Predicting Semantically Linkable Knowledge in Online Forums via Convolutional Neural Network

Bowen Xu\textsuperscript{1} ∗, Deheng Ye\textsuperscript{2} ∗, Zhenchang Xing\textsuperscript{2}, Xin Xia\textsuperscript{1} †, Guibin Chen\textsuperscript{2}, Shanping Li\textsuperscript{1}

\textsuperscript{1}College of Computer Science and Technology, Zhejiang University, China
\textsuperscript{2}School of Computer Science and Engineering, Nanyang Technological University, Singapore

max_xbw@zju.edu.cn, ye0014ng@e.ntu.edu.sg, zcxing@ntu.edu.sg, xxia@zju.edu.cn, gbchen@ntu.edu.sg, shan@zju.edu.cn

ABSTRACT

Consider a question and its answers in Stack Overflow as a knowledge unit. Knowledge units often contain semantically relevant knowledge, and thus linkable for different purposes, such as duplicate questions, directly linkable for problem solving, indirectly linkable for related information. Recognising different classes of linkable knowledge would support more targeted information needs when users search or explore the knowledge base. Existing methods 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 (i.e., have lexical gap). In this paper, we formulate the problem of predicting semantically linkable knowledge units as a multiclass classification problem, and solve the problem using deep learning techniques. To overcome the lexical gap issue, we adopt neural language model (word embeddings) and convolutional neural network (CNN) to capture word- and document-level semantics of knowledge units. Instead of using human-engineered classifier features which are hard to design for informal user-generated content, we exploit large amounts of different types of user-created knowledge-unit links to train the CNN to learn the most informative word-level and document-level features for the multiclass classification task. Our evaluation shows that our deep-learning based approach significantly and consistently outperforms traditional methods using traditional word representations and human-engineered classifier features.

CCS Concepts

- Software and its engineering → Software libraries and repositories;
- Information systems → Social networking sites;

\textsuperscript{∗}Joint first authors, contributed equally.
\textsuperscript{†}Corresponding author.

Keywords

Link prediction, Semantic relatedness, Multiclass classification, Deep learning, Mining software repositories

1. INTRODUCTION

In Stack Overflow, computer programming knowledge has been shared through millions of questions and answers. We consider a Stack Overflow question with its entire set of answers as a knowledge unit regarding some programming-specific issues. The knowledge contained in one unit is likely to be related to knowledge in other units. When asking a question or providing an answer in Stack Overflow, users reference existing questions and answers that contain relevant knowledge by URL sharing \cite{46}, which is strongly encouraged by Stack Overflow \cite{2}. Through URL sharing, a network of linkable knowledge units has been formed over time \cite{46}.

Unlike linked pages on Wikipedia that follows the underlying knowledge structure, questions and answers are specific to individual’s programming issues, and URL sharing in Q&As is opportunistic, because it is based on the community awareness of the presence of relevant questions and answers. A recent study by Ye et al. \cite{46} shows that the structure of the knowledge network that URL sharing activities create is scale free. A scale free network follows a power law degree distribution, which can be explained using preferential attachment theory \cite{4}, i.e., “the rich get richer”. On one hand, this means that a small proportion of the knowledge units is attracting a large proportion of users’ attention. On the other hand, this means that large amounts of knowledge units in the long tail of the power law distribution are mostly under linked.

To mitigate this issue, Stack Overflow recommends related questions when users are viewing a question. Stack Overflow’s recommendation of related questions is essentially based on lexical similarity of word overlap between questions \cite{1}. However, linkable knowledge units often share few words in common (i.e., lexical gap) due to two main reasons. First, users could formulate the same question in many different ways. For example, Table \ref{table:1} shows two duplicate questions from Stack Overflow. The question (Id 510357) has been recognized as a duplicate to another question (Id 19477465) by the Stack Overflow users, because they can be answered by the same answer. However, literally, we can hardly see common words and phrases between these two questions. Second, two questions could discuss different but...
relevant knowledge. For example, Table 2 shows two directly linkable questions, one asks “iterating over a stack (reverse list)” while the other asks “best way to check if a list is empty”. Again, the two questions share few common words, but they embody strongly relevant knowledge (stack iteration and list empty checking). Furthermore, questions may discuss relevant knowledge but not directly for solving the questions, as the examples in Table 3 shows. As such questions discuss indirectly linkable knowledge, they are unlikely to share many common words. Traditional word representations (e.g., BM25 [26], LDA [5]) cannot reliably recognize many cases of potentially linkable knowledge units when lexical gaps exist between them.

As shown in the above examples, knowledge units can be linkable for different purposes. Ye et al.’s empirical study [46] on the knowledge network in Stack Overflow confirms this phenomenon. 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. However, such multiclass semantic relatedness in software engineering knowledge is not considered in existing work [20, 6], which focuses on only binary classification, i.e., related or not. Furthermore, existing work heavily relies on human-engineered word- and document-level features to train the classifier, which are hard to design for informal user-generated content in Stack Overflow.

In this paper, we propose a novel approach for predicting multiclass semantically linkable knowledge units in Stack Overflow. Inspired by Ye et al. [46], we consider four classes of semantic relatedness: duplicate, direct linkable, indirectly linkable, and isolated. To overcome the sequential gap issue, Our approach recognizes and quantifies semantic relatedness between knowledge units using word embeddings [21, 20] and Convolutional Neural Network (CNN) [13], which are the state-of-the-art deep learning techniques for capturing word-level and document-level semantics for Natural Language Processing (NLP) tasks. Furthermore, our approach does not rely on human-engineered features to classify semantic relatedness. Instead, we collect large amounts of different types of user-created knowledge-unit links (duplicate, direct-linked, indirect-linked) from Stack Overflow to train the CNN to automatically learn the most informative word- and document-level features to classify semantic relatedness between knowledge units.

We conduct extensive experiments to compare our deep-learning based approach with the baseline methods using traditional word representations (TF-IDF) and the human-engineered features for determining semantic similarity. Our results show that: 1) our approach significantly outperforms the baseline methods, and performs much more consistently for different classes of semantic relatedness; 2) both word embeddings and CNN help improve the performance of multiclass classification of linkable knowledge units. CNN plays a more important role for predicting duplicate, directly linkable, and isolated knowledge unit, due to its capability of capturing document-level semantic features, while word embeddings are good at predicting indirectly linkable knowledge units; 3) domain-specific training corpus help improve the performance of deep-learning techniques for specific tasks.

Our contributions are as follows:

- We formulate the research problem of predicting multiclass linkable knowledge units in Stack Overflow;
- We propose a deep learning based approach to tackle the problem;
- We evaluate the effectiveness of our approach against the traditional methods for predicting relevant software engineering knowledge [40].

2. RELATED WORK

Many studies have been carried out on Stack Overflow [35, 44, 36, 11]. Our work predicts semantically linkable knowledge in developers’ discussions in Stack Overflow. It is related to two lines of research: link prediction in complex networks and semantic relatedness in software data.

2.1 Link Prediction in Complex Networks
Link Prediction focuses on detecting potentially linkable objects in an observed network that is complex and evolves dynamically. Link prediction in complex networks has attracted enormous attention from physics, biochemical, and computer science communities. A significant proportion of the related research falls into link prediction in social networks, many of which have been found to be complex networks, e.g., the user-user network in online Q&A forums, in Twitter. These social network research works aim to understand the network evolution patterns and identify potential social relations between users.

Ye et al. report that the knowledge network formed by developers’ URL sharing activities in Stack Overflow is also a complex network, i.e., the network structure is scale-free. They show that “the rich get richer” effect influences the knowledge sharing activities and the growth of the knowledge network in Stack Overflow. As a result, a small portion of questions and answers attracts a large portion of developers’ attention, while large amounts of relevant questions and answers in the long tail of the power law distribution are under linked. This finding motivates our work to predict potentially linkable knowledge units, which could enhance the knowledge sharing and search in Stack Overflow.

Unlike link prediction in social network which predicts only whether the two persons are linkable or not (i.e., binary classification), our work predicts different types of relatedness (duplicate, direct link, or indirect link) between potentially linkable knowledge units (i.e., multiclass classification). This multiclass link prediction is again inspired by the study of Ye et al., which reveals that users link relevant questions and answers for different purposes. They suggest that mixing different types of linkable knowledge units together could hinder the knowledge search as users often need different types of relevant information in different situations. Therefore, we treat the linkable prediction as a multiclass classification problem.

### 2.2 Semantic Relatedness in Software Data

Measuring the relatedness (or similarity) between two pieces of textual contents has long been studied. The underlying mathematic model of textual contents has evolved from Vector-Space-Model (e.g., TF-IDF), n-gram language model, topic modeling (e.g., LDA), to the recent development of neural language model (e.g., distributed word representations (or word embeddings)). The trend is towards semantically richer similarity measures. In 2013, Mikolov et al. propose two neural language models (referred to as word2vec in the literature) (continuous bag-of-words and continuous skip-gram) and efficient negative sampling method for learning word embeddings from large amounts of text. In this work, we adopt continuous skip-gram model for learning domain-specific word embeddings from Stack Overflow text.

In the software engineering community, researchers utilize the textual similarity between two software artifacts to solve various software engineering tasks, such as duplicate or similar bug report detection, relevant software tweets detection, duplicate online question detection. For example, Sun et al. use a Support Vector Machine (SVM) classifier to classify and an IR based similarity measure in respectively, to detect duplicate bug reports. Zhang et al. detect duplicate Stack Overflow questions by comparing their titles and question descriptions using topic model (LDA). These existing work are formulated as a binary prediction problem, e.g., duplicate or non-duplicate.

In the NLP community, the development of CNN architectures for sentence-level and document-level text processing is under intensive research. Some recent work utilizes CNN to learn the semantic relations between two pieces of texts. For example, Kim proposes a simple CNN trained on top of Mikolov’s word embeddings, and then apply the CNN to sentence classification. Bogdanova et al. apply CNN with word embeddings to duplicate question detection in online Q&A sites. These CNN-based methods outperform traditional NLP methods for sentence classification and duplication question detection.

Comparing with these existing works on semantic relatedness between two pieces of text, we leverage CNN with domain-specific word embeddings for the problem of multiclass classification of potentially linkable knowledge units in Stack Overflow, beyond the binary prediction of duplicate content or not.

### 3. THE APPROACH

This section formulates our research problem and describes the technical details of our approach.

#### 3.1 Problem Formulation

We consider a question together with its entire set of answers in Stack Overflow as a knowledge unit regarding some programming-specific issues. If two knowledge units are semantically related, we consider them as linkable. We further predict the types of the relatedness between the two knowledge units. Inspired by Ye et al.’s study on the purposes of URL sharing, we define four types of the relatedness between the two knowledge units:

- **Duplicate**: Two knowledge units discuss the same question in different ways, and can be answered by the same answer.
- **Direct link**: 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 link**: 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.

We formulate our task as a multiclass classification problem. Specially, given two knowledge units, our task is to predict whether the two knowledge units have one of the above four types of relatedness.

#### 3.2 Overview of Main Steps

In NLP tasks, words are usually represented as vectors. In this work, we use word embeddings (distributed representations of words in a vector space) as word representations, because word embeddings require only large amounts of unlabeled text to train, and have been shown to be able to capture rich semantic and syntactic features of words. We use continuous skip-gram model (the Python implementation in Gensim) to learn domain-specific word embeddings from large amounts of software engineering text from Stack Overflow questions and answers. The output is a dictionary of word embeddings for each unique word.
Word embeddings encode word-level semantics. To predict semantic relatedness between knowledge units, we need to capture semantics at sentence and knowledge unit level. To this end, we resort to convolutional neural networks [13].

Given two knowledge units, they are first converted into word vector representations by looking up the word embeddings dictionary, which are then fed into a CNN to produce a feature vector for each knowledge unit. The similarity of the two knowledge units are measured by the cosine similarity of their feature vectors. Based on the similarity score, the relatedness of the two knowledge units are classified as one of the four classes defined in Section 3.1.

To train the CNN for the prediction task, we collect a set of user-linked knowledge-unit pairs from Stack Overflow for each type of relatedness: duplicate, direct link, indirect link, and isolated. This set of training data is fed into the CNN and the training is guided by the cosine loss of the relatedness of the knowledge-unit pairs against the user-linked relatedness. During the training process, the CNN automatically learns the most informative word and sentence features for predicting multiclass linkable knowledge units.

### 3.3 Learning Word Representations

Distributed word representations assume that words appear in similar context tend to have similar meanings [11]. Therefore, individual words are no longer treated as unique symbols, but mapped to a dense real-valued low-dimensional vector space. Each dimension represents a latent semantic or syntactic feature of the word. Semantically similar words are close in the embedding space. Figure 2 shows some examples of domain-specific word embeddings we learn from Stack Overflow java text, visualized in a two-dimensional space using the t-SNE dimensionality reduction technique [29]. Semantically close words, such as JPanel, JButton, JFrame and JLabel which belong to GUI component are close in the vector space.

Word embeddings are unsupervised word representations, which requires only large amounts of unlabeled text to learn. In this work, we collect body content of questions and answers from Stack Overflow as software engineering text. We preprocess the body content by removing large code snippets in <pre> HTML tag and cleaning HTML tags. Short code elements in <code> HTML tag in natural language sentences are kept. We use the software-specific tokenizer developed in [45, 47] to tokenize the sentences. This tokenizer preserves the integrity of code tokens. For example, it tokenizes dataframe . apply () as a single token, instead of a sequence of 5 tokens: dataframe . apply () . This supports more accurate domain-specific word embeddings learning because code tokens will not be mixed with normal words.

Word embeddings are typically learned using neural language models. In this work, we use continuous skip-gram model, a popular word to vector (word2vec) model proposed by Mikolov et al. [21, 20]. As shown in Figure 3, continuous skip-gram model learns the word embedding of a center word (i.e., \( w_i \)) that is good at predicting the surrounding words in a context window of 2k + 1 words (k = 2 in this example). The objective function of skip-gram model is to maximize the sum of log probabilities of the surrounding context words conditioned on the center word:

\[
\sum_{i=1}^{n} \sum_{-k \leq j \leq k, j \neq 0} \log p (w_{i+j} | w_i)
\]

where \( w_i \) and \( w_{i+j} \) denote the center word and the context word in a context window of length 2k + 1. \( n \) denotes the length of the word sequence.
The log \( p(w_{i+j} | w_i) \) is the conditional probability defined using the softmax function:

\[
p (w_{i+j} | w_i) = \frac{\exp(v_{w_{i+j}}^T w_v)}{\sum_{w \in W} \exp(v_{w}^T w_v)}
\]

where \( v_w \) and \( v_{w'} \) are respectively the input and output vectors of a word \( w \) in the underlying neural model, and \( W \) is the vocabulary of all words. Intuitively, \( p(w_{i+j} | w_i) \) estimates the normalized probability of a word \( w_{i+j} \) appearing in the context of a center word \( w_i \) over all words in the vocabulary. Mikolov et al. [2] propose an efficient negative sampling method to compute this probability.

The output of the model is a dictionary of words, each of which is associated with a vector representation. Given a knowledge unit, its title, description and answers content (after proper preprocessing) will be converted into a knowledge-unit vector by looking up the dictionary of word embeddings and concatenating the word embeddings of words comprising the body content. Let \( w_v \in \mathbb{R}^d \) be the \( d \)-dimensional word vector corresponding to the \( i \)-th word in a knowledge unit. The knowledge unit of \( n \) words is represented as, \( knv = w_v1 \oplus w_v2 \oplus \ldots \oplus wvn \), which is used as input to the CNN.

### 3.4 The CNN Architecture

Figure 3 presents the architecture of our CNN. Our CNN takes as input a knowledge-unit vector \( knv = w_v1 \oplus w_v2 \oplus \ldots \oplus wvn \), and captures the most informative word and sentence features in the knowledge unit for determining semantic relatedness. Treating the knowledge-unit vector as an “image”, we perform convolution on it via linear filters. Because each word is represented as a \( d \)-dimensional vector, we use filters with widths \( k \) equal to the dimensionality of the word vectors (i.e., \( k = d \)). Let \( wv_{i+h-1} \) refer to the concatenation vector of \( h \) adjacent words \( w_{i}, \ldots, w_{i+h-1} \). A convolution operation involves a filter \( w \in \mathbb{R}^{h \times k} \) (a vector of \( h \times k \) dimensions) and a bias term \( b \in \mathbb{R}^h \), which is applied to \( h \) words to produce a new value \( o_i \in \mathbb{R} \):

\[
o_i = w^T \cdot wv_{i+h-1} + b \tag{1}
\]

where \( i = 1 \ldots n-h+1 \), and \( \cdot \) is the dot product between the filter vector and the word vector. This filter is applied repeatedly to each possible window of \( h \) words in the sentences (zero padding where necessary) in the knowledge unit (i.e., \( wv_{1:h}, wv_{2:h+1}, \ldots, wv_{n-h+1} \)) to produce an output sequence \( o \in \mathbb{R}^{n-h+1} \), i.e., \( o = [o_1, o_2, \ldots, o_{n-h+1}] \). We apply the non-linear activation function \( ReLu \) to each \( o_i \) to produce a feature map \( c_i \in \mathbb{R}^{n-h+1} \) where \( c_i = ReLu(o_i) = \max(0, o_i) \).

To capture features of phrases of different length, we vary the window size \( h \) of the filter, i.e., the number of adjacent words considered jointly. In our CNN, we use filters of 5 different window size (1, 3, 5, 7, 9), which can capture features of different n-grams respectively. For each window size, we use 128 filters to learn complementary features from the same word windows. The dimensionality of the feature map generated by each filter will vary as a function of the knowledge-unit length and the filter’s window size. Thus, a pooling function should be applied to each feature map to induce a fixed-length vector. In this work, we use 1-max pooling, which extracts a scalar (i.e., a feature vector of length 1) with the maximum value in the feature map for each filter. Together, the outputs from each filter can be concatenated into a top-level feature vector, denoted as \( fv_{ku} \). This feature vector would capture the most informative 1, 3, 5, 7 and 9-grams in the input knowledge unit.

Given two knowledge units \( ku_x \) and \( ku_y \), the CNN outputs their respective feature vectors \( fv_x \) and \( fv_y \). For this pair of feature vectors \( fv_x \) and \( fv_y \), the CNN computes a similarity score between \( fv_x \) and \( fv_y \) in the last layer, using the cosine similarity, i.e.,

\[
relatedness(ku_x, ku_y) = \frac{fv_x \cdot fv_y}{\|fv_x\| \|fv_y\|}
\]

In the training phase, the computed similarity score will be used to compute the mean square error against the ground-truth similarity score for the given pair of knowledge units (see Section 3.5). In the prediction phase, the computed similar score will be used to determine the class of semantic relatedness of the given two knowledge units. Because the computed similarity score is a continuous value, we need to bin the similarity score as four discrete classes of semantic relatedness between the two knowledge units: \([0, 0.25] \) as isolated, \([0.25, 0.5] \) as indirect link, \([0.5, 0.75] \) as direct link, and \([0.75, 1] \) as duplicate.

### 3.5 Training the CNN

Training of the proposed CNN follows supervised learning paradigm. According to our task objective, the training data must consist of sufficient examples of four types of related knowledge-unit pairs, i.e., duplicate, direct link, indirect link and isolated, so that the CNN can learn to capture informative word and sentence features for classifying knowledge-unit relatedness.

Fortunately, URL sharing activities by Stack Overflow users create all types of the needed relatedness. User-created links between knowledge units are stored in the \( post\_links \) table. We parse the \( post\_links \) to generate the training dataset as follows. First, we randomly select pairs of user-linked knowledge-unit pairs. If the \( LinkTypeId \) is 3 (i.e., duplicate posts), we mark the selected pair as an instance of duplicate knowledge units. Otherwise, the selected pair is an instance of direct-link. Second, we randomly select pairs of knowledge units that are only transitively linked (i.e., the shortest path between the knowledge units is at least 2). These pairs of knowledge units are instances of indirect-link knowledge units. Finally, we randomly select pairs of knowledge units that do not have links in the \( post\_links \) table. These pairs are instances of isolated knowledge units. We select equal number of instances for different types of relatedness.

Let the tuple \( < ku_x, ku_y, simValue > \) be a pair of knowledge units in the training dataset \( TD \), and \( simValue \) is the ground-truth similarity score between \( ku_x \) and \( ku_y \). In this work, we set 0.125, 0.375, 0.625 and 0.875 as the ground-
We add the $l^2$ norm regularization loss to the mean-squared error data loss. The $l^2$ norm constraint penalizes large weight parameters. It helps avoid model overfitting, because no input dimension can have a very large influence on the predicted probabilities all by itself. The parameters to be estimated include: the convolution filters $w$ and the bias terms $b$ (see Eq. 1). We solve the optimization problem using Adam update algorithm [17]. Once we learn the CNN parameters, the CNN can be used to predict the semantic relatedness of an unseen pair of knowledge units.

4. EXPERIMENT

We conduct a set of experiments to evaluate the effectiveness of our approach, compare it against a well-designed baseline, and investigate the impact of domain-specific word embeddings. As our approach relies on deep learning techniques, we also report the training cost of our approach.

4.1 Baseline Building

As our task is essentially to predict relatedness between two pieces of text, we design two baselines, each of which is a multiclass SVM classifier with different textual features, i.e., traditional Term Frequency (TF) and Inverse Document Frequency (IDF) versus word embeddings.

4.1.1 Feature Extraction with TF and IDF

TF and IDF are widely used in predicting textual similarity with cosine distance [30]. In this baseline, we define in total 36 features based on the TF and IDF values of the words in a pair of knowledge units. That is, the pair of the knowledge units is represented as a 36-dimensional feature vector to be used as input to the SVM classifier.

Given a knowledge unit $KU_x$, we split its text into three sub-documents, denoted as $C^i_x$ ($1 \leq i \leq 3$), which correspond to question title ($i = 1$), question body ($i = 2$), and question title plus question body plus body of all answers ($i = 3$). Let $KU_x, KU_y > 0$ be a pair of knowledge units.

For TF-based features, we first obtain a vocabulary $V_{2,j}^i_x$ with all words in a combination of different sub-documents of the two knowledge units, i.e., $V_{2,j}^i_x = \{ w \mid w \in C^i_x \cup C^i_y \}$. The TF value of each word in the vocabulary $V_{2,j}^i_x$ is then computed. These word TF values are used to convert the sub-document $C^i_x$ of knowledge unit $KU_x$ into a vector representation $v_{i}^x$. The dimensionality of the $v_{i}^x$ is the size of the vocabulary, and each dimension is the TF value of the word in the sub-document $C^i_x$, or 0 otherwise. We compute the cosine similarity of the two vectors $v_{i}^x$ and $v_{j}^y$ to measure the similarity of the sub-document $C^i_x$ of the knowledge unit $KU_x$ and $KU_y$, respectively. As each knowledge unit has three sub-documents, i.e., $1 \leq i, j \leq 3$, we obtain $3 \times 3 \times 9$ TF-based features for a pair of knowledge units.

For IDF-based features, we first combine the respective sub-documents $C^i_x$ of all knowledge units in a dataset as a corpus $C$. The IDF value of a word $w$ in the corpus $C$ is then computed, denoted as $idf(w)$. We measure the IDF-based similarity between the sub-documents $C^i_x$ of the knowledge unit $KU_x$ and the sub-document $C^j_y$ of the knowledge unit $KU_y$ as:

$$Sim_{IDF}(C^i_x, C^j_y) = \sum_{w \in C^i_x \cap C^j_y} idf(w)$$

i.e., the sum of the IDF value of the common words of $C^i_x$ and $C^j_y$ in the corpus $C$. As $1 \leq i, j, k \leq 3$, we obtain $3 \times 3 \times 3 \times 27$ IDF-based features for a pair of knowledge units.

4.1.2 Feature Extraction with Word Embeddings

The semantic information of a piece of text can be captured by taking the mean of the word embeddings of the words comprising the text [15]. Following this treatment, we compute the mean of the word embeddings of the words in a knowledge unit $KU_x$ (including question title plus question body plus body of all answers) as the word embedding of the knowledge unit, i.e.,

$$wv(KU_x) = \frac{1}{n} \sum_{w \in B_x} wv(w)$$

where $B_x$ is the bag of words of the $KU_x$ and $n$ is the size of $B_x$. A pair of knowledge units $KU_x$ and $KU_y$ is then represented as the mean of the two knowledge-unit word embeddings, i.e.,

$$wv(KU_x, KU_y) = \frac{1}{2}(wv(KU_x) + wv(KU_y))$$

which will be used as input to the SVM classifier. In this work, we use 200-dimensional word vector.

4.1.3 Multiclass SVM Classifier

We develop a multiclass SVM classifier to predict the relatedness of the two knowledge units. Based on the feature vector used (TF-IDF based or word embedding based as described above), we have two baselines: Baseline1 (TF-IDF+SVM) and Baseline2 (WordEmbed+SVM). The two baseline SVM classifiers are trained using the same dataset as that for training our approach. As our task is a multiclass classification problem, we use RBF kernel [34] in the SVM model. We set the $\gamma$ parameter of the SVM to $1/k$, $k$ denotes the dimensionality of the feature vector, i.e., $1/36$ for the TF-IDF+SVM baseline and $1/200$ for the WordEmbed+SVM baseline. We use grid search [12] to optimize the SVM parameters.

4.2 Dataset

Our experimental data is from Stack Overflow data dump of March 2010[1].

**Word Embedding Corpora.** From the posts table, we randomly select 100,000 knowledge units tagged with "java" as the word embedding corpora. Note that our corpora contains not only questions but also question answers.

**Training and test linkable knowledge units.** From the postlinks table, we randomly select in total 6,400 pairs of knowledge units that are tagged with "java", i.e., 1,600 pairs
for each type of relatedness defined in Section 3.1 (Duplicate, Direct Link, Indirect Link and Isolated). These 6,400 pairs of knowledge units are used as training data. For test data, we select in total 1,600 pairs of knowledge units that are tagged with “java”, i.e., 400 pairs for each type of relatedness. User-created links are considered as ground truth label for the semantic relatedness of the selected pairs of knowledge units. The knowledge units in the test data do not overlap with those in the training data.

4.3 Evaluation Metrics

We represent the multiclass classification results as a $K \times K$ matrix $A$, where $K$ is the number of classes (in our work $K = 4$). The rows represent the ground truth labels and the columns represent the predicted labels. The value of the $A_{ij}$ is the number of times that a pair of knowledge units with the ground truth label $L_i$ that is classified as the label $L_j$. Therefore, the value of $A_{ij}$ is the number of correct classification for the label $L_i$, $\sum_{1 \leq j \leq K} A_{ij}$ is the number of the ground truth label $L_i$ in a dataset, $\sum_{1 \leq i \leq K} A_{ij}$ is the number of predicted label $L_j$ in a dataset, and $\sum_{i} \sum_{j} A_{ij}$ is the number of knowledge-unit pairs in a dataset.

Precision, recall, and F1-scores as the evaluation metrics are standard and widely used to evaluate the effectiveness of a prediction technique [37, 38, 39, 40, 42, 43, 51]. Thus, we use them to compare our approach with the two baselines:

**Accuracy** is defined as the proportion of numbers of correct classification in a dataset, i.e.,

$$\text{Accuracy} = \frac{\sum_{i} A_{ii}}{\sum_{i} \sum_{j} A_{ij}}$$

**Precision** for a class $j$ is the proportion of knowledge-unit pairs correctly classified as the class $j$ among all pairs classified as the class $j$. Precision for all classes is the mean of the precision for each class.

$$\text{Precision}_j = \frac{A_{jj}}{\sum_{1 \leq i \leq K} A_{ij}}$$

**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 for all classes is the mean of the recall for each class.

$$\text{Recall}_i = \frac{A_{ii}}{\sum_{1 \leq j \leq K} A_{ij}}$$

**F1-score** for a class $i$ is a harmonic mean of precision and recall for that class. F1-score for all classes is the mean of the F1-score for each class.

$$F1_i = \frac{2 \times \text{Precision}_i \times \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}$$

F1-score evaluates if an increase in precision (or recall) outweighs a loss in recall (or precision). As there is often a trade off between precision and recall, F1-score is usually used as the main evaluation metric in many software engineering papers [23, 48]. In this paper, we also choose F1-score as the main evaluation metric.

4.4 Research Questions and Findings

In our experiment, we are interested in the following two research questions:

**Table 4: Accuracy Comparison**

<table>
<thead>
<tr>
<th>Approach</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline1 TF-IDF + SVM</td>
<td>0.625</td>
</tr>
<tr>
<td>Baseline2 Word Embedding + SVM</td>
<td>0.660</td>
</tr>
<tr>
<td>Our Approach Word Embedding + CNN</td>
<td>0.841</td>
</tr>
</tbody>
</table>

4.4.1 Overall Comparison

**RQ1:** How much improvement can our approach achieve over the baseline approaches?

**Motivation.** Our approach uses deep learning techniques (word embeddings and CNN-derived features) to quantify semantic relatedness between two pieces of software engineering text, which is very different from the baseline approaches using traditional TF-IDF word representations and human-engineered features. Answer to this research question will shed light on whether and to what extent deep learning techniques can improve the results of the multiclass classification task on software engineering text.

**Approach.** We apply our approach and the baseline approaches to the test data, i.e., 1,600 pairs of linkable knowledge units. We compare the accuracy, precision, recall, and F1-score metrics of different approaches.

**Result.** Table 4 and Table 5 present the results. We can see that the accuracy of our approach outperforms the Baseline1 and Baseline2 by 34.6% and 25.7%, respectively. Overall (see the last column of Table 5), our approach outperforms the Baseline1 and Baseline2 in terms of F1-score by 36.5% and 28.2%, respectively. Similar improvement on overall precision and recall can be observed.

**Overall, our approach achieves the best performance in all the evaluated metrics by a substantial margin, compared with the two baseline approaches.**

4.4.2 Comparison by Different Classes

**RQ2:** How effective is our approach in predicting different classes of semantic relatedness, compared with the baseline approaches?

**Motivation.** Different from existing work on binary text classification, our approach is for multiclass text classification. Different classes of semantic relatedness may exhibit different word and sentence features, which may affect the effectiveness of different semantic similarity measures. We would like to investigate the advantages and disadvantages of our approach and the baseline approaches for predicting different classes of semantic relatedness.

**Approach.** We compare the precision, recall and F1-score for each class of relatedness (i.e., duplicate, direct link, indirect link, and isolated) by our approach and the two baseline approaches (see Table 5).

**Result.** The performance of our approach and the baseline approaches do vary for different classes of relatedness. Our approach performs the best for three of the four classes (i.e., duplicate, direct link and isolated) on all the evaluation metrics, while the Baseline2 performs the best for the indirect link class only on recall and F1-score. In fact, the Baseline2 achieves the extremely high recall (0.980) but low precision (lower than our approach) for the indirect link class. The performance variance of our approach for different classes is small (0.05-0.09 in terms of F1-score), compared with the performance variance of the two baseline methods for different classes (in terms of F1-score, 0.1-0.27 for the Baseline1 and 0.27-0.39 for the Baseline2).
4.4.3 Effects of Word Embeddings and CNN

RQ3: What is the impact of word embeddings and CNN on the performance improvement respectively?

**Motivation.** Word embeddings and CNN are two deep learning techniques our approach relies on. They help capture semantics at word-level and sentence/document-level respectively. Answer to this research question helps us understand the importance of different levels of semantics for the multiclass text classification task.

**Approach.** To understand the importance of word embedding, we compare the performance of the Baseline1 (i.e., TF-IDF+SVM) and the Baseline2 (i.e., WordEmbedding+SVM). To understand the importance of CNN, we compare the performance of the Baseline2 (i.e., WordEmbedding+SVM) and our approach (i.e., WordEmbedding+CNN).

**Result.** Overall, the Baseline2 outperforms the Baseline1 by a small margin (see Accuracy in Table 4 and the Overall Column in Table 5). This suggests that word embeddings can moderately improve the text classification performance compared with the traditional TF-IDF word representation. Overall, our approach outperforms the Baseline2 by a substantial margin. This suggests that CNN plays a more important role than word embeddings for improving the multiclass classification performance.

The performance of the Baseline2 is not always better than that of the Baseline1 for difference classes of relatedness. For some classes, the Baseline2 has better precision but worse recall than the Baseline1, and vice versa for other classes. In terms of F1-score, the two baselines are almost the same for duplicate and direct link classes, while the Baseline2 is significantly better than the Baseline1 for indirect link class, but significantly worse for isolated classes. This suggests that word-level semantics encoded in word embeddings may not be appropriate for determining all classes of relatedness. For isolated knowledge units, traditional TF-IDF representation which is sensitive to lexical gap performs better than word embeddings. Taking the mean of word embeddings as knowledge-unit representations may blur the semantic distinction between the knowledge units, which helps the Baseline2 recall indirectly linkable knowledge units, but degrades the performance of the Baseline2 for isolated knowledge units.

In contrast, our approach consistently outperforms the Baseline2 on all the evaluation metrics for different classes of relatedness, except the recall and F1-score for indirectly linkable knowledge units. This suggests that word-level semantics are especially useful for determining semantic similarity of indirectly linkable knowledge units. As indirectly linkable knowledge units may not exhibit semantic similarity at sentence/document level, considering sentence/document-level semantics by the CNN could rule out false-negative links, which degrades its recall.

Both word embeddings and CNN help improve the performance of multiclass text classification, but CNN has a bigger impact than word embeddings. Word-level semantics are especially useful for predicting indirectly linkable knowledge units, while sentence/document-level semantics plays more significant role for predicting duplicate, directly linkable and isolated knowledge units.

4.4.4 Impact of Domain-Specific Word Embeddings

RQ4: How sensitive is our approach to word embeddings learned from different corpus?

**Motivation.** In this work, we predict linkable knowledge in developers’ online forum. Therefore, we learn word embeddings from Stack Overflow text which is representative of the ways people ask/answer questions and the vocabulary people use. We would like to investigate whether and to what extent word embeddings learned from a different corpora affect the performance of our approach. This will help us understand the importance of suitable corpus for a software-specific machine learning task.

**Approach.** We collect a corpus of Wikipedia web pages from the Wikipedia data dump. The number of sentences and the size of the vocabulary of the Wikipedia corpus is comparable to that of the corpus of the 100,000 knowledge units from Stack Overflow. We learn word embeddings from the Wikipedia corpus and use it to train the CNN subsequently. We compare the performance of the two CNNs, one trained with Stack Overflow word embeddings, and the other trained with Wikipedia word embeddings.

<table>
<thead>
<tr>
<th>Domain-specific Corpus</th>
<th>General Corpus</th>
<th>From Wikipedia</th>
<th>From Stack Overflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.784</td>
<td>0.790</td>
<td>0.776</td>
</tr>
<tr>
<td>Precision</td>
<td>0.831</td>
<td>0.847</td>
<td>0.842</td>
</tr>
<tr>
<td>Recall</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F1-Score</td>
<td>0.841</td>
<td>0.847</td>
<td>0.842</td>
</tr>
</tbody>
</table>

**Result.** Table 6 presents the accuracy, the overall precision, recall and F1-score of our approach with domain-specific word embeddings (i.e., Stack Overflow) versus general word embeddings (i.e., Wikipedia). General word embeddings degrade the performance of our approach, compared to the domain-specific word embeddings.

https://dumps.wikimedia.org/enwiki/latest/
pared with domain-specific word embeddings. However, the
degrade is moderate, in terms of accuracy, precision, recall
and F1-score by 7%, 5%, 7% and 7%, respectively. Furthermore,
although the performance of our approach degrades
with general word embeddings, it still outperforms the two
baseline methods (at least 11% on all the evaluation
metrics).

Our approach demonstrates certain level of reliability even
with word embeddings learned from a general corpus that is
different completely from Stack Overflow discussions. How-
ever, suitable domain-specific word embeddings leads to bet-
ter performance.

4.4.5 Training Cost

RQ5: What is the time cost for training the underlying
deep learning models?

Motivation. The underlying word embeddings and CNN
models need to be trained before they can be used for pred-
tiction task. Training of word embeddings and the CNN
model is done only once offline. After model training, using
the model to predict the relatedness of a pair of knowledge
units takes negligible time. Understanding the training time
cost helps us understand the practicality of our approach.

Approach. We record the start time and the end time of
the program execution to obtain the training time cost of the
word embeddings and the CNN model. The experimental
environment is an Intel(R) Core(TM) i7 2.5 GHz PC with
16GB RAM running Windows7 OS (64-bit).

Result. For learning word embeddings model, the 100,000
knowledge units contain 23,759,119 words (as bags of words),
and the vocabulary size (i.e., the number of unique words)
is 434,836. It takes about 15 minutes to analyze the text of
these 100,000 knowledge units to learn the word embeddings.
For training the CNN model, it takes about 14 hours for the
CNN model to achieve the loss convergence ($< e^{-3}$).

Training of the word embeddings model and the CNN model
can be done efficiently offline and only need to be done once.
Our approach is practical for large dataset.

5. DISCUSSION

This section presents the qualitative analysis of some ex-
amples to illustrate the capability of our approach and dis-
cusses threats to validity of our experiments.

5.1 Qualitative Analysis

Table 1 presents one example for each class of linkable
knowledge unit and the classification results by our approach
and the two baselines. The first example is a pair of dupli-
cate questions. However, the two questions do not have
many words in common. The two baseline methods classify
them as indirectly linkable knowledge units. In contrast, our
approach can capture semantic similarity between terms like
“standard input / output” and “System.out.println()”). This
helps our approach classify the two questions as duplicate.

In the second example, the two knowledge units contain
directly relevant knowledge, i.e., the “GUI repaint() prob-
lem” is directly relevant to “text not displaying correctly”
problem. Unfortunately, the two knowledge units share few
words in common, which makes the baseline methods classify
them as isolated. Our approach can capture the semantic
relatedness between the two technical problems, and thus
correctly classify the two knowledge units as direct link.

In the third example, the two indirectly linked questions
share some common words. Based on these overlap words,
the Baseline1 which essentially relies on lexical similarity
of overlapping words classifies the two questions as duplic-
ate. The Baseline2 which considers word-level semantics
by word embeddings makes a better judgment, classifying
them as isolated. Our approach makes the most accurate
prediction by taking into account not only word-level but
also document-level semantics. Similarly, in the fourth ex-
ample, the Baseline1 makes the least appropriate prediction
based on overlapping words, the Baseline2 makes a better
judgment based on word-level semantics, and our approach
makes the most accurate prediction.

5.2 Error Analysis

We also analyze the cases in which our approach makes the
wrong prediction. We find that many cases that fail our ap-
proach involve knowledge units whose essential information
is presented as a code snippet3 or an image4. In this work,
we remove code snippets and images during preprocessing.
In the future, we could incorporate code snippets and images
into our approach. Incorporating image semantics into our
approach would be relatively easy because CNN is originally
invented for image classification. Incorporating code seman-
tics into our approach could be a challenging task. A recent
work by Mou et al. [22] proposes to use CNN to encode the
program ASTs for program analysis tasks. However, their
approach is not directly applicable to code snippets in Q&A
discussion, which is usually incomplete and cannot be com-
plied. We plan to tackle this challenge as our future work.

5.3 Threats to Validity

There are several threats that may potentially affect the
validity of our experiments. Threats to internal validity re-
late to errors in our experimental data and tool implement-
ation. We have double checked our experimental data and
tool implementation. We have also manually checked the
selected knowledge units in our dataset to ensure that they
are really tagged with “java” and have the right types of
knowledge-units links. Threats to external validity relate to
the generalizability of our results. In this study, we use a
medium-size training and test dataset (100,000 knowledge
units for word embedding learning, 6,400 pairs of knowl-
edge units for CNN training, and 1,600 pairs of knowledge
units for testing). This allows us to perform some manual
analysis to understand the capability and limitations of our
approach. In the future, we will reduce this threat by ex-
tending our approach to larger word embeddings corpus and
more knowledge-unit pairs for training and testing.

6. CONCLUSION AND FUTURE WORK

In this paper, we propose a novel deep-learning based ap-
proach for predicting multiclass semantically linkable knowl-
edge units in Stack Overflow. Our approach can predict four
types of semantic relatedness, duplicate link, indirect link, in-
direct link and isolated. At word level, our approach adopts

3For example, http://stackoverflow.com/questions/5085912/simplesdateformat-bug
4For example, http://stackoverflow.com/questions/4382178/android-sdk-installation-doesnt-find-jdk

\[ e^{-3} \]
the state-of-the-art distributed word representations (i.e., word embedding) to encode word semantics in dense low-dimensional real-valued vectors. At document level, we train a CNN to automatically learn the most informative word and sentence features for classifying the semantic relatedness between two knowledge units. The training of the CNN is guided by sufficient examples of different types of semantically linkable knowledge units. Due to the adoption of word-level and document-level semantics, our approach is robust to the lexical gap (i.e., sharing few words in common) between linkable knowledge units. Our experiments confirm the effectiveness and consistency of our approach for predicting multiclass semantically linkable knowledge units, compared with the well-designed baselines using TF-IDF word representations and human-crafted word and sentence/decors features. In the future, we will enhance our approach by incorporating image and code-snippet semantics into our framework. We will also develop automated tool to help developers search and explore different types of linkable knowledge in Stack Overflow.

**Acknowledgments.** This work was partially supported by Singapore MOE AcRF Tier-1 grant M4011165.020 and the NSFC Program (No.61572426), and National Key Technology R&D Program of the Ministry of Science and Technology of China under grant 2015BAH17F01.

### Table 7: Examples of Different Classes of Semantically Linkable Knowledge Units

<table>
<thead>
<tr>
<th>Duplicate</th>
<th>Question Id : 2103820</th>
<th>Question Id : 10561540</th>
<th>Our approach : Duplicate Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title: Java, JUnit - Capture the standard input / Output for use in a unit test</td>
<td>Body: I'm writing integration tests using JUnit to automate the testing of a console based application. /.../ I only need to have the test execute and verify the the output is what is expected given the input. /.../</td>
<td>In JUnit testing is it possible to check the method System.out.println()</td>
<td>Baseline1: Indirect Link</td>
</tr>
<tr>
<td>Direct Link</td>
<td>Question Id : 1775858</td>
<td>Question Id : 369823</td>
<td>Our approach : Direct Link</td>
</tr>
<tr>
<td>Title: Text in Label not displaying correctly with setText method</td>
<td>Body: I'm trying to set the text in a label dynamically by calling the setText method whenever a button is clicked. /.../ that are passed to the setText method aren't visible on the screen when the submit button is clicked until I click on the window and drag to resize it. /.../ On a PC, the strings are are visible but obscured until I resize the window. /.../ On the Mac, the strings appear as intended, however, they're obscured until the window is resized. What am I missing?</td>
<td>Java GUI repaint() problem?</td>
<td>Baseline1 : Isolated</td>
</tr>
<tr>
<td>Indirect Link</td>
<td>Question Id : 465099</td>
<td>Question Id : 6973820</td>
<td>Our approach : Indirect Link</td>
</tr>
<tr>
<td>Title: Best way to build a Plugin system with Java</td>
<td>Body: How would you implement a Plugin-system for your Java application? Is it possible to have an easy to use (for the developer) system which achieves the following: Users put their plugins into a subdirectory of the app. The Plugin can provide a configuration screen If you use a framework, is the license compatible with commercial development?</td>
<td>Base pattern for web app</td>
<td>Baseline1 : Duplicate Link</td>
</tr>
<tr>
<td>Isolated</td>
<td>Question Id : 2438387</td>
<td>Question Id : 644814</td>
<td>Our approach : Isolated</td>
</tr>
<tr>
<td>Title: Hibernate - How to store java.net.URL into a database through Hibernate</td>
<td>Body: I have a field URL countryURL, in a Country class. I want to store its data into a COUNTRY table in a database through Hibernate. Which Hibernate type I should use in the hibernate mapping file /.../ I is not accepting string and text type.</td>
<td>Does Hibernate EntityManager include Core?</td>
<td>Baseline1 : Direct Link</td>
</tr>
<tr>
<td></td>
<td>Body: I'm using Hibernate's implementation of JPA. /.../ My question is, does Hibernate EntityManager depend on or use the Hibernate Core code? /.../ I want to know if EntityManager uses Core and thus has this fix.</td>
<td>Hibernate depend on or include Core?</td>
<td>Baseline2 : Indirect Link</td>
</tr>
</tbody>
</table>
7. REFERENCES


