top 5 ranked Stack Overflow questions by Google Search Engine for the query “differences HashMap HashTable Java” . . 86

5.2 Desirable Answer Summary with Relevant, Salient and Diverse Information 86
5.3 Examples of Irrelevant and Low-quality Answer Paragraphs . . 86
5.4 Redundant Answer Paragraphs 87
5.5 Semantic Patterns For Salient Information 93
5.6 Task Allocation to Participants 97
5.7 Mean of Relevance, Usefulness and Diversity of Our Approach and the Baseline Approaches (RQ1) 99
5.8 Top@k Accuracy and MRR of Our Approach and the Baseline Approaches in Relevant Question Retrieval (RQ2) 101
5.9 Top@k Accuracy and MRR of Feature Ablation for Answer Paragraph Selection (RQ3) 102
5.10 Comparison of the Best Community-Provided Answers and the Answer Summary Generated by Our Approach 105

Acknowledgements

First of all, I would like to express my deepest appreciation to my supervisor, Prof. David Lo, for his dedicated support and guidance. His immense knowledge and plentiful experience have encouraged me in all the time of my academic research and daily life. Moreover, he is also a lifelong friend who has been inspiring me to overcome many difficulties and move forward during my rough PhD journey. I would also like to express my deepest gratitude to Dr. Xin Xia, for his invaluable advice during the last four years. I am also grateful to Prof. Alipour's help in guiding me on one of my research projects and writing a recommendation letter for me.

I would like to extend my sincere thanks to all the committee members, i.e., Prof. David Lo, Prof. Lingxiao Jiang, Prof. Wei Gao, and Dr. Xin Xia for your precious time and effort to review my dissertation.

I would like to thank all the current and former members of our 'family-like' research group: Dr. Duy, Dr. Ferdian, Dr. Yuan, Dr. Pavneet, Dr. Bach, Dr. Abhishek, Dr. Agus, Dr. Lingfeng, Dr. Zhiyuan, Dr. Weiqin, Dr. Yun, Dr. Thong, Dr. Le, Dr. Xuan, Dr. Xinli, Artha, HongJin, Ting, Zhipeng, Stefanus, Hilmi, Ratnadira, Xin, Chengran, and Zhou.

I would also like to thank the Singapore Management University who offered me a great opportunity to start my studies. I would like to say a special thank you to Ms. Yong Fong KOH, who has been caring for my study life.

Last but not least, I would like to express my gratitude to my family, especially my wife Liuhan Yue. Without their tremendous understanding and encouragement in the past four years, it would be impossible for me to complete my study. I would like to say "I love you!" to my little boy Zhizhi. I already had a deep impression of his great curiosity, and I wish he enjoy his life journey and discover his own research interests about this beautiful world.

Bowen Xu

May 2021, Singapore

Chapter 1

Introduction

1.1 Motivation

In recent years, software question and answer (SQA) sites have grown rapidly and nowadays they already become an essential part of developers' day-to-day work for various purposes, e.g., problem-solving and self-learning. Particularly, Stack Overflow is the most popular and also the largest SQA site[1]. According to the latest developers survey of SO[2], as of July 2020, it already has over 13 million registered users and 20 million questions. Moreover, about 50 million people visit Stack Overflow each month and more than 7.5 thousand questions are raised every day. The huge amount of data in SQA sites constitutes a core knowledge asset for the software engineering domain. At the same time, based on the data in SQA sites, a large number of machine learning approaches have been proposed to interpret and make use of the knowledge from those sites. For example, some works aim to boost SQA sites by improving relevant question retrieval [118, 20], post classification [3, 10], and tag recommendation [114, 105]. Also, there are various works that propose solutions to automate software development activities by leveraging SQA data, e.g., program repair [65, 132], refactoring [98, 112], etc.

[1]https://stackoverflow.com
[2]https://insights.stackoverflow.com/survey/2020

n this dissertation, extending on the body of work that analyzes SQA, we propose the following three new angles to interpret knowledge from SQA sites: knowledge linking, representation, and summarization. We refer to a Stack Overflow thread consisting of a question along with all its answers as a *knowledge unit*. As shown in Figure 1.1, knowledge representation focuses on producing meaningful distributed representation for individual knowledge units, knowledge linking aims to identify the relatedness between any pair of knowledge units, and knowledge summarization is mainly about generating summary based on a cluster of related knowledge units. For each of them, we raise a specific research problem as follows.

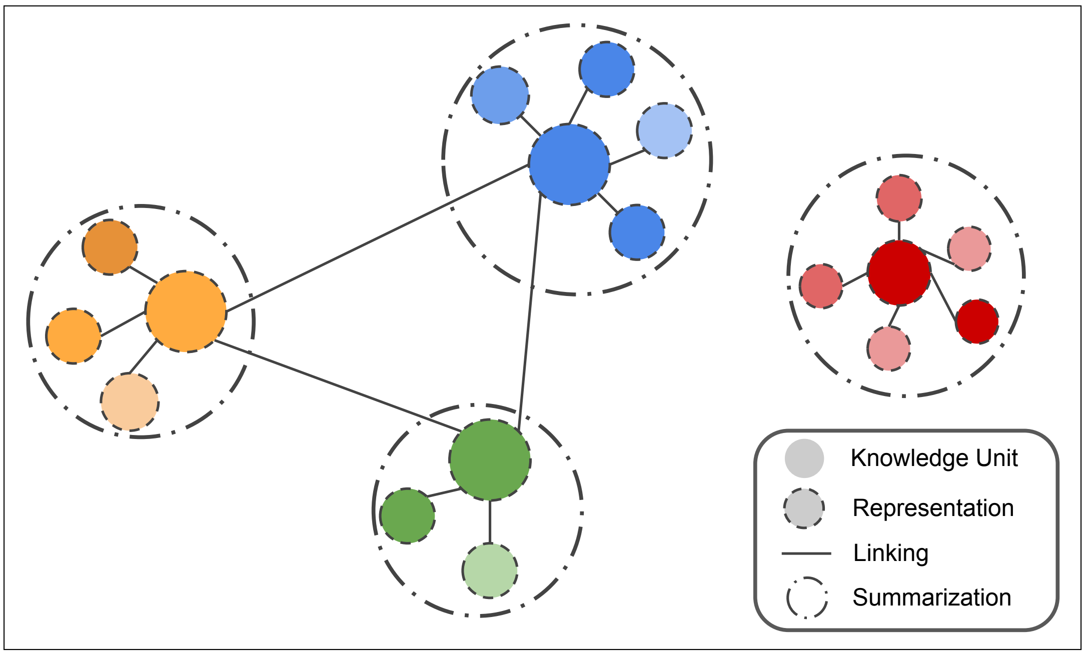


Figure 1.1: Three Angles to Interpret Knowledge Units in Software Q&A Sites

1. **Knowledge linking**. The research problem for knowledge linking is about linkable knowledge unit prediction. In this problem, following [126], we consider a question and its answers within a post in Stack Overflow as a *knowledge unit*. Knowledge units often contain semantically relevant knowledge, and thus linkable for different purposes. Being able to classifying different classes of linkable knowledge units would support more targeted information needs when users search or explore the linkable

knowledge. For example, duplicate questions allow users to understand a problem from different aspects, directly linkable questions help explain concepts or sub-steps of a complex problem, while indirectly-linkable questions provide extended knowledge. Prior works [94, 75] focus on binary relatedness (i.e., related or not), and are not robust to recognize different classes of semantic relatedness when linkable knowledge units share few words in common. In [119], Xu et al. used a deep neural network (DNN) technique to classify the degree of relatedness between two knowledge units (question-answer threads) on Stack Overflow. More recently, extending Xu et al.'s work, Fu and Menzies proposed a simpler classification technique based on a fine-tuned support vector machine (SVM) that achieves similar performance but in a much shorter time [29]. Thus, they suggested that researchers need to compare their sophisticated methods against simpler alternatives. However, we found that both Xu et. al. [119], and Fu and Menzies [29] perform experiments on a small dataset that contains only 8,000 pairs of knowledge units. Thus, it motivates us to further extend the research problem and aim to replicate the previous studies and further investigate the validity of Fu and Menzies' claim by evaluating the DNN- and SVM-based approaches on a larger dataset.

2. **Knowledge representation**. The research problem for knowledge representation is about distributed representation for Stack Overflow posts. Past studies have proposed solutions that analyze Stack Overflow content to help users find desired information or aid various downstream software engineering tasks. We found that a common step performed by those solutions is to extract suitable representations of posts; typically, in the form of meaningful vectors. These vectors are then used for different tasks, for example, tag recommendation [114, 105], relatedness prediction [119, 29], and post classification [10, 2]. Intuitively, following the

"garbage in, garbage out" principle, the quality of the vector representations of posts determines the effectiveness of the solutions in performing the respective tasks. It inspires us to propose an approach that learns distributed representations of posts. Once the post representations are learned, they can be used to boost the effectiveness of many supervised learning tasks, especially those rely only on a limited set of labeled data.

3. **Knowledge summarization**. The research problem for knowledge summarization is about answer summary generation for technical questions. Although search engines can return a list of questions relevant to a user query of some technical question, the abundance of relevant posts and the sheer amount of information in them makes it difficult for developers to digest them and find the most needed answers to their questions. We survey 72 developers in two IT companies with two questions: *(1) whether you need a technique to provide direct answers when you post a question/query online and why?* and (2) *what is your expectation of the automatically generated answers, e.g., must the answers be accurate?* All the developers agree that they need some automated technique to provide direct answers to a question/query posted online. The reasons they give include (1) sometimes it is hard to describe the problem they meet, so some hints would be useful, (2) there is too much noisy and redundant information online, (3) the answers in long posts are hard to find, and (4) even the answer they found may cover only one aspect of the problem. Developers expect the answer generation tool to provide a succinct and diverse summary of potential answers, which can help them understand the problem and refine the queries/questions. Our survey reveals a great need to provide improved techniques for information retrieval and exploration. Motivated by the survey result, we develop an automated technique for generating answer summary to developers' technical questions, instead of merely returning answer posts.

1.2 Contribution

The contributions of this dissertation are as follows:

- **Linkable knowledge prediction**. We replicate two prior studies on predicting relatedness of Stack Overflow knowledge units using a much larger and cleaner dataset [119, 29]. Our study confirms some findings reported in prior works, highlights and explains some discrepancies, and points out to challenges unsolved by prior works. To address one of the challenges (i.e., high runtime cost of Xu et al.'s and Fu and Menzies' approaches), we investigate the value of an alternative lightweight method (SimBow). We demonstrate that it can outperform prior baselines in terms of runtime cost by a large margin, while achieving a similar accuracy.

- **Representations of Stack Overflow posts**. We propose a specialized deep learning architecture Post2Vec that learns vector representations of SQA posts. To the best of our knowledge, this work is the first to explore the learning of generic representation of SQA posts. We focus on Stack Overflow posts, but the solution presented here can be easily adapted for posts from other SQA sites. We empirically compare end-to-end Post2Vec with the state-of-the-art neural network based approach for tag recommendation task based on a 10 million posts dataset, and demonstrate that Post2Vec can achieve 11-20% improvements and also complete the learning process around 19 hours faster. We empirically investigate the benefit of considering code snippets for post representation, by implementing a reduced version of Post2Vec that ignore code snippets in posts (i.e., Post2Vec^{-Code}). The experimental results show that the consideration of code snippets improves the quality of the learned post representation. We empirically investigate the value of integrating the post vectors generated by Post2Vec and feature vectors used by two state-of-the-art approaches of two tasks (relatedness prediction and post classification) and demonstrate substantial improvements.

- **Answer summary generation for technical questions**. We conduct a formative study to assess the necessity of automated question answering techniques to provide answer summary to developers' technical questions. We formulate the problem of automated question answering as a query-focused multi-answer-posts summarization task for an input technical question. We propose a three-stage framework to solve the task, i.e., 1) relevant question retrieval, 2) answer paragraphs selection, 3) answer summary generation. We conduct user studies to evaluate the effectiveness of our approach and its components, and identify several areas for future improvements.

1.3 Dissertation Structure

The remaining part of this dissertation is as follows. We first review related works in Chapter 2. Chapter 3 presents our work on linkable knowledge unit prediction which we systematically compare traditional machine learning and deep learning approaches. Chapter 4 describes our work on the distributed representation of Stack Overflow posts. Chapter 5 presents our work on answer summary generation for technique questions by leveraging Stack Overflow data. Finally, we summarize our works and describe future research directions (Chapter 6).

Chapter 2

Related Work

2.1 Overview of Interpreting Knowledge in SQA

In the literature, there are plenty of research works that have been done which aim to interpret knowledge in SQA for software development. These works analyze SQA knowledge for various purposes. For simplicity's sake, we divide all the purposes into two categories, "SQA knowledge design and usage" and "SQA knowledge-based applications". Next, we introduce related works for the two categories respectively.

SQA knowledge design and usage. Many works aim to understand and discover the SQA knowledge in a deeper way to indirectly serve for developers. Ye et al. observed that due to the cross-reference of questions and answers via URL, knowledge is diffused in the Q&A site, forming a large knowledge network [126]. And they further performed quantitative analysis to study the structural and dynamic properties of the emergent knowledge network in Stack Overflow. Through this study, they obtained an in-depth understanding of the knowledge diffusion process in Stack Overflow and exposed the implications of URL sharing behavior for SQA site design, developers who use crowdsourced knowledge in Stack Overflow, and future research on knowledge representation and search. One of the most common reasons that two posts are linked in SQA sites is because they are duplicate and many approaches have been proposed for duplicate posts detection, e.g., [3, 89, 131]. For example, Ahasanuzzaman et al. found that the top-3 reasons on why the duplicate posts are submitted are (1) questioners do not search before they ask, (2) the title of the duplicate questions do not match their questions, (3) compare with previous questions, the question is for a different application or domain category [6]. And based on the result, Ahasanuzzaman et al. proposed a classification approach that uses a number of handcrafted features to identify duplicate questions. Moreover, various types of analysis on SQA data have been performed on knowledge sharing [31, 119], learning [129, 130], and searching [62, 18].

SQA knowledge-based applications. A large number of studies focus on leveraging the knowledge in SQA for certain software tasks. Take code search by leveraging SQA knowledge as an example, Mohammad and Chanchal proposed a technique that automatically identifies relevant and specific API classes from SQA sites for a programming task written as a natural language query [82]. Similarly, Sirres et al. proposed a code search engine on top of Stack Overflow and Github [90]. There are also many works that utilize SQA knowledge for bug fixing. For example, Islam et al. studied 415 repairs from Stack Overflow and Github to understand challenges in repairs and bug repair patterns [47]. Moreover, they also contributed a benchmark of 667 deep neural network (bug, repair) instances. Gao et al. proposed an automatic approach to fix recurring crash bugs via analyzing SQA sites [30]. By extracting queries from crash traces and retrieving a list of SQA posts, they analyze the pages and generate edit scripts. Then they apply these scripts to target source code and filter out the incorrect patches.

The three research topics of this dissertation cover both two above mentioned categories. More specifically, knowledge summarization aims to directly serve developers while knowledge linking and representation target better manage the knowledge.

2.2 Knowledge Linking

Ye et al. defined a question and its answers on Stack Overflow as a *knowledge unit* [126]. They found that knowledge units often contain semantically relevant knowledge, and thus linkable for different purposes. Recognising different classes of linkable knowledge would support more targeted information needs when programmers search or explore the knowledge base. Naturally, the linkable knowledge prediction problem is formulated as a multi-class classification problem.

Xu et al. firstly demonstrate that being able to classify different classes of linkable knowledge units would support more targeted information needs when users search or explore the linkable knowledge [119]. For example, two knowledge units will be labeled as duplicate if they talk about the same technical problems but in different ways. Then they divided all the relationships between two knowledge units into four categories based on the relatedness, duplicate, direct, indirect, and isolated. To identify related content, a convolutional neural network (CNN) model can be trained to predict the relatedness between knowledge unit pairs. The model first utilizes word embedding technique [71] to capture word-level semantics and represent knowledge units into vectors. Then, the vectors are fed to a basic CNN which is used to capture the sentence-level semantics. More recently, extending Xu et al.'s work, Fu and Menzies proposed a simpler classification technique based on a fine-tuned support vector machine (SVM) that achieves similar performance but in a much shorter time [29]. They adopt differential evolution [95] as the tuning algorithm for the SVM. Thus, they suggested that researchers need to compare their sophisticated methods against simpler alternatives.

Previous approaches measure the text similarity between the content of knowledge units and then map the similarity to a particular level of relatedness. Thus, the problem is related to the pairwise multi-class document classification in the field of natural language processing (NLP). There are many document similarity measurements that have been proposed in the NLP domain. Typically, the documents are represented as numeral features that can be calculated directly, e.g., bag-of-words model. And then, distance measures are used to compute the similarity, e.g., cosine distance. The most related work is that Ostendorff et al. built a new dataset for predicting the relationships (e.g., country of citizenship, different from) between pairs of main entities in Wikipedia articles into nine categories based on their content [76]. And they investigated a series of techniques, such as GloVe [78], Paragraph-Vectors [57], vanilla BERT [25], and vanilla XLNet [122]. They find that vanilla BERT achieves the best performance. The above mentioned and more recent approaches should be taken into consideration as baselines for linkable knowledge units prediction in the future works.

To tackle the problem, the approach we proposed in this dissertation aims to not achieve better effectiveness but also be more scalable than the existing approaches. To achieve this, we design more effective handcraft feature representation for knowledge units and utilize a scalable variant of SVM classifiers at the same time. For more details, please refer to Chapter 3 (Linkable Knowledge Prediction).

2.3 Knowledge Representation

To serve certain tasks, knowledge in SQA sites are represented in various ways which are designed to carry meaningful features. Intuitively, the better representation for the knowledge tends to better performance of approaches.

One of the most common means for knowledge representation is treating knowledge (or knowledge units) as text and applying various text representation techniques from the field of natural language processing. For example, Xia et al. proposed a tag prediction approach for Stack Overflow posts named TagCombine. It uses the bag-of-words (BOW) model to represent posts [114].

Another example is relevant question retrieval in Stack Overflow, Xu et al. propose an approach that represents a Stack Overflow question as a set of keywords [118]. Fischer et al. utilized the TF-IDF model to represent code snippets in Stack Overflow and evaluated their security score by using a stochastic gradient descent classifier [28]. The above-mentioned text representation techniques belong to a family of the text representation techniques named discrete text representation, i.e., words are represented by their corresponding indexes to their position in a dictionary from a larger corpus. The advantage of discrete text representation is that it is easy to understand, implement, and interpret. More recently, distributed text representations have gained great attention which is typically learned by deep learning approaches. For example, Zhou et al. have proposed four different deep learning approaches, TagCNN, TagRNN, TagHAN and TagRCNN, which are based on convolutional neural networks (CNN), recurrent neural networks (RNN), hierarchical attention networks (HAN), and recurrent convolutional neural network (RCNN), respectively [133]. All four approaches concatenate the text in the title and description of posts as the input data and then represent each post as a distributed vector to predict the posts' tags. Chen et al. proposed an approach based on word embeddings and convolutional neural network (CNN) to capture word and sentence-level semantics [20]. The approach was trained based on pairs of posts with labeled relevance, each of the posts is represented as a vector and the two vectors of the posts are mapped in a dual vector space. The labeled relevance is used to guide the approach to embed semantically similar posts close in vector space.

Different with above mentioned works, we propose a deep learning-based approach to represent posts in a more general way. Thus, the learned representation can be further leveraged to serve multiple downstream tasks, e.g., API recommendation. Moreover, to learn the knowledge representation with high quality, we consider the knowledge units (i.e., posts) as structured data units with multiple components and utilize different neural networks to learn the representation of each component separately. Furthermore, we also address several limitations of the existing approaches for knowledge representation. For more details, please refer to Chapter 4 (Distributed Knowledge Representation).

2.4 Knowledge Summarization

In the field of NLP, there are many text summarization techniques that have been proposed. There are two main forms of them, extractive and abstractive [74]. A typical extractive text summarization technique finds the most informative sentences within a large body of text which are used to form a summary (e.g., LexRank [27]) while a typical abstractive technique generates concise phrases that are semantically consistent with the large body of text (e.g., encoder-decoder recurrent neural network [73]).

In the literature, there are many works that focus on knowledge summarization for SQA data, e.g., summarization for API and code snippets. For API summarization, to investigate the usefulness of the APIs, Gias and Foutse developed an extractive summarization approach that presents summaries of opinions based on Stack Overflow [101]. Specifically, based on the API reviews from developers in Stack Overflow, the approach generates summaries of opinions for an API from a set of pre-defined aspects (e.g., performance, security). For each aspect, a corresponding classifier (i.e., SVM) is built to predict whether a given review belongs to the aspect. Another example is code summarization, Peddamail et al. investigated the performance of two abstractive code summarization approaches (Codenn [48] and DeepCom [44]) on a dataset constructed based on Stack Overflow data [77]. The dataset is in the form of a set of pairs and each pair corresponds to a natural language question, code solution. Codenn uses an end-to-end generation system to perform content selection and surface realization jointly. The core component of Codenn is a LSTM-based recurrent neural network with an attention mechanism [66], which models the probability of a natural language summary conditioned on the given code snippet. DeepCom is built upon advances in Neural Machine Translation (NMT) by considering generating summaries in natural language as a variant of the NMT problem, where code snippets written in a programming language needs to be translated to text in natural language.

The knowledge summarization task formulated in this dissertation is different from the above mentioned tasks from several perspectives. First, we focus on generating answer summaries for technical questions in general. In other words, the query can be a mixture of text and code. Second, the above mentioned works do not consider the redundancy across the different descriptions of knowledge. In our approach design, we remove the redundancy by calculating the similarity between knowledge description candidates and apply a text summarization technique by Maximal Marginal Relevance (MMR) [14] to select a most representative subset of them. For more details, please refer to Chapter 5 (Automated Knowledge Summarization).

Chapter 3

Linkable Knowledge Prediction

3.1 Introduction

Using machine learning techniques in software engineering research has been commonplace, such as [50, 96, 119] to name few. The applicability of machine learning techniques depends on the hypothesis class that can be represented by them. That is, the functions that they can represent. For examples, linear regression models are very effective for linearly separable problems (i.e., classes can be separated with a single decision surface), but they cannot be used for problems with higher complexity.

Neural networks constitute a powerful class of machine learning models with large hypothesis class. For example, a multilayer feed-forward network is called a universal approximator [42]; that is, it can essentially represent any function. Deep neural networks methods are representation learning methods that allow a method to use raw data and extract the representation of the data [58]; it can substantially reduce the burden of feature engineering. Deep learning has produced promising results in complex tasks such as object detection [97], natural language understanding [87], text classification [56] and many more.

Nowadays, there has been a surge in adoption of deep learning[1] in software

[1]We use two terms deep learning, and deep neural networks interchangeably.

engineering research. It has been applied successfully to problems such as [119, 35, 128]. A common issue raised in the application of deep learning techniques is that sometimes deep neural networks are applied to problems that do not require the rich, complex hypothesis class that deep learning offers, and simpler techniques can be used as effectively instead. The simplicity of models is desirable for two main reasons. First, simpler models are easier to interpret and comprehend and comprehension of relations between variables can afford useful insights about the underlying phenomena. Second, simpler models can be trained more efficiently, and potentially with smaller dataset.

Recently, Xu et al. [119] and Fu and Menziess [29] investigated the problem of predicting relatedness between Stack Overflow knowledge units. Xu et al. use deep neural networks (DNN) for the task, while Fu and Menzies [29] use a support vector machine (SVM) tuned by using differential evolution (DE). Fu and Menzies reported benefits of using the simpler model; that is, similar accuracy can be achieved with lower runtime cost. In this paper, we replicate the evaluation of the two techniques on the same software engineering task, but using a much larger dataset. Our goal in this study is to evaluate the consistency of claims made by these prior studies. Replication studies are often instrumental to assess the validity of previous findings, uncover new insights, as well as investigate the impact of some threats to validity affecting prior work [9].

In our experiments, we find that the dataset used to evaluate both approaches has a number of shortcomings. Once we addressed those shortcomings, by creating a larger dataset that is subjected to a more thorough data cleaning step, we observed that the performance of the both techniques (evaluated using F1-score) drops sharply by more than 20%. We found that still Fu and Menzies' SVM-based model performs slightly better than Xu et al.'s DNN-based model – consistent with the findings in [29]. However, in terms of time efficiency, the runtime cost required to tune SVM using DE grows by a large amount when the dataset is increased in size. As a result, the performance benefit of using Fu and Menzies' SVM-based model is no longer observed when it is evaluated on the new dataset. Addressing this drawback, we adapt a lightweight award-winning SVM-based model named SimBow [17] for the task and evaluate its effectiveness. We demonstrate that SimBow requires much less runtime cost as compared to Xu et al. and Fu and Menzies approaches, while achieving similar accuracy.

The contributions of this work are as follows:

- We replicate two previously presented studies on predicting relatedness of Stack Overflow knowledge units using a much larger and cleaner dataset. Our study confirms some findings reported in prior works, highlights and explains some discrepancies, and points out to challenges unsolved by prior works.

- To address one of the challenges (i.e., high runtime cost of Xu et al.'s and Fu and Menzies' approaches), we investigate the value of an alternative lightweight method (SimBow). We demonstrate that it can outperform prior baselines in terms of runtime cost by a large margin, while achieving a similar accuracy.

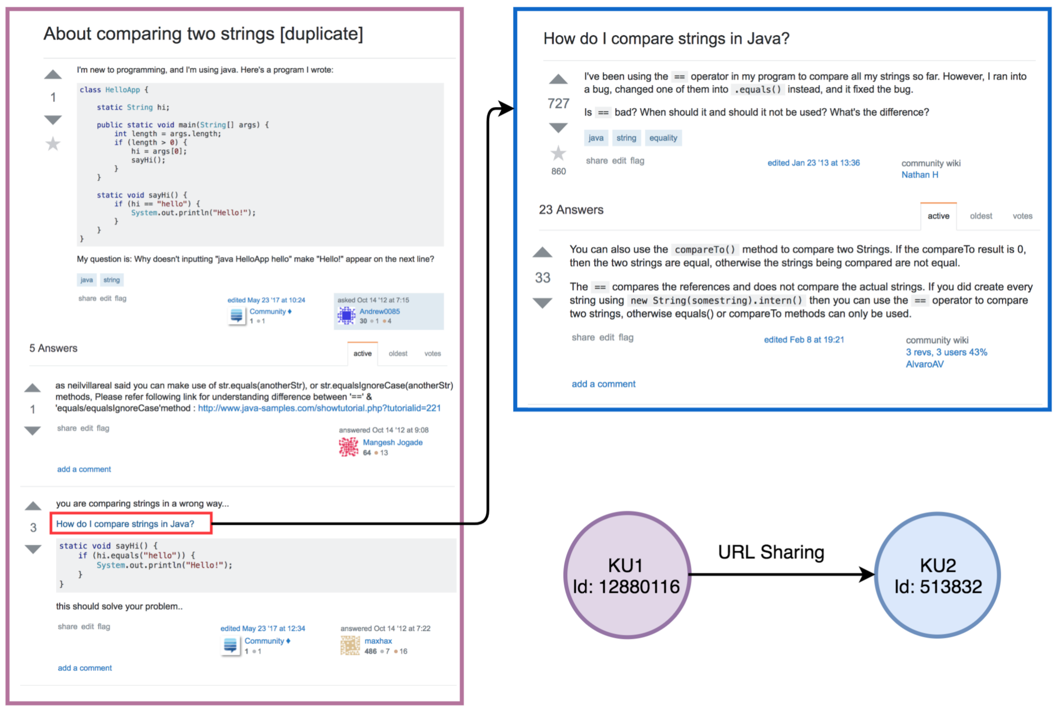
3.2 Task and Evaluation Metrics

3.2.1 Predicting Relatedness in Stack Overflow

Software developers must solve numerous programming, algorithmic, and system problems to write, maintain, or deploy programs. Knowledge about these problems is dispersed in many books and user manuals that are hard to locate and use. Therefore, developers often use technical forums to use crowd's knowledge and seek solutions to those problems.

Among technical forums, Stack Overflow is the most popular resource for programming related discussions. Stack Overflow reputation system has attracted many developers to participate actively and contribute to this forum. Most Stack Overflow questions are answered within 11 minutes after posting them [68]. Stack Overflow allows users to search, post, or answer questions. It also allows users to vote up and down questions and answers. Nowadays, Stack Overflow is an indispensable tool for programmers; a recent study shows that about 50 million developers visit it monthly, and over 85% users visit Stack Overflow more than four times a week.[2]

Following Xu et al. [119], we refer to a Stack Overflow thread consisting of a question along with all its answers as a *knowledge unit* (KU). Despite Stack Overflow's vibrant community, knowledge in Stack Overflow is disconnected and developers must search for *related knowledge units* that provide additional insights about their problem and possible solutions that can be very time-consuming.


About comparing two strings [duplicate]

I'm new to programming, and I'm using java. Here's a program I wrote:

```
class HelloApp {

    static String hi;

    public static void main(String[] args) {
        int length = args.length;
        if (length > 0) {
            hi = args[0];
            sayHi();
        }
    }

    static void sayHi() {
        if (hi == "hello") {
            System.out.println("Hello!");
        }
    }
}
```

My question is: Why doesn't inputting "java HelloApp hello" make "Hello!" appear on the next line?

edited May 23 '17 at 10:24 Community ♦ 1 • 1

asked Oct 14 '12 at 7:15 Andrew0085 30 • 1 • 4

5 Answers

as neilvillareal said you can make use of str.equals(anotherStr), or str.equalsIgnoreCase(anotherStr) methods, Please refer following link for understanding difference between '==' & 'equals/equalsIgnoreCase'method : http://www.java-samples.com/showtutorial.php?tutorialid=221

answered Oct 14 '12 at 9:08 Mangesh Jogade 64 • 13

you are comparing strings in a wrong way...

How do I compare strings in Java?

```
static void sayHi() {
    if (hi.equals("hello")) {
        System.out.println("Hello!");
    }
}
```

this should solve your problem..

edited May 23 '17 at 12:34 Community ♦ 1 • 1

answered Oct 14 '12 at 7:22 maxhax 486 • 7 • 16

How do I compare strings in Java?

I've been using the `==` operator in my program to compare all my strings so far. However, I ran into a bug, changed one of them into `.equals()` instead, and it fixed the bug.

Is `==` bad? When should it and should it not be used? What's the difference?

edited Jan 23 '13 at 13:36

community wiki Nathan H

23 Answers

You can also use the `compareTo()` method to compare two Strings. If the compareTo result is 0, then the two strings are equal, otherwise the strings being compared are not equal.

The `==` compares the references and does not compare the actual strings. If you did create every string using `new String(somestring).intern()` then you can use the `==` operator to compare two strings, otherwise equals() or compareTo methods can only be used.

edited Feb 8 at 19:21

community wiki 3 revs, 3 users 43% AlvaroAV

Figure 3.1: Linked Knowledge Units by URL Sharing

Identifying relatedness of knowledge units would accelerate developer's abil-

[2]Stack Overflow 2018 Developer Survey, `https://insights.stackoverflow.com/survey/2018/`

Table 3.1: Classes of Knowledge Unit Pairs

Link Type	Definition
Duplicate	Two knowledge units discuss the same question in different ways, and can be answered by the same answer.
Direct	One knowledge unit can help solve the problem in the other knowledge unit, for example, by explaining certain concepts, providing examples, or covering a sub-step for solving a complex problem.
Indirect	One knowledge unit provides related information, but it does not directly answer the question in the other knowledge unit.
Isolated	The two knowledge units are not semantically related.

ity in navigating the rich and yet diverse information in Stack Overflow. Thus, Stack Overflow encourages developers to link related knowledge units by URL sharing [93]. Figure 3.1 shows a real example of how two knowledge units are linked by developers. A network of linkable knowledge units constitutes a *knowledge unit network* over time through URL sharing [125]. As shown in Table 3.1, Xu et al. divided all the relationship between two knowledge units into four categories based on relatedness, i.e., duplicate, direct, indirect and isolated [119]. To identify related contents, a model can be trained to predict the relatedness between KU pairs. There are multiple challenges for predicting relatedness of KUs in Stack Overflow. First, there is informal, redundant, irrelevant information in KUs. Secondly, in addition to natural text, KUs contain source code, which is of a different nature. Thirdly, different developers exhibit different discursive habits in posting questions and answers; e.g., some questions or answers are very terse, while some are very long and tend to include much information.

Table 3.2 shows real examples of pairs of knowledge units with different degrees of relatedness. The original knowledge unit is talking about *String comparison in Java*. Another knowledge unit on Stack Overflow is labeled as duplicate with the original knowledge unit because they actually talk about the same problem but in different ways. Thus, the answers of original knowledge unit and duplicate knowledge unit can be shared. Another knowledge unit talks about a similar but not identical problem, i.e., *how does == works in case of String concatenation in Java.* Thus, based on the definition, there is a *direct* relationship between the two knowledge units. Consider yet another knowledge unit that discusses *memory change during string concatenation in Java.* We regard it as an indirect knowledge unit to the original knowledge unit, because it is directly linked to one of the direct knowledge units of the original knowledge unit. The order of semantic relatedness between two knowledge units is: *Duplicate* > *Direct* > *Indirect* > *Isolated.* For the details of dataset building, please refer to Section 3.4.

3.2.2 Evaluation Metrics

To evaluate the performance of the proposed approaches in the prediction of relatedness between knowledge units, we use the same metrics as used in previous works [119, 29], i.e., precision, recall and f1-score. In this task, the classifier has to classify each pair of knowledge units into four classes. Table 3.3 depicts the confusion metrics when we have four classes.

Base on the confusion matrix, the definitions of precision, recall and F1-score are as below:

Precision for a class i is the proportion of knowledge-unit pairs correctly classified as the class i among all pairs classified as the class i.

$$Precision_j = \frac{C_{ii}}{\sum_{1 \le j \le K} C_{ji}}$$

Recall for a class i is the percentage of knowledge-unit pairs correctly classified as the class i compared with the number of ground truth label L_i in the dataset.

$$Recall_i = \frac{C_{ii}}{\sum_{1 \le j \le K} C_{ij}}$$

Table 3.2: Example of Duplicate, Direct, Indirect Knowledge Units Pairs

[Original KU] (`https://stackoverflow.com/questions/513832`)	
Title	How do I compare strings in Java?
Description	I've been using the == operator in my program to compare all my strings so far. However, I ran into a bug, changed one of them into .equals() instead, and it fixed the bug. Is == bad? When should it and should it not be used? What's the difference?
[Duplicate KU] (`https://stackoverflow.com/questions/3281448`)	
Title	Strings in Java : equals vs ==

Description	`String s1 = "andrei";`
	`...`
	`System.out.println((s1==s2) + "␣" + (s2==s3));`
	Giving the following code why is the second comparison s2 == s3 true ? What is actually s2.toString() returning ?
	Where is actually located (s2.toString()) ?

[Direct KU] (`https://stackoverflow.com/questions/34509566`)	
Title	"==" in case of String concatenation in Java

Description	`String a = "devender";`
	`...`
	`System.out.println(a == e);  //case 3: o/p false`
	a & b both are pointing to the same String Literal in string constant pool. So true in case 1
	`String d = "dev" + "ender";`
	should be internally using something like -
	`String d = new StringBuilder().append("dev")`
	`.append("ender").toString();`
	How a & d are pointing to the same reference & not a & e ?

[Indirect KU] (`https://stackoverflow.com/questions/11989261`)	
Title	Does concatenating strings in Java always lead to new strings being created in memory?

Description	I have a long string that doesn't fit the width of the screen. For eg.
	`String longString = "This␣string␣is␣very␣long...";`
	To make it easier to read, I thought of writing it this way -
	`String longString = "This␣string␣is␣very␣long..."`
	`+ "This␣string␣is␣very␣long..." + ...;`
	However, I realized that the second way uses string concatenation and will create 5 new strings in memory and this
	might lead to a performance hit. Is this the case? Or would the compiler be smart enough to figure out that all I need
	is really a single string? How could I avoid doing this?

Table 3.3: Confusion Matrix

		Predicted as			
		C1	C2	C3	C4
Actual Label	C1	C_{11}	C_{12}	C_{13}	C_{14}
	C2	C_{21}	C_{22}	C_{23}	C_{24}
	C3	C_{31}	C_{32}	C_{33}	C_{34}
	C4	C_{41}	C_{42}	C_{43}	C_{44}

F1-score for a class i is a harmonic mean of precision and recall for that class.

$$F1_i = \frac{2 \times Precision_i \times Recall_i}{Precision_i + Recall_i}$$

3.3 Replication

This section overviews the techniques for predicting relatedness. The techniques are as follows, we refer to Xu et al., Fu and Menzies, and SimBow techniques as CNN Model, Tuning SVM, and SoftSVM, respectively.

- **CNN Model, Xu et al.** [**119**]: Appeared in ASE 2016.
- **Tuning SVM, Fu and Menzies** [**29**]: Appeared in FSE 2017.
- **SoftSVM, SimBow** [**17**]: Appeared in SemEval-2017 Task 3: Community Question Answering.

3.3.1 Xu et al.'s Study (CNN Model)

At ASE 2016, Xu et al. [119] presented the task of predicting relatedness of knowledge units, and proposed a deep learning approach for it. In this section, we briefly review their approach. For more technical details, please refer to the original paper [119].

Deep learning is a class of machine learning techniques that can be used for classification or regression tasks. Deep learning has produced impressive results in domains such as image processing and natural language processing where feature engineering has been traditionally challenging.

Deep learning trains a weighted neural network for the learning task. A neural network comprises a group of interconnected *neurons* organized in multiple layers. A neuron is the smallest unit of computation in the networks. Each neuron performs a dot product on the input vector X and weights vector W, then, it adds the bias b; finally, it applies the activation function f (or non-linearity) to the result.

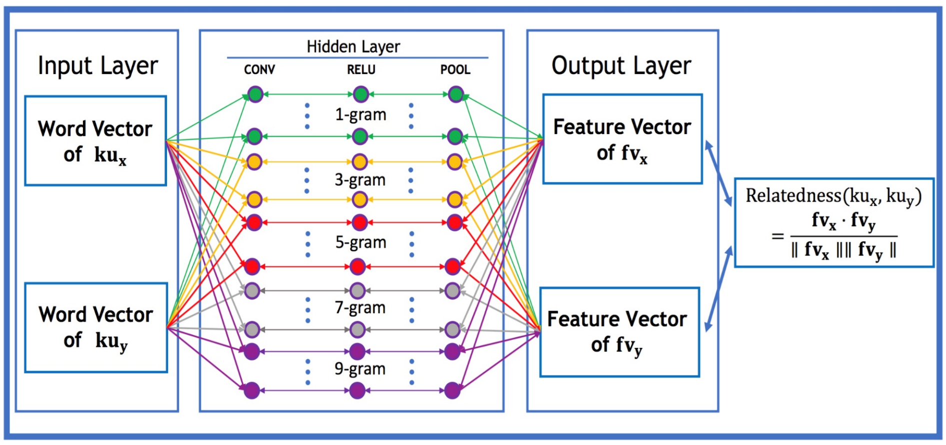


Figure 3.2: CNN Architecture in CNN Model

Overview of Approach To predict the relatedness between knowledge units, Xu et al. built a convolutional neural network (CNN) model [59] using a word embedding trained on Stack Overflow data to capture low- and high-level representations of KU pairs.

To extract low level (i.e., word-level) semantic features, each word is represented by a 200-dimension vector by utilizing a word2vec model [71]. The word2vec model is created using a corpus of 100,000 Java knowledge units (i.e., posts tagged with “java”). And continuous skip-gram model [71] is used to learn domain-specific word embeddings from the corpora. The embeddings for the words were initialized using the trained word embeddings. Zero vector is used for padding the shorter sequences and representing the missing words in the pre-trained vectors.

Then, a convolutional neural network model is built on top of that to extract high level (i.e., document-level) semantic features. The convolutional neural network is a class of deep learning techniques, feed-forward artificial neural networks. A convolutional neural network consists of an input and an output layer, as well as hidden layers. The hidden layer's parameters consist of a set of learnable filters. Figure 3.2 shows the overview of the CNN model which is fed by pairs of knowledge units as input and then uses CNN to extract features from a knowledge unit (in the form of a sequence of word vectors). The filters of five different window sizes (the number of adjacent words considered jointly, in their case, i.e., 1, 3, 5, 7, 9) are utilized to capture the most informative n-grams in the text. For each window size, there are 128 filters to learn complementary features from the same word windows. $Relu$ is used as activation function (i.e., $Relu(x) = max(0, x)$) and Max Pooling is used in the sampling process.

The input of the model is two high-dimensional text vectors of two given knowledge units and the output are two low-dimensional semantic feature vectors. The relatedness between two knowledge units are computed as the following equation:

$$Relatedness(KU_x, KU_y) = \frac{fv_x \cdot fv_y}{\|fv_x\| \, \|fv_y\|}$$

where fv_x and fv_y denote two low-dimensional (in this case, 50-dimension) feature vectors generated by CNN. Then, the loss is computed as the absolute difference between cosine similarity of two feature vectors and ground truth relatedness.

Replication To replicate the experiments described in [119], we use the source code released by Xu et al. [3] and apply it to our dataset. Although the neural networks are usually trained using GPUs, the implementation of this approach is CPU-based. In our replication, we ran the experiment on a MacBook Pro with Intel(R) Core(TM) i7-4870HQ 2.5 GHz, 16GB RAM, running macOS High Sierra(64-bit).

[3] `https://github.com/XBWer/ase16-CNN`

3.3.2 Fu and Menzies' Study (Tuning SVM)

In FSE 2017, Fu and Menzies [29] proposed a different technique for predicting relatedness of pairs of knowledge units. This section provides a brief overview of their technique.

Fu and Menzies argue that CNN models described in the Section 3.3.1 is computationally too expensive for the task of predicting relatedness of knowledge units. They propose tuned support vector machines for this problem.

Feature representation To represent a pair of knowledge units, Fu and Menzies first build a corpus with 100,000 knowledge units and then train a word embedding model. By querying the trained model, each word is converted to its vector representation. Next, the whole knowledge unit is represented by the mean of the word embeddings of the words in a knowledge unit.Then, a pair of knowledge units is represented by the mean of the two knowledge unit vectors which is used as the input data to SVM.

SVMs Support Vector Machines (SVMs) are supervised learning models used mainly for classification. In their basic form, SVMs learn linear threshold function. These learners seek to minimize misclassification errors by selecting a boundary or hyper-plane that leaves the maximum margin between two classes [51].

Parameter Tuning Fu and Menzies [29] use word2vec representation [71] as features and SVM as a classifier in this study. The word2vec model is trained on 100,000 Java knowledge units in Stack Overflow using the skip-gram model. They use differential evolution (DE) [95] to tune the conventional support vector machine model. Authors use the same training and testing knowledge unit pairs as in Xu et al.'s study [119], where 6,400 pairs of knowledge units for training and 1,600 pairs for testing. During the parameter tuning procedure, 10-fold cross-validation is performed to reduce the potential variance caused by how the original training data is divided. Therefore, all the performance scores used for tuning are averaged values over 10 runs. They use F1-score to score the candidate parameters because it controls the trade-offs between precision and recall.

Replication We carefully followed the steps outlined in [29] to replicate the study. We used the source code released for Tuning SVM [4] and apply it to our dataset. That is, we use the same word2vec as theirs and we also apply DE to find the optimal parameters for the SVM training. The objective of the parameter optimization is to maximize the F1-score of the underlying SVM. Then the SVM model with optimal parameters is evaluated on testing data.

Unfortunately, Fu and Menzies's implementation spent more than one week without returning any result. We found that the approach executes through 10 pre-trained word2vec models with different seeds and perform 10 fold-cross validation for each model. Thus, to further improve the time efficiency, we modify the code to execute the code on ten word2vec instances in parallel. We deploy it on an HPC cluster with Intel Xeon E5-2680 v2 2.8 GHz CPUs, and 64 GB RAM nodes.

3.3.3 SimBow: A Lightweight Alternative (SoftSVM)

In this section, we describe the paper "SimBow at SemEval-2017 Task 3: Soft-Cosine Semantic Similarity between Questions for Community Question Answering" (SimBow). Author's proposed approach is a supervised combination of different unsupervised textual similarities such as soft-cosine similarity. Unlike two previous system, this system is a re-ranking problem and is evaluated on natural text from Qatar living forum. The task aims at re-ranking 10 related questions proposed by a search engine, regarding the relevance to the original question.

Soft-Cosine Similarity Measure Classic cosine similarity measure between 2 vectors is directly related to the number of words which are in common in both which texts are represented by a vector of TF-IDF coefficients

[4]`https://github.com/WeiFoo/EasyOverHard`

(Equation 3.1).

$$cosine(a,b) = \frac{\sum_{n=1}^{N} a_i b_i}{\sqrt{\sum_{i=1}^{N} a_i^2}\sqrt{\sum_{i=1}^{N} b_i^2}} \tag{3.1}$$

The problem with traditional cosine similarity is that when there are no words in common between texts a and b, cosine similarity is null. However, two texts can semantically convey the similar meaning by using different words. This problem occurs repeatedly in our case where two semantically similar questions are depicted in different ways. Therefore, cosine similarity alone cannot be enough here.

Hence authors propose to take into account word-level relations by introducing the soft-cosine similarity formula with computing a relation matrix M, as suggested in equation 3.2.

$$soft-cosine(a,b) = \frac{\sum\sum_{i,j}^{N} a_i m_{ij} b_j}{\sqrt{\sum\sum_{i,j}^{N} a_i m_{ij} a_j}\sqrt{\sum\sum_{i,j}^{N} b_i m_{ij} b_j}} \tag{3.2}$$

where M is a matrix whose element $m_{i,j}$ expresses some relation between word i and word j. When computing this metric, the similarity between two texts is not null when the texts share related words, even if they have no words in common. Different ways are suggested for computing the matrix M. To obtain relevant semantic relations between words, authors compute soft-cosine similarity features based on two pre-trained word embeddings (Qatar living word2vec and Wikipedia word2vec) and one based on the Edit distance. Matrix M can be computed in different ways. Authors use the following equation(3.3) for computing soft-cosine similarity based on word embedding.

$$m_{ij} = max(0, cosine(v_i, v_j))^2 \tag{3.3}$$

where v_i stands for the word2vec representation of word w_i. Grounding to 0 is to avoid having negative cosines between words and is obtained empirically.

Likewise, for Edit distance-based measure, the matrix M is calculated as follows:

$$m_{ij} = \alpha * (1 - \frac{Levenshtein(w_i, w_j)}{max(||w_i||, ||w_j||)})^{\beta} \tag{3.4}$$

Where $||w||$ is the number of characters of the word, α is a weighting factor relatively to diagonal elements, and β is a factor that enables to emphasize the score dynamics. Authors set $\alpha = 1.8$ and $\beta = 5$ empirically.

SimBow for Relatedness Prediction We followed several simple, preprocessing steps: We replaced URLs and numbers with `URL` and `CC` respectively. Stop words and punctuations are removed and all letters are converted to lowercase. There are many technical terms in Stack Overflow so we need to have more data specific preprocessing steps. Therefore besides mentioned preprocessing steps, we split words by underline and capital letters. We also removed characters like $<$, $>$, (and).

We re-implement the SimBow's features from scratch on Stack Overflow data. In total four different features are extracted from the text. Cosine similarity, soft-cosine similarity based on Stack Overflow data (soft-SO), soft-cosine similarity based on pre-trained Google word2vec (soft-Google) [71] and soft-cosine similarity based on Levenshtein distance (soft-Edit).

For soft-cosine similarity features based on word embedding and based on Edit distance, we follow the same formulations as in SimBow. To compute soft-cosine similarity based on Stack Overflow data (soft-SO), we train a word2vec model on 223,466 knowledge units tagged with `java` from Stack Overflow posts table (include titles, bodies and answers). The skip-gram model [72] is used with vectors dimension 200 and only the words with a minimum frequency of 20 are taken into account.

We apply the same algorithm suggested for weighting the word2vec vectors with TF-IDF, where IDF is derived from the train and development sets.

We train a SVM model to classify question pairs into four classes. We tune the regularization parameter (C) using grid search technique over the best feature combination that includes all of the four extracted features. We

Table 3.4: Precision, Recall and F1-Score of CNN Model and Tuning SVM on Original Dataset

			Duplicate	Direct Link	Indirect Link	Isolated	Overall
Precision	Xu et al.	CNN Model	**0.89**	0.75	0.84	0.89	0.84
	Fu et al.	Tuning SVM	0.88	**0.85**	**0.94**	**0.90**	**0.89**
Recall	Xu et al.	CNN Model	**0.89**	**0.90**	0.77	0.79	0.842
	Fu et al.	Tuning SVM	0.86	0.82	**0.99**	**0.90**	**0.89**
F1-Score	Xu et al.	CNN Model	**0.89**	0.82	0.80	0.84	0.84
	Fu et al.	Tuning SVM	0.87	**0.84**	**0.96**	**0.90**	**0.89**

use the best parameter value (C=100) with the linear kernel for training the model with all the training and development data and used that model for predicting the labels in the test data. This system is performed on Intel(R) Xeon(R) CPU E5-2667 v4 @ 3.20GHz. We report the runtime of this system on the Stack Overflow dataset to be around 1.5h for feature extraction and 1.5h for tuning the parameters.

3.4 Data

Both Xu et. al. [119], and Fu and Menzies [29] perform experiments on a small dataset that we call OriginalDataset. OriginalDataset contains only 8,000 pairs of Java knowledge units (i.e., tagged with "Java"): 2,000 pairs of knowledge units for each type of relationships, among them, 1,600 pairs are used for training and 400 pairs are used for testing.

3.4.1 Creating LargeDataset

To further evaluate the effectiveness of the techniques proposed, we created a larger dataset that we refer to as LargeDataset. Note that the relatedness (i.e., label) between knowledge units cannot be directly extracted from Stack Overflow, further processing is required. Figure 3.3 depicts the process of creating the new dataset by an example; it includes three main steps:

1. extracting duplicate and direct link pairs from Stack Overflow data dump,

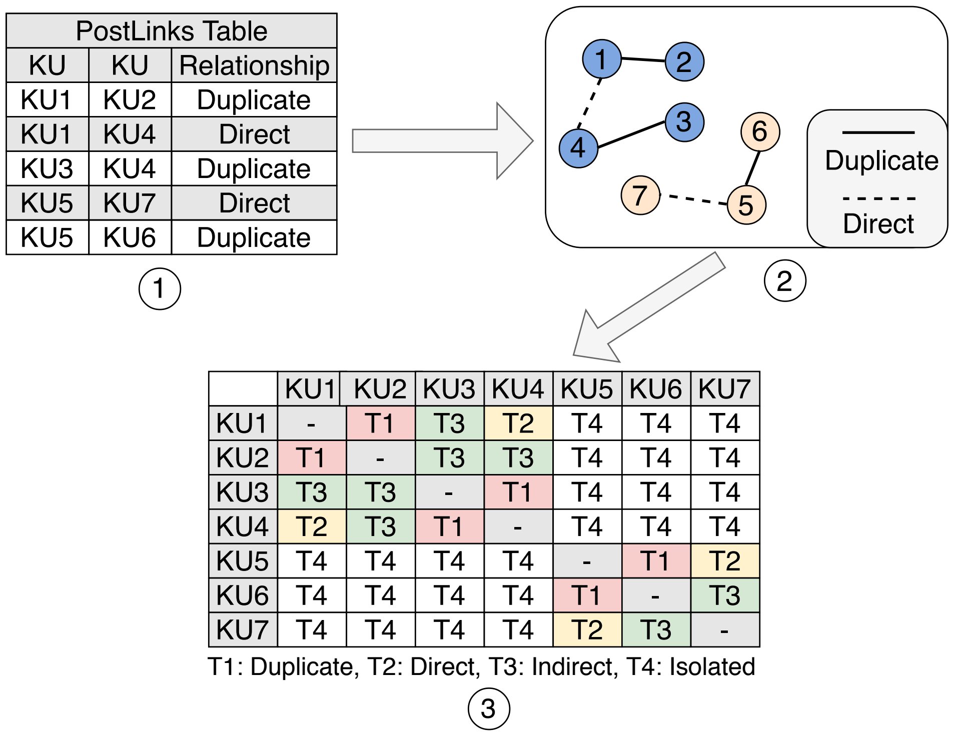
2. building a knowledge units network (KUN) using the link information,
3. extracting relations between all pairs of knowledge units in the knowledge network.

First, the experiment data is from Stack Overflow data dump [5]. Specifically, a table named `PostLinks` includes *duplicate* and *direct* knowledge units pairs. Similar to ORIGINALDATASET, only Java knowledge units are considered in LARGEDATASET. Then, *duplicate* and *direct* links information is used to create a knowledge unit network (KUN). The KUN is used to extract the relationships between any two knowledge units.Direct and duplicate relations are readily extracted from the information in the `PostLinks`. The relation between a pair of knowledge unit is *indirect* if two knowledge units are connected in the KUN with a certain range of distance (in this case, length of shortest path $\in$ [2,5]), but the relationship between them belongs neither to *duplicate* nor *direct*. Two knowledge units are *isolated* if they are not connected in the KUN (i.e., they belong to different clusters).

3.4.2 Characteristics of the new dataset

We followed the steps outlined in Section 3.4.1 and created a new, larger dataset (LARGEDATASET). More specifically, the new dataset contains 40,000 pairs of knowledge units which is five times the small dataset. We applied Latent Dirichlet Allocation (LDA) to investigate the top-50 topic distribution of two datasets. Figures 3.4 and 3.5 shows the graphical distribution of topics in ORIGINALDATASET and LARGEDATASET, respectively. As is shown, compared to LARGEDATASET, ORIGINALDATASET covers fewer topics.

[5]Stack Overflow data dump, `https://archive.org/download/stackexchange`



PostLinks Table		
KU	KU	Relationship
KU1	KU2	Duplicate
KU1	KU4	Direct
KU3	KU4	Duplicate
KU5	KU7	Direct
KU5	KU6	Duplicate

	KU1	KU2	KU3	KU4	KU5	KU6	KU7
KU1	-	T1	T3	T2	T4	T4	T4
KU2	T1	-	T3	T3	T4	T4	T4
KU3	T3	T3	-	T1	T4	T4	T4
KU4	T2	T3	T1	-	T4	T4	T4
KU5	T4	T4	T4	T4	-	T1	T2
KU6	T4	T4	T4	T4	T1	-	T3
KU7	T4	T4	T4	T4	T2	T3	-

T1: Duplicate, T2: Direct, T3: Indirect, T4: Isolated

Figure 3.3: Dataset Building Process

3.5 Research Questions and Results

3.5.1 Research Questions

We seek to address the following four research questions.

- **RQ1:** How well do CNN Model and Tuning SVM perform in predicting relatedness of knowledge units on LargeDataset?
- **RQ2:** What is the run-time cost of CNN Model and Tuning SVM in LargeDataset?
- **RQ3:** Is there any major difference between performance of CNN Model and Tuning SVM on LargeDataset and their performance in OriginalDataset?
- **RQ4:** Does SoftSVM perform better than CNN Model and Tuning SVM on LargeDataset?

RQ1 and RQ2 seek to investigate the performance of the prediction techniques on LargeDataset. Specifically, we are interested to learn if the high

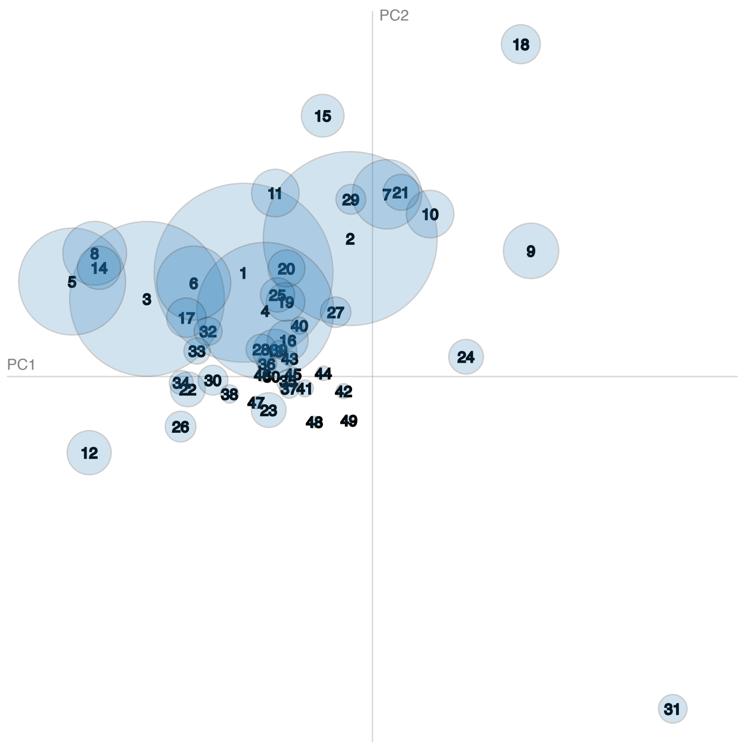


Figure 3.4: Topic Distribution of OriginalDataset

performance of those models stems from the characteristics of the dataset that they used. In other words, we investigate a more realistic situation by having more instances and covering more topics available in LargeDataset.

RQ3 is concerned with the consistency of performance of techniques between the OriginalDataset and LargeDataset. RQ4 addresses the question whether an SVM model with fewer but more effective textual features can perform better than other techniques.

3.5.2 Results

In this section, we present the results of our experiments. Consistent with the previous studies, we use precision, recall and F1-score as performance metrics of models.

RQ1: Performance of CNN Model and Tuning SVM on LargeDataset

Table 3.5 depicts performance of models trained by CNN Model and Tuning SVM and SoftSVM for individual classes along with the overall

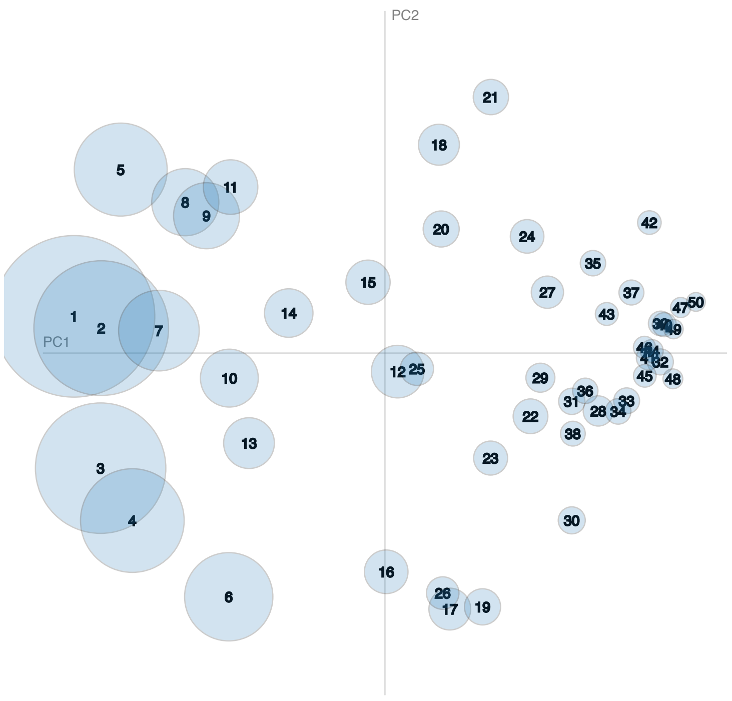


Figure 3.5: Topic Distribution of LargeDataset

scores. Overall, Tuning SVM outperforms CNN Model system almost by 10 percentage points. In all classes except Direct, Tuning SVM outperforms CNN Model, as it seems that the Direct class is the hardest class for Tuning SVM to predict. On the other hand, Duplicate class is the hardest class for CNN Model to classify. Across all metrics, Isolated class obtains the highest score, which means identification of this class is easier than the rest.

Table 3.5: Performance of Three Systems on Large Dataset

			Duplicate	Direct Link	Indirect Link	Isolated	Overall
Precision	CNN Model	Word Embedding + CNN	**0.55**	0.33	0.32	**0.79**	0.50
	Tuning SVM	DE + SVM	0.49	0.33	**0.49**	0.68	0.49
	SoftSVM	Tuned SVM+ Soft-cosine	0.51	**0.45**	0.42	0.75	**0.53**
Recall	CNN Model	Word Embedding + CNN	0.21	**0.62**	0.39	0.41	0.41
	Tuning SVM	DE + SVM	**0.59**	0.22	0.56	0.67	0.51
	SoftSVM	Tuned SVM+ Soft-cosine	0.48	0.21	**0.58**	**0.90**	**0.54**
F1-Score	CNN Model	Word Embedding + CNN	0.31	**0.43**	0.35	0.54	0.41
	Tuning SVM	DE + SVM	**0.54**	0.26	**0.52**	0.68	0.50
	SoftSVM	Tuned SVM+ Soft-cosine	0.50	0.29	0.49	**0.82**	**0.52**

RQ2: Computation cost of CNN Model and Tuning SVM on LargeDataset

Table 3.6 compares the time efficiency of building classification models in each approach. Note that, in interpreting the results, the heterogeneous computing infrastructure may have a moderate impact on the computation time. The Table 3.6 shows that TUNING SVM takes considerably more computation time than other techniques—around 2.5x and 12.5x more than CNN MODEL and SOFTSVM, respectively. We find that this is due to the fact that TuningSVM uses a large number of features, adapts several kernels and C parameters one by one to tune the SVM. Also, some kernels (such as RBF kernel) used in TuningSVM are not suitable for the large dataset because the time cost will increase exponentially when the number of features becomes large [43]. Training model in SOFTSVM was by far the faster than others.

Table 3.6: Training Time in Different Techniques

	Approach	Time
CNN MODEL.	Word Embedding + CNN (CPU-based)	15h 21m 24s
TUNING SVM	DE + SVM (CPU-based)	38h 24m 46s
SOFTSVM	Tuned SVM+ Soft-cosine (CPU-based)	2h 54m

RQ3: Discrepancies between the performance of techniques on ORIGINALDATASET and LARGEDATASET Note that in this research question, we aim to investigate whether the conclusion of Fu and Menzies [29] still holds on the large dataset. Thus, we analyze the performance of CNN MODEL and TUNING SVM on both ORIGINALDATASET and LARGEDATASET.

Comparison of performance Table 3.4 compares the performance of CNN MODEL and TUNING SVM on ORIGINALDATASET. CNN MODEL achieved 0.84 F1-score, 0.84, 0.84 precision and recall, respectively. TUNING SVM achieves 0.89 F1-score, 0.89 and 0.89 precision and recall, respectively.

Performance of CNN MODEL and TUNING SVM on LARGEDATASET are shown in Table 3.5. CNN MODEL achieves 0.41 F1-score, 0.50, 0.41 precision and recall, respectively. TUNING SVM achieves 0.50 F1-score, 0.49 and 0.51 precision and recall, respectively.

By comparing the same approach on OriginalDataset and LargeDataset, we find that the effectiveness of both CNN Model and Tuning SVM, as measured by F1-score, drop sharply, around 40 percentage points. By comparing the performance of CNN Model and Tuning SVM on the same dataset, our experimental results confirm that the conclusion of Fu and Menzies [29] still holds on the LargeDataset, i.e., CNN Model and Tuning SVM can achieve similar (and sometimes better) results.

Comparison of training computation cost According to the data reported in [119, 29], training CNN Model and Tuning SVM on OriginalDataset take almost 14 hours, and 10 minutes, respectively, on regular machines. Table 3.6 shows the training time of approaches on LargeDataset. Training time of all techniques increases on LargeDataset, but with different slopes. For example, computation time for training in Tuning SVM from 10 minutes on the OriginalDataset, jumps to 38 hours on LargeDataset, while computation cost of training a model using CNN Model increases from 14 hours to 15.3 hours, on OriginalDataset and LargeDataset, respectively.

Comparing to each other, Tuning SVM (>38 hours) spends 2.5x as much time as CNN Model (>15 hours). We find that this is due to the fact that Tuning SVM approach by using a large number of features, adapts several kernels and C parameters one by one to tune the SVM. To reduce the potential variance caused by how the original training data, this process repeats 10 times. Also, some kernels (such as RBF kernel) used in Tuning SVM is not suitable for LargeDataset because the time cost will increases exponentially when the number of features becomes large [43]. On the other hand, when using LargeDataset, there is only a slight increase in the runtime of CNN Model approach which proves that the scale of the dataset has slight impacts on the runtime of the technique.

***RQ4: Performance of* SoftSVM**

Performance Rows corresponding to SoftSVM in Table 3.5 contain performance of SoftSVM, i.e., 0.52 F1-score, 0.53, 0.54 precision and recall, respectively. The results show that SoftSVM achieves better overall performance than Tuning SVM and CNN Model in terms of F1-score, precision, and recall.

Comparing results of individual classes, it is clearly visible that SoftSVM has a performance advantage in predicting the Isolated class over the other methods. In other three classes, SoftSVM performs better than at least one of Tuning SVM and CNN Model. For example, for Duplicate class it achieves 0.04 F1-score worse than Tuning SVM but 0.19 better than CNN Model; for Direct class, achieves 0.03 F1-score better than Tuning SVM and 0.14 lower than CNN Model.

Comparison of computation time In terms of time efficiency, SoftSVM spends much less time than the other two approaches. As shown in Table 3.6, SoftSVM needs only less than 3 hours for training on LargeDataset, while CNN Model and Tuning SVM require 5x and 12x more time for training.

3.6 Discussion

3.6.1 Shortcomings of the original dataset

We found that the dataset used in previous studies have several limitations. First, we found that in the creation of the OriginalDataset, clusters larger than a certain size KUNet has been removed mistakenly (creators of the OriginalDataset confirmed this mistake), which results in only a small set of clusters with limited topic distribution. Second, we found that the OriginalDataset covers far fewer topics than LargeDataset. There is also a larger overlap among topics covered by knowledge units in OriginalDataset. Therefore, we believe that the results on LargeDataset are more reliable than the results reported on OriginalDataset in previous studies.

3.6.2 Performance of techniques

Our results on LargeDataset show that, consistent with the [29], Tuning SVM keeps its performance advantage over CNN Model. Our reproducibility study confirms that, in this task, Tuning SVM performs better than deep learning technique. However, as the number of instances for tuning increases in OriginalDataset, Tuning SVM's efficiency is diminished and it becomes slower than CNN Model deep learning techniques.

Our results also suggest that an SVM model with lightweight features, i.e. SoftSVM, can outperform CNN Model and Tuning SVM. On the other hand, previous results showed high performances of techniques (F1-score as high as 0.88) which leaves little room for improvement. In this work, we improved the quality of the dataset by covering more topics, adding more instances and correcting improper preprocessing process. Our results on LargeDataset, shows that, contrary to previous results, the performance of techniques are as low as F1-score=0.41.

Low performance of models suggests that proposed prediction models, our approach is still highly inadequate for large, diverse dataset. An alternative interpretation of low performance on LargeDataset can be that the relations between knowledge units are purely stochastic and there is no feature to capture any relation.

Table 3.7: Feature Scale and Performance of the Considered Approaches

Approach	Feature Scale	Feature Extraction Technique	Effectiveness	Efficiency
CNN Model	640	CNN model	The worst	Intermediate
TuningSVM	200	Word embedding	Intermediate	The worst
SoftSVM	4	TF-IDF + Word embedding, Edit distance	The best	The best

Moreover, we are also interested in the potential relationship between feature scale and performance of the considered approach. As shown in Table 3.7, the order of feature scale is CNN model > TuningSVM > SoftSVM. While the effectiveness rank is SoftSVM > TuningSVM > CNN model and the efficiency rank is SoftSVM > CNN model > TuningSVM. The results of effectiveness indicate that the feature used by SoftSVM is not only low-scale but also effective. And in terms of the efficiency comparison, we can find that the feature scale is not directly related to effectiveness in our task.

3.7 Threats to Validity

There are several threats that may potentially affect the validity of our experiments. Threats to internal validity relate to errors in our experimental data and tool implementation. To mitigate this threat for the new dataset, i.e., LargeDataset, we manually checked the selected knowledge units in the dataset to ensure that they are really tagged with "java" and correctly labeled. Another threat to internal validity is modifications to Tuning SVM to make it parallel. However, we note the implementation of Tuning SVM is simple, and we only execute each fold in parallel without modify any internal implementation. Threats to external validity relate to the generalizability of our results. In this study, we followed the same steps as previous work [119] to create LargeDataset. Thus, only the knowledge units tagged with "Java" are considered. The threat is limited by the fact that "Java" has been consistently on the top-5 list of most popular tags on Stack Overflow [6].

3.8 Conclusion and Future Work

In this paper, we performed a reproducibility study of multiple techniques proposed for predicting relatedness of knowledge units on Stack Overflow. We find that there are several limitations in the original dataset used in the previous studies, thus we created a new dataset to address these limitations. We observed that performance of proposed approaches (as measured using F1-score) drop sharply on the new dataset, however similar to the previous finding, performance of SVM-based approaches (Fu and Menzies' approach and SimBow)

[6]https://stackoverflow.com/tags

are slightly better than the DNN-based approach, however, contrary to the previous findings, Fu and Menzies' approach runs much slower than DNN-based approach on the larger dataset – its runtime grows sharply with increase in dataset size.

We conclude that, for this task, simpler approaches based on SVM performs similarly to DNN-based approach. We also illustrate the challenges brought by the increased size of data and show the benefit of a lightweight SVM-based approach for this task.

In the future, we plan to investigate an ordinal classification algorithm for the task as the categorical labels of the relatedness are naturally ordinal. Thus, we believe ordinal classification algorithms have a great potential to perform better than classic ones.

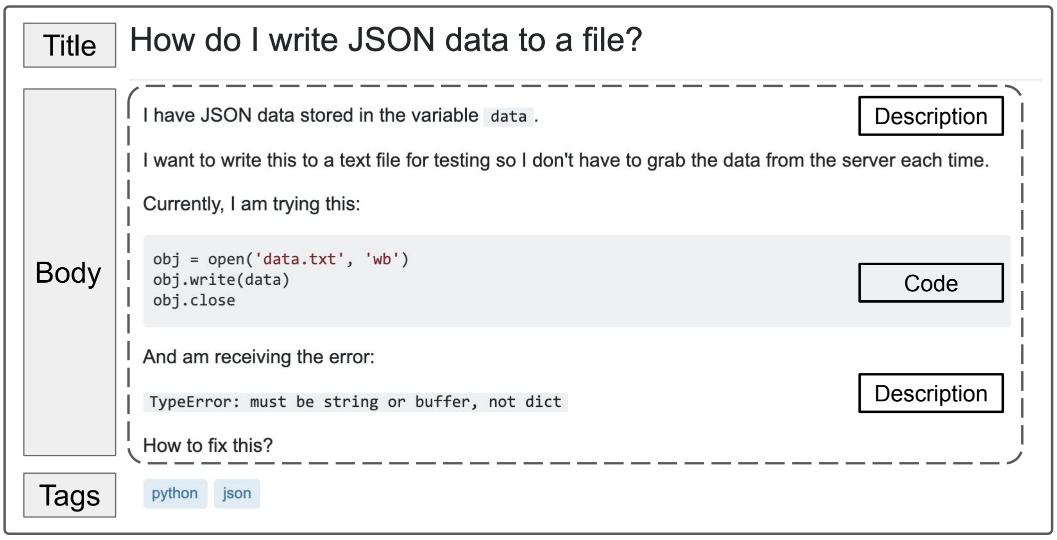
Chapter 4

Distributed Knowledge Representation

4.1 Introduction

Presently, software question and answer (SQA) sites are an essential part of developers' day-to-day work for problem-solving and self-learning. A SQA site (such as Stack Overflow) is a collection of posts where a post contains several components, e.g., title, body, and tags as illustrated in Figure 4.1. To help developers navigate SQA sites and find relevant information efficiently, many solutions have been proposed. They address concrete tasks such as recommending tags to posts (aka. tag recommendation) [114, 105, 133], identification of related posts (aka. relatedness prediction) [119, 29, 115, 70], post classification [86, 4, 10], and many more. A common step in previous works is to extract a suitable vector to represent a post for further processing. Intuitively, following "garbage in, garbage out" principle, the quality of a post's vector representation plays a major role in determining the eventual outcomes.

In this work, to boost the effectiveness of existing solutions that analyze SQA site contents, we want to learn distributed representations of posts that can be used for a number of downstream tasks. We propose a new deep learning



How do I write JSON data to a file?

I have JSON data stored in the variable `data` .

I want to write this to a text file for testing so I don't have to grab the data from the server each time.

Currently, I am trying this:

```
obj = open('data.txt', 'wb')
obj.write(data)
obj.close
```

And am receiving the error:

`TypeError: must be string or buffer, not dict`

How to fix this?

python json

Figure 4.1: Example of a Stack Overflow Post ID=12309269

architecture named POST2VEC that can effectively embed posts in a space where the distance between similar posts is small. Specifically, considering the nature of tags, i.e., semantic labels provided by developers and shared by many posts, we utilize them to supervise the learning of post representations.

Intuitively, this problem is challenging because it requires learning a correspondence between the entire content of a post and only a few semantic labels among hundreds of thousands of candidates. That is, it requires associating contents from different components of a post, which typically include hundreds of words and several pieces of code snippets, into a few descriptive labels. Moreover, to develop a robust and useful post representation, POST2VEC needs to address the following three limitations of prior works:

- Posts from a particular SQA site typically follow a standard structure, which consists of different components (e.g., title and body) carrying information at different levels of detail. The title summarizes a specific problem, while the body expands the title with more details. However, we found that many prior studies, e.g., [119, 133], simply treated the different components equally and represented a post by concatenating content of the various components together into one single piece of text.

- Many previous works, e.g., [99, 120, 107], have removed code snippets during the preprocessing step due to these code snippets are short, have poor structure, and are written in different programming languages. However, from Figure 4.1 we can see that if we consider the code snippets, it becomes effortless to identify that the post is *Python* related even though the user does not mention Python in the title and description. Thus, intuitively, code snippets should be considered while learning an effective post representation. Moreover, we found that for more than 70% posts in the Stack Overflow data dump (dated September 5, 2018), their body contained at least one code snippet. In other words, code snippets are vital sources of information if they can be properly captured while learning representations of posts.

- Early attempts [114, 105] focused only on extracting text from posts and then used the text to create well known but simplistic representations such as bag-of-words (BOW). A BOW representation of a post is a multiset of words that appear in the post. However, BOW ignores the semantic relation between words.

POST2VEC uses Word2Vec [72] to map words with similar meaning to vectors with small distances between them. Additionally, POST2VEC leverages not only the natural language content (i.e., title and description) presented in the post but also the code snippets. Individually, title, description, and code snippets are considered as three components and they are fed into three different neural network models to produce three fixed length feature vectors. Finally, considering the nature of tags, we utilize them to supervise the learning of post representations from the three feature vectors. More specifically, POST2VEC optimizes the vector representations of posts to predict appropriate tags that are assigned to the respective posts by learning the relationship between the vectors and tags.

Once the post representations are learned, they can be used to boost the effectiveness of many supervised learning tasks, especially those rely only on a limited set of labeled data. Although Stack Overflow data is vast, many prior studies only made use of a subset of posts that are labeled [119, 10]. By using the learned post representations, in effect, we are using the power of vast unlabeled data to help in a particular supervised learning task (aka. semi-supervised learning). Additionally, many prior approaches, e.g., [115, 116], proposed handcrafted features for a specific task and represented a post by a low dimensional feature vector. The effectiveness of those approaches may benefit from the post representations that are pre-trained based on a large dataset. Such representations can be used to embellish the low dimensional feature vector to boost effectiveness potentially.

The closest work to ours is by Zhou et al. [133], who proposed a deep neural network architecture to predict tags that are assigned to SQA posts. Our work is different from them in the following aspects. First, the main goal of our work and theirs differ; Zhou et al. aim to improve tag recommendation, while we aim to produce representations of posts that can be used for multiple downstream tasks. Second, the components considered in our work and theirs differ; Zhou et al. only utilize title and description while we consider code snippets as well. Consider many tasks (e.g., tag prediction [105, 133], similar post detection [29, 115], etc) are expected to be performed as soon as the post is created, hence our approach is designed to construct vector representations of newly created posts, i.e., those without answers and comments. With this design, we can immediately infer the vector representations of posts once they are created, and directly use the representations for other tasks. Third, the way the components are processed differ too; Zhou et al. concatenate title and description to a piece of text in the data preprocessing step while we feed different components to different models separately. Another closely related work is Doc2Vec [57] that generates a vector representation for textual documents; however, there are significant differences between "flat" textual documents considered in Doc2Vec and posts. We consider a post as a structured document with three components (i.e., title, description, and code snippets). Moreover, different from Doc2Vec, we use tags as guides in the generation of post representations.

To evaluate Post2Vec, we first investigate the effectiveness of our approach in the tag recommendation task and compare it with the state-of-the-art approach [133]. Post2Vec achieves 11-20% improvement in terms of F1-score@5. Moreover, we further evaluate the value of the representation learned by Post2Vec in two other tasks, i.e., relatedness prediction [119] and post classification [10]. We found that these state-of-the-art approaches are improved by integrating our post representation, i.e., 11% and 9% for relatedness prediction and post classification, respectively.

The main contributions of this work are as follows:

- We propose a specialized deep learning architecture Post2Vec that learns vector representations of SQA posts. To the best of our knowledge, this work is the first to explore the learning of generic representation of SQA posts. We focus on Stack Overflow posts, but the solution presented here can be easily adapted for posts from other SQA sites.

- We empirically compare end-to-end Post2Vec with the state-of-the-art neural network based approach for tag recommendation task based on a 10 million posts dataset, and demonstrate that Post2Vec can achieve 15-25% improvements and also complete the learning process by up to 1.5 days faster.

- We empirically investigate the benefit of design decision from three aspects. For each aspect, we implement and compare pairs of Post2Vec variants. The experimental results show that, (1) CNN-based Post2Vec outperform Lstm-based Post2Vec significantly and consumes fewer computing resources during model training, (2) consideration of code snippets can boost effectiveness but consumes more computing resources during model training, (3) considers different post components separately significantly boosts the performance but consumes more computing resources than considers different components together during model training.

- We empirically investigate the value of integrating the post vectors generated by POST2VEC and feature vectors used by two state-of-the-art approaches of three tasks (relatedness prediction, post classification, and API recommendation) and demonstrate substantial improvements.

The rest of this paper is organized as follows. Section 4.2 elaborates the design of our approach POST2VEC. Section 4.3 presents our experiments that compare POST2VEC with the state-of-the-art tag recommendation approach and their results. Section 4.4 investigates the value of our learned post representations to aid in the two downstream tasks (relatedness prediction and post classification). Section 4.5 presents a qualitative study and some threats to validity. We conclude and mention future work in Section 4.6.

4.2 Proposed Approach

4.2.1 Framework Overview

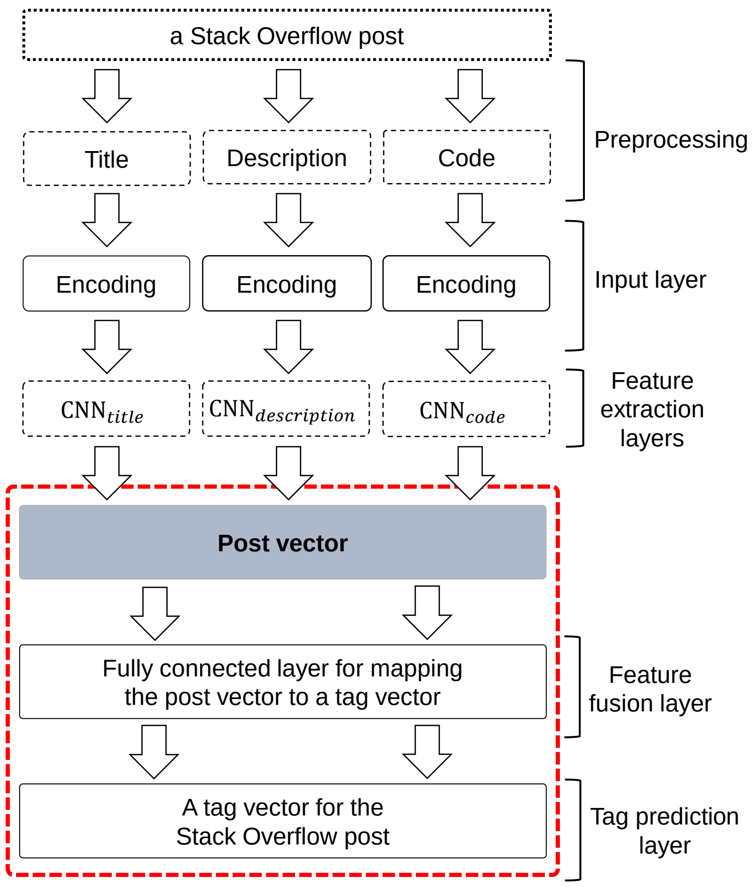


Figure 4.2: The overall framework of Post2Vec.

Figure 4.2 illustrates the overall framework of Post2Vec that takes Stack Overflow (SO) posts as input and outputs their distributed representations in the form of post vectors. More specifically, Post2Vec consists of four parts:

- *Preprocessing*: This part extracts three pieces of information from a SO post (i.e., title, description, and code snippets), cleans them, and represents each of them as a sequence of tokens.

- *Input layer*: This part encodes the sequences of tokens corresponding to the title, description, and code snippets of posts into two-dimensional matrices as inputs to the NNs in the feature extraction layers.

- *Feature extraction layers*: This part extracts the embedding vectors (aka. features) of the title, description, and code snippets. These embedding vectors are then concatenated to form the post vector representation of the input SO post.

- *Feature fusion and tag prediction layers*: This part maps the post vector to a tag vector; the tag vector indicates the probabilities of various tags being assigned to the SO post.

Post2Vec leverages the tags assigned to SO posts to guide the learning of suitable post vectors. And then, the learned vectors are used as the distributed representations of SO posts. We use tags as a guide since they are semantic labels provided by developers and shared by many posts. Specifically, we define a learning task to construct prediction function $\mathbf{f} : P \rightarrow \mathcal{Y}$, where $y_i \in \mathcal{Y}$ indicates a list of tags of the post $p_i \in P$. The prediction function $\mathbf{f}$ is learned by minimizing the differences between predicted and actual tags assigned to posts. After the prediction function $\mathbf{f}$ is learned, for each post, we can obtain its post vector from the intermediate output between the feature extraction and feature fusion layers (see Figure 4.2). We explain the details of each part of Post2Vec in the following subsections.

4.2.2 Preprocessing

We process the title, description, and code snippets of each post by the following four steps that are applied one after the other:

1. **Separate description and code snippets from the body**. Code snippets in the body of a post are enclosed in a pair of HTML tags *"⟨pre⟩⟨code⟩"* and *"⟨\code⟩⟨\pre⟩"* [116, 119]. Hence, we use a regular expression *"⟨pre⟩⟨code⟩([\s\S]*?)⟨\\code⟩⟨\\pre⟩"* to extract those code snippets. We concatenate all code snippets in a post into one document, and the remaining parts of the post into another document.

2. **Remove HTML tags**. We remove HTML tags that appear in the extracted code snippets and description documents. These HTML tags typically correspond to formatting instructions that may not carry much semantic meaning [119]. To achieve this, we utilize a popular HTML parser Beautiful Soup[1].

3. **Tokenize title, description, and code snippets**. At this step, we would like to split each title, description, and code snippet into a sequence of tokens. To do this step, we employ a popular tool named NLTK[2]. More precisely, the function *word_tokenize* was used in our implementation.

4. **Construct component-specific vocabulary**. Based on the posts in training data, we build three vocabularies $\mathcal{V}^{\mathrm{T}}$, $\mathcal{V}^{\mathrm{D}}$, $\mathcal{V}^{\mathrm{C}}$; each vocabulary corresponds to a set of tokens along with their occurrence frequencies in the collection of title, description and code snippet, respectively. A common practice for constructing a vocabulary from large scale data is to discard the tokens that occur less than k times [72]. In this work, we set k equals to 50 which is a common threshold used in prior works, e.g., [61].

At the end of the preprocessing steps, title, description, and code snippets of each post are extracted as three *sequences of tokens*. These sequences are fed to the input layer of our proposed framework for further processing.

4.2.3 Input Layer

The input layer performs three main steps:

1. **Indexing and padding**. We convert each component of a post output by the preprocessing step (which is a sequence of tokens of arbitrary length) to a *fixed length* sequence of *token indices*. To achieve this, we

[1]Beautiful Soup, `https://www.crummy.com/software/BeautifulSoup`.
[2]NLTK Tokenizer, `https://www.nltk.org/api/nltk.tokenize.html`.

first replace each token in each sequence by its index in the corresponding vocabulary (if it exists in the vocabulary; otherwise we skip the token). For example, given a title as a sequence of tokens $\mathrm{T} = [w_1, \ldots, w_{|\mathrm{T}|}]$, we convert it to $[index_1, \ldots, index_{|\mathrm{T}|}]$, where $|\mathrm{T}|$ is a length of the sequence of tokens and $index_i$ is the index of w_i in $\mathcal{V}^{\mathrm{T}}$. Since components of different posts contain different numbers of tokens, and CNN requires each input to be represented in the same way for the purpose of parallelization, we perform padding or truncating which is a standard solution. Specifically, we set three parameters $\mathcal{L}^T$, $\mathcal{L}^D$, and $\mathcal{L}^C$ to be the target length of the sequence of tokens in the title, description, and code snippets respectively. For each sequence belonging to a component (i.e., title, description, or code snippets), if its length is smaller than the predefined length (i.e., $\mathcal{L}^T$, $\mathcal{L}^D$, or $\mathcal{L}^C$), we add a special token[3] (one or more times) so that all sequences have the same length. If the length of a sequence belonging to a component is longer than the predefined length, we truncate it to only its first $\mathcal{L}^T$, $\mathcal{L}^D$, or $\mathcal{L}^C$ tokens. We calculate the distribution of the number of tokens from each component among all posts. And we found that by setting the value of $\mathcal{L}^T$, $\mathcal{L}^D$, $\mathcal{L}^C$ as 100, 1,000, and 1,000, we can preserve more than 95% of the original tokens for each component. In other words, more than 95% posts' title, description and code length are smaller than 100, 1,000, and, 1,000. Thus only a minimal loss of data is incurred while keeping the representation to a smaller size, for a boost in performance (esp. training time).

2. **Token representation**. We represent each token as a vector of d values. By default, we set d to be 128. We initialize the d values randomly and they will be updated and learned during the training process.

3. **Component representation**. We encode the title, description, and code snippets of a post as two-dimensional matrices (i.e., $\mathcal{X}^{\mathrm{T}}$, $\mathcal{X}^{\mathrm{D}}$, and

[3]We use zero as the padded token.

$\mathcal{X}^{\mathrm{C}}$). For example, given a title as a sequence of tokens T, we construct its matrix representation $\mathcal{X}^{\mathrm{T}} \in \mathbb{R}^{\mathcal{L}^T \times d}$, where d is the dimension of the token representation. At the end, the input layer outputs $\mathcal{X}^{\mathrm{T}}$, $\mathcal{X}^{\mathrm{D}}$, and $\mathcal{X}^{\mathrm{C}}$ and these matrices are used as an input to the feature extraction layers.

4.2.4 Feature Extraction Layers

Under the framework of Post2Vec, we utilize NN models to extract feature from posts. Specifically, we input $\mathcal{X}^{\mathrm{T}}$, $\mathcal{X}^{\mathrm{D}}$, and $\mathcal{X}^{\mathrm{C}}$, which are the encoded data of the title, description, and code snippets, for each Stack Overflow (SO) post to three independent NN models. The NN models produce embedding vectors $\mathbf{z_T}$, $\mathbf{z_D}$, and $\mathbf{z_C}$, which represent the features extracted from each component of the SO post, respectively. Considering code snippets in Stack Overflow are short, have poor structure, and are written in different programming languages [99, 120, 107], the existing code representation techniques are unable to be applied as they are applicable for either a single type of program language (e.g., [108, 100]) or the code should be at least a complete function or method (e.g., [100, 16]). Hence, Post2Vec processes code snippets and other two textual components (i.e., title and description) in the same way. Noted that different types of NN models can be used as feature extractors. And we investigate the performance of widely used NN models in our experiment and corresponding hyperparameters in Section 4.3.3.

4.2.5 Feature Fusion and Tag Prediction Layers

Figure 4.3 shows the details of the part of the architecture shown inside the red dashed box in Figure 4.2. The inputs of this part are the three embedding vectors (i.e., $\mathbf{z_T}$, $\mathbf{z_D}$, and $\mathbf{z_C}$) which represent the features extracted from the title, description, and code snippets respectively. These embedding vectors are concatenated to construct a new embedding vector representing a Stack

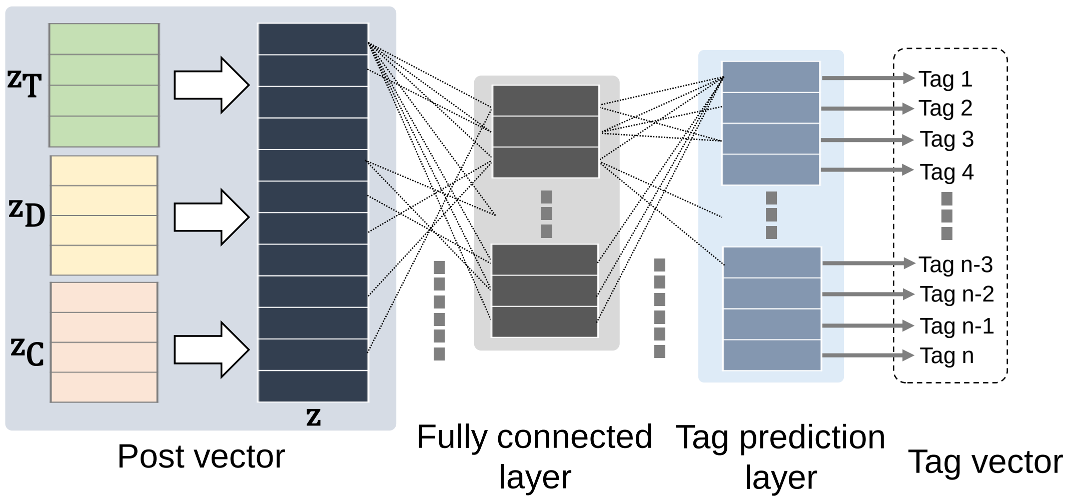


Figure 4.3: The details of the red dashed box in Figure 4.2. $\mathbf{z_T}$, $\mathbf{z_D}$, and $\mathbf{z_C}$ are the embedding vectors of the title, description, and code snippet, respectively. $\mathbf{z}$ is the post vector and is fed to a fully-connected layer (FC) to produce the tag vector (i.e., the probability distribution over tags).

Overflow post.

We put the embedding vector $\mathbf{z}$ into a fully connected (FC) layer to produce a vector $\mathbf{h}$:

$$\mathbf{h} = \alpha(\mathbf{w_h} \cdot \mathbf{z} + b_\mathbf{h}) \tag{4.1}$$

where $\mathbf{w_h}$ is the weight matrix used to connect the embedding vector $\mathbf{z}$ with the FC layer, $\cdot$ is the dot product between $\mathbf{w_h}$ and $\mathbf{z}$, and $b_\mathbf{h}$ is the bias value. Finally, the vector $\mathbf{h}$ is passed to a tag prediction layer to produce the following:

$$\mathbf{t} = -\mathbf{h} \cdot \mathbf{w_t} \tag{4.2}$$

where $\mathbf{w_t}$ is the weight matrix between the FC layer and the tag prediction layer, and $\mathbf{t} \in \mathbb{R}^{\mathbb{T}\times 1}$ ($\mathbb{T}$ is a total number of tags). We then apply the sigmoid function [40] to get the probability distribution over tags as follows:

$$\mathrm{p}(t_i|p_i) = \frac{exp(t_i)}{exp(t_i) + 1} \tag{4.3}$$

where $t_i \in \mathbf{t}$ is the score of the i^{th} tag and p_i is the post for which we want to recommend tags to.

4.2.6 Parameters Learning

Post2Vec aims to learn the following parameters: the embedding matrices of the title, description, and code snippets of each SO post, the convolutional layers' matrices, and the weights matrices and bias values of the fully connected layer and the tag prediction layer. After these parameters are learned, the post vector of each SO post can be determined. These parameters of Post2Vec are learned by minimizing the following objective function:

$$\begin{aligned} \mathcal{O} = \sum_{y_i \in \mathbf{y}} y_i \times (-\log(\mathrm{p}(t_i|p_i))) + (1 - y_i) \\ \times (-\log(1 - \mathrm{p}(t_i|p_i))) + \frac{\lambda}{2}\|\theta\|_2^2 \end{aligned} \tag{4.4}$$

where $\mathrm{p}(t_i|p_i)$ is the predicted tag probability defined in Equation 4.3, $y_i = \{0, 1\}$ indicates whether the i^{th} tag is assigned to the post p_i, and θ contains all parameters our model. The term $\frac{\lambda}{2}\|\theta\|_2^2$ is used to prevent overfitting in the training process [15]. The dropout technique [92] is employed to improve the robustness of Post2Vec. We use Adam [55] to minimize the objective function since Adam has been shown to be computationally efficient and it requires low memory consumption as compared to other optimization techniques [55, 6, 7]. We also use backpropagation [38], a simple implementation of the chain rule of partial derivatives, to efficiently compute parameter updates during the training process.

4.3 Post2Vec for tag recommendation

To evaluate Post2Vec's performance, we investigate its performance for tag recommendation task. The problem formulation of tag recommendation task is as follows: given a set of existing software posts that are labeled with tags, how to automatically predict a set of appropriate tags for a new unseen software post. Arguably, if Post2Vec can learn good embeddings, its end-to-end performance for the task of learning suitable tags to be assigned to Stack Overflow posts should be high. In this section, we conduct experiments to mainly answer the following three research questions:

RQ1: *Compared with the state-of-the-art approach, how effective and efficient is* Post2Vec *in tag recommendation?*

Recently, Zhou et al. [133] proposed deep learning based tag recommendation approaches. To demonstrate the value of Post2Vec for tag recommendation task, we compare our approach with theirs.

RQ2: *What is the impact of different types of NNs as feature extractors?*

One of the novelties of Post2Vec over the state-of-the-art tag recommendation approaches is the NNs used for extracting features can be replaced by other types. Here, we want to investigate the impact of different types of NNs on the performance of Post2Vec.

RQ3: *Does* Post2Vec *benefit from code snippets?*

One novelty of Post2Vec over the state-of-the-art tag recommendation approaches is the consideration of code snippets. Considering the fact that code snippets in Stack Overflow are incomplete, have poor structure, and are written in different programming languages, it is unclear if their consideration can be a net benefit to Post2Vec. Furthermore, many previous works, e.g., [99, 107], remove the code snippets when analyzing posts. In this research question, we want to investigate if code snippets boost (or harm) Post2Vec performance.

RQ4: *What is the impact of handling post components separately rather than combining them?*

Another novelty of Post2Vec over the state-of-the-art tag recommendation approaches is the deep learning architecture that considers different post components (title, description, and code snippets) separately using three different NN models (i.e., NN_{title}, $\text{NN}_{description}$, and NN_{code}). Here, we would like to investigate if handling post components separately can boost the performance

Table 4.1: Differences between Variants of Post2Vec

Approach	Used NN Model	Considered Components			Architecture
		Title	Description	Code Snippets	
Post2Vec$_{LSTM,All,Sep}$	BiLSTM	✓	✓	✓	Multiple LSTMs for different components
Post2Vec$_{CNN,-Code,Sep}$	CNN	✓	✓		Multiple CNNs for different components
Post2Vec$_{CNN,All,Com}$	CNN	✓	✓	✓	Single CNN model
Post2Vec$_{CNN,All,Sep}$	CNN	✓	✓	✓	Multiple CNNs for different components

of Post2Vec.

4.3.1 Baseline Approaches

To answer RQ1, we compare the Post2Vec with the state-of-the-art tag recommendation approach proposed by Zhou et al. [133]. In particular, they proposed four types of neural network based approaches, TagCNN, TagRNN, TagHAN, and TagRCNN, which were based on convolutional neural network (CNN) [54], recurrent neural network (RNN) [64], hierarchical attention network (HAN) [123], and recurrent convolutional neural network (RCNN) [56], respectively. Moreover, they also compared the four proposed approaches with the three traditional approaches, EnTagRec [106], TagMulRec [134], and FastTagRec [63]. Their experimental results showed that TagCNN and TagRCNN were the top-2 best performers in terms of effectiveness. Moreover, Zhou et al. also concluded that TagCNN and TagRCNN are practical for use on the various-scale datasets. Thus, we use both TagCNN and TagRCNN as our baseline approaches. To replicate Zhou et al.'s approach, we carefully use their released source code[4].

Moreover, we raise three research questions (i.e., RQ2, RQ3, RQ4) evaluating the impact of Post2Vec design decisions: RQ2 investigates the impact of using different type of NN models in Post2Vec. RQ3 investigates the impact of inclusion or exclusion of code snippets in the post components considered by Post2Vec. RQ4 investigates the impact of concatenating or separating different post components. To answer these three research questions, we implement four variants of Post2Vec. Table 4.1 summarizes the

[4]`https://pan.baidu.com/s/1pKCpodP`.

differences between all investigated variants considering three perspectives: used NN models, used components, and architecture. To answer RQ2, we compare the performance of pairs of variants that are only different on used NN models, i.e., Post2Vec$_{LSTM,All,Sep}$ and Post2Vec$_{CNN,All,Sep}$. We consider two kinds of NNs to serve as feature extractors, i.e., convolutional neural network (CNN) and bidirectional long short term memory (BiLSTM). We pick these two because the effectiveness of both CNN [133, 119] and BiLSTM [32, 104] have been demonstrated in prior works on processing Stack Overflow textual data. Thus, we investigate these two types of NN models in our experiment. Noted that they can be replaced by other NN models and we leave it for future work. To answer RQ2, we compare the performance of a pair of variants that are only different on used NN models, i.e., Post2Vec$_{CNN,All,Sep}$ and Post2Vec$_{LSTM,All,Sep}$. Similarly, to answer RQ3 and RQ4, we implement and compare pairs of variants of Post2Vec that are only different in terms of considered components (with or without code snippets, i.e., Post2Vec$_{CNN,All,Sep}$ and Post2Vec$_{CNN,-Code,Sep}$) and architecture (single NN model or multiple NN models with different components, i.e., Post2Vec$_{CNN,All,Com}$ and Post2Vec$_{CNN,All,Sep}$).

4.3.2 Dataset

In this work, we use a snapshot from the Stack Overflow data dump following prior studies [80, 127, 79]. Zhou et al. [133] also use this dataset in the evaluation of their deep learning solutions. However, after careful investigation, we find that the dataset used by Zhou et al. has some limitations:

1. Zhou et al. only used the posts before July 1, 2014. However, this dataset is too old (more than four years ago) to demonstrate the effectiveness of tag recommendation approaches on the recent posts, especially due to the rapid growth of Stack Overflow in recent years.

2. In the experiments performed by Zhou et al., posts in test data were randomly selected from the whole Stack Overflow dataset without consideration of when the posts were made. In other words, some of the newer posts were used to train the model while some older posts were used in the test dataset which introduced bias for evaluation.

3. While millions of posts were considered by Zhou et al. in their experiments, only 10,000 (i.e., less than 0.1%) of them were selected to form the test set. Thus, the proportion of test data may not be representative enough to evaluate the approaches properly.

To address these limitations, we construct a more comprehensive dataset in this work. To address the first limitation, we use the version of Stack Overflow data dump released on September 5, 2018[5]. A commonly used first data preprocessing step in tag recommendation studies [133, 114, 105] is to identify the rare tags. A tag is *rare* if it appears less or equal to a predefined threshold *ts*. Rare tags are less important and less useful to serve as *representative tags* to be recommended to users [114]. Following the same setting as Zhou et al., we set the value to the threshold *ts* as 50. Using this setting, we identify 29,357 rare tags. We also obtain 10,379,014 posts after removing the posts that only contain rare tags. To address the second limitation, we sort all the posts by the attribute *creation date*. To avoid the bias, we use the latest posts as test data and the earlier posts as training data. Lastly, to address the third limitation, the latest 100,000 posts are selected as test data which is 20 times larger than the size of test data considered by Zhou et al. In this way, we build a new dataset; it has 10,279,014 posts as training data, and 100,000 posts as test data. Moreover, in the preprocessing step mentioned in Section 4.2.2, we also construct a vocabulary for each component. They are, $\mathcal{V}^{\mathrm{T}}$ (title's vocabulary), $\mathcal{V}^{\mathrm{D}}$ (description's vocabulary), and $\mathcal{V}^{\mathrm{C}}$ (code snippets' vocabulary) that include 28,213, 119,190, and 502,466 unique tokens respectively.

[5]`https://archive.org/download/stackexchange`.

4.3.3 NNs and Hyperparameter Setting

We employ independent NN models (i.e., NN_{title}, $\text{NN}_{description}$, and NN_{code} in Figure 4.2) to extract features from the title, description, and code snippets. In our experiment, we investigate two types of NN models, i.e., CNN and BiLSTM. For a fair comparison, both CNN and BiLSTM represent posts in the same dimension. Moreover, we use Adam as the optimization algorithm [55] and shuffled mini-batches to train them [60]. And we set the number of epochs as 16. The parameter setting is widely used in many deep learning based approaches for software engineering tasks, e.g., [37, 21, 34]. Next, we describe the specific hyperparameter settings for both CNN- and BiLSTM-based Post2Vec.

Convolutional Neural Network (CNN) based Post2Vec. Each CNN model includes a convolutional layer with multiple filters and a non-linear activation function (i.e., RELU). Each filter, followed by a RELU activation function, is employed to a window of k tokens to extract the corresponding features of this window. We apply a max pooling operation [12] to obtain the highest feature value of this window. Post2Vec uses multiple filters (with K different window sizes) to obtain features from each component. Following prior works (e.g., [37]), we follow a parameter tuning step to estimate good values of these parameters. We set aside 10% of the training data as a validation set (10%), while the remainder of the data is used by Post2Vec to learn post representation. We use the validation set for the parameter tuning procedure. Specifically, we vary different values of the parameters and measure the corresponding performance on the validation set. Take learning rate as an example; we try setting it as 0.01, 0.001, or 0.0001. And then, we measure the performance of our models with different learning rates and find that Post2Vec achieves the best performance when the training rate is set to 0.0001. In a similar way, we identify the optimal values of some other parameters. As a result of this parameter tuning step, we set the learning rate as 0.0001, the batch size as 128, and the number of filters as 100 for each of the CNN models [54]. We set window sizes (K) as 1, 2, 3 for title and description and 2, 3, 4 for code snippets (see Section 4.2.4). In this way, Post2Vec represents each component (i.e., title, description, or code snippets) of a post as a 300-dimensional (3×100) vector, while each post is represented as a 900-dimensional vector.

Bidirectional Long Short Term Memory (BiLSTM) based Post2Vec. Each BiLSTM model includes a recurrent layer with a hidden state. We set the number of features in the hidden state to 300. As a result, each component is represented as a 300-dimensional vector while each post is represented as a 900-dimensional vector.

4.3.4 Evaluation Metrics

To evaluate the performance of their proposed approaches, Zhou et al. [133] used Precision@k, Recall@k, and F1-score@k. The 3 metrics are defined as the averages of Precision@k$_i$, Recall@k$_i$, and F1-score@k$_i$ for each post p_i respectively. The latter metrics are defined below:

Precision@k$_i$ is the percentage of a post's ground truth tags GT_i that are in the top-k recommended list TR_i^k, formally defined as:

$$\text{Precision}\,@k_i = \frac{|TR_i^k \cap GT_i|}{k} \tag{4.5}$$

Recall@k$_i$ is the percentage of tags in the top-k recommended list TR_i^k that are among the set of ground truth tags GT_i. However, Recall@k_i value disfavors small k [63, 133]. For example, for a post p_i with 2 ground truth tags, even if the first 2 recommended tags are correct, the Recall@1_i value is still 50% (i.e., 1/2). Hence, we follow the definition of Recall@k_i that was used to evaluate the state-of-the-art tag recommendation approach by Zhou et al. [133]. In Zhou et al.'s work, if the number of ground truth tags of a post is larger than k, the value of Recall@k_i is computed using the same formula as Precision@k_i. In this way, for the aforementioned example, Recall@1_i will be 100%. This perfect score better reflects the correctness of the recommended tags at top-1 position (as it is the best result possible). Zhou et al.'s Recall@k_i is formally defined as:

$$\text{Recall}\,@k_i = \begin{cases} \frac{\left|TR_i^k \cap GT_i\right|}{k}, |GT_i| > k \\ \frac{\left|TR_i^k \cap GT_i\right|}{|GT_i|}, |GT_i| \leq k \end{cases} \tag{4.6}$$

F1-score@k_i is a harmonic mean of *Precision@k_i* and *Recall@k_i*, and it is formally defined as:

$$\text{F1-score}\,@k_i = 2 \times \frac{\text{Precision}\,@k_i \times \text{Recall}\,@k_i}{\text{Precision}\,@k_i + \text{Recall}\,@k_i} \tag{4.7}$$

Note that, based on the definition of Precision@k (Equation 4.5), Recall@k (Equation 4.6), and F1-score@k (Equation 4.7), their values will always be the same when k equals to 1; this is the case because each post is labeled by at least one tag, and thus $|GT_i| = k$, when $k = 1$. Moreover, the value of Zhou et al.'s Recall@k_i formula may not always increase with an increase in the value of k because of the second subformula in Equation (4.6).

Zhou et al. [133] used Precision@k, Recall@k, and F1-score@k, where k $\in \{5, 10\}$. However, we find that Stack Overflow only allows users to assign at most 5 tags for each post. Thus, in this work, we use more proper settings of k, i.e., k $\in \{1, 2, 3, 4, 5\}$. Similar to Zhou et al.'s work, we regard F1-score@k as the main evaluation metric as it takes into consideration both Precision@k and Recall@k[6].

4.3.5 Experimental Result

Effectiveness Comparison Table 4.2 shows a comparison in terms of effectiveness. The results show that Post2Vec$_{CNN,All,Sep}$ achieves the best performance among all variants and baselines on all evaluation metrics. And

[6]In this work, by default, we use the evaluation metrics that were used to evaluate the state-of-the-art approach [133].

Table 4.2: Effectiveness of POST2VEC and its baselines

	Precision@1	Precision@2	Precision@3	Precision@4	Precision@5
TagRCNN	0.68	0.53	0.42	0.34	0.29
TagCNN	0.70	0.56	0.45	0.37	0.32
POST2VEC$_{LSTM,All,Sep}$	0.78	0.62	0.50	0.41	0.35
POST2VEC$_{CNN,-Code,Sep}$	0.74	0.59	0.48	0.40	0.34
POST2VEC$_{CNN,All,Com}$	0.77	0.62	0.50	0.41	0.35
POST2VEC$_{CNN,All,Sep}$	0.79	0.63	0.51	0.42	0.36
	Recall@1	Recall@2	Recall@3	Recall@4	Recall@5
TagRCNN	0.68	0.57	0.53	0.52	0.54
TagCNN	0.70	0.60	0.56	0.56	0.58
POST2VEC$_{LSTM,All,Sep}$	0.78	0.67	0.62	0.63	0.65
POST2VEC$_{CNN,-Code,Sep}$	0.74	0.64	0.60	0.61	0.63
POST2VEC$_{CNN,All,Com}$	0.77	0.66	0.62	0.63	0.65
POST2VEC$_{CNN,All,Sep}$	0.79	0.68	0.64	0.64	0.66
	F1-score@1	F1-score@2	F1-score@3	F1-score@4	F1-score@5
TagRCNN	0.68	0.54	0.46	0.40	0.36
TagCNN	0.70	0.57	0.49	0.43	0.39
POST2VEC$_{LSTM,All,Sep}$	0.78	0.63	0.54	0.48	0.44
POST2VEC$_{CNN,-Code,Sep}$	0.74	0.61	0.52	0.46	0.42
POST2VEC$_{CNN,All,Com}$	0.77	0.63	0.54	0.48	0.43
POST2VEC$_{CNN,All,Sep}$	0.79	0.64	0.55	0.49	0.45

POST2VEC$_{LSTM,All,Sep}$ achieves the second best performance, while TagRCNN performs the worst. POST2VEC$_{CNN,All,Sep}$ outperforms POST2VEC$_{LSTM,All,Sep}$, POST2VEC$_{CNN,-Code,Sep}$, POST2VEC$_{CNN,All,Com}$, TagCNN, and TagRCNN by 2%, 7%, 5%, 15%, and 25% in terms of F1-score@5, respectively.

Efficiency Comparison For a fair comparison, all the approaches were run on the same machine: a 64-bit, 14.04.4 LTS GPU server with Tesla P100-SXM2-16GB[7]. Table 4.3 shows the comparison in terms of training time cost; we observed that all approaches cost more than 5 days to train the model due to the large dataset. Specifically, TagRCNN requires 8.5 days, TagCNN requires 7.1 days, POST2VEC$_{CNN,All,Sep}$ requires 7 days, POST2VEC$_{LSTM,All,Sep}$ requires 12.7 days, POST2VEC$_{CNN,-Code,Sep}$ requires 6.9 days, and POST2VEC$_{CNN,All,Com}$ requires 5.4 days. Although the training time takes days, the time these approaches take to recommend a set of tags to a post (i.e., model application time) is very fast. It took each approach 0.08 seconds per post on average. Note that training can be done once in a long while.

[7]`https://www.nvidia.com/en-us/data-center/tesla-p100`.

Table 4.3: Training time cost comparison

Approach	Time cost (Mins)
TagRCNN	12,287
TagCNN	10,278
Post2Vec$_{LSTM,All,Sep}$	18,233
Post2Vec$_{CNN,-Code,Sep}$	9,916
Post2Vec$_{CNN,All,Com}$	7,811
Post2Vec$_{CNN,All,Sep}$	10,130

4.3.6 Research Questions and Analysis

We perform further analysis to answer the three research questions.

RQ1: *Compared with the state-of-the-art approach, how effective and efficient is* Post2Vec *in tag recommendation?*

To answer RQ1, we compare Post2Vec with the state-of-the-art tag recommendation approaches (i.e., TagCNN and TagRCNN). In terms of effectiveness, our experimental results show that Post2Vec largely outperforms the baseline approaches, i.e., it improves over TagCNN and TagRCNN in terms of F1-score@5 by 15% and 25%, respectively. Moreover, in terms of efficiency, Post2Vec costs 0.1 day less than TagCNN and 1.5 days less than TagRCNN in terms of model training time. In terms of model application time (i.e., the time it takes to recommend tags to a new SO post), all approaches take a fraction of a second.

Post2Vec *outperforms the state-of-the-art approaches in terms of both effectiveness (by 15-25% in terms of F1-score@5) and efficiency (by up to 1.5 days in terms of model training time).*

RQ2: *What is the impact of different types of NNs as feature extractors?*

To answer RQ2, we compare the performance of a pair of Post2Vec's variants that only differ in the way of used NN model, i.e., Post2Vec$_{CNN,All,Sep}$ and Post2Vec$_{LSTM,All,Sep}$. From Table 4.2, we find that Post2Vec$_{CNN,All,Sep}$ outperforms Post2Vec$_{LSTM,All,Sep}$ by 3% in term of F1-score@5. We apply the Wilcoxon Signed Rank Test [110] at a 95% confidence level (i.e., p-value < 0.05) on the paired data which corresponds to the F1-score@5 of CNN-

and LSTM-based Post2Vec. We find that CNN-based Post2Vec (i.e., Post2Vec$_{CNN,All,Sep}$) significantly outperforms LSTM-based Post2Vec (i.e., Post2Vec$_{LSTM,All,Sep}$). From Table 4.3, we find that Post2Vec$_{LSTM,All,Sep}$ costs 5.6 more days than Post2Vec$_{CNN,All,Sep}$ for model training.

CNN-based Post2Vec *outperforms LSTM-based* Post2Vec *significantly and consumes fewer computing resources during model training.*

RQ3: *Does* Post2Vec *benefit from code snippets?*

To answer RQ3, we compare the performance of Post2Vec$_{CNN,All,Sep}$ and Post2Vec$_{CNN,-Code,Sep}$ since the only difference between them is the consideration of code snippets. The experimental results show that Post2Vec$_{CNN,All,Sep}$ outperforms Post2Vec$_{CNN,-Code,Sep}$ in terms of effectiveness (7% better) but requires more time to train (0.1 day more). Post2Vec$_{CNN,-Code,Sep}$ is faster to train than Post2Vec$_{CNN,All,Sep}$ since Post2Vec$_{CNN,-Code,Sep}$ has a similar but simpler architecture (i.e., it ignores the code snippets).

Consideration of code snippets can boost effectiveness (i.e., a boost in F1-score@5 by 7%) but consumes more computing resources during model training.

RQ4: *What is the impact of handling post components separately rather than combining them together?*

To answer RQ4, we compare the performance of Post2Vec$_{CNN,All,Com}$ and Post2Vec$_{CNN,All,Sep}$ because both of them utilized title, description, and code snippets as input data but Post2Vec$_{CNN,All,Sep}$ separately employs three CNN models for the title while description, and code snippets, Post2Vec$_{CNN,All,Com}$ concatenates them as one document and feed it to a single CNN model. The experimental results show that Post2Vec$_{CNN,All,Sep}$ outperforms Post2Vec$_{CNN,All,Com}$ over all the evaluation metrics; in terms of F1-score@5, Post2Vec$_{CNN,All,Com}$ is 5% better than Post2Vec$_{CNN,All,Com}$. We apply the Wilcoxon Signed Rank Test [110] at a 95% confidence level (i.e., p-value < 0.05) on the paired data which corresponds to the F1-score@5 and we find that Post2Vec$_{CNN,All,Sep}$ significantly outperforms Post2Vec$_{CNN,All,Com}$. But Post2Vec$_{CNN,All,Com}$ is 22% more efficient.

Post2Vec*'s architecture that considers different post components (i.e., title, description, and code snippets) separately significantly boosts the performance by 5% in terms of F1-score@5 but consumes more computing resources than considers different components together during model training.*

For the rest of the dissertation, we refer the variant Post2Vec$_{CNN,All,Sep}$ to Post2Vec.

4.4 Post2Vec for Downstream Tasks

As stated earlier, the goal of this work is to build a Stack Overflow post representation that can be useful for multiple downstream tasks. In the training phase, our approach uses tag as a guide to aid the learning of post representation. A tag is a word or phrase that is attached to a post as a result of a crowd sourced process and serves as a semantic label of the post. Downstream tasks that are poised to benefit more by representations learned by our approach are those where: (1) the downstream task can benefit from an abstraction of the post; (2) the tags in our training data capture suitable semantics that are needed for the task. In this section, we pick three tasks that satisfy the above two criteria.

4.4.1 Baseline Approaches

We first compare against state-of-the-art techniques proposed by Xu et al. [115] and Beyer et al. [10] for relatedness prediction and post classification tasks, respectively. Furthermore, we also compare the performance of our approach with two deep learning based techniques, i.e., TagCNN [133] and Doc2Vec [57], that can also be used to generate post representations.

TagCNN forms a single output vector (i.e., post vector) in the penultimate layer, and subsequently pass the vector to a fully connected softmax layer to compute the probability distribution over tags. Thus, we fed the posts used in the downstream tasks to the TagCNN model which was trained for the tag recommendation task (refer to Section 4.3) and collected the corresponding vectors of those posts. Moreover, as posts are also documents, we use Doc2Vec (i.e., a document embedding algorithm) as another baseline approach. We follow the Doc2Vec setting used in [116] as they also use Doc2Vec to analyze Stack Overflow posts. We use Gensim [84] to implement the Doc2Vec model. For a fair comparison, the same training dataset is used to build Post2Vec, TagCNN, and Doc2Vec models, i.e., the training dataset used for the tag recommendation task.

4.4.2 Task 1: Relatedness Prediction

The task of automatically identifying such related posts or knowledge units is described as *relatedness prediction*, aka., linkable knowledge prediction. It can support developers for different purposes; for example, searching for related solutions to better solve a particular problem, or gathering new knowledge by browsing related SQA posts [119, 115].

4.4.2.1 Problem Formulation

The *relatedness prediction* is formulated as a multi-class classification problem. Given a set of pairs of software posts, labels are assigned to them to describe how related they are. The labels are based on the order of relatedness from most to least related, i.e., *Duplicate* > *Direct* > *Indirect* > *Isolated.* There is only one possible relatedness label between two posts. For more details, please refer to Chapter 3.

4.4.2.2 State-of-the-art Approach

The state-of-the-art *relatedness prediction* approach named SoftSVM was proposed by Xu et al. [115]. SoftSVM introduced a soft-cosine similarity which is able to capture the degree of relatedness when the text of posts share related words. To measure the degree of relatedness, SoftSVM constructs a four-dimensional vector which corresponds to four types of features, 1) cosine similarity based on Stack Overflow data cos_{SO}, 2) soft-cosine similarity based on Stack Overflow data $soft_{SO}$, 3) soft-cosine similarity based on pre-trained Google word2vec $soft_{Google}$, and 4) soft-cosine similarity based on Levenshtein distance $soft_{Edit}$. For more details on these four features, please refer to [115]. To summarize, for a given pair of posts, a four-dimensional feature vector $FV_{SoftSVM}$ is constructed where each dimension corresponds to each feature value, i.e., $\langle cos_{SO}, soft_{SO}, soft_{Google}, soft_{Edit}\rangle$.

4.4.2.3 Dataset

We use the same dataset used in the state-of-the-art work [115]. The dataset contains 40,000 pairs of posts with the four types of relatedness. Detailedly, each type of relatedness corresponds to 10,000 pairs of posts, i.e., the dataset is balanced with respect to class.

4.4.2.4 Experimental Setting

We hypothesize that there is a correlation between the post vector distance and the relatedness, i.e., the smaller the distance between two generated post vectors, the more related these two posts are. To evaluate the post representation generated by the three approaches (i.e., Post2Vec, TagCNN and Doc2Vec), we consider post vector distance as an extra feature and combine it with the features used in the state-of-the-art relatedness prediction approach. In particular, we add the Euclidean distance [85] (see Equation 4.8) between two post vectors as an additional feature, namely *Dist*.

$$Dist(\mathbf{p},\mathbf{q}) = Dist(\mathbf{q},\mathbf{p}) = \sqrt{\sum_{i=1}^{Dimension} (q_i - p_i)^2} \qquad (4.8)$$

Next, we append *Dist* to the feature vector used by the state-of-the-art ap-

Table 4.4: Performance Comparison for Relatedness Prediction

	Feature Vector	Duplicate	Direct	Indirect	Isolated	**Overall**
Precision	$FV_{SoftSVM}$	0.47	0.41	0.43	0.73	0.51
	$FV_{SoftSVM+D2V}$	0.48	0.41	0.44	0.73	0.51
	$FV_{SoftSVM+TagCNN}$	0.53	0.44	0.46	0.76	0.55
	$FV_{SoftSVM+P2V}$	0.53	0.45	0.48	0.77	**0.56**
Recall	$FV_{SoftSVM}$	0.54	0.19	0.49	0.91	0.53
	$FV_{SoftSVM+D2V}$	0.50	0.23	0.50	0.91	0.54
	$FV_{SoftSVM+TagCNN}$	0.52	0.33	0.49	0.91	0.56
	$FV_{SoftSVM+P2V}$	0.61	0.31	0.48	0.91	**0.58**
F1-score	$FV_{SoftSVM}$	0.51	0.26	0.46	0.81	0.51
	$FV_{SoftSVM+D2V}$	0.49	0.30	0.47	0.81	0.52
	$FV_{SoftSVM+TagCNN}$	0.52	0.38	0.47	0.83	0.55
	$FV_{SoftSVM+P2V}$	0.57	0.36	0.48	0.83	**0.56**

proach. In this way, we construct a five-dimensional feature vector $FV_{SoftSVM+Dist}$ for each pair of posts, i.e., $\langle cos_{SO}, soft_{SO}, soft_{Google}, soft_{Edit}, Dist\rangle$. We name the feature vector based on POST2VEC, TagCNN, Doc2Vec as $FV_{SoftSVM+P2V}$, $FV_{SoftSVM+TagCNN}$, and $FV_{SoftSVM+D2V}$, respectively. To facilitate comparison, we feed the feature vector $FV_{SoftSVM+Dist}$ to the same kind of traditional machine learning classifier (i.e., SVM) with the same setting as described in [115]. Implementation-wise, we reuse the source code released by Xu et al.[8] and modify the feature vector part. Then, ten times ten-fold cross-validation is performed to evaluate predictive models.

4.4.2.5 Evaluation Metrics

We use precision, recall, and F1-score, which were also used by Xu et al. as evaluation metrics. For the definitions of precision, recall, and F1-score, please refer to Section 3.2.2. Higher values of the above metrics indicate better performance. Following [115], F1-score is regarded as the main metric.

4.4.2.6 Experimental Result

Table 4.4 shows the performance obtained using the feature vector $FV_{SoftSVM}$ and $FV_{SoftSVM+Dist}$ generated by the three approaches. The results of the

[8] `https://github.com/XBWer/ESEM2018`.

experiment show that $FV_{SoftSVM+P2V}$ achieves the best performance. And $FV_{SoftSVM+TagCNN}$ achieves the second best, and $FV_{SoftSVM}$ achieves worst in terms of precision, recall, and F1-score. We also find that when using $FV_{SoftSVM+P2V}$ as feature vector, in terms of overall F1-score, it outperforms $FV_{SoftSVM+TagCNN}$, $FV_{SoftSVM+D2V}$, and $FV_{SoftSVM}$ by 2%, 8%, and 10%, respectively. We apply the Wilcoxon Signed Rank Test [110] at a 95% significance level (i.e., p-value < 0.05) on the paired data which corresponds to the F1-score of our approach, i.e., Post2Vec, and the best performing baseline approach, i.e., TagCNN. We find that Post2Vec significantly outperforms TagCNN in terms of F1-score.

Overall, we are able to boost the performance of the state-of-the-art relatedness prediction approach by leveraging the post vectors generated by Post2Vec.

4.4.3 Task 2: Post Classification

Beyer et al. [10] presented a taxonomy for Android-related Stack Overflow (SO) posts and manually labeled 500 posts. The taxonomy includes seven categories in which an Android post can be classified, namely, API change, API usage, Conceptual, Discrepancy, Learning, Errors, and Review.

4.4.3.1 Problem Formulation

The post classification problem is formulated as a multi-label classification problem. Given a set of software question and answer posts that are already labeled with one or multiple categories, a new post needs to be classified into a set of appropriate categories (i.e., API change, API usage, Conceptual, Discrepancy, Learning, Errors, and Review).

4.4.3.2 State-of-the-art Approach

To classify posts into multiple categories, Beyer et al. [10] constructed a binary classifier for each category. To construct a classifier, they considered 82 different configurations based on, e.g., whether data balancing is performed, which classifier is used, whether parts-of-speech tags are included in the feature vector representing a post, etc. The best configuration is then determined for each category, and the corresponding results reported.

4.4.3.3 Dataset

To facilitate comparison, we have used the same dataset used by Beyer et al. [10]. The dataset contains 500 posts which have been manually labeled, with 30, 206, 145, 129, 79, 30, and 93 posts labeled as API CHANGE, API USAGE, CONCEPTUAL, DISCREPANCY, LEARNING, ERRORS, and REVIEW, respectively.

4.4.3.4 Experimental Setting

Our goal is to investigate whether leveraging post representation generated using Post2Vec, TagCNN, and Doc2Vec can boost the effectiveness of Beyer et al.'s state-of-the-art approach. Thus, we reuse most of Beyer et al.'s proposed architecture that considers the 82 configurations and determines the best one. We refer to Beyer et al.'s original approach as FV_{STOA}. The only change that we made is to enhance the feature vector that FV_{STOA} uses to represent a post. Specifically, we construct a new feature vector for each post by appending FV_{STOA}'s feature vector with the corresponding post's representation that is learned by using either POST2VEC, TagCNN or Doc2Vec. We refer to the approaches that use the feature vectors enhanced with post representation learned using POST2VEC, TagCNN or Doc2Vec as $FV_{STOA+P2V}$, $FV_{STOA+TagCNN}$, and $FV_{STOA+D2V}$ respectively. Implementation-wise, we reused the source code released by Beyer et al.[9] and only modified the feature vector part.

Beyer et al. divided their manually-labeled set of 500 posts into training and test sets, with 90% (i.e., 450) of the posts used for training and 10% (i.e., 50) for testing. Since the dataset is unbalanced, random stratified sampling [53] is applied to ensure that the test set contains 10% or at least three posts of each category. The experiment is repeated 50 times using the random stratified sampling, and the average scores of the evaluation metrics (described below) are reported.

4.4.3.5 Evaluation Metrics

We use precision, recall, and F1-score, which were also used as evaluation metrics by Beyer et al. For each category, these metrics are defined for both sides of the classification: whether a post is classified correctly as belonging to the category ($class_T$) and whether a post is classified correctly as not belonging to the category ($class_F$). For all of the above metrics, higher values indicate better performance. F1-score for $class_T$ (denoted as f_T) is regarded as the main metric since the dataset is unbalanced and the minority is $class_T$ (i.e., belonging to a category).

4.4.3.6 Experimental Result

Table 4.5 shows the performance of the post classification task focusing on f_T and the other two metrics used to derive it (pre$_T$ and rec$_T$). The experimental results show that $FV_{STOA+P2V}$ achieves the best average number of f_T and also the best performance for the most number of categories. Specifically, across the 7 categories, on average $FV_{STOA+P2V}$ outperforms FV_{STOA} by 7% in terms of f_T. We apply the Wilcoxon Signed Rank Test [110] at a 95% significance level (i.e., p-value < 0.05) on the paired data which corresponds to the f_T of the feature vector based on our approach, i.e., $FV_{STOA+P2V}$, and the best perform-

[9]`https://github.com/icpc18submission34/icpc18submission34`.

ing baseline approach, i.e., FV_{STOA}. We find that $FV_{STOA\,+\,P2V}$ significantly outperforms FV_{STOA} in terms of f_T. Moreover, we find that $FV_{STOA\,+\,D2V}$ and $FV_{STOA\,+\,TagCNN}$ perform worse than FV_{STOA}. Also, none of them achieve the best performance for a single category. It indicates that Post2Vec's learned representation enhances the original feature vector, while the same cannot be said for representations learned by TagCNN and Doc2Vec.

Table 4.5: Performance Comparison for Post Classification. pre$_T$, rec$_T$, f$_T$ = precision, recall and F1-score for $class_T$.

Category	pre$_T$			
	FV_{STOA}	$FV_{STOA\,+\,D2V}$	$FV_{STOA\,+\,TagCNN}$	$FV_{STOA\,+\,P2V}$
API_CHANGE	0.59	0.60	0.74	0.67
API_USAGE	0.91	0.82	0.94	0.88
CONCEPTUAL	0.59	0.47	0.62	0.58
DISCREPANCY	0.41	0.46	0.48	0.53
DOCUMENTATION	0.56	0.35	0.40	0.64
ERRORS	0.88	0.86	0.56	0.93
REVIEW	0.42	0.38	0.47	0.53
Average	0.62	0.56	0.60	**0.68**
	rec$_T$			
	FV_{STOA}	$FV_{STOA\,+\,D2V}$	$FV_{STOA\,+\,TagCNN}$	$FV_{STOA\,+\,P2V}$
API_CHANGE	0.95	0.76	0.73	0.77
API_USAGE	0.78	0.58	0.58	0.71
CONCEPTUAL	0.62	0.74	0.46	0.68
DISCREPANCY	0.87	0.63	0.69	0.76
DOCUMENTATION	0.55	0.27	0.18	0.38
ERRORS	0.33	0.21	0.34	0.70
REVIEW	0.54	0.49	0.25	0.48
Average	0.66	0.53	0.46	0.64
	f$_T$			
	FV_{STOA}	$FV_{STOA\,+\,D2V}$	$FV_{STOA\,+\,TagCNN}$	$FV_{STOA\,+\,P2V}$
API_CHANGE	0.71	0.65	0.71	0.68
API_USAGE	0.84	0.68	0.71	0.78
CONCEPTUAL	0.59	0.57	0.52	0.62
DISCREPANCY	0.56	0.53	0.56	0.62
DOCUMENTATION	0.53	0.28	0.24	0.45
ERRORS	0.46	0.33	0.37	0.79
REVIEW	0.46	0.42	0.31	0.48
Average	0.59	0.49	0.49	**0.63**

4.4.4 Task 3: API Recommendation

According to a survey conducted by Huang et al. [45], API recommendation approach is desired and Stack Overflow is an important resource for developers.

4.4.4.1 Problem Formulation

In the literature, the API recommendation problem is formed as given a query in natural language that describes a technical question, recommending a certain API or API sequence to answer the question [83, 35, 45]. Depending on the granularity of the recommended API, the problem is divided into two categories, method- and class-level API recommendation.

4.4.4.2 State-of-the-art Approach

Huang et al. [45] focus on the API recommendation at the method level and present an approach named BIKER which recommends API based on Stack Overflow data. The framework of BIKER mainly contains three stages. First, given a query and Stack Overflow data dump, BIKER collects the top k most similar questions for the query by measuring the similarity between the questions and the given query. Second, it extracts API from the answers of the k similar questions and ranks them by computing similarity between each API candidate and the query. Third, BIKER selects the top 5 most similar APIs and summarizes them by considering multiple sources, e.g., the information in the official description, title of the similar questions, and the code snippets exist in the answers of the similar questions.

4.4.4.3 Experimental Setting

In the first stage of BIKER, it computes the similarity between each question in the Stack Overflow data dump and the query based on a word IDF (inverse document frequency) vocabulary and a word embedding model. Based on the similarity, all the questions are ranked and top k of them are selected as similar questions to the query.

To investigate whether leveraging the post representation generated by using POST2VEC, TagCNN, and Doc2Vec can boost the effectiveness of BIKER, we embed the post representation into the rank list of the similar questions based on a new perspective, i.e., considering similarity between questions based on the post representation. Given a query, we first collect the rank list with top k most similar questions and their similarities to the query returned by BIKER, i.e.,

$$\begin{aligned} RankList = \{TQ_1 &: \{Sim\langle TQ_1, Query\rangle\}, \\ TQ_2 &: \{Sim\langle TQ_2, Query\rangle\}, \\ &..., \\ TQ_k &: \{Sim\langle TQ_k, Query\rangle\}\} \end{aligned}$$

And then we collect 5 questions in the data dump with the least euclidean distance to each of the similar questions TQ_i based on the post representation, i.e.,

$$\begin{aligned} TQ_i \Rightarrow TQSimQs_i = \{&TQSimQ_i^1, \\ &TQSimQ_i^2, \\ &..., \\ &TQSimQ_i^5\} \end{aligned}$$

We set the similarity between two questions as $Sim\langle Q_i, Q_j\rangle = 1/(1+dist\langle Q_i, Q_j\rangle)$. Last, we add the top 5 questions (i.e., $TQSimQs$) to the rank list $RankList$. For each question $TQSimQ_i^j$, we set its similarity to the query by multiplying two similarities, (1) similarity between the corresponding top question and query, (2) similarity between the corresponding top question and the question, i.e.,

$$\begin{aligned} Sim\langle TQSimQ_i^j, Query\rangle = &Sim\langle Q_i, Query\rangle \\ &\times Sim\langle TQ_i, TQSimQ_i^j\rangle \end{aligned}$$

In this way, the rank list of similar questions is expanded and its top k questions are fed to the next stage of BIKER. We follow the same setting in [45] and set k as 50.

4.4.4.4 User Study

To investigate whether BIKER can help developers find the appropriate APIs more efficiently and accurately in practice, Huang et al. conducted a user study [45]. Thus, we conduct a user study to evaluate the performance of two versions of BIKER, i.e., before and after integrating the post representation.

Experimental queries and ground-truth APIs. In [45], 10 queries are selected as experimental queries. In this work, we randomly select 50 queries which are 5 times more. And for each query, we collect its corresponding API or API sequence which is regarded as the ground truth.

Participants. We recruited 8 participants from the first author's university, 1 postdoc, 3 PhDs, and 4 research engineers. The years of their developing experience vary from 3 to 7 years, with an average of 4.6 years.

Experimental groups. We divided the participants uniformly based on years of development experience into four groups, (1) **BIKER**: the full-feature of original BIKER, (2) **BIKER+D2V**: the enhanced BIKER which integrated the post representation learned by doc2vec, (3) **BIKER+TagCNN**: the enhanced BIKER which integrated the post representation learned by TagCNN, (4) **BIKER+Post2Vec**: the enhanced BIKER which integrated the post representation learned by POST2VEC.

Evaluation metrics. To measure the accuracy of BIKER, two evaluation metrics have been used in [45], correctness and time cost. Correctness is used to evaluate whether a participant can find the correct APIs. For a query that only needs one API method, correctness is 1 if the participant submitted only one API and it's correct, otherwise, it is 0. For a query that needs an API sequence, correctness is the proportion of the correct APIs submitted by the participant among all APIs in the correct API sequence. Time cost evaluates how fast a participant can answer a question.

Result analysis. From Table 4.6, we find that BIKER+Post2Vec achieves the best performance, BIKER achieves the second-best, BIKER+TagCNN performs the worst, in terms of both correctness and time cost. We find that the correctness of BIKER is improved by 10% and the time cost is reduced by 18% after integrating the post representation learned by Post2Vec. Moreover, we also find that only the representation learned by Post2Vec boosts the performance of BIKER and the other two baselines harm the performance.

Table 4.6: Performance Comparison for API Recommendation

	Correctness	Time Cost (in Seconds)
BIKER	0.40	67
BIKER+D2V	0.39	80
BIKER+TagCNN	0.37	108
BIKER+P2V	**0.44**	**55**

Table 4.7 presents the recommendation result returned by BIKER and BIKER+Post2Vec for an experimental query, *Converting a time String to ISO 8601 format.* We find that the correct API (i.e., *java.time.Instant.toString()*) does not exist in all the APIs recommended by BIKER but it is ranked first by BIKER+Post2Vec. It indicates that BIKER+Post2Vec successfully identifies similar questions which cover the ground truth but BIKER fails. Thus, the result shows that integrating the post representation learned by Post2Vec can boost the performance of BIKER.

4.5 Discussion

4.5.1 Qualitative Analysis

Recall that the goal of Post2Vec is to embed posts to a vector space where it is easy to compute meaningful distances. Ideally, the distance measure between closely related posts should be small to capture the perception of similarity. Also recall that, for the relatedness prediction task, there are four relatedness

Table 4.7: Recommendation returned by BIKER and BIKER+Post2Vec for an experimental query

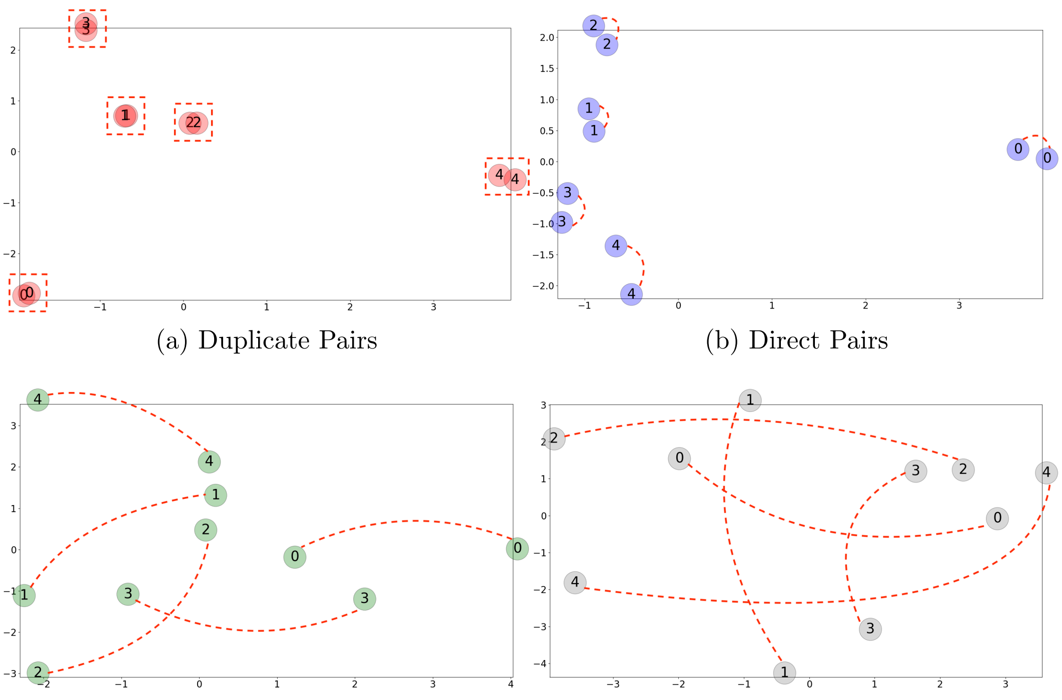
Query: ***Converting a time String to ISO 8601 format***		
1st Recommendation	**BIKER**	**BIKER+Post2Vec**
API	java.lang.String.format	**java.time.Instant.toString**✓
Java Doc	Returns a formatted string using the specified format string and arguments.	A string representation of this instant using ISO-8601 representation.
Relevant SO Questions	1.Convert String to UUID formatted string (25895225*) 2.String.format runtime formatting (19803405*) 3.Formatting: String cannot be converted to String[] (35702926*)	1.Format Instant to String (25229124*)

Code Snippets	/***code snippet1***/
	String plain = "...";
	String uuid = String.format("...", plain.substring(0,7));
	
	/***code snippet2***/
	int width = ...;
	String fmt = String.format("...", width);
	String formatted = String.format (fmt, "Hello", "World");
	
	/***code snippet3***/
	dataArray += String.format("...", b.getTitle(), b.getIsbn());
		/***code snippet1***/
		DateTimeFormatter formatter = DateTimeFormatter.ofPattern("...");
		String text = date.toString(formatter);
		LocalDate date = LocalDate.parse(text, formatter);

*: Unique question Id in Stack Overflow.

types among pairs of posts, and the order of relatedness from most to least related is *Duplicate* > *Direct* > *Indirect* > *Isolated*. The experimental results in Section 4.4.3 show that the state-of-the-art approach is improved by adding distance between post vectors as an additional feature.

To further investigate the capability of Post2Vec, we visualize the vector space of randomly selected 20 pairs of posts with different types of relatedness labels (details are shown in Table 4.8) and collected their corresponding post vectors. Figure 4.4 presents the post vector space learned by Post2Vec, visualized in a two-dimensional space using the t-SNE dimensionality reduction technique [67]. Specifically, Figures 4.4a, 4.4b, 4.4c, and 4.4d present vector


(a) Duplicate Pairs (b) Direct Pairs

(c) Indirect Pairs (d) Isolated Pairs

Figure 4.4: Vector Space of POST2VEC on Relatedness Prediction

space of POST2VEC for pairs of post with different types of relatedness labels, *duplicate*, *direct*, *indirect*, and *isolated*, respectively. The figures show that the rank list of average distance between pairs of posts which belong to the same type of relatedness is: *duplicate* (Figure 4.4a) $<$ *direct* (Figure 4.4b) $<$ *indirect* (Figure 4.4c) $<$ *isolated* (Figure 4.4d). In other words, post vectors are closer to each other when the corresponding pair of posts are more related. This further demonstrates the quality of post vectors generated by POST2VEC.

4.5.2 Quantitative Analysis

4.5.2.1 Stability of Post2Vec Representation

We find that the performance rankings of different post representations for different downstream tasks are not always consistent. Take TagCNN as an example, it performs second-best in relatedness prediction but performs worst in post classification and API recommendation. Moreover, after integrating the post representation learned by D2V and TagCNN into the state-of-the-

Table 4.8: Pairs in Figure 4.4a, 4.4b, 4.4c, and 4.4d

pairId	id1	id2	label
0	10499846	10499780	duplicate
1	41995162	41985606	duplicate
2	25265734	25269382	duplicate
3	12399031	12399168	duplicate
4	21347995	21579452	duplicate
0	9701688	9690628	direct
1	4807299	27781975	direct
2	46692249	43321202	direct
3	12884573	40770990	direct
4	16821771	13944155	direct
0	24791298	27667086	indirect
1	7183010	37101318	indirect
2	32147303	23247539	indirect
3	36966266	40257154	indirect
4	23018048	29306473	indirect
0	25441021	45652298	isolated
1	42996077	35741584	isolated
2	25335343	22510809	isolated
3	25763556	2081159	isolated
4	40324266	33151168	isolated

art approaches, the performance becomes worse for the post-classification (see Table 4.5) and API recommendation (see Table 4.6) tasks. It demonstrates that the performance could be even harmed when integrating unsuitable post representation. Overall, the experimental result demonstrates that the post representation learned by Post2Vec is not only higher quality but also more stable than others.

4.5.2.2 Robustness of Post2Vec Representation

Recall that there are four handcrafted features proposed in the original work of relatedness prediction and we append the distance between posts' vectors generated by Post2Vec (denoted by $P2V_{Dist}$) to boost the features (Section 4.4.2.4). In this way, we construct a five-dimensional feature vector (FV) for each pair of posts, i.e., $\langle cos_{SO}$, $soft_{SO}$, $soft_{Google}$, $soft_{Edit}$, $P2V_{Dist}\rangle$.

To measure the relative importance of the feature generated by Post2Vec, we remove each component of FV one-at-a-time and measure the corresponding performance (in terms of F1-score; which is the summary metric). The most important feature should result in the largest drop in performance when it is removed. As shown in Table 4.9, we find that F1-score decreases the most when Dist (the feature generated by Post2Vec) is removed. This shows that the representation learned by Post2Vec is the most important feature among the five. The rank of the relative importance of the five features is $P2V_{Dist} > soft_{SO} > soft_{Google} > cos_{SO} = soft_{Edit}$.

Table 4.9: Related Importance of Different Combinations of Features for Relatedness Prediction

	Feature Vector	Duplicate	Direct	Indirect	Isolated	**Overall**
Precision	$FV_{w/o\ cos_{SO}}$	0.53	0.45	0.48	0.77	0.55
	$FV_{w/o\ soft_{SO}}$	0.53	0.45	0.46	0.69	0.53
	$FV_{w/o\ soft_{Google}}$	0.53	0.44	0.48	0.76	0.55
	$FV_{w/o\ soft_{Edit}}$	0.53	0.45	0.48	0.77	0.55
	$FV_{w/o\ P2V_{Dist}}$	0.47	0.41	0.43	0.73	0.51
	FV_{all}	0.53	0.45	0.48	0.77	**0.56**
Recall	$FV_{w/o\ cos_{SO}}$	0.60	0.31	0.48	0.91	0.57
	$FV_{w/o\ soft_{SO}}$	0.60	0.32	0.38	0.93	0.55
	$FV_{w/o\ soft_{Google}}$	0.61	0.31	0.47	0.91	0.57
	$FV_{w/o\ soft_{Edit}}$	0.60	0.30	0.48	0.91	0.57
	$FV_{w/o\ P2V_{Dist}}$	0.54	0.19	0.49	0.91	0.53
	FV_{all}	0.61	0.31	0.48	0.91	**0.58**
F1-score	$FV_{w/o\ cos_{SO}}$	0.56	0.37	0.48	0.83	0.56
	$FV_{w/o\ soft_{SO}}$	0.57	0.37	0.42	0.79	0.53
	$FV_{w/o\ soft_{Google}}$	0.57	0.37	0.47	0.83	0.55
	$FV_{w/o\ soft_{Edit}}$	0.56	0.36	0.48	0.83	0.56
	$FV_{w/o\ P2V_{Dist}}$	0.51	0.26	0.46	0.81	0.51
	FV_{all}	0.57	0.36	0.48	0.83	**0.56**

4.5.3 Vector Space Visualization

To further investigate the clustering quality of post vectors learned by Post2Vec, we randomly sample a subset of posts with a significant number from the dataset of relatedness prediction and visualize the vector space learned by Post2Vec. Out of the 51,918 posts from the dataset, we randomly consider 8,105 posts (i.e., the exact number provided by the Sample Size Calculator[10]

[10]`https://www.surveysystem.com/sscalc.htm`

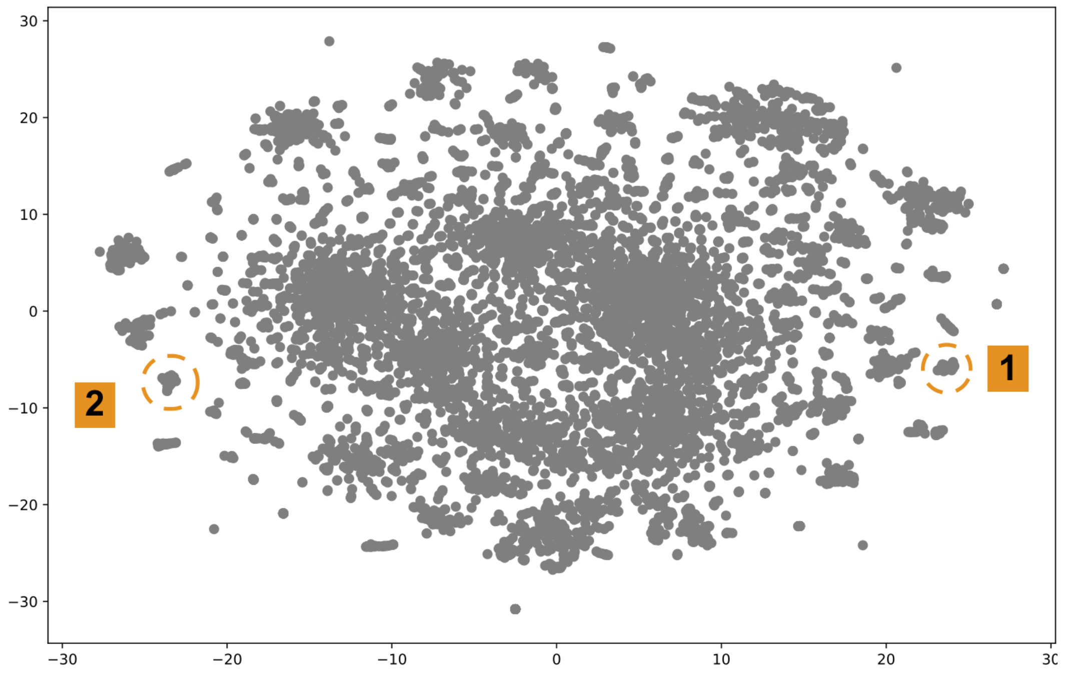
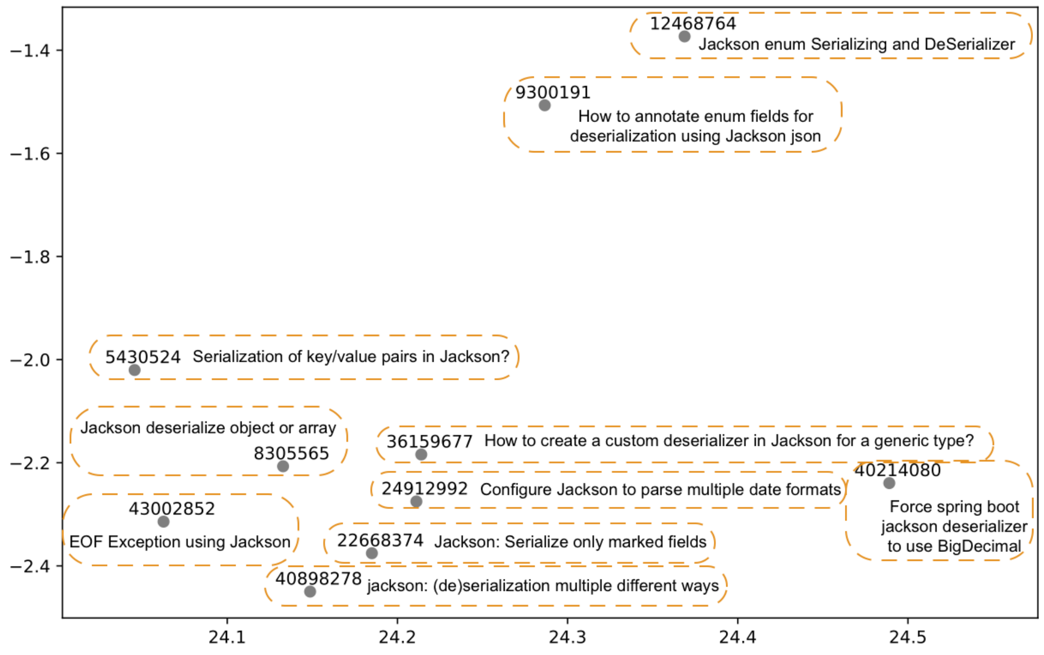


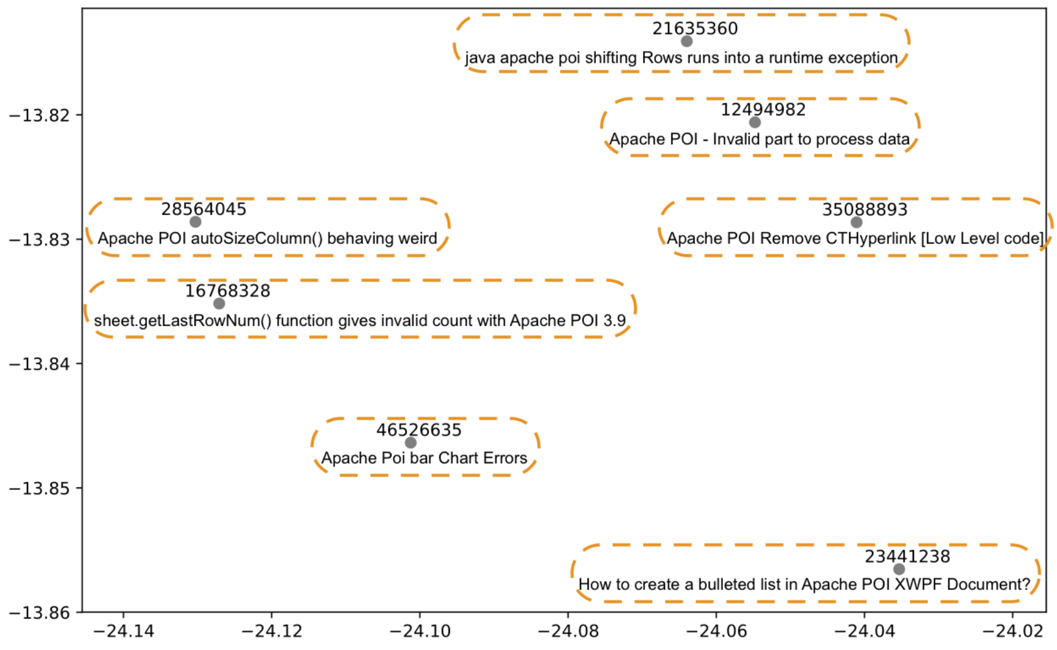
Figure 4.5: Vector Space



*: The numbers represent posts' Ids in Stack Overflow.

Figure 4.6: An *Jackson*-related Cluster (#1) of Posts in Figure 4.5

using 95% for the confidence level). Different from the vector space visualization for the relatedness prediction in Section 4.5.1, we omit the relatedness labels between posts. We utilize the t-SNE dimensionality reduction technique [67] to visualize the vector space learned by our approach. We find that



*: The numbers represent posts' Ids in Stack Overflow.

Figure 4.7: An *Apache POI*-related Cluster (#2) of Posts in Figure 4.5

the learned vector space contains different sizes of clusters. Furthermore, we arbitrarily pick two clusters (as shown in Figure 4.6 and Figure 4.7) and manually read all the included posts. We find that posts in a cluster are related as they are mainly about one or a few specific technologies. For example, all the posts in cluster #1 (i.e., Figure 4.6) are related to a Java API named *Jackson* while most of the posts (except one) in cluster #2 are related to another API called *Apache POI*. To some extent, it also explains that the boundaries between clusters are not always clear as the posts may be related to multiple technologies instead of only one.

Within a cluster, we find that the closer two posts' vectors are, the more sentiment similar they are. Take a pair of posts in cluster #1 as an example (i.e., Figure 4.6), the two posts (Id = 9300191 and 12468764) are close to each other in the vector space but far from others. The reason is that only these two posts are related to deserializing enumeration by using Jackson. The same observation can also be found in cluster #2 which is related to another technology named *Apache POI*. Within cluster #2, the pair of posts that most close to each other (i.e., Id = 28564045 and 16768328) are related to version 3.9 of the API *Apache POI* and the usage of API function calls for editing sheets. Above all, two posts' vectors are likely to be close to each other in the vector space learned by our approach if they are similar semantically.

4.5.4 Threats to Validity

A threat to internal validity relates to our assumption that tags are labeled correctly by users in Stack Overflow; however, mislabeling could happen sometimes. Still, we believe that Stack Overflow effective crowdsourcing process helps to reduce the number of such cases. Also, some works utilize a validation dataset to tune hyperparameters during training to reduce overfitting likelihood. However, similar to many prior works [41, 46], for efficiency reason, we did not tune the hyperparameters. Tuning them can potentially boost POST2VEC effectiveness. Still, the experimental results have demonstrated that POST2VEC can improve the effectiveness of state-of-the-art approaches in several tasks, which suggests that the learned post representation does not overfit training data.

One threat to external validity relates to the generality of the distributed representation generated by POST2VEC. Related works like fun2vec [24] and code2vec [5], that aim to generate a representation for functions and methods, only consider at most one downstream task. In this work, we have reduced this threat by considering two downstream tasks. Another threat to external validity is that the effectiveness of our distributed representation depends on the richness of the tags. We observed that there are more than 58k tags in Stack Overflow on August 2019 and on average every post is labeled with 3 tags. Moreover, Stack Overflow encourages users to label their questions with existing tags and requires them to avoid meta-tags (i.e., generic tags that do not describe the content of the question, e.g., *beginners*) when creating new tags. [11] The aforementioned facts suggest that the tags in Stack Overflow are commonly used to characterize posts and they are meaningful for capturing the contents of posts. In the future, we will explore other potential information (in addition to the tags) that can also be used to guide the representation learning process.

Threats to construct validity relate to the evaluation metrics that we consider. We reuse the evaluation metrics that were used in the original tag prediction, relatedness prediction, and post classification work that we extend. Most of the metrics are also well-known. Thus, we believe that this threat is minimal.

4.6 Conclusion and Future Work

In this work, we present a novel neural network based architecture, namely Post2Vec, which learns distributed representation of Stack Overflow posts by using the tags assigned by users to such posts as a guide. Post2Vec architecture takes advantage of code snippets and handles different components (i.e., title, description, and code snippets) separately.

To evaluate the quality of Post2Vec's deep learning architecture, we first investigate its end-to-end effectiveness in tag recommendation task. The experimental results showed that Post2Vec outperforms the best performing state-of-the-art deep learning based tag recommendation approaches, suggesting its ability to learn better post representations. Furthermore, to evaluate the value of representation learned using Post2Vec, we integrated the generated post representations into the learning pipeline of three downstream tasks, i.e., relatedness prediction, post classification, and API recommendation. We found that the post representation learned using Post2Vec can boost the effectiveness of state-of-the-art approaches' performance by a substantial margin.

[11] Users guide on tags in Stack Overflow, `https://stackoverflow.com/help/tagging`.

In the future, we would like to explore the potential improvements of post representation by integrating more information, e.g., comments and answers. We also plan to investigate the effectiveness of other deep neural network models (beyond CNN and RNN) to be used in the feature extraction layer of Post2Vec. We are also interested in investigating different (component) combination strategies to further boost the performance. Moreover, we plan to further investigate the effectiveness of post representation learned by POST2VEC in additional downstream tasks. It would also be interesting to investigate the applicability of POST2VEC to additional software question and answer sites beyond Stack Overflow.

Chapter 5

Automated Knowledge Summarization

5.1 Introduction

Answers on Stack Overflow have become an important body of knowledge for solving developers' technical questions. Typically, developers formulate their questions as a query to some search engine, and the search engine returns a list of relevant posts that may contain answers. Then, developers need to read the returned posts and digest the information in them to find the answers to their questions. Information seeking is rendered difficult by the sheer amount of questions and answers available on the Q&A site.

We survey 72 developers in two IT companies (i.e., Hengtian and Insigma Global Service) with two questions: *(1) whether you need a technique to provide direct answers when you post a question/query online and why?* and (2) *what is your expectation of the automatically generated answers, e.g., must the answers be accurate?* All the developers agree that they need some automated technique to provide direct answers to a question/query posted online. The reasons they give include (1) sometimes it is hard to describe the problem they meet, so some hints would be useful, (2) there is too much noisy and redundant information online, (3) the answers in long posts are hard to find, and (4) even the answer they found may cover only one aspect of the problem. Developers expect the answer generation tool to provide a succinct and diverse summary of potential answers, which can help them understand the problem and refine the queries/questions.

Our survey reveals a great need to provide improved techniques for information retrieval and exploration. In this work, we aim to develop an automated technique for generating answer summary to developers' technical questions, instead of merely returning answer ports containing answers. Many developers' technical questions are non-factoid questions [91], for example, *what are differences between HashTable and HashMap?*, *How do I write logs and display them realtime in Java Swing?* For such non-factoid technical questions, multiple sparse and diverse sentences may make up the answer summary together.

We formulate our task as a *query-focused multi-answer-posts summarization* task for a given input question. This task is closely related to question answering task [102, 49, 8], which aims to find information from a huge text base to answer a question. An answer summary from the text base should provide related information with respect to the query question. However, the answer sentences and the query question are highly asymmetric on the information they convey. They may not share lexical units. Instead, they may only be semantically related (see examples in Table 5.1 and Table 5.2). The inherent lexical gap between the answer sentences and the questions imposes a major challenge for the non-factoid question answering task.

To tackle this challenge, we develop a three-stage framework to achieve the goal of generating an answer summary for a non-factoid technical question. In the first stage, we retrieve a set of relevant questions based on the question titles' relevance to a query question. The question titles and the query question are "parallel text" whose relevance is easier to determine. However, the question titles and the query question often still have lexical gaps. Inspired by the recent success of word embeddings in handling document lexical gaps [121], we learn word embeddings from a large corpus of Stack Overflow posts to encode the question titles and the query question and measure their relevance.

In the second stage, we collect all the answers of the relevant questions retrieved in the first stage. We extract answer paragraphs from the collected answers. We develop a multi-criteria ranking method to select a small subset of relevant and salient answer paragraphs for summarization, based on query-related, user-oriented, and paragraph content features. In the third stage, given a set of answer paragraphs to be summarized, the generated answer summary needs to cover as much diverse information as possible. To generate a diverse summary, we measure novelty and diversity between the selected answer paragraphs by Maximal Marginal Relevance (MMR) [14].

We build a question repository of 228,817 Java questions and their corresponding answers from Stack Overflow. We randomly select another 100 Java questions as query questions and use our approach to generate an answer summary for each query question. Our user studies confirm the relevance, usefulness and diversity of the generated summary, and the effectiveness of our approach's relevant question retrieval and answer paragraphs selection components. Our error analysis identifies four main challenges in generating high-quality answer summary: vague queries, lexical gap between query and question description, missing long code-snippet answers, and erroneous answer paragraph splitting, which reveal the future enhancements of our automated answer generation technique.

The main contributions of this paper are the following:

- We conduct a formative study to assess the necessity of automated question answering techniques to provide answer summary to developers' technical questions.

- We formulate the problem of automated question answering as a query-focused multi-answer-posts summarization task for an input technical question.

- We propose a three-stage framework to solve the task, i.e., 1) relevant question retrieval, 2) answer paragraphs selection, 3) answer summary generation.

- We conduct user studies to evaluate the effectiveness of our approach and its components, and identify several areas for future improvements.

5.2 Formative Study of Answer Summary

Table 5.1: The top 5 ranked Stack Overflow questions by Google Search Engine for the query "differences HashMap HashTable Java"

No.	Question Id	Title	#Answers	#Words
1	40471	Differences between HashMap and Hashtable?	37	4135
2	8875680	Difference between Hashtable and Collections.synchronizedMap(HashMap)	5	847
3	36313817	What are the differences between hashtable and hashmap? (Not specific to Java)	3	747
4	32274953	Difference between HashMap and HashTable purely in Data Structures	3	565
5	30110252	What are the differences between Hashmap vs Hashtable in theory?	3	477

We contacted 120 developers by emails in two IT companies. We received 72 replies which help us understand the developers' difficulties in the current information retrieval (IR) practice and assess the necessity of automated question answering techniques. Some comments we received are listed as follows:

- *"Google will **return a number of "relevant" links for a query**, and I have to click into these links, and read a number of paragraphs ... It is really time-consuming ... Some links*

Table 5.2: Desirable Answer Summary with Relevant, Salient and Diverse Information

No.	Answer Id	Content	Aspect
1	764418	HashMap is non synchronized whereas Hashtable is synchronized.	Synchronization/Thread Safety
2	25526024	Hashmap can store one key as null. Hashtable can't store null.	Null Keys/ Null Values
3	22491742	Second important difference between Hashtable and HashMap is performance, since HashMap is not synchronized it perform better than Hashtable.	Performance
4	16018266	HashTable is a legacy class in the jdk that shouldn't be used anymore.	Evolution History
5	20519518	Maps allows you to iterate and retrieve keys, values, and both key-value pairs as well, Where HashTable don't have all this capability.	Iteration

Table 5.3: Examples of Irrelevant and Low-quality Answer Paragraphs

No.	Type	Answer Id	Example
1	Irrelevant	42003464	The explaination between hashmap and hashtable is quite correct as it also fits to the header of a string hash map implementated in strmap. c where thestringmap is a hashtable for strings satisfying the properties of a key,value-structure. Here it says : /...code.../
3	Low-quality	36325577	The interviewer may have been looking for the insight that. A hash table is a lower-level concept that doesn't imply or necessarily support any distinction or separation of keys and values...even if in some implementations they're always stored side by side in memory, e.g. members of the same structure / std::pair<>...

Table 5.4: Redundant Answer Paragraphs

Aspect	Set of Redundant Answers' Id	Example
Synchronization/ Thread Safety	40512, 40878, 764418, 39785829, 30108941, 42622789, 10372667, 20519518,	[764418] HashMap is non synchronized whereas Hashtable is synchronized.
	17815037, 28426488, 27293997, 8876192, 34618895, 41454, 25348157, 22084149,	[40512] Hashtable is synchronized, whereas HashMap isn't. That makes Hashtable slower than Hashmap.
	8876289, 8876205, 42315504, 22491742, 14452144, 25526024, 11883473	[39785829] HashTable is internally synchronized. Whereas HashMap is not internally synchronized.
Null Keys/ Null Values	40878, 7644118, 40548, 10372667,	[25526024] Hashmap can store one key as null. Hashtable can't store null.
	39785829, 14452144, 17815037, 28426488, 20519518, 25526024, 34618895, 42622789,	[40878] Hashtable does not allow null keys or values. HashMap allow sone null key and any number of null values.
	25348157, 30108941	[10372667] HashTable can only contain non-null object as a key or as a value. HashMap can contain one null key and null values.
Performance	13797704, 40848, 30108941, 39785829,	[40848] For threaded apps, you can often get away with ConcurrentHashMap- depends on your performance requirements.
	22491742, 34618895, 28426488, 24583680	[22491742] Second important difference between Hashtable and HashMap is performance, since HashMap is not synchronized it perform better than Hashtable.
		[34618895] As HashMap is not synchronized it is faster as compared to Hashtable.
Evolution History	1041798, 22629569, 40522, 10372667,	[16018266] HashTable is a legacy class in the jdk that shouldn't be used anymore.
	30108941, 39785829, 16018266, 14627155,	[39785829] HashMap extends AbstractMap class where as HashTable extends Dictionary class which is the legacy class in java.
	34618895, 42315504	[42315504] Second difference is HashMap extends Map Interface and whether HashSet Dictionary interface.
Iteration	40878, 7644118, 8832544, 40483, 7344090, 41454, 10372667, 30108941,	[30108941] Iterating the values: Hashmap object values are iterated by using iterator. HashTable is the only class other than vector which uses enumerator to iterate the values of HashTable object.
	39785829, 20519518, 14452144, 34618895,	[20519518] Maps allows you to iterate and retrieve keys, values, and both key-value pairs as well, Where HashTable don't have all this capability.
	42622789	[42622789] Iterator in HashMap is fail-fast. Enumerator in Hashtable is not fail-fast.

even contain viruses. A tool which generates potential answers can ***save my time wasted on reading a lot of irrelevant content****. If the generated answer accurately solves my question, it is good. But I think it would be difficult. Anyway, I believe it is no harm to use an answering tool, at least* ***I can get some hints to solve my problem****."*

- *"Sometimes I* ***cannot accurately describe my questions****, which made it hard to find the answer. I have to browse a number of posts online to learn how to refine my query, and search again. Thus, I expect that the answer generation tool can* ***help me understand the information space better, so I can refine my query faster without browsing those noisy posts."***

- *"I notice even* ***the best answers*** *in Stack Overflow often answer the questions* ***only in one aspect****. Sometimes I need to know a diversity of aspects to understand the problem better, but they cannot be found in a single best answer. Thus, I expect the tool should* ***provide a diversity of potential answers****, even if some answers are not accurate. "*

- *"... Some questions received* ***too many long answers****, and many of these answers have* ***redundant content****. I expect the answer generation tool should* ***return succinct answers which covers many aspects of potential solutions. So I could have a high-level understanding of the question I posted.*** *"*

- *"****Even if the accuracy of the tool is only 10%, I will still use it.*** *In the worst case, I will use your tool first, and then search on Google again to find the solutions."*

We use a real-world example to illustrate the difficulties mentioned in the developers' replies and the desirable properties of automated answer generation tool. Assume a developer was interested in the differences between *HashMap* and *HashTable* in Java. He used Google to search for Stack Overflow posts and Table 5.1 lists the top 5 ranked Stack Overflow questions returned by Google.

The question titles are very relevant to the developer's information need and he should be able to find the needed information in the answers to these questions. However, information overload can be detrimental to the developer. There are 51 answers which have 6,771 words in total. Reading all these answers may take 30 minutes (based on the average readers' reading speed of 200 words per minute [52]). Even just reading the best answers (i.e., the accepted answers) or top-voted answers may still take some time.

It would be desirable to have a answer summary extracted from the answers to the top 5 ranked questions, as the one shown in Table 5.2. This answer summary helps the developer quickly capture the key points of the answers relevant to his technical question. These points reveal the salient and diverse differences between *HashMap* and *HashTable*. They may help the developer decides which API is more appropriate for his task, or provide *information scents* for guiding the developer performing further search or learning [111].

However, manually generating this answer summary is not an easy task. First, there is much low-quality and irrelevant information [113]. Table 5.3 shows two examples (eight answers in total). The first example discusses *HashMap* and *HashTable* in C. The second example discusses how to answer *HashMap* and *HashTable* related interview questions. These answers are valid in a particular questions context, but have nothing to do with the developer's technical question.

Another issue is information redundancy. As shown in Table 5.4, the same aspect may be mentioned in many answer posts. The information redundancy and diversity creates a dilemma for the developer. If he reads every post, he is likely to come across the same information again and again, which is a waste of time. If he skips some posts, he risks missing some important aspects he has not seen. Reading only the best answers can address the information overload issue, but not the information redundancy and diversity. For example, the best answer[1] to the 1st ranked question in Table 5.1 discusses only three aspects (*Synchronization or Thread Safety*, *Null Keys and Null Values*, and *Iteration*) listed in Table 5.2. To tackle the above information relevance, redundancy and diversity issues for finding answers to developers' technical questions, we need an effective technique to generate an answer summary with relevant, salient and diverse information from unstructured text of answer posts.

5.3 Proposed Approach

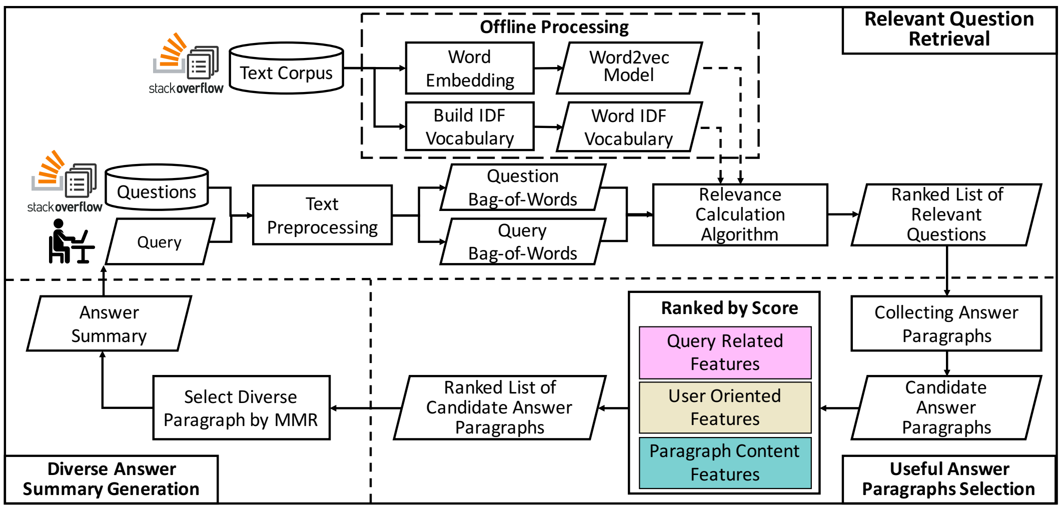
As shown in Figure 5.1, our approach (called AnswerBot) takes as input a software-engineering-related technical question as a query from the user, and produces as output an answer summary for the question. Next, we describe the three components of our approach, i.e., relevant question retrieval, useful answer paragraphs selection, and diverse answer summary generation.

5.3.1 Relevant Question Retrieval

The relevant question retrieval component takes a technical question as an input query q and ranks all questions Q in a large question repository (e.g., questions from Q&A sites like Stack Overflow). The questions that are ranked at the top are more likely to have answers that can answer the input technical question. We combine word embedding technique and traditional IDF metric to measure the relevance between the input query and the questions in the repository. Word embedding has been shown to be robust in measuring text relevance in the presence of lexical gap [119]. IDF metric helps to measure the importance of a word in the corpus.

To train the word embedding model and compute the word IDF metrics, we build a domain-specific text corpus using the question title and body of Stack Overflow questions. Each question is considered as a document in the corpus.

[1]`http://stackoverflow.com/a/40878`



Offline Processing
Relevant Question Retrieval
stackoverflow
Text Corpus
Word Embedding
Word2vec Model
Build IDF Vocabulary
Word IDF Vocabulary
stackoverflow
Questions
Query
Text Preprocessing
Question Bag-of-Words
Query Bag-of-Words
Relevance Calculation Algorithm
Ranked List of Relevant Questions
Answer Summary
Ranked by Score
Query Related Features
User Oriented Features
Paragraph Content Features
Collecting Answer Paragraphs
Select Diverse Paragraph by MMR
Ranked List of Candidate Answer Paragraphs
Candidate Answer Paragraphs
Diverse Answer Summary Generation
Useful Answer Paragraphs Selection

Figure 5.1: The Framework of AnswerBot

As the text is from the website, we follow the text cleaning steps commonly used for preprocessing web content [19]. We preserve textual content but remove HTML tags. We remove long code snippets enclosed in HTML tag ⟨*pre*⟩, but not short code fragments in ⟨*code*⟩ in natural language paragraphs. We use software-specific tokenizer [124] to tokenize the sentence. This tokenizer can preserve the integrity of code-like tokens and the sentence structure. We use Gensim (a Python implementation of the word2vec model [84]) to learn the word embedding model on this domain-specific text corpus. To compute the word IDF metrics, we build a vocabulary from the text corpus by removing stop words based on the list of stop words for English text[2] and using a popular stemming tool [11] to reduce each word to its root form. We then compute the IDF metric of each word in the vocabulary over the text corpus.

Given a query q and the title of a question Q in the repository, our relevance calculation algorithm computes their relevance based on an IDF-weighted word embedding similarity between the query and the question title. We use question title in relevance calculation because query and question title are "parallel text" [20]. The query and the question title are transformed into a bag of words, respectively, following the same text preprocessing steps described above. Let W_q be the bag of words for the query q and W_Q be the bag of words for the

[2]`http://snowball.tartarus.org/algorithms/english/stop.txt`

title of the question Q. An asymmetric relevance $rel(W_q \to W_Q)$ is computed as:

$$rel(W_q \to W_Q) = \frac{\sum_{w_q \in W_q} rel(w_q, W_Q) * idf(w_q)}{\sum_{w_q \in W_q} idf(w_q)} \tag{5.1}$$

where $idf(w_q)$ is the IDF metric of the word w_q, $rel(w_q, W_Q)$ is $\max_{w_Q \in W_Q} \; rel(w_q, w_Q)$, and $rel(w_q, w_Q)$ is the cosine similarity of the two word embeddings w_q and w_Q. Intuitively, the word embedding similarity of a more important word in the query and the words in the question title carries more weight towards the relevance measurement between the query and the question title. An asymmetric relevance $rel(W_Q \to W_q)$ is computed in the same way. Then, the symmetric relevance between the query q and the question Q is the average of the two asymmetric relevance between W_q and W_Q, i.e., $rel(q, Q) = (rel(W_q \to W_Q) + rel(W_Q \to W_q))/2$.

Based on the symmetric relevance between the query and each question in the repository, the questions in the repository are ranked and the top N ranked questions are returned as the relevant questions for the query.

5.3.2 Useful Answer Paragraphs Selection

Given a ranked list of relevant questions, all the answer paragraphs (split by HTML tag $\langle p \rangle$) in the answers to these questions are collected. We decide to use the granularity of answer paragraphs because they are the logical text units that answerers create when writing the posts. To select relevant and salient answer paragraphs for summarization, our approach ranks answer paragraphs based on three kinds of features, i.e., query related features, paragraph content features and user oriented features.

Query related features measure the relevance between an answer paragraph and the query.

- *Relevance to query.* As the query and the answer paragraphs usually have lexical gap between the information they convey, it is hard to directly measure their relevance. In this work, we set the relevance between a query and an answer paragraph as the relevance between the query and the question from which the answer paragraph is extracted.[3] The underlying intuition is that the more relevant the question is to the query, the more likely the answers to the question contain relevant answer paragraphs.

- *Entity overlap.* If an answer paragraph contains software-specific entities mentioned in the query, it is very likely that the paragraph is relevant to the query. For example, all desirable answer paragraphs in Table 5.2 contain *HashMap* and/or *HashTable* mentioned in the query. Software-specific entities can be programming languages, libraries/frameworks, APIs, data format, and domain-specific concepts [124]. In this work, we consider tags and tag synonyms on Stack Overflow as entities. We identify entity mentions in a query or answer paragraph by matching words in the query or answer paragraph with tag names and tag synonyms. Let E_q and E_{ap} be the set of entities mentioned in the query and the answer paragraph, respectively. The entity overlap between the query and the answer paragraph is computed as $|E_q \bigcap E_{ap}| \, / \, |E_q|$ $(|E_q| \neq 0)$. If the query does not mention any entities $(|E_q| = 0)$, we set entity overlap at 0.

Paragraph content features measure the salience of an answer paragraph's content.

- *Information entropy.* Salient answer paragraphs would contain high-entropy words. A word with higher IDF metric indicates that the word is less common in the corpus (i.e., higher entropy). Thus, we sum the IDF metrics of words (after removing stop words and stemming) in a paragraph to represent the paragraph's entropy. Using this feature, many paragraphs with low information entropy, e.g., "*I cannot agree more.*", will be filtered out.

[3]The relevance between the query and question is calculated during the relevant question retrieval process – see Section 5.3.1.

Table 5.5: Semantic Patterns For Salient Information

No.	Pattern	No.	Pattern
1	Please check XX	7	In short, XX
2	Pls check XX	8	The most important is XX
3	You should XX	9	I'd recommend XX
4	You can try XX	10	In summary, XX
5	You could try XX	11	Keep in mind that XX
6	Check out XX	12	I suggest that XX

- *Semantic patterns.* We observe that there are certain sentence patterns that often indicate recommendation or summarization of salient information in Stack Overflow discussions (see Table 5.5 for examples). For example, a question on Stack Overflow asks "*Array or List in Java. Which is faster?*". The best answer to this question is "***I suggest that** you use a profiler to test which is faster.*". In this work, we summarize 12 sentence patterns based on our empirical observations of 300 randomly selected best answers on Stack Overflow. If an answer paragraph contains at least one of the sentence patterns, we set the paragraph's pattern value at 1, otherwise 0.

- *Format patterns.* We observe that HTML tags are often used to emphasize salient information in the discussions. For example, ⟨*strong*⟩ highlights some text by bold font and ⟨*strike*⟩ points out some incorrect information. If an answer paragraph contains such HTML tags, we set its format pattern score at 1, otherwise 0.

User oriented features select summary and high-quality answer paragraphs based on user behavior patterns.

- *Paragraph position.* We observe that when answerers write answer posts, they usually start with some summary information and then go into details. For example, a question asks "*How do I compare strings in Java?*", The first three paragraphs of the best answer of this question present "*== for reference equality*", "*.equals() for value equality*", and "*Objects.equals() checks for nulls*". Therefore, we set a paragraph's summary value to be inversely proportional to the paragraph's position in the post for the first m paragraphs, i.e., $summary = 1/pos$ ($1 \leq pos \leq m$) ($m = 3$ in our current implementation). The summary values of the subsequent (beyond the m^{th}) paragraphs are set at 0.

- *Vote on answer.* Answers with higher vote indicate that the community believes that they contain high-quality information to answer the corresponding question. In this work, we set an answer paragraph's vote as the vote on the answer post from which the paragraph is extracted.

Based on the above seven features, an overall score is computed for each answer paragraph by multiplying the normalized value of each feature. To avoid the feature scores being 0, all the feature scores are normalized to (0,1] by adding a smooth factor 0.0001 [22]. Answer paragraphs are ranked by their overall scores and the top M ranked answer paragraphs are selected as candidate answer paragraphs for summarization.

5.3.3 Diverse Answer Summary Generation

As shown in Table 5.4, there are often many redundant answer paragraphs from the answers to relevant questions. The generated answer summary should avoid such redundant information. Given a list of candidate answer paragraphs, maximal marginal relevance (MMR) algorithm is used to select a subset of answer paragraphs in order to maximize novelty and diversity between the selected answer paragraphs [14]. MMR first builds a similarity matrix between each pair of candidate answer paragraphs. The similarity is computed as the symmetric relevance between the two answer paragraphs as described in Section 5.3.1. It then iteratively selects K candidate answer paragraphs with maximal marginal relevance. The selected answer paragraphs form an answer summary to the user's technical question.

5.4 Experiments & Results

We conduct three user studies to answer the following three research questions, respectively:

RQ1 How effective is our approach in generating answer summaries with relevance, useful and diverse information for developers' technical questions?

RQ2 How effective is our approach's relevant question retrieval component?

RQ3 How effective is our approach's answer paragraph selection component?

In this section, we first describe our repository of questions and answers and tool implementation. We then describe our experimental query questions, and how we select participants and allocate tasks in our user studies. Finally, we elaborate the motivation, approach and results for the three research questions.

5.4.1 Question Repository and Tool Implementation

We collect 228,817 Java questions (i.e., questions tagged with Java) and their corresponding answers from Stack Overflow Data Dump of March 2016. These questions have at least one answer. To ensure the quality of question repository, we require that at least one of the answers of the selected questions is the accepted answer or has vote > 0. When collecting questions, we avoid duplicate questions of the already-selected questions, because duplicate questions discuss the same question in different ways and can be answered by the same answer. We use these Java questions and their answers as a repository for answering Java-related technical questions. We build a text corpus using the title and body of these Java questions to train the word embedding model and build the word IDF vocabulary. Considering the conciseness of the generated answer summary and the fact that searchers tend to browse only the top ranked search results [1], our current implementation returns top 5 relevant questions for a query and selects top 10 candidate answer paragraphs for summarization. The generated answer summary contains 5 answer paragraphs.

5.4.2 Experimental Queries

We randomly select another 100 questions[4] and use the titles of these questions as query questions. We ensure that our question repository does not contain these 100 query questions and their duplicate questions. The randomly selected 100 query questions cover a diversity of aspects of Java programming. For example, some of them are related to language features, such as multi-threading (e.g., *How does volatile actually work?*) and I/O (e.g., *Can BufferedReader read bytes?*), while others are related to many third-party libraries, such as TEST-Assertions (e.g., *Testing API which returns multiple values with JUnit*) and REST (e.g., *Is there an equivalent to ASP.NET WEB API in JAVA world?*). Some of the query questions are easy to answer (e.g., *How to convert String into DateFormat in java?*), while others are difficult (e.g., *How does volatile actually work?*). The diversity of these 100 questions can improve the generality of our study, and reduce the bias that our approach might be only effective for a specific type of questions. We index these 100 questions as Q1 to Q100.

5.4.3 Participant Selection and Task Allocation

We recruited participants through our school's mailing lists and select 2 postdoctoral fellows (P1 and P2) and 6 PhD students (D1 to D6) to join our user study. All the selected participants have industrial experience on Java development, and they have used Java to develop commercial projects in their work before they went to graduate school. The years of their working experience on Java are vary from 2 to 8 years, with an average 4.6 years. The diversity of these participants' working experience on Java can improve the generality

[4]See our replication package at `http://bit.ly/2qBEUhi`

Table 5.6: Task Allocation to Participants

RQs	Group 1 (P1, D1, D2, D3)	Group 2 (P2, D4, D5, D6)
RQ1	Q1-Q50	Q51-Q100
RQ2	Q51-Q100	Q1-Q50
RQ3	Q51-Q100	Q1-Q50

of our results. In practice, our tool aims to help all levels of developers, from novice to senior developers. During our user study, no participants report being unable to understand the query questions and answers.

We divided the eight participants into two groups, i.e., P1, D1, D2 and D3 in Group1 and P2, D4, D5 and D6 in Group2. Furthermore, we divided the 100 questions into two tasks, i.e., Q1-Q50 in Task1 and Q51-Q100 in Task2. Table 5.6 present the task allocation to the two participant groups. For RQ1, Group1 worked on Task1 questions, while Group2 worked on Task2 questions. For RQ2 and RQ3, Group1 worked on Task2 questions, while Group2 worked on Task1 questions. All four participants in these two groups were required to review the answers of the assigned 50 questions independently. With this task allocation, we have all 100 query questions evaluated for the three research questions. As RQ1 evaluates the overall performance of our approach, and RQ2 and RQ3 evaluates its components, using the same set of query questions to evaluate RQ1 and RQ2/RQ3 may bias the participants' results. However, since RQ2 and RQ3 evaluates the two components independently and the two components deal with completely different input/output, using the same questions would have little impact on the participants' results. We asked the participants to complete the study in three 2-hour sessions; the first session evaluates RQ1, while the second and third evaluate RQ2 and RQ3 respectively.

5.4.4 Research Questions

RQ1: Effectiveness of the overall approach and the relevance, usefulness and diversity of answer summary

Motivation. In the current IR practice, developers retrieve relevant questions by entering their technical questions to a search engine. Then they have to manually browse the answers of the returned relevant questions to find the needed information. In contrast, our approach can automatically generate an answer summary of the key points in the answers of the returned relevant questions. We would like to investigate the overall performance of our approach in terms of the relevance, usefulness and diversity of the generated summary, compared with manual information seeking.

Approach. We compare our approach against two baselines built based on Google search engine and Stack Overflow search engine, respectively. We add "site:stackoverflow.com" to the query of Google search engine so that it searches only posts on Stack Overflow. For a query question, we use the first ranked Stack Overflow question returned by a search engine as the most relevant question. We assume that a developer would read the best answer (i.e., the accepted answer) or the answer with the highest vote if there is no accepted answer of the relevant question. We refer to the collected best or highest-vote answer of the two baseline approaches as the *Baseline_Google* answer summary and the *Baseline_SO* answer summary, respectively.

For each query question, we provide the participants the answer summary generated by *Baseline_Google*, *Baseline_SO* and our approach, respectively. The participants do not know which answer summary is generated by which approach. They are asked to score the three answer summaries from three aspects, i.e., *relevance*, *usefulness* and *diversity. Relevance* refers to how relevant the generated summary is to the query. *Usefulness* refers to how useful the generated summary is for guiding the developer's further search or learning. *Diversity* refers to whether the generated summary involves diverse aspects of information. The score is a 5 point likert scale, with 1 being "Highly Irrelevant/Useless/Identical" and 5 being "Highly Relevant/Useful/Diverse".

Results. Table 5.7 shows the mean of relevance, usefulness and diversity scores of our approach and the two baselines. The result shows that our ap-

Table 5.7: Mean of Relevance, Usefulness and Diversity of Our Approach and the Baseline Approaches (RQ1)

	Relevance	Usefulness	Diversity
Our Approach	**3.450**	**3.720**	**3.830**
Baseline_Google	3.440	3.480*	2.930***
Baseline_SO	2.576***	2.712***	2.305***

***p<0.001, **p<0.01, *p<0.05

proach achieves the best performance in all three aspects, while the *Baseline_SO* achieves the worst performance. Our approach and *Baseline_Google* have comparable relevance score, but our approach has higher score in usefulness and diversity, especially diversity. The average number of paragraphs in *Baseline_Google* and *Baseline_SO* answer summaries are 4.18 and 4.09, respectively.

We use Wilcoxon signed-rank test [109] to evaluate whether the differences between our approach and the baseline approaches are statistically significant. The improvement of our approach over the *Baseline_SO* is statistically significant on all three aspects at the confidence level of 99.9%. The improvement of our approach over the *Baseline_Google* on usefulness and diversity is statistically significant at the confidence level of 95%. We use the best answer of the most relevant question returned by Google as the *Baseline_Google*'s answer for the query. Considering the Google's capability, it is not surprising the difference in relevance is not statistically significant. However, our approach achieves statistically significant better performance on usefulness and diversity (especially diversity). This indicates that the best or highest-vote answers may not cover as diverse information as the developers need. Therefore, it is worth reading more answer posts to summarize more complete information. Our approach automates this summarization process for a diversity of answers.

RQ2: The Effectiveness of relevant question retrieval

Motivation. An answer summary is generated from the answers of some questions relevant to a query question. If the retrieved questions are not relevant to the query question, it is unlikely to generate a high-quality answer summary for the query question. Our question retrieval approach combines word embeddings with traditional IDF metrics. We would like to investigate the effectiveness of our method, compared with the traditional TF-IDF based methods and other word- or document-embedding based methods.

Approach. We build three baseline approaches: TF-IDF based IR [39], word-embedding based document retrieval [121], and document-to-vector (Dov2Vec) based document retrieval [57]. TF-IDF is a traditional IR metric that is often used to rank a document's relevance to a user query in software engineering tasks, such as question retrieval [13] and code search [88]. Yang et al. [121] average word embeddings of words in a document to obtain a document vector which can be used to measure document relevance. Dov2Vec learns document embeddings together with the underlying word embeddings using a neural network and is also applied to measure document relevance [57].

For each query question, we collect the top-10 ranked questions retrieved by our question retrieval approach or one of the baseline approaches. We ask the participants to identify the relevant questions in the top-10 ranked questions. The participants do not know which approach generates which list of top-10 ranked questions. We use the following metrics in comparison of different approaches.

Top-K Accuracy: Given a query question q, if at least one of the top-k ranked questions is relevant, we consider the retrieval to be successful, and set the value $Success(q, top-k)$ to 1. Otherwise, we consider the retrieval to be unsuccessful, and set the value $Success(q, top-k)$ to 0. Given a set of queries, denoted as qs, its top-k accuracy Top@k is computed as: $Top@k(qs) = \sum_{q \in qs} Success(q, top\text{-}k)/|qs|$. The higher the top-k accuracy score is, the better a relevant question retrieval approach ranks the first relevant question. In this paper, we set k = 1, 5 and 10.

Mean Reciprocal Rank: Given a query question, its reciprocal rank is the multiplicative inverse of the rank of the first relevant question in a ranked list of

Table 5.8: Top@k Accuracy and MRR of Our Approach and the Baseline Approaches in Relevant Question Retrieval (RQ2)

	Top@1	Top@5	Top@10	MRR
Our Approach	**0.460**	**0.610**	**0.660**	**0.520**
Doc2Vec	0.180***	0.580	0.650	0.335***
Word Embeddings	0.340*	0.520	0.550*	0.408*
TF-IDF	0.320*	0.490*	0.530**	0.388*

***p<0.001, **p<0.01, *p<0.05

questions. Mean Reciprocal Rank (MRR) is the average of the reciprocal ranks of all queries in a set of queries: $MRR(qs) = \frac{1}{|qs|}\sum_{q \in qs} \frac{1}{Rank(q)}$. We denote qs as the set of queries. *Rank(q)* refers to the position of the first relevant question in the ranked list of questions returned by a relevant question retrieval approach for a query. The higher the MRR is, the higher the first relevant questions is ranked for a set of queries.

Results. Table 5.8 shows the Top@k and MRR metrics of our approach and the three baseline approaches for retrieving relevant questions for the 100 query questions. We notice that our approach achieves the best performance in all the evaluated metrics, especially for Top@1 and MRR. The *Doc2Vec* baseline achieves comparable performance as our approach on Top@5 and Top@10, but it has the worst performance on Top@1 and MRR. The *word-embedding* baseline performs slightly better than the *TF-IDF* baseline in Top@k and MRR. The differences between the three baseline approaches actually indicate that the traditional TF-IDF based method and the word or document-embedding based methods can complement each other. In fact, our approach that combines word embeddings and IDF metrics achieves the best performance than either TF-IDF or word/document embedding alone.

We apply Wilcoxon signed-rank test to test the statistical significance of the differences between our approach and the baseline approaches. The improvement of our approach over the *TF-IDF* baseline is statistically significant on all the metrics at the confidence level of 95%. Compared with the *word-embedding* and *Doc2Vec* baselines, our approach is statistically signifi-

Table 5.9: Top@k Accuracy and MRR of Feature Ablation for Answer Paragraph Selection (RQ3)

	Top@1	Top@5	Top@10	MRR
all features	**0.660**	**0.990**	**1.000**	**0.803**
w/o query related	0.610*	0.970	1.000	0.758**
w/o user oriented	0.500***	0.980	1.000	0.699***
w/o paragraph content	0.570*	1.000	1.000	0.744**

***p<0.001, **p<0.01, *p<0.05

cant better on Top@1 and MRR at the confidence level of 95%. Although the differences between our approach and the *word-embeddings* and *Doc2Vec* baselines on Top@5 and Top@10 are not statistically significant, the significantly better MRR indicates that our approach can rank relevant questions higher than the other two baselines.

RQ3: The Effectiveness of answer paragraph selection

Motivation. To select the most relevant and salient answer paragraphs for summarization, our approach considers three types of features of answer paragraphs, i.e., query-related, user-oriented and paragraph content features. We would like to investigate the impact of different types of features on the results of answer paragraphs selection.

Approach. We remove one type of features at a time from the full feature set and reserve the other two types. Thus, three baseline approaches are built, i.e., without (w/o) query-related features, w/o user-oriented features, and w/o paragraph content features. We let each approach output a ranked list of 10 answer paragraphs with the highest overall score. We take the union of the 40 answer paragraphs selected by our approach (with all features) and the three baselines. Participants are asked to judge whether the selected paragraphs contain salient information relevant to the query question. They do not know which answer paragraph is selected by which approach. We use Top@k accuracy and MRR in the top 10 ranked candidate answer paragraphs to evaluate the performance of answer paragraph selection with and without certain type of features.

Results. Table 5.9 presents Top@k and MRR metrics of using all features or adopting certain type of features for answer paragraph selection. Using all features achieves the best performance in all metrics compared with adopting certain type of features. This suggests that all types of features are useful for answer paragraph selection. Using query-related features achieves better performance than adopting the other two types of features, and using user-oriented features achieves the worst performance. This suggests that user-oriented features play the most important role for answer paragraph selection, paragraph content features take a second place, and query-related features are relatively less important.

Wilcoxon signed-rank test shows that the improvement of using all features over adopting certain type of features is statistically significant on Top@1 and MMR at the confidence level of 95%. Top@5 and Top@10 results in Table 5.9 show that there is almost always at least one relevant answer paragraph in the top 5 or 10 ranked list of candidate answer paragraphs. This demonstrates the general effectiveness of our answer paragraph selection features. Therefore, the differences between using all features and adopting certain type of features on Top@5 and Top@10 are not statistically significant.

5.5 Discussion

In this section, we qualitatively compare our question answering approach with community question answering practice. We then discuss cases where our approach is ineffective, followed by some threats to validity.

5.5.1 Comparison with Community Question Answering

In community question answering, developers post their questions on a Q&A site like Stack Overflow and wait for answers from the community of developers.

To understand the differences between the community-provided answers to a technical question and the answer summary that our approach generates from answers of relevant questions, we manually compare the best answers of the questions we use as queries and the answer summary that our approach generates for these questions.

Table 5.10 presents two examples. The query question in first example is "calculating time difference" in which the developer runs into some errors in using *getTime*() to calculate time difference. The best answer of this question suggests to use *long* to store *getTime*()'s return value, rather than casting it to *int*. For this query question, our generated answer summary consists of five paragraphs from the four answers of two relevant questions. Except for the fifth paragraph "You can try this" (due to the limitation of paragraph splitting, see Section 5.5.2), the other four paragraphs provide two alternative solutions (using *System.nanoTime*() or *currentTimeMillis*()) to calculate time difference. They also describe the reason and cautions of using the two APIs, such as *System.nanoTime*() is more precise, *System.nanoTime*() must not be used as a wall clock, *currentTimeMillis*() may not be a good method for time due to method overhead.

The query question in the second example is about hostname mismatch problem in *HTTPClient*. The best answer provides a solution and explains the advantage of the provided solution. Our answer summary consists of five paragraphs from the five answers of four relevant questions. Except for the fifth paragraph from the fourth question (about a fact of Amazon AWS service), the other four paragraphs provide valuable information which can complement the best answer of the query question. The third and fourth paragraphs provide two different solutions for solving the problem, i.e., using *HttpClientBuilder.create*().*build*() or using a fixed version of *HTTPClient* instead of the buggy version. Although the first and second paragraphs do not provide direct solutions, they point out some important aspects related

Table 5.10: Comparison of the Best Community-Provided Answers and the Answer Summary Generated by Our Approach

#1 Query Question: Calculating time diffrence			
Best Answer of Query Question(Id: 18575058)	Answer Summary Generated by Our Approach	Relevant Questions	
		Id	Title

Try changing
int diff=(int)d2.getTime()-(int)d1.getTime();
to long diff=d2.getTime()-d1.getTime();
Explicit typecasting from long to int
will cause precision loss and may result in
a negative value on subtraction.
long a = 90000000000000L;
long b = 10000001000000L;
a>b (int)a-(int)b=>negative value
	[9707986] Try this
	long start_time = System.nanoTime();
	resp = GeoLocationService.getLocationByIp(ipAddress);
	long end_time = System.nanoTime();
	double difference = (end_time - start_time)/1e6;
		9707938	Calculating time difference in Milliseconds

	[9707972] No, it doesn't mean it's taking 0ms - it shows it's taking a smaller amount of time than you can measure with currentTimeMillis(). That may well be 10ms or 15ms. It's not a good method to call for timing; it's more appropriate for getting the current time.		
	[9707972] To measure how long something takes, consider using System.nanoTime instead. The important point here isn't that the precision is greater, but that the resolution will be greater... but only when used to measure the time between two calls. It must not be used as a "wall clock".		
	[9707982] Since Java 1.5, you can get a more precise time value with System.nanoTime(), which obviously returns nanoseconds instead.		
	[24907491] You can try this :	24907002	Calculating Time Difference in Java
#2 Query Question: Android HttpClient - hostname in certificate didn't match <example.com>!= <*.example.com>			
Best Answer of Query Question(Id: 3136980)	Answer Summary Generated by Our Approach	Relevant Questions	
		Id	Title
This is my (edited) solution: /...code.../ It has the advantage of not changing the default behavior unless there is a wildcard domain, and in that case it revalidates as though the 2 part domain (e.g., someUrl.com) were part of the certificate, otherwise the original exception is rethrown. That means truly invalid certs will still fail.	[15497467] Important - if you are allowing all hosts (that is, disabling host name verification), then it is certainly NOT safe. You shouldn't be doing this in production.	15497372	Java HTTP post using HTTPS Confusion - javax.net.ssl.SSLException: hostname in certificate didn't match
	[7257060] The certificate verification process will always verify the DNS name of the certificate presented by the server, with the hostname of the server in the URL used by the client.	7256955	Java SSLException: hostname in certificate didn't match
	[24526126] This HttpClientBuilder.create().build() will return org.apache.http.impl.client.InternalHttpClient. It can handle the this hostname in certificate didn't match issue.		
	[34494091] This problem is described in Apache HttpClient resolving domain to IP address and not matching certificate. It appears to be a bug in the version of HTTPClient you are using, where it compares the target IP instead of the target hostname with the subject certificate. Please use a fixed version of HTTPClient instead.	34493872	SSLException: hostname in certificate didn't match <50.19.233.255>!=<*.heroku.com>
	[12755039] Buckets whose name contains periods can now be correctly addressed again over HTTPS.	12755038	SSL problems with S3/AWS using the Java API: hostname in certificate didn't match

to the hostname mismatch problem, such as avoiding the action which allows all hosts without any verification, checking the DNS name of the certificate presented by the server.

As the above two examples show, community answers to a technical question usually focus on a specific aspect technical issue in the question. Our generated answer summary derived from answers of several relevant questions can well complement community answers to a technical question by providing more alternative solutions and/or broader information useful for further search and learning.

5.5.2 Error Analysis

Through the analysis of the cases in which our approach generates a low-quality even unrelated answer summary, we identify four main challenges in generating high-quality answer summary: vague queries, lexical gap between query and question description, answers involving long code snippet, and erroneous answer paragraph splitting.

In our study, we randomly select 100 Stack Overflow questions and use their titles as queries to search the question repository. However, question titles are short and some of them are vague, e.g., "*Why is this code working without volatile?*", "*Fast processing of data*". It is hard to understand such question titles without looking into the question bodies. Furthermore, the lexical gap between query question and titles of questions in the repository makes it difficult to measure their semantic relevance. Due to these two challenges, our relevant question retrieval component fails to retrieve at least one relevant question in the top 10 results for 34 of the 100 query questions. The low-quality question retrieval results directly result in the low quality of the generated answer summary. To improve relevant question retrieval, we will consider question bodies which contain richer information and adopt deep neural network [20] which are more robust for handling lexical gaps in text.

Our current approach keeps short code fragments (enclosed in HTML tag ⟨*code*⟩) in natural language paragraphs but removes long code snippets (enclosed in HTML tag ⟨*pre*⟩). However, for some query questions, such as "*How to send an Image from Web Service in Spring*", "*How to implement a db listener in Java*" and "*XML to Json using Json-lib*", relevant answers are code snippets, rather than natural language paragraphs. Thus, to generate high-quality answer summary for this type of questions, we must take into account long code snippets.

Our current approach splits the text in an answer post into answer paragraphs by HTML tag ⟨*p*⟩. This simple strategy may sometimes break the logic relationships between several consecutive physical paragraphs. For example, people often write some introductory paragraph like "*Try with the following:* ", "*From here you can go to*", and "*There are many solutions to this problem*", followed by a separate paragraph explaining the details. To generate more sensible answer summary, an introductory paragraph and the following detailed explanation should be treated as a whole, using more robust paragraph splitting method.

5.5.3 Threats to Validity

Threats to internal validity are related to experimental bias of participants in manual examination of relevant questions and answer summaries. First, the participants' lack of knowledge on Java may affect their judgements about question/answer's relevance and usefulness. This threat is limited by selecting only participants who have at least 2 years industrial experience on Java development. Still, there could be errors because an experienced Java developer is not necessarily familiar with all Java frameworks and APIs. On the other hand, the participants' degree of carefulness and effort in manual examination may affect the validity of judgements. We minimize this threat by choosing participants who express interests in our research, and giving the participants enough time to complete the evaluation tasks.

Threats to external validity are related to the generalizability of our research and experiments. Stack Overflow questions are related to many domains other than Java (e.g. Python, Eclipse, database), or combinations of multiple domains. Our approach is general, but considering the background knowledge of available participants for the user studies, we use only Java questions in this work. Furthermore, as user studies require significant human efforts, we only use 100 query questions in this study. In the future, we will use more queries, larger question repository, and questions of more domains to reduce these threats.

Threats to construct validity are related to the suitability of our evaluation metrics. Relevance [81], usefulness [26] and diversity [69] are widely used to evaluate summarization tasks in software engineering, Both relevant question retrieval and answer paragraphs selection are ranking problems. Thus, we use Top@k accuracy (k=1, 5, 10) and MRR. These two metrics are the most widely used metrics for evaluating IR techniques [118, **?**, 117].

5.6 Conclusion and Future Work

Our formative study indicates that developers need some automated answer generation tools to extract a succinct and diverse summary of potential answers to their technical questions from the sheer amount of information in Q&A discussions. To meet this need, we propose a three-stage framework for automated generation of answer summary. Our user studies demonstrate the relevance, usefulness and diversity of our automated generated answer summaries.

In the future, we will investigate more robust relevant question retrieval algorithms, and explore more features for answer paragraph selection. We will develop our approach into practical tools (e.g., browser plugins) to provide developers with direct answers and extended suggestions to help them find the needed information more efficiently on the Internet. Moreover, considering our approach is three-stage (relevant post retrieval, useful information identification and answer summarization), we will compare the performance of each stage with the (NLP) techniques which aim the same/similar goal. Particularly, considering the similarity between the query and every single question in the repository need to be measured and the size of the repository could be big, we also plan to improve its efficiency from two directions. First, as the similarity measurement is independent across the repository, the computation could be implemented under the framework of parallel computing, such as MapReduce [23]. Second, intuitively, if two questions are semantically equivalent (i.e., duplicate questions), then the query only needs to compute the similarity with one of them. Thus, following this spirit, it is also worthy to investigate the feasibility of pre-computing the similarity between questions in the repository and then leverage the information to reduce the online computation cost. Moreover, besides MMR [14], we plan to investigate more automated summarization algorithms (such as LexRank [27]) in the stage of answer summary generation.

Chapter 6

Conclusion, Impact, and Future Work

6.1 Conclusion

As SQA data grows significantly in both size and complexity, leveraging it to support various software activities becomes challenging, time-consuming, and very costly. This dissertation focuses on analyzing SQA sites to save developers' time and effort in several ways. Specifically, we propose solutions for three tasks: linkable knowledge prediction, knowledge representation, and answer summarization. The contributions of this dissertation are as follows:

1. We introduce a low-complexity but more effective machine learning model, namely SoftSVM, to predict relatedness between knowledge units in Stack Overflow. SoftSVM takes four feature vectors from different perspectives as its input and employs a linear kernel for the support vector machine.

2. We propose a deep learning framework Post2Vec to learn the distributed representation of posts in SQA sites. It not only considers the textual data from posts but also leverages the code snippets inside the posts. Once the model is trained, the learned representation can be used

for many software engineering tasks based on the SQA data.

3. We design an end-to-end approach (named AnswerBot) to automatically generate a summary for a given technical question. AnswerBot is capable to retrieve the relevant questions, and identify the useful paragraphs from the answers to the relevant questions, and produce a summary based on the useful paragraphs.

6.2 Impact

We summarize the impact of this dissertation to the whole area of the mining crowd-knowledge platforms from three aspects as follows.

- **Reproducibility study should be more advocated for the research of mining crowd-knowledge**. Typically, in the area of the mining crowd-knowledge platforms, researchers demonstrate their approaches on a single dataset. Differently, in the work of linkable knowledge units prediction, we conducted a reproducibility study on how well the traditional machine learning-based and deep learning-based approaches perform over different scales of datasets. The experimental result demonstrates that the performance of traditional machine learning-based and deep learning-based approaches are not persistent. More specifically, we find that although the traditional machine learning approach achieves better performance than deep learning-based approach on a small dataset, it achieves much worse efficiency when we increase the size of the dataset. Inspired by this finding, we advocate researchers should always carefully take the characteristic (e.g., complexity) of their datasets into consideration to propose an appropriate approach design. More importantly, we encourage more reproducibility studies in the whole area of the mining crowd-knowledge platforms to improve the generalizability of research findings. Inspired by our work, Guo et al. propose an approach for identifying self admitted technical debt and the experimental results show that their approach is not only effective but also with low complexity [36].

- **Crowd-knowledge platforms should not be treated as information silos**. Instead, knowledge is linkable and can be leveraged for many software tasks. In this dissertation, both works of knowledge linking and representation show that knowledge units in SQA are linkable and provide two different ways to measure their relatedness. In the work of knowledge linking, the relatedness is divided into four categories, i.e., duplicate, directly link, indirectly link, and isolated. While in the work of knowledge representation, knowledge units are mapped into a vector space and the relatedness between them can be measured by the distance of their corresponding vectors. More importantly, we show that by leveraging the relationship between knowledge units, the performance of many software tasks can be boosted, e.g., API recommendation.

- **Knowledge summarization for crowd-knowledge platforms is desired**. In the work of knowledge summarization, we demonstrate that developers are suffering from a large amount of redundant information and it barriers them to identify the relevant information. Moreover, based on the result of our survey, we find that a relevant, useful, and diverse summary for their queries are desired. Thus, we proposed our approach AnswerBot for automatic answer summary generation. Motivated by our work AnswerBot, Huang et al. proposed an approach which not only recommends API but also summarizes API description and code examples in Stack Overflow posts to help developers select the APIs that are most relevant to their tasks [45]. Gao et al. proposed an answer recommendation approach which follows the same question retrieval method of AnswerBot to search for similar questions [33]. Wang et al. migrated the idea of AnswerBot for automatic solution summarization for crash bugs [103].

6.3 Future Work

In this dissertation, we have proposed several machine learning-based models to interpret the SQA data. The experimental results demonstrate that they perform well in terms of both effectiveness and efficiency. As future work, we want to investigate the following directions.

- **Explainable program repair by leveraging SQA data**. Many automated program repair approaches have been proposed to generate patches for given buggy programs. However, digesting and verifying generated patches is non-trivial as it requires programmers to equip a deep understanding of the patches. Thus, an explainable program repair approach is desired to save programmers' time. To achieve the goal, SQA data becomes a precious resource to be considered as it naturally contains solutions for many certain types of bugs with clear explanation. For example, a question with buggy code in Stack Overflow is titled as *Prevent SQL injection attacks in a Java program*[1]. The corresponding accepted answer[2] contains a correct patch by using the class *PreparedStatement*. More importantly, it also explains how does *PreparedStatement* work and why it can be used to prevent injection attacks. With such informative explanation, programmers can easily judge the correctness of the patch. Motivated by the above observations, we plan to propose an explainable program repair approach by leveraging SQA data.

- **Domain-Specific Search Engine**. Inspired from AnswerBot, we find that general search engines (e.g., Google) fail in many instances to return the desired results, and they are not fulfilling all needs. Thus,

[1]https://stackoverflow.com/questions/9516625
[2]https://stackoverflow.com/a/9516672/5022378

developing domain-specific search engines is one solution to help address the limitations of general search engines for a software-engineering related search. Such a domain-specific search engine should understand software engineering terminologies and concepts, and be able to disambiguate query terms and website contents based on these terminologies and concepts. It should also automatically identify useful software information sites, and prioritize results from these sites while ignoring sites that are irrelevant to software development. Although AnswerBot has demonstrated it can achieve better performance than general search engines, we observe that AnswerBot can be further improved by capturing the semantic links among software concepts, e.g., both *JQuery* and *Servlet* are two related techniques which correspond to front-end and back-end of java web development. To develop a more advanced version of AnswerBot, we plan to do the following work:

– **Domain-Specific knowledge graph construction**. Software engineering has its own terminologies and concepts. To identify their relationships, the first step is to construct a domain-specific knowledge graph from various software information sites like Stack Overflow and GitHub. We plan to leverage techniques from natural language processing and ontology construction to build the knowledge graph.

– **Domain-Specific semantic search**. We plan to use the knowledge graph to refine the search queries proposed by developers, and the semantic links between different software engineering concepts to improve the accuracy of search results, e.g., by integrating semantic links into a language model and proposing a domain-specific semantic language model.

– **Automated answer generation**. Developers' goal in using search engines is to find desired answers to their questions. General search engines typically return a list of relevant posts, and developers have to manually check these posts to find the desired answers. By leveraging a knowledge graph, the domain-specific search engine can identify not only a list of more relevant posts but also the semantic relationship between these posts. We plan to apply and extend text summarization techniques to generate the answers from the posts.

- **Automatic summarization for developers' activities**. Inspired by the work of knowledge summarization, we realize that many software activities may also need a promising summary to improve the developers' productivity. Hence, we plan to do the following work to further expand this idea:

  – **User study on potential applicable scenarios for knowledge/information summarization**. To understand how summarization techniques can better serve developers and also develop practical summarization tools, we plan to conduct an open survey. More specifically, we want to recognize what activities need summarization techniques the most and the least. Moreover, it's also useful to know what the expected summary looks like in developers' minds.

  – **Multi-resource software knowledge summarization**. Inspired by the work of API summarization done by Gias et al., developers expect the API summary can be generated based on multiple resources, e.g., API documentation and SQA [101]. Intuitively, considering multiple resources provides a big potential to improve the quality of the generated summary. Thus, to achieve this, we plan to develop a more advanced approach. Note that the task is non-trivial as the data from different resources may carry various data structures and styles.

Bibliography

[1] Models of the Information Seeking Process. `http://searchuserinterfaces.com/book/sui_ch3_models_of_information_seeking.html`.

[2] Md Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K Roy, and Kevin A Schneider. Classifying stack overflow posts on api issues. In *2018 IEEE 25th international conference on software analysis, evolution and reengineering (SANER)*, pages 244–254. IEEE, 2018.

[3] Muhammad Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K Roy, and Kevin A Schneider. Mining duplicate questions of stack overflow. In *Mining Software Repositories (MSR), 2016 IEEE/ACM 13th Working Conference on*, pages 402–412. IEEE, 2016.

[4] Miltiadis Allamanis and Charles Sutton. Why, when, and what: analyzing stack overflow questions by topic, type, and code. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 53–56. IEEE, 2013.

[5] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages*, 3(POPL):40, 2019.

[6] Marios Anthimopoulos, Stergios Christodoulidis, Lukas Ebner, Andreas Christe, and Stavroula Mougiakakou. Lung pattern classification for

interstitial lung diseases using a deep convolutional neural network. *IEEE transactions on medical imaging*, 35(5):1207–1216, 2016.

[7] Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 244–253, 2018.

[8] Muhammad Asaduzzaman, Ahmed Shah Mashiyat, Chanchal K Roy, and Kevin A Schneider. Answering questions about unanswered questions of stack overflow. In *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*, pages 97–100. IEEE, 2013.

[9] C Glenn Begley and John PA Ioannidis. Reproducibility in science: improving the standard for basic and preclinical research. *Circulation research*, 116(1):116–126, 2015.

[10] Stefanie Beyer, Christian Macho, Massimiliano Di Penta, and Martin Pinzger. Automatically classifying posts into question categories on stack overflow. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, pages 211–21110. IEEE, 2018.

[11] Steven Bird. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pages 69–72. Association for Computational Linguistics, 2006.

[12] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118, 2010.

[13] Xin Cao, Gao Cong, Bin Cui, and Christian S Jensen. A generalized framework of exploring category information for question retrieval in

community question answer archives. In *Proceedings of the 19th international conference on World wide web*, pages 201–210. ACM, 2010.

[14] Jaime Carbonell and Jade Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336. ACM, 1998.

[15] Rich Caruana, Steve Lawrence, and C Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in neural information processing systems*, pages 402–408, 2001.

[16] S. Chakraborty, Y. Ding, M. Allamanis, and B. Ray. Codit: Code editing with tree-based neural models. *IEEE Transactions on Software Engineering*, pages 1–1, 2020.

[17] Delphine Charlet and Geraldine Damnati. Simbow at semeval-2017 task 3: Soft-cosine semantic similarity between questions for community question answering. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 315–319, 2017.

[18] Chunyang Chen and Zhenchang Xing. Towards correlating search on google and asking on stack overflow. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 83–92. IEEE, 2016.

[19] Chunyang Chen, Zhenchang Xing, and Ximing Wang. Unsupervised Software-Specific Morphological Forms Inference from Informal Discussions.

[20] Guibin Chen, Chunyang Chen, Zhenchang Xing, and Bowen Xu. Learning a dual-language vector space for domain-specific cross-lingual ques-

tion retrieval. In *Ieee/acm International Conference on Automated Software Engineering*, pages 744–755, 2016.

[21] Jieshan Chen, Chunyang Chen, Zhenchang Xing, Xiwei Xu, Liming Zhut, Guoqiang Li, and Jinshui Wang. Unblind your apps: Predicting natural-language labels for mobile gui components by deep learning. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 322–334. IEEE, 2020.

[22] Stanley F Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318. Association for Computational Linguistics, 1996.

[23] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[24] Daniel DeFreez, Aditya V Thakur, and Cindy Rubio-González. Path-based function embedding and its application to error-handling specification mining. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 423–433. ACM, 2018.

[25] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[26] Andrea Di Sorbo, Sebastiano Panichella, Carol V Alexandru, Junji Shimagaki, Corrado A Visaggio, Gerardo Canfora, and Harald C Gall. What would users change in my app? summarizing app reviews for recommending software changes. In *Proceedings of the 2016 24th ACM*

SIGSOFT International Symposium on Foundations of Software Engineering, pages 499–510. ACM, 2016.

[27] Günes Erkan and Dragomir R Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of artificial intelligence research*, 22:457–479, 2004.

[28] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. Stack overflow considered harmful? the impact of copy&paste on android application security. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 121–136. IEEE, 2017.

[29] Wei Fu and Tim Menzies. Easy over hard: A case study on deep learning. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 49–60. ACM, 2017.

[30] Qing Gao, Hansheng Zhang, Jie Wang, Yingfei Xiong, Lu Zhang, and Hong Mei. Fixing recurring crash bugs via analyzing q&a sites (t). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 307–318. IEEE, 2015.

[31] Sa Gao, Zhenchang Xing, Yukun Ma, Deheng Ye, and Shang-Wei Lin. Enhancing knowledge sharing in stack overflow via automatic external web resources linking. In *2017 22nd International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 90–99. IEEE, 2017.

[32] Zhipeng Gao, Xin Xia, John Grundy, David Lo, and Yuan-Fang Li. Generating question titles for stack overflow from mined code snippets. *ACM Trans. Softw. Eng. Methodol.*, 29(4), September 2020.

[33] Zhipeng Gao, Xin Xia, David Lo, and John Grundy. Technical q8a site answer recommendation via question boosting. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(1):1–34, 2020.

[34] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. Deep code search. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 933–944. IEEE, 2018.

[35] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. Deep api learning. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 631–642. ACM, 2016.

[36] Zhaoqiang Guo, Shiran Liu, Jinping Liu, Yanhui Li, Lin Chen, Hongmin Lu, Yuming Zhou, and Baowen Xu. Mat: A simple yet strong baseline for identifying self-admitted technical debt. *arXiv preprint arXiv:1910.13238*, 2019.

[37] Huong Ha and Hongyu Zhang. Deepperf: performance prediction for configurable software with deep sparse neural network. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1095–1106. IEEE, 2019.

[38] Martin T Hagan and Mohammad B Menhaj. Training feedforward networks with the marquardt algorithm. *IEEE transactions on Neural Networks*, 5(6):989–993, 1994.

[39] Sonia Haiduc, Gabriele Bavota, Andrian Marcus, Rocco Oliveto, Andrea De Lucia, and Tim Menzies. Automatic query reformulations for text retrieval in software engineering. In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 842–851. IEEE, 2013.

[40] Jun Han and Claudio Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International Workshop on Artificial Neural Networks*, pages 195–201. Springer, 1995.

[41] Thong Hoang, Hoa Khanh Dam, Yasutaka Kamei, David Lo, and Naoyasu Ubayashi. Deepjit: an end-to-end deep learning framework for just-in-time defect prediction. In *Proceedings of the 16th International Conference on Mining Software Repositories*, pages 34–45. IEEE Press, 2019.

[42] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[43] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification. In *Technical Report*. Department of Computer Science and Information Engineering, National Taiwan University, Taiwan., 2003.

[44] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. Deep code comment generation. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, pages 200–20010. IEEE, 2018.

[45] Qiao Huang, Xin Xia, Zhenchang Xing, David Lo, and Xinyu Wang. Api method recommendation without worrying about the task-api knowledge gap. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 293–304. IEEE, 2018.

[46] Xuan Huo and Ming Li. Enhancing the unified features to locate buggy files by exploiting the sequential nature of source code. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1909–1915, 2017.

[47] Md Johirul Islam, Rangeet Pan, Giang Nguyen, and Hridesh Rajan. Repairing deep neural networks: Fix patterns and challenges. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 1135–1146. IEEE, 2020.

[48] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2073–2083, 2016.

[49] Mohit Iyyer, Jordan L Boyd-Graber, Leonardo Max Batista Claudino, Richard Socher, and Hal Daumé III. A Neural Network for Factoid Question Answering over Paragraphs. In *EMNLP*, pages 633–644, 2014.

[50] Siyuan Jiang, Ameer Armaly, and Collin McMillan. Automatically generating commit messages from diffs using neural machine translation. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*, pages 135–146, 2017.

[51] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.

[52] Marcel Adam Just and Patricia A Carpenter. Speedreading. 1987.

[53] C Kadilar and HB Cingi. Ratio estimators in stratified random sampling. *Biometrical Journal: Journal of Mathematical Methods in Biosciences*, 45(2):218–225, 2003.

[54] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar,*

A meeting of SIGDAT, a Special Interest Group of the ACL, pages 1746–1751, 2014.

[55] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[56] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *AAAI*, volume 333, pages 2267–2273, 2015.

[57] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1188–1196, 2014.

[58] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[59] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[60] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670, 2014.

[61] Xiaoya Li, Yuxian Meng, Xiaofei Sun, Qinghong Han, Arianna Yuan, and Jiwei Li. Is word segmentation necessary for deep learning of chinese representations? In *Proceedings of the 57th Conference of the Association for Computational Linguistics*, pages 3242–3252, 2019.

[62] Jiakun Liu, Sebastian Baltes, Christoph Treude, David Lo, Yun Zhang, and Xin Xia. Characterizing search activities on stack overflow. In *Proceedings of the 2021 29th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2021.

[63] Jin Liu, Pingyi Zhou, Zijiang Yang, Xiao Liu, and John Grundy. Fasttagrec: fast tag recommendation for software information sites. *Automated Software Engineering*, 25(4):675–701, 2018.

[64] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Recurrent neural network for text classification with multi-task learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 2873–2879, 2016.

[65] Xuliang Liu and Hao Zhong. Mining stackoverflow for program repair. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 118–129. IEEE, 2018.

[66] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[67] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[68] Lena Mamykina, Bella Manoim, Manas Mittal, George Hripcsak, and Björn Hartmann. Design lessons from the fastest q&a site in the west. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2857–2866, New York, NY, USA, 2011. ACM.

[69] Senthil Mani, Rose Catherine, Vibha Singhal Sinha, and Avinava Dubey. Ausum: approach for unsupervised bug report summarization. In *Pro-*

ceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, page 11. ACM, 2012.

[70] Tim Menzies, Suvodeep Majumder, Nikhila Balaji, Katie Brey, and Wei Fu. 500+ times faster than deep learning:(a case study exploring faster methods for text mining stackoverflow). In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pages 554–563. IEEE, 2018.

[71] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[72] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[73] Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*, 2016.

[74] Ani Nenkova and Kathleen McKeown. A survey of text summarization techniques. In *Mining text data*, pages 43–76. Springer, 2012.

[75] Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N Nguyen, David Lo, and Chengnian Sun. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 70–79. ACM, 2012.

[76] Malte Ostendorff, Terry Ruas, Moritz Schubotz, Georg Rehm, and Bela Gipp. Pairwise multi-class document classification for semantic relations

between wikipedia articles. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries in 2020*, pages 127–136, 2020.

[77] Jayavardhan Peddamail, Zhen Wang, Ziyu Yao, and Huan Sun. A comprehensive study of staqc for deep code summarization. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Lond, UK*, 2018.

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

[79] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. Mining stackoverflow to turn the ide into a self-confident programming prompter. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 102–111. ACM, 2014.

[80] Luca Ponzanelli, Andrea Mocci, Alberto Bacchelli, Michele Lanza, and David Fullerton. Improving low quality stack overflow post detection. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 541–544. IEEE, 2014.

[81] Dragomir R Radev and Weiguo Fan. Automatic summarization of search engine hit lists. In *Proceedings of the ACL-2000 workshop on Recent advances in natural language processing and information retrieval: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 11*, pages 99–109. Association for Computational Linguistics, 2000.

[82] Mohammad Masudur Rahman and Chanchal Roy. Effective reformulation of query for code search using crowdsourced knowledge and extra-

large data analytics. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 473–484. IEEE, 2018.

[83] Mohammad Masudur Rahman, Chanchal K Roy, and David Lo. Rack: Automatic api recommendation using crowdsourced knowledge. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 349–359. IEEE, 2016.

[84] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[85] Chris Rorres and Howard Anton. *Elementary linear algebra: applications version*. Wiley, 1994.

[86] Christoffer Rosen and Emad Shihab. What are mobile developers asking about? a large scale study using stack overflow. *Empirical Software Engineering*, 21(3):1192–1223, 2016.

[87] Ruhi Sarikaya, Geoffrey E Hinton, and Anoop Deoras. Application of deep belief networks for natural language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(4):778–784, 2014.

[88] David Shepherd, Kostadin Damevski, Bartosz Ropski, and Thomas Fritz. Sando: an extensible local code search framework. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 15. ACM, 2012.

[89] Rodrigo FG Silva, Klérisson Paixão, and Marcelo de Almeida Maia. Duplicate question detection in stack overflow: A reproducibility study. In *2018 IEEE 25th international conference on software analysis, evolution and reengineering (SANER)*, pages 572–581. IEEE, 2018.

[90] Raphael Sirres, Tegawendé F Bissyandé, Dongsun Kim, David Lo, Jacques Klein, Kisub Kim, and Yves Le Traon. Augmenting and structuring user queries to support efficient free-form code search. *Empirical Software Engineering*, 23(5):2622–2654, 2018.

[91] Hongya Song, Zhaochun Ren, Shangsong Liang, Piji Li, Jun Ma, and Maarten de Rijke. Summarizing Answers in Non-Factoid Community Question-Answering.

[92] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[93] StackOverflow. How to ask a good question?, `http://stackoverflow.com/help/how-to-ask`, 2018.

[94] stackoverflow.com. Why are some questions marked as duplicate?, 2018.

[95] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[96] Yan Sun, Celia Chen, Qing Wang, and Barry W. Boehm. Improving missing issue-commit link recovery using positive and unlabeled data. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*, pages 147–152, 2017.

[97] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In *Advances in neural information processing systems*, pages 2553–2561, 2013.

[98] Amjed Tahir, Aiko Yamashita, Sherlock Licorish, Jens Dietrich, and Steve Counsell. Can you tell me if it smells? a study on how developers

discuss code smells and anti-patterns in stack overflow. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, pages 68–78, 2018.

[99] Christoph Treude and Markus Wagner. Predicting good configurations for github and stack overflow topic models. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 84–95. IEEE, 2019.

[100] Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. Deep learning similarities from different representations of source code. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pages 542–553. IEEE, 2018.

[101] Gias Uddin and Foutse Khomh. Automatic summarization of api reviews. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 159–170. IEEE, 2017.

[102] Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. Template-based question answering over RDF data. In *Proceedings of the 21st international conference on World Wide Web*, pages 639–648. ACM, 2012.

[103] Haoye Wang, Xin Xia, David Lo, John Grundy, and Xinyu Wang. Automatic solution summarization for crash bugs. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1286–1297. IEEE, 2021.

[104] Liting Wang, Li Zhang, and Jing Jiang. Detecting duplicate questions in stack overflow via deep learning approaches. In *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, pages 506–513. IEEE, 2019.

[105] Shaowei Wang, David Lo, Bogdan Vasilescu, and Alexander Serebrenik. Entagrec: An enhanced tag recommendation system for software information sites. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 291–300, 2014.

[106] Shaowei Wang, David Lo, Bogdan Vasilescu, and Alexander Serebrenik. Entagrec: An enhanced tag recommendation system for software information sites. In *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014*, pages 291–300. IEEE Computer Society, 2014.

[107] Shaowei Wang, David Lo, Bogdan Vasilescu, and Alexander Serebrenik. Entagrec++: An enhanced tag recommendation system for software information sites. *Empirical Software Engineering*, 23(2):800–832, 2018.

[108] Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk. Deep learning code fragments for code clone detection. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 87–98. ACM, 2016.

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

[110] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

[111] Wan-Ching Wu. How far will you go?: characterizing and predicting online search stopping behavior using information scent and need for cognition. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 1149–1149. ACM, 2013.

[112] Yuhao Wu, Shaowei Wang, Cor-Paul Bezemer, and Katsuro Inoue. How do developers utilize source code from stack overflow? *Empirical Software Engineering*, 24(2):637–673, 2019.

[113] Xin Xia, Lingfeng Bao, David Lo, Pavneet Singh Kochhar, Ahmed E Hassan, and Zhenchang Xing. What do developers search for on the web? *Empirical Software Engineering*, pages 1–37, 2017.

[114] Xin Xia, David Lo, Xinyu Wang, and Bo Zhou. Tag recommendation in software information sites. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 287–296. IEEE Press, 2013.

[115] Bowen Xu, Amirreza Shirani, David Lo, and Mohammad Amin Alipour. Prediction of relatedness in stack overflow: deep learning vs. svm: a reproducibility study. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, page 21. ACM, 2018.

[116] Bowen Xu, Zhenchang Xing, Xin Xia, and David Lo. Answerbot: automated generation of answer summary to developersź technical questions. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, pages 706–716. IEEE Press, 2017.

[117] Bowen Xu, Zhenchang Xing, Xin Xia, David Lo, and Xuan-Bach D Le. Xsearch: a domain-specific cross-language relevant question retrieval tool. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 1009–1013. ACM, 2017.

[118] Bowen Xu, Zhenchang Xing, Xin Xia, David Lo, Qingye Wang, and Shanping Li. Domain-specific cross-language relevant question retrieval. In *Proceedings of the 13th International Conference on Mining Software Repositories*, pages 413–424. ACM, 2016.

[119] Bowen Xu, Deheng Ye, Zhenchang Xing, Xin Xia, Guibin Chen, and Shanping Li. Predicting semantically linkable knowledge in developer online forums via convolutional neural network. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 51–62. ACM, 2016.

[120] Xin-Li Yang, David Lo, Xin Xia, Zhi-Yuan Wan, and Jian-Ling Sun. What security questions do developers ask? a large-scale study of stack overflow posts. *Journal of Computer Science and Technology*, 31(5):910–924, 2016.

[121] Xinli Yang, David Lo, Xin Xia, Lingfeng Bao, and Jianling Sun. Combining Word Embedding with Information Retrieval to Recommend Similar Bug Reports. In *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on*, pages 127–137. IEEE, 2016.

[122] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

[123] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016.

[124] Deheng Ye, Zhenchang Xing, Chee Yong Foo, Zi Qun Ang, Jing Li, and Nachiket Kapre. Software-specific named entity recognition in software engineering social content. In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, volume 1, pages 90–101. IEEE, 2016.

[125] Deheng Ye, Zhenchang Xing, and Nachiket Kapre. The structure and dynamics of knowledge network in domain-specific q&a sites: a case study of stack overflow. *Empirical Software Engineering*, 2016.

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

[127] Xin Ye, Hui Shen, Xiao Ma, Razvan Bunescu, and Chang Liu. From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th International Conference on Software Engineering*, pages 404–415. ACM, 2016.

[128] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. Droid-sec: deep learning in android malware detection. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 371–372. ACM, 2014.

[129] Haoxiang Zhang, Shaowei Wang, Tse-Hsun Peter Chen, Ying Zou, and Ahmed E Hassan. An empirical study of obsolete answers on stack overflow. *IEEE Transactions on Software Engineering*, 2019.

[130] Tianyi Zhang, Ganesha Upadhyaya, Anastasia Reinhardt, Hridesh Rajan, and Miryung Kim. Are code examples on an online q&a forum reliable?: a study of api misuse on stack overflow. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 886–896. IEEE, 2018.

[131] Yun Zhang, David Lo, Pavneet Singh Kochhar, Xin Xia, Quanlai Li, and Jianling Sun. Detecting similar repositories on GitHub. In *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*, pages 13–23. IEEE, 2017.

[132] Cheng Zhou. Intelligent bug fixing with software bug knowledge graph. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 944–947, 2018.

[133] Pingyi Zhou, Jin Liu, Xiao Liu, Zijiang Yang, and John Grundy. Is deep learning better than traditional approaches in tag recommendation for software information sites? *Information and Software Technology*, 2019.

[134] Pingyi Zhou, Jin Liu, Zijiang Yang, and Guangyou Zhou. Scalable tag recommendation for software information sites. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 272–282, 2017.

List of Papers

1. **Xu, B.**, Hoang, T., Sharma, A., Yang, C., Xia, X. & Lo, D., "Post2Vec: Learning Distributed Representations of Stack Overflow Posts," in IEEE Transactions on Software Engineering, doi: 10.1109/TSE.2021.3093761.

2. **Xu, B.**, An, L., Thung, F., Khomh, F., & Lo, D. (2020). Why reinventing the wheels? An empirical study on library reuse and re-implementation. Empirical Software Engineering, 25(1), 755-789. **Presented during ICSE 2020 as part of the Journal First Paper Track**

3. Zhang, T., **Xu, B.** (*Corresponding Author*), Thung, F., Haryono, S. A., Lo, D., & Jiang, L. (2020, September). Sentiment analysis for software engineering: How far can pre-trained transformer models go?. In 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 70-80). IEEE.

4. Cai, L., Wang, H., **Xu, B.**, Huang, Q., Xia, X., Lo, D., Xing, Z. (2019, August). AnswerBot: an answer summary generation tool based on stack overflow. In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 1134-1138).

5. Li, H., Shi, S., Thung, F., Huo, X., **Xu, B.**, Li, M., & Lo, D. DeepReview: Automatic code review using deep multi-instance learning.(2019). In Advances in knowledge discovery and data mining: 23rd Pacific-Asia Conference, PAKDD 2019, Macau, China, April 14-17: Proceedings (Vol. 11440, pp. 318-330).

6. **Xu, B.**, Shirani, A., Lo, D., & Alipour, M. A. (2018, October). Prediction of relatedness in stack overflow: deep learning vs. SVM: a reproducibility study. In Proceedings of the 12th ACM/IEEE International

Symposium on Empirical Software Engineering and Measurement (pp. 1-10). **Highly Commended Full Paper Award**

7. **Xu, B.**, Xing, Z., Xia, X., & Lo, D. (2017, October). AnswerBot: Automated generation of answer summary to developers' technical questions. In 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 706-716). IEEE.