Prediction of Relatedness in Stack Overflow: Deep Learning vs. SVM

A Reproducibility Study

Bowen Xu*
Singapore Management University
bowenxu.2017@smu.edu.sg

Amirreza Shirani*
University of Houston
ashirani@uh.edu

David Lo
Singapore Management University
davidlo@smu.edu.sg

Mohammad Amin Alipour
University of Houston
alipour@cs.uh.edu

ABSTRACT

Background 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. Thus, they suggested that researchers need to compare their sophisticated methods against simpler alternatives.
Aim The aim of this work is 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. We also compare the effectiveness of these two approaches against SimBow, a lightweight SVM-based method that was previously used for general community question-answering.
Method We (1) collect a large dataset containing knowledge units from Stack Overflow, (2) show the value of the new dataset addressing shortcomings of the original one, (3) re-evaluate both the DNN- and SVM-based approaches on the new dataset, and (4) compare the performance of the two approaches against that of SimBow.
Results We find that: (1) there are several limitations in the original dataset used in the previous studies, (2) effectiveness of both Xu et al.'s and Fu and Menzies' approaches (as measured using F1-score) drop sharply on the new dataset, (3) similar to the previous finding, performance of SVM-based approaches (Fu and Menzies' approach and SimBow) are slightly better than the DNN-based approach, (4) 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, and (5) SimBow outperforms both Xu et al. and Fu and Menzies' approaches in terms of runtime.
Conclusion We conclude that, for this task, simpler approaches based on SVM performs adequately well. We also illustrate the challenges brought by the increased size of the dataset and show the benefit of a lightweight SVM-based approach for this task.

*Equal contribution.

ESEM '18, October 11–12, 2018, Oulu, Finland
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5823-1/18/10…$15.00
https://doi.org/10.1145/3239235.3240503

CCS CONCEPTS

• **Theory of computation → Support vector machines**; • **Computing methodologies → Neural networks**; • **Software and its engineering → Software libraries and repositories**;

KEYWORDS

Relatedness Prediction, Deep Learning, Support Vector Machine

ACM Reference format:
Bowen Xu, Amirreza Shirani, David Lo, and Mohammad Amin Alipour. 2018. Prediction of Relatedness in Stack Overflow: Deep Learning vs. SVM . In *Proceedings of ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), Oulu, Finland, October 11–12, 2018 (ESEM '18),* 10 pages.
https://doi.org/10.1145/3239235.3240503

1 INTRODUCTION

Using machine learning techniques in software engineering research has been commonplace, such as [9, 22, 30] 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 [7]; 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 [13]; it can substantially reduce the burden of feature engineering. Deep learning has produced promising results in complex tasks such as object detection [23], natural language understanding [19], text classification [11] and many more.

Nowadays, there has been a surge in adoption of deep learning[1] in software engineering research. It has been applied successfully

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

to problems such as [6, 30, 33]. 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. [30] and Fu and Menziess [4] 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 [4] 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 [2].

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 [4]. 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 [3] 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.
- We release source code for SimBow and the new dataset, along with the experiment results at https://github.com/XBWer/ESEM2018.

Organization The rest of this paper is organized as follows. Section 2 defines the problem of predicting relatedness of Stack Overflow knowledge units and the evaluation metrics. Sections 3 provides replication of the two approaches proposed by Xu et al. and Fu and Menzies along with one new adopted approach SimBow. Section 4 explains the creation process of the dataset. Section 5 presents the research questions and corresponding results. Section 6 discusses the possible shortcomings with the previous dataset used in Xu et al. and Fu and Menzies's studies. Section 8 describes related works. Section 9 concludes the paper.

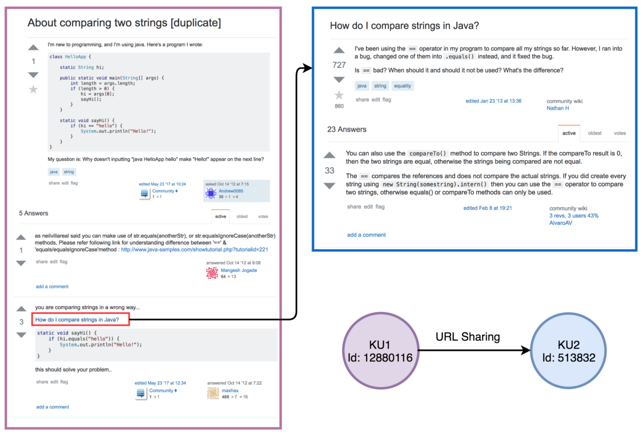
2 TASK AND EVALUATION METRICS

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 [15]. 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; 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. [30], 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]

1

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?

java string

share edit flag — edited May 23 '17 at 10:24 Community ♦ 1 • 1 — asked Oct 14 '12 at 7:15 Andrew0085 30 • 1 • 4

5 Answers — active oldest votes

1

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

share edit flag — answered Oct 14 '12 at 9:08 Mangesh Jogade 64 • 13

add a comment

3

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..

share edit flag — edited May 23 '17 at 12:34 Community ♦ 1 • 1 — answered Oct 14 '12 at 7:22 maxhax 486 • 7 • 16

add a comment

How do I compare strings in Java?

727

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?

java string equality

860

share edit flag — edited Jan 23 '13 at 13:36 — community wiki Nathan H

23 Answers — active oldest votes

33

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.

share edit flag — edited Feb 8 at 19:21 — community wiki 3 revs, 3 users 43% AlvaroAV

add a comment

Figure 1: Linked Knowledge Units by URL Sharing

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

Table 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.

Identifying relatedness of knowledge units would accelerate developer's ability in navigating the rich and yet diverse information in Stack Overflow. Thus, Stack Overflow encourages developers to link related knowledge units by URL sharing [20]. Figure 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 [32]. As shown in Table 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 [30]. 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 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 4.

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 [4, 30], 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 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.

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

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 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 Soft SVM, respectively.

- **CNN Model, Xu et al. [30]**: Appeared in ASE 2016.
- **Tuning SVM, Fu and Menzies [4]**: Appeared in FSE 2017.
- **Soft SVM, SimBow [3]**: Appeared in SemEval-2017 Task 3: Community Question Answering.

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

At ASE 2016, Xu et al. [30] 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 [30].

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.

Overview of Approach To predict the relatedness between knowledge units, Xu et al. built a convolutional neural network (CNN) model [14] 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 [16]. 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 [16] is used to learn domain-specific word

Table 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"; String s2 = "andrei"; String s3 = s2.toString();`
	`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"; String b = "devender"; String c = "dev"; String d = "dev" + "ender";`
	`String e = c + "ender";`
	`System.out.println(a == b);  //case 1: o/p true`
	`System.out.println(a == d);  //case 2: o/p true`
	`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: 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}

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. As shown as Figure 2, 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:

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

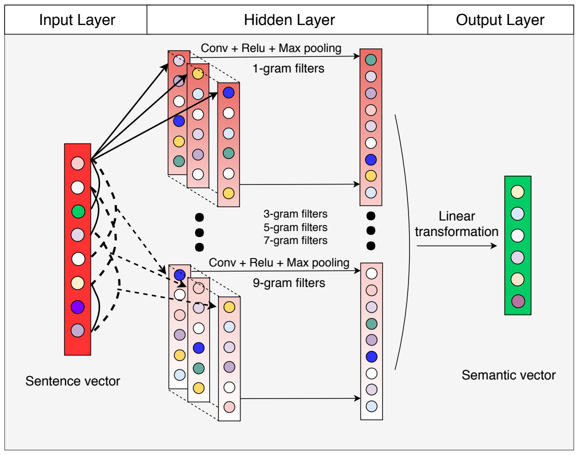


Figure 2: CNN Architecture in CNN Model

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 [30], 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).

3.2 Fu and Menzies' Study (Tuning SVM)

In FSE 2017, Fu and Menzies [4] 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 3.1 is computationally too expensive for the task of predicting relatedness of knowledge units. They propose tuned support vector machines for this problem.

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 [10].

Parameter Tuning Fu and Menzies [4] use word2vec representation [16] 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) [21] 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 [30], 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 [4] 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 SimBow: A Lightweight Alternative (Soft SVM)

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 (Equation 1).

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

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 2.

$$soft-cosine(a,b) = \frac{\sum\sum_{i,j}^{N} a_i m_{ij} b_j}{\sqrt{\sum\sum_{i,j}^{N} a_i m_{ij} a_j}\sqrt{\sum\sum_{i,j}^{N} b_i m_{ij} b_j}} \tag{2}$$

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

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

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

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) for computing soft-cosine similarity based on word embedding.

$$m_{ij} = max(0, cosine(v_i, v_j))^2 \tag{3}$$

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:

$$m_{ij} = \alpha * (1 - \frac{Levenshtein(w_i, w_j)}{max(||w_i||, ||w_j||)})^{\beta} \tag{4}$$

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) [16] 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 [17] 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 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.

4 DATA

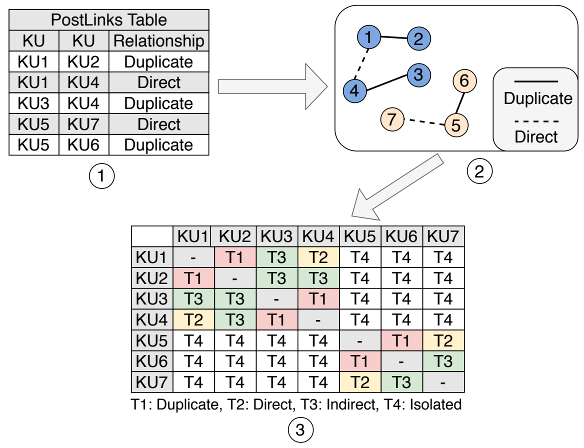
Both Xu et. al. [30], and Fu and Menzies [4] perform experiments on a small dataset that we call OriginalDataset. OriginalDataset contains only 8,000 pairs of Java knowledge units (i.e., tagged with "Java"): 2,000 pairs of knowledge units for each type of relationships, among them, 1,600 pairs are used for training and 400 pairs are used for testing.

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 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 ∈ [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).


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: Dataset Building Process

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

Table 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**

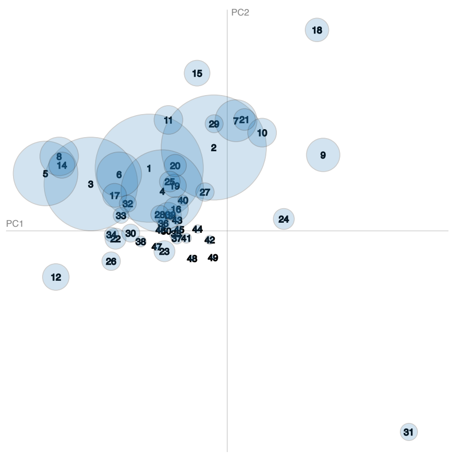


Figure 4: Topic Distribution of OriginalDataset

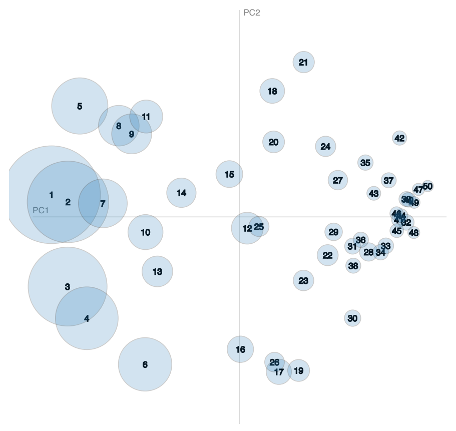


Figure 5: Topic Distribution of LargeDataset

4.2 Characteristics of the new dataset

We followed the steps outlined in Section 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 4 and 5 shows the graphical distribution of topics in OriginalDataset and LargeDataset, respectively. As is shown, compared to LargeDataset, OriginalDataset covers fewer topics.

5 RESEARCH QUESTIONS AND RESULTS

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 Soft SVM 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 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.

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 5 depicts performance of models trained by CNN Model and Tuning SVM and Soft SVM for individual classes along with the overall 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.

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

Table 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

Table 5: Performance of Three Systems on Large Dataset

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

impact on the computation time. The Table 6 shows that Tuning SVM takes considerably more computation time than other techniques—around 2.5x and 12.5x more than CNN Model and Soft SVM, respectively. Training model in Soft SVM was by far the faster than others.

Table 6: Training Time in Different Techniques

	Time
CNN Model.	15h 21m 24s
Tuning SVM	38h 24m 46s
Soft SVM	2h 54m

RQ3: Discrepancies between the performance of techniques on OriginalDataset and LargeDataset

Comparison of performance Table 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 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 [4] still holds on the LargeDataset, i.e., CNN Model and Tuning SVM can achieve similar results.

Comparison of training computation cost According to the data reported in [4, 30], training CNN Model and Tuning SVM on OriginalDataset take almost 14 hours, and 10 minutes, respectively, on regular machines. Table 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 [8]. On the other hand, when using LargeDataset, there is only a slight increase in the run-time of CNN Model approach which proves that the scale of the dataset has slight impacts on the runtime of the technique.

RQ4: Performance of Soft SVM

Performance Rows corresponding to Soft SVM in Table 5 contain performance of Soft SVM, i.e., 0.52 F1-score, 0.53, 0.54 precision and recall, respectively. The results show that Soft SVM 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 Soft SVM has a performance advantage in predicting the Isolated class over the other methods. In other three classes, Soft SVM 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, Soft SVM spends much less time than the other two approaches. As shown in Table 6, Soft SVM needs only less than 3 hours for training on LargeDataset, while CNN Model and Tuning SVM require 5x and 12x more time for training.

6 DISCUSSION

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.

6.2 Performance of techniques

Our results on LargeDataset show that, consistent with the [4], 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. Soft SVM, 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, Soft SVM, 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.

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 [30] 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].

8 RELATED WORK

8.1 Knowledge Analysis in Stack Overflow

Many studies have been done on leveraging the knowledge on Stack Overflow to improve developers’ productivity [25, 28, 29, 32]. Ye et al. conducted an empirical study on analyzing the structure of knowledge network in Stack Overflow [32]. They defined a question and its answers on Stack Overflow as a *knowledge unit*. To better understand the knowledge diffusion process, they presented a methodology to analyze URL sharing activities in Stack Overflow. They found that knowledge units often contain semantically relevant knowledge, and thus linkable for different purposes. For example, two knowledge units will be labeled as duplicate if they talk about the same technical problems but in different ways. The structure of the knowledge network with respect to in-degree distribution is scale-free, in spite of the ad-hoc and opportunistic nature of URL sharing activities, while the out-degree distribution of the knowledge network is not scale-free. Gao et al. leveraged knowledge in Stack Overflow to fix recurring bugs in software systems [5]. Firstly, they extracted queries from crash traces and retrieved a list of Q&A pages from Stack Overflow. Secondly, they analyzed the pages and generate edit scripts. Thirdly, those generated scripts are applied to target source code and filter out the incorrect patches.

In this work, we focus on the task of predicting relatedness between know ledge units to support more targeted information needs when users search or explore the knowledge base. compared to previous studies which only focus on binary relationship (e.g., duplicate question prediction [1, 34]) between knowledge units, our task is more challenging since there are multiple classes of relatedness.

8.2 Traditional Machine Learning v.s. Deep Learning

Deep learning approaches are widely applied in software engineering tasks [6, 12, 31, 33]. However, there are very limited studies to compare performance of deep learning and traditional machine learning techniques.

Lam et al. proposed a bug localization approach which combines three deep neural network (DNN) models for different goals [12]. The first DNN model is built to bridge the lexical gap between bug reports and source code. The second DNN model is built for feature combination. The third DNN model is built to perform dimension reduction for feature vectors. The experiment data are six projects with more than 26,000 source files and the experiment is performed based on a PC with CPU Intel Xeon CPU E5-2650 2.00GHz (32 cores), 126 GB RAM. Based on the experiment results, their approach outperforms the baseline approach which is based on machine learning approach (i.e., Naive Bayes). However, the runtime information of the baseline methods was not reported and they just claim that training time of DNN is large if only run in one thread.

Yuan et al. presented a deep learning based approach for Android malware detection [33]. More specifically, they adopted a deep belief network (DBN) to classify malware from normal apps. They crawled 250 malware apps and 250 normal apps from app store. In the experiment, they compare the proposed deep learning based approach outperforms other five machine learning approaches (i.e., SVM, C4.5, Naive Bayes, Linear Regression and Multi-layer Perceptron). Unfortunately, neither execution time nor experiment environment are provided.

Yang et al. also built a deep learning based classifier for defect prediction [31]. In particular, they use DBN which contains three stacked restricted boltzmann machines (RBMs) and a logistic regression classifier. They compare their approach against two baselines, a standard logistic regression classifier and a random under-sampling and logistic regression classifier without DBN. The experiment dataset are six open source projects (i.e., Bugzilla, Columba, Eclipse

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

JDT, Eclipse Platform, Mozilla and PostgreSQL) which contains a total of 137,417 changes. The experiment results show that their approach achieves the best performance on 4 out of the 6 projects. However, the proposed approach (9.98s) spend 12 times more time than the baseline (0.79s) under the experimental environment that an Intel(R) Core(TM) T6570 2.10 GHz CPU, 4GB RAM desktop running Windows 7 (32-bit).

Above all, many previous works only presented the high effectiveness of deep learning approach without showing any information of execution time [18, 27, 33]. On the other hand, some works just simply report the time cost of the proposed approaches [6, 12, 24, 26]. In this work, we focus on the task that predicting relatedness between knowledge units on Stack Overflow and further evaluate effectiveness and efficiency of different approaches on a larger dataset.

9 CONCLUSION

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) 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.

REFERENCES

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

[2] Begley, C. G., and Ioannidis, J. P. Reproducibility in science: improving the standard for basic and preclinical research. *Circulation research 116*, 1 (2015), 116–126.

[3] Charlet, D., and Damnati, G. 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)* (2017), pp. 315–319.

[4] Fu, W., and Menzies, T. Easy over hard: A case study on deep learning. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (2017), ACM, pp. 49–60.

[5] Gao, Q., Zhang, H., Wang, J., Xiong, Y., Zhang, L., and Mei, H. Fixing recurring crash bugs via analyzing q&a sites (t). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on* (2015), IEEE, pp. 307–318.

[6] Gu, X., Zhang, H., Zhang, D., and Kim, S. Deep api learning. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (2016), ACM, pp. 631–642.

[7] Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural networks 2*, 5 (1989), 359–366.

[8] Hsu, C.-W., Chang, C.-C., Lin, C.-J., et al. A practical guide to support vector classification. In *Technical Report* (2003), Department of Computer Science and Information Engineering, National Taiwan University, Taiwan.

[9] Jiang, S., Armaly, A., and McMillan, C. 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* (2017), pp. 135–146.

[10] Joachims, T. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning* (1998), Springer, pp. 137–142.

[11] Lai, S., Xu, L., Liu, K., and Zhao, J. Recurrent convolutional neural networks for text classification. In *AAAI* (2015), vol. 333, pp. 2267–2273.

[12] Lam, A. N., Nguyen, A. T., Nguyen, H. A., and Nguyen, T. N. Combining deep learning with information retrieval to localize buggy files for bug reports (n). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on* (2015), IEEE, pp. 476–481.

[13] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature 521*, 7553 (2015), 436.

[14] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural computation 1*, 4 (1989), 541–551.

[15] Mamykina, L., Manoim, B., Mittal, M., Hripcsak, G., and Hartmann, B. Design lessons from the fastest q&a site in the west. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2011), CHI '11, ACM, pp. 2857–2866.

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

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

[18] Mou, L., Li, G., Zhang, L., Wang, T., and Jin, Z. Convolutional neural networks over tree structures for programming language processing. In *AAAI* (2016), vol. 2, p. 4.

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

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

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

[22] Sun, Y., Chen, C., Wang, Q., and Boehm, B. W. 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* (2017), pp. 147–152.

[23] Szegedy, C., Toshev, A., and Erhan, D. Deep neural networks for object detection. In *Advances in neural information processing systems* (2013), pp. 2553–2561.

[24] Wang, S., Liu, T., and Tan, L. Automatically learning semantic features for defect prediction. In *Proceedings of the 38th International Conference on Software Engineering* (2016), ACM, pp. 297–308.

[25] Wang, S., Lo, D., and Jiang, L. An empirical study on developer interactions in stackoverflow. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing* (2013), ACM, pp. 1019–1024.

[26] White, M., Tufano, M., Vendome, C., and Poshyvanyk, D. Deep learning code fragments for code clone detection. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (2016), ACM, pp. 87–98.

[27] White, M., Vendome, C., Linares-Vásquez, M., and Poshyvanyk, D. Toward deep learning software repositories. In *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on* (2015), IEEE, pp. 334–345.

[28] Xu, B., Xing, Z., Xia, X., and Lo, D. Answerbot: automated generation of answer summary to developersź technical questions. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering* (2017), IEEE Press, pp. 706–716.

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

[30] Xu, B., Ye, D., Xing, Z., Xia, X., Chen, G., and Li, S. 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* (2016), ACM, pp. 51–62.

[31] Yang, X., Lo, D., Xia, X., Zhang, Y., and Sun, J. Deep learning for just-in-time defect prediction. In *Software Quality, Reliability and Security (QRS), 2015 IEEE International Conference on* (2015), IEEE, pp. 17–26.

[32] Ye, D., Xing, Z., and Kapre, N. The structure and dynamics of knowledge network in domain-specific q&a sites: a case study of stack overflow. *Empirical Software Engineering* (2016).

[33] Yuan, Z., Lu, Y., Wang, Z., and Xue, Y. Droid-sec: deep learning in android malware detection. In *ACM SIGCOMM Computer Communication Review* (2014), vol. 44, ACM, pp. 371–372.

[34] Zhang, Y., Lo, D., Xia, X., and Sun, J.-L. Multi-factor duplicate question detection in stack overflow. *Journal of Computer Science and Technology 30*, 5 (2015), 981–997.