Software-Specific Synonyms Discovery

Towhidur Rahman Bhuiyan

Technische Universität Berlin

t.bhuiyan@campus.tu-berlin.de

Abstract. *An enormous amount of programming knowledge has appeared in the discussions on social platforms like Stack Overflow, GitHub, CodeProject, etc. in the form of natural language text. This programming knowledge can be used as a foundation for different software engineering tasks by applying various Natural language processing (NLP) techniques. Precisely measuring the similarity of words is important in representing and comparing documents, and it improves the results of many natural language processing (NLP) tasks. The WordNet, a lexical database that contains relationships between many pairs of words and the NLP community has proposed various measurements based on WordNet. However, WordNet does not contain software-specific words because this is a general-purpose resource. Also, the meanings of words in WordNet are usually different than when they are used in the software engineering context. In this work, we proposed an automatic approach that builds a software-specific dictionary by leveraging the posts of StackOverflow. Finally, we evaluate the coverage and accuracy of constructed dictionary against community approved lists of synonyms provided by Stack Overflow.*

Keywords. *Software-specific Dictionary, Natural Language Processing, Word Embedding, Synonyms, Word2Vec, Stack Overflow.*

1. Introduction

A basic natural language processing(NLP) problem is synonyms detection. In NLP, different words used to express the same meaning. That's why we need to compute the semantic distance between two words. For example, words such as a desktop, laptop, and tablet are more semantically similar than the words computer, book, and phone. The NLP community invested a good amount of time to figure out the similarity measurement problem. Humans can naturally measure the semantic distance of the word but it is much harder for the machines. But word similarity measurement helps to improve machine learning tasks, e.g., information retrieval, text categorization, machine translation, duplicate detection, program prediction, program summarization, etc.

WordNet is created by Prinston University to measure the similarity of words in the English language. It groups verbs, adjectives, nouns, and adverbs into cognitive synonym sets. And it is a general-purpose lexical database that used to capture the semantic distance between two words. But, it does not contain many software specific words because this is a general-purpose dictionary. The semantic meaning stored in WordNet is usually different. For instance, the word "Python" relates to snake in WordNet, rather than a Programming language. Same for "Java" also, it shows the result as an island located in Indonesia. Hence, the necessity of constructing software specific word similarity database for the software engineering community is a time demanding. [2]

In our approach, we propose an automatic way for building a software-specific synonyms dictionary named SE_Dictionary from informal software engineering text(e.g., discussions from Stack Overflow). In the next section, we will discuss the overview of the solution. The third section will discuss the detail of the solution. The fourth section will do different experiments and observations. Finally, this work will conclude with the summary of works and possible future work.

2. Solution Overview

Our solution to the problem states a sequence of tasks to be performed to obtain the output. In the first step, we will collect Stack Overflow data which is provided by stack exchange from the Internet archive. In the second step, cleaning and transformation will begin. In the third step, we will apply the word embedding techniques such as word2vec to learn the term semantics. In the fourth step, we will extract semantically related terms. In the end, we will create a software-specific dictionary. The full architecture of the system is as follows:

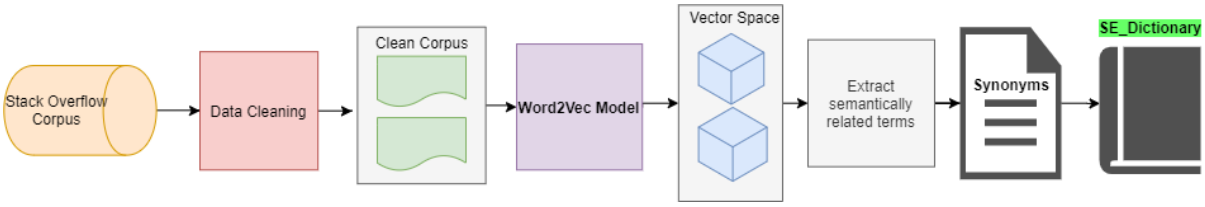

Figure 1: System Architecture.

3. Solution Approach

3.1 Data Collection

We have collected datasets from the Internet Archive where stack overflow dump data is archived. We used "Stack Overflow" as a software-specific corpus and downloaded 25 GB posts.xml of Stack overflow Posts. Each posts.xml contains Id, Body, Tags.

For each post:

- get Id
- get Tags
- get Code snippet from each body
- save them based on the post id in CSV

3.2. Pre-Processing Dataset

Since datasets are downloaded from websites, we need to perform the data cleaning steps, which are commonly used for pre-processing web content. So, before we dig into the next step, we will have to do some cleaning to break the things down into a format that the model can easily understand and produce a meaningful word embedding.

- Removal of HTML tags
  - long code snippets in <pre>, <code> in the posts
  - hyperlinks

- Removal of punctuations
  - symbols like !"#$%&\'()*+,-./:;<=>?@[\\]^_{|}~`

- Removal of stop words
  - stopwords are commonly occurring words like 'the', 'a' and so on.

- Tokenization
  - it takes the corpus and splits it into “tokens” based on a specified pattern using Regular Expressions aka RegEx. The pattern we chose to use this time (r”\w+”)

3.3. Word Embedding

Word embedding is one of the most important techniques in natural language processing(NLP), where words are mapped to vectors of real numbers. It is a Collective term for models that learned to map a set of words or phrases in vocabulary to vectors of numerical values. Word Embedding is really all about improving the ability of the Neural network. It is capable of capturing the meaning of a word in a corpus, semantic and syntactic similarity, relation with other words. Popular state-of-the-earth word embedding models in use today:

- Word2Vec (by Google)
- GloVe (by Stanford)
- FastText (by Facebook)

Word2Vec: One of the most popular techniques to learn word embeddings using a two-layer neural network is Word2Vec. It takes text corpus as input and produces a set of vectors as output. Word embedding using word2vec makes natural language computer-readable, then we can implement different mathematical operations on words that used to detect their similarities. A well-trained set of word vectors will place similar words close to each other in the vector spaces. For example, the words HTML, CSS, and Web might cluster in one corner, while Database, SQL and Oracle cluster together in another. [5]

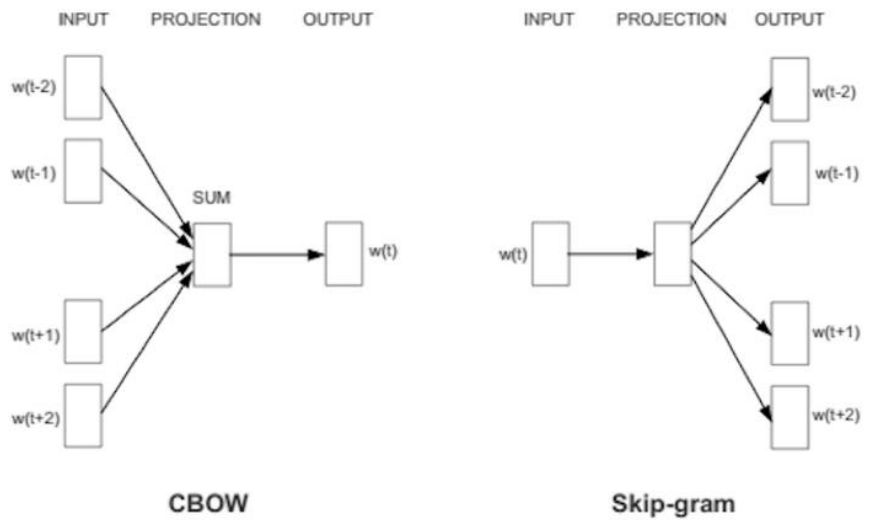

Figure 2: Architecture of CBOW and Skip-gram

Word2Vec has two main training algorithms, one is the continuous bag of words(CBOW), another is called skip-gram. The main difference between these two methods is that CBOW is using context to predict a target word while skip-gram is using a word to predict a target context. Generally, the skip-

gram method can have a better performance compared with the CBOW method because it can capture two semantics for a single word.

Robust Word2Vec Models with Gensim:

Gensim is an open-source python library for natural language processing(NLP) and it was developed and is maintained by the Czech natural language processing researcher Radim Řehůřek. Gensim library will enable us to develop word embeddings by training our own word2vec models on a custom corpus either with CBOW of skip-grams. After that, we can perform different similarity measurements. There are three main building blocks of Word2Vec. [6]

- Vocabulary Builder.
- Context builder.
- Neural network with two layers.

Vocabulary Builder: The basic building block of the Word2Vec model is the vocabulary builder. It takes the raw data in the form of sentences and extracts unique words from them to build the vocabulary of the system.

Context builder: This is the next level to convert words to vector. The output of the Vocabulary builder is the input of the Context builder. The output vocabulary object which contains the index and count. It takes all the words in the range of the context window rather than any particular word. The context window is the number of words that have to be considered while taking in the input. Generally, three to ten words are considered as standard.

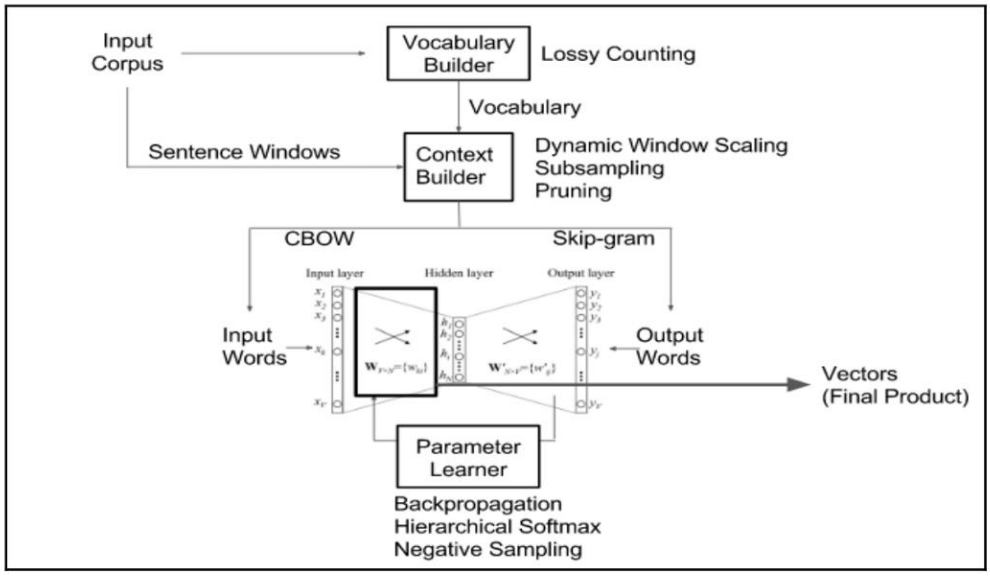

Figure 3: Word2Vec architecture.

Neural network with two layers: Word2vec uses the neural network for training. The output of the Context builder is the input of the Neural network. The context builder forms the word pairings, center and the words from the context window, which are fed to the neural network now. They are the followings:

- There is one input layer that has as many neurons as there are words in the vocabulary for training.
- The second layer is the hidden layer, layer size in terms of neurons is the dimensionality of the resulting word vectors.

- The third and final layer is the output layer which has the same number of neurons as the input layer.

Gensim Word2Vec Model Training:

We can train the genism word2vec model with our own custom corpus as following:

```
model = Word2Vec(sent, min_count=5, size= 100, workers=3, window =5, sg = 0)
```

- **size:** The number of dimensions of the embeddings and the default is 100.
- **window:** The maximum distance between a target word and words around the target word. The default window is 5.
- **min_count:** The minimum count of words to consider when training the model; The default for min_count is 5.
- **workers:** The number of partitions during training and the default workers are 3.
- **sg:** The training algorithm, either CBOW(0) or skip-gram (1). The default training algorithm is CBOW.

3.4. Extracting Semantically Related Terms

Word similarity has to determine how 'close' two pieces of the word are both in surface closeness (lexical similarity) and meaning (semantic similarity). In this section, we will go through two popular similarity measurement techniques. One is Euclidean similarity and another is Cosine similarity.

Euclidean Distance: Comparing the shortest distance among two words. It uses Pythagorean Theorem.

Cosine Similarity: Compute cosine angle between the two words to estimate how similar the words are. It calculates similarity by measuring the cosine of the angle between two vectors. The range of scores is 0 to 1. If the score is 1, it means that they are the same in orientation (not magnitude). [7]

For our case, we choose cosine similarity because euclidean similarity cannot work well for the high-dimensional word vectors. And euclidean similarity will increase as the number of dimensions increases. On the other hand, the cosine similarity is advantageous because even if the two similar words are far apart by the euclidean distance (due to the size of the word), chances are they may still be oriented closer together. The smaller the angle, the higher the cosine similarity.

4. Experiments

Experimental Setup:

Hardware: Intel Core i7 8th generation 3.60 GHz with 16GB RAM and 512 SSD, OS Linux distribution.

Software: Jupyter Notebook, Pandas, Gensim, Python 3.8, NLTK, Spacy.

Experiment 1:

Goal: The primary focus of our experiment is to build up a synonyms dictionary to determine the similarity between the words from the corpus.

Approach:

Dataset: We load our dataset into Pandas Dataframe. For this experiment, we consider 100 thousand rows from Dataframe. (nrows = 100K)

Pre-Processing: We applied all the pre-processing techniques mentioned in the solution approach section.

Word Embedding with Gensim Word2Vec: Now, we can train the genism word2vec model as mentioned in the approach by changing the parameter value.

> **size:** The number of dimensions of the embeddings is 200.
> **window:** The maximum distance between a target word and words around the target word is 3.
> **min_count:** The minimum count of words to consider when training the model is 2.
> **workers:** The number of partitions during training and the default workers are 2.
> **sg:** The training algorithm is skip-gram 1.

Extracting semantically related terms: Now we can use Word2vec to compute the similarity between two words by invoking the model.most_similarity( ) and passing in the relevant words. Cosine similarity measurement used here for extracting semantically related terms.

Then, we create a dictionary file and save all the semantically related terms in **SE_Dictionary**.

Evaluation:

DataSet: Our approach SE_Dictionary identifies 24,337 synonym pairs.

Ground Truth: Stack Overflow community maintains a list of tag synonym pairs in a collaborative way. Each pair has a synonym tag and a master tag. 3,663 synonym pairs.

Baselines: To demonstrate the effectiveness of the approach, compare it with state-of-the-art baselines WordNet. 117 000 synonym pairs.

Method	**Covered Synonym**	**Correct Synonym**	**Accuracy**
SE_Dictionary	1493 (40.7%)	913	61.16%
WordNet	828 (22.6%)	281	33.93 %

Experiment 2:

Our second experiment's goal, approach, and evaluation methodologies are the same as the first one. The difference is here we consider more rows and changed the value of the parameter in the Gensim Word2Vec training model.

Dataset: We consider 500 thousand rows from Dataframe. (nrows = 500K)

Gensim Word2Vec Model:

> **size:** The number of dimensions of the embeddings is 300.
> **window:** The maximum distance between a target word and words around the target word is 4.
> **min_count:** The minimum count of words to consider when training the model is 1.
> **workers:** The number of partitions during training and the default workers are 1.
> **sg:** The training algorithm is skip-gram 1.

Evaluation: Our approach SE_Dictionary identifies 54,001 synonyms pairs and accuracy is 67.4 %

From the above experiments, we understand that if we consider more data for training in the Gensim Word2Vec model, it consistently improving the Synonyms coverage of our dictionary(**SE_Dictionary**)

as well as accuracy. However, because of our hardware limitations, we couldn't able to perform more experiments.

5. Related Work

One of the fundamental natural language processing (NLP) tasks is measuring the similarity between two words. Various approaches have proposed by many papers to measure this similarity. Most of the popular techniques leverage a lexical database (e.g, WordNet). Meng Qu, Xiang Ren works also used a lexical database for automatic synonym discovery in text corpora. But our work is different from theirs because we use the software-specific domain rather than a general domain. [10]

Software developers often use abbreviations and domain-specific terms during software development that must be typed in source code and documentation. This is a challenge to the effectiveness of NLP-based techniques. Previous works have looked at string frequencies from source code, string edit distance, word co-occurrence data, or a combination of various techniques. These approaches inspire the design of lexical rules and transformations in our approach. [1]

However, rather than analyzing code, we analyze software-specific informal discussions on social platforms (e.g., StackOverflow). Because a broader range of programming knowledge is covered in the discussion on social platforms rather than traditional software text that all the previous work has focused on. We also used a different algorithm to train the datasets. In this work, we generate semantically related words.

6. Conclusion

In this work, we proposed a technique that automatically constructs a dictionary called **SE_Dictionary** that stores the similarity of words used in the software engineering context. The evaluation shows that SE_Dictionary covers a large set of software-specific terms and synonyms with moderate accuracy.

In the future, the plan is to construct a larger dictionary by training it with more question-and-answer threads from StackOverflow and other resources that contain software-related document corpus. In fact, if we include more threads from StackOverflow, the runtime would not dramatically increase. Furthermore, we will try FastText for word embedding because FastText has the ability to achieve good performance for word representations by making use of character-level representations. Our additional plan is to allow open access to an expanded SE_Dictionary as a web service. Finally, we will also extend our study into other domains for constructing a domain-specific dictionary like sports or medicine.

References

[1] Z. X. a. X. W. Chunyang Chen, "Unsupervised software-specific Morphological," in *IEEE/ACM 39th International Conference on Software Engineering*, 2017.

[2] D. L. J. L. Y Tian, "Automated construction of a software-specific word similarity database," in *Software Evolution Week-IEEE*, 2014.

[3] I. Stack Exchange, "Internet Archive," [Online]. Available: https://archive.org/details/stackexchange.

[4] R. Koenig, "Medium," [Online]. Available: https://towardsdatascience.com/nlp-for-beginners-cleaning-preprocessing-text-data-ae8e306bef0f.

[5] Z. Li, "Medium," [Online]. Available: https://towardsdatascience.com/a-beginners-guide-to-word-embedding-with-gensim-word2vec-model-5970fa56cc92.

[6] Viswash, "Medium," [Online]. Available: https://medium.com/@vishwasbhanawat/the-architecture-of-word2vec-78659ceb6638.

[7] E. Ma, "Medium," [Online]. Available: https://towardsdatascience.com/3-basic-distance-measurement-in-text-mining-5852becff1d7.

[8] K. Ganesan, "Build Beautiful NLP Applications.," [Online]. Available: http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.Xlqj7qhKiUm.

[9] S. Agarwal, "Medium," [Online]. Available: https://towardsdatascience.com/what-the-heck-is-word-embedding-b30f67f01c81.

[10] X. R. J. H. M Qu, "Automatic synonym discovery with knowledge bases," in *Proceedings of the 23rd ACM SIGKDD- dl.acm.org*, 2017.