Word Embeddings for the Software Engineering Domain

Vasiliki Efstathiou, Christos Chatzilenas, Diomidis Spinellis
{vefstathiou,xatzilenas16,dds}@aueb.gr
Athens University of Economics and Business
Athens, Greece

ABSTRACT

The software development process produces vast amounts of textual data expressed in natural language. Outcomes from the natural language processing community have been adapted in software engineering research for leveraging this rich textual information; these include methods and readily available tools, often furnished with pre–trained models. State of the art pre–trained models however, capture general, common sense knowledge, with limited value when it comes to handling data specific to a specialized domain. There is currently a lack of domain-specific pre–trained models that would further enhance the processing of natural language artefacts related to software engineering. To this end, we release a word2vec model trained over 15GB of textual data from Stack Overflow posts. We illustrate how the model disambiguates polysemous words by interpreting them within their software engineering context. In addition, we present examples of fine-grained semantics captured by the model, that imply transferability of these results to diverse, targeted information retrieval tasks in software engineering and motivate for further reuse of the model.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**; *Information extraction*; *Lexical semantics*; • **Software and its engineering**;

KEYWORDS

Natural Language Processing, Skip-gram, word2vec, Stack Overflow

ACM Reference Format:
Vasiliki Efstathiou, Christos Chatzilenas, Diomidis Spinellis. 2018. Word Embeddings for the Software Engineering Domain. In *MSR '18: MSR '18: 15th International Conference on Mining Software Repositories , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3196398.3196448

1 INTRODUCTION

Textual artefacts in software projects have always offered opportunities for a multitude of information retrieval tasks. Besides source code, text in natural language has been traditionally used for the analysis of formal documentation such as manuals, requirements and design specifications [19], [4], [26], among others. Nowadays, with the adoption of tool-assisted collaborative practices for processes such as code reviews and issue reports, as well as the emergence of online open source repositories such as GitHub, the textual footprint of software projects is constantly increasing and becoming more diverse, capturing a variety of aspects related to the development lifecycle. The study of this data is rising in popularity in empirical software engineering due to its potential for revealing social and technical aspects of collaborative projects.

Indicative software engineering processes that provide grounds for applying natural language processing techniques include issue reports, code reviews and source code documentation. Examples of research topics include the detection of duplicate bug reports [25], [30], the identification of textual characteristics reflected in useful code review comments [6], [23], the detection and classification of different types of self-admitted technical debt in source code comments [14], the assessment of the quality of source code comments [12] and sentiment analysis in developer communications [21]. Furthermore, implications of the aspects revealed though linguistic analysis are studied with the aim of explaining insights of the development process such as issue fixing time [20] productivity of developers [15], and the need for improving collaborative work conditions [8].

The increasing adoption of outcomes from the Natural Language Processing (NLP) community in software engineering research comes in line with recent advances in NLP which provide opportunities for efficiently processing large amounts of data. Besides methodological aspects, practical outcomes include NLP libraries that come as well-documented portable toolkits. These are usually furnished with readily available models, pre–trained over corpora of billions of words, that can be loaded and exploited for empowering various NLP tasks. One of the major drawbacks of using pre–trained models for assisting information retrieval tasks, is that they may be trained over data that carry meanings irrelevant to the domains they are aimed to be used with, thus leading to ambiguous results. The word 'kill' for instance, would be expected to carry different meanings in the contexts of news and software engineering. Negative results on applying off-the-shelf natural language processing tools and methods have started to emerge in software engineering research [11]. The need for carefully choosing the appropriate NLP toolkits [3], [29] and for employing calibration mechanisms for software engineering data has been highlighted in the literature [22], [1]. In addition, even though the lack of domain-specific software engineering data has been identified in related research, approaches for filling this gap tend towards the opposite extreme, focusing on the collection of data that are related to specific studies, [16] [21], thus leading to ad-hoc results that narrow down to task-specific rather than domain-specific solutions.

To this end, we propose a general-purpose model for capturing knowledge specific to the domain of software engineering. In terms

MSR '18, May 28–29, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5716-6/18/05…$15.00
https://doi.org/10.1145/3196398.3196448

of methodology, our approach capitalizes on the potential of continuous vector space models, specifically the word2vec model [17]. In terms of data, we exploit the abundance of related information available online on Stack Overflow (hereafter SO) posts. We discuss the representational power of the model at different levels of semantic abstraction. At a high level of abstraction, we demonstrate how the pre–trained model associates polysemous English words with their domain specific context. At a lower level of abstraction, we demonstrate the capacity of the model for capturing fine-grained information specific to particular software development processes. Finally, as a direct practical contribution, we release our model for further reuse by the community.

2 WORD2VEC FOR STACK OVERFLOW

In this section we briefly motivate our choice of method and dataset towards producing a general-purpose lexical model for representing software engineering knowledge. We describe the data and the training set up and discuss the resulting model.

2.1 Motivation

SO questions and answers encompass rich software engineering knowledge. Opportunities for exploiting this information have attracted substantial research focus, as attested also by the mining challenge of MSR 2013 where SO data provided the basis for the challenge. Indicative research directions for leveraging the information conveyed in SO posts include the identification of discussion topics [5], the effective detection of duplicate [2] or semantically linked [31] questions, usability of code snippets in SO posts [32] and social aspects [33], [27], among others.

Besides the above-mentioned targeted investigations of facets underlying SO posts, the resource of SO itself provides the grounds for general models of software engineering knowledge which may serve as a subsidiary resources of background information. The idea of leveraging the resource of SO in full for building a software-specific lexical database has been investigated in the past [28]. The proposed approach, however, follows the paradigm of traditional linguistics by aiming to replicate a WordNet-like [18] domain specific resource and suffers from scalability shortcomings. Based on recent advances in NLP [17], in this study we propose a continuous vector space model for representing software engineering knowledge. Word embeddings are based on the distributional hypothesis by Harris [10], which states that words that occur in the same contexts tend to have similar meanings. Along these lines, we trained word embeddings in a continuous vector space model using word2vec algorithm [17] over SO posts for capturing the related context. We argue that the model operates at an appropriate level of abstraction by being specific enough for capturing fine grained meanings of software engineering concepts; yet, it is broad enough for representing diverse topics of the domain and for disambiguating special meanings that polysemous English words carry in the field. In addition, besides technical terminology, it also involves informal language which captures developer jargon ubiquitous throughout the software development lifecycle. We therefore believe that it represents a broad spectrum of information related to processes, artefacts and people, which renders it applicable to a wide range of software engineering information retrieval tasks. We demonstrate this potential through examples in the next section. To the best of our knowledge this is the first study to train word embeddings on a full SO dump and release the pre–trained portable model [1] for further reuse by the community.

Progr. language	Full SO word2vec	Java-specific word2vec
Python	flake8	Ruby
C++	PVS-Studio	SuprressWarnings
C#	FoxCop	'*word 'C#' not in vocabulary*'
C	Cppcheck	compiler

Table 1: Programming language–static analysis tool analogy

2.2 Dataset and instrumentation

We used a SO dump [2] that spans a temporal interval from August 2008 to December 2017. We used the posts part where at preprocessing we removed code snippets and HTML tags. We also performed conservative punctuation removal, keeping symbols that often appear in programming commands, or symbols that are essential for keeping the meaning of a term tied together. For example, we kept symbols such as '+' and '#' which are essential for differentiating 'C', 'C++', and 'C#' from one another. Finally, we converted the whole text into lowercase in order to avoid information loss due to the informally written language of the resource, where different spellings and cases appear for referring to the same word. After initial preprocessing, 15GB of clean text remained, amounting to over 11 billion words. Further preprocessing involved removal of stop words, i.e., words that frequently appear in text and do not carry meaning associated with concrete concepts (typical examples of stop words include: *and, or, it, the, is, are, if, a, me*). After stop word removal, over 6 billion of words remained, which comprised the final data set that was fed into the training algorithm. We performed our experiment on a Dell PowerEdge machine equipped with an Intel(R) Xeon(R) CPU clocking 2.80GHz with 4 cores and 12GB RAM running Debian 4.9.65-3. For training the model we used the open source Python library gensim [24] [9], which provides a wrapper for the training algorithms originally ported from the C package. [3] Gensim comes with a comprehensive documentation that makes it straightforward to use for non-NLP experts, thus encouraging adoption of our released word2vec model for further reuse.

We performed the training over 3 epochs, which took about 11 hours to complete. We trained the word2vec model with vector dimensions of 200 features and the window for checking word co-occurrences set to 5. We chose the dimensionality to be lower than the 300-feature state of the art Google news vectors since for our much smaller dataset with limited vocabulary, a lower dimension should be sufficient for capturing the necessary features. The resulting model consists of a vocabulary of 1,787,145 keywords.

2.3 Results

We discuss our results in the following three directions: (1) We provide examples that illustrate the representational power of the model for capturing abstract notions that need to disambiguate to their software-specific meaning (e.g. 'smell'). (2) We provide

[1] http://doi.org/10.5281/zenodo.1199620
[2] https://archive.org/download/stackexchange
[3] https://code.google.com/p/word2vec/

Keyword	Most similar in full SO word2vec	Most similar in Java-specific word2vec	Most similar in Google news word2vec
abort	interrupted, terminate, aborting, aborts, exit	interrupted, terminate, shutdown, disconnect, abnormally	abort, aborting, aborts, abort_fetuses, terminating_pregnancy
cookie	sessionid, coockie, aspxauth, session, coookie	certificate, URL, JSESSIONID, password, Cookies	cupcake, Trefoils, cookie_recipe, oatmeal_cookie, oatmeal_raisin_cookies
fork	forking, fork/exec, forks, /execve, vfork	task, forked, forking, upi1985, forks	pancake_turner, forking, ##mm_Marzocchi, wooden_skewer, ricer
smell	antipattern, smells, code-smell, spaghetti, smelling	valid, rife, messy, duplicated, smelly	odor, smelling, aroma, pungent_odor, pungent_smell
virus	malwarebytes, malware, mcafee, anti-malware, viruses	Netstat, Havok_, antivirus, BorlandC, rocksaw	avian_flu_virus, viruses, flu_virus, bird_flu_virus, swine_flu_virus

Table 2: Examples of polysemous words along with top 5 semantically related words in the full SO word2vec model, a small-scale Java-specific word2vec model [7] and Google news [17] word2vec model.

examples that illustrate the representational power of the model for capturing fine grained semantics specific to software engineering. (3) We provide examples that illustrate the representational power of the model for capturing finer grained semantics, specific to a Java-related information retrieval task.

As a reference for general, domain independent knowledge we use the state of the art Google news word2vec model released by Mikolov et al. [17]. As a reference for specialised domain-specific knowledge, we use a word2vec model released by Fu and Menzies [7], trained over a small fraction of randomly selected SO question-answer units that were tagged with the word 'Java'. We will refer to this model as Java-specific model hereafter. We illustrate how our model is general enough for capturing broad software engineering knowledge, yet sufficiently specific for capturing fine grained concepts.

Representing abstract notions in software engineering: Figurative use of non-technical English words is common in software Engineering. Table 2 shows an indicative list of such words. Both our SO word2vec model and the Java-specific word2vec model appear to perform well in disambiguating polysemous words.

Representing fine-grained notions in software engineering: Besides semantic relatedness, continuous vector space models are good in representing analogies, such as the famous '*king - man + woman = queen*' example. In the SO word2vec, we found some analogies such as the programing language – IDE analogy, and the programming language – static analysis tool analogy, among others. Given for instance, the example SonarQube - Java + Python, the model derived the word 'flake8'. Other instances of this analogy derived by the model are listed in Table 1, along with the corresponding results derived by the Java-specific word2vec. The latter imply that the Java-specific model did not perform very well in the analogy task; note that this model is limited to a vocabulary of 157,414 words which differs by one order of magnitude compared to the SO word2vec model. We did not investigate analogies in the Google news word2vec, but as a note, SonarQube did not even exist in the dictionary of the model.

Representing more fine grained task-specific notions in software engineering: The studies presented in references [7] and [31] are focused on Java-related issues; to this effect, the word2vec models they use are trained on Java-specific topics. We argue that our SO word2vec model can capture comparable degree of Java-related detail even though it is trained across a wide spectrum of data. Xu et al. [31] demonstrate that semantically related words such as JPanel, JButton, JFrame and JLabel which belong to a GUI component are close in the vector space, whereas, words such as jdk, jre, Ant, Maven appear close with one another but further away from the GUI component words. We found very similar results by querying our model. Due to space constraints we do not discuss the details here, but the model is openly available [4] and instructions for replicating results can be found in our repository. [5] Note that we could not replicate the two-dimensional representation of Figure 2, reference [31], as it probably does not correspond to a straightforward application of t-SNE dimension reduction on word embeddings. By performing standard t-SNE dimensionality reduction technique for visualizing the data [13] we produced scatter plots for the same set of keywords. We produced two versions: one in the vector space of the Java-specific word2vec model and one in the vector space of our SO word2vec model. The two comparable plots are depicted in Figure 1 where each colour corresponds to a group of words that are semantically close.

Our concluding remark here is that the SO word2vec model presented in this study appears to perfom well at different levels of detail. It disambiguates the metaphorical use of English words in the domain of software engineering, it captures fine grained relations of analogy in the field and a comparable level of detail with an ad-hoc model which has been trained on a custom data set.

3 DISCUSSION AND OPPORTUNITIES

Continuous vector space models have already been used in line with other information retrieval methods in software engineering. Rather than training custom, ad–hoc models, we propose the reuse of general purpose, domain specific models. Indicative grounds for application of such models include, among others, the recommendation of APIs and components for software reuse, clustering of repositories, evolutionary aspects of software artefacts and the identification of untagged topics of discussion.

4 CONCLUSIONS

We presented what we believe to be a generic model of software Engineering knowledge. The preliminary results in this study imply that the model adequately captures detailed relational structures, and a broad range of topics in the field. In future work we aim to empirically evaluate the model against human judgment and to practically assess it by using it as a subsidiary knowledge resource in various information retrieval tasks in software engineering.

[4] http://doi.org/10.5281/zenodo.1199620
[5] https://github.com/vefstathiou/SO_word2vec

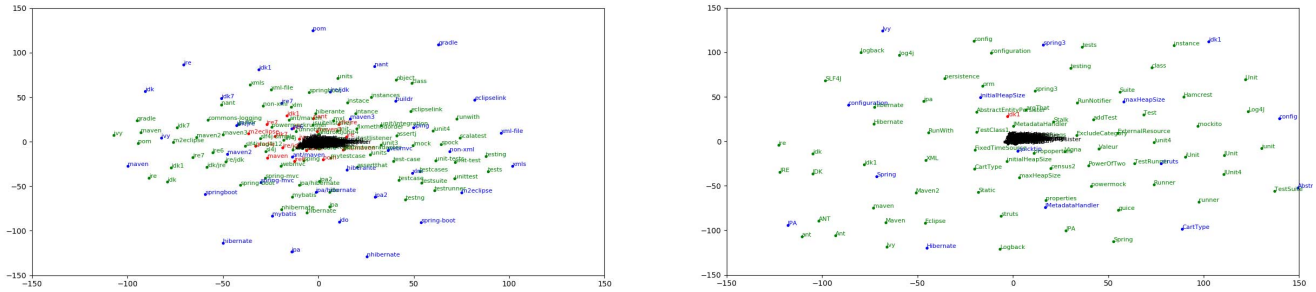

Figure 1: Word embeddings for a set of words related to Java. The plot on the left is the representation in the full SO word2vec model and the one on the right shows the representation of the same set of words in a word2vec model trained on Java-specific subset of SO.

ACKNOWLEDGMENTS

The project associated with this work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732223.

REFERENCES

[1] Amritanshu Agrawal, Wei Fu, and Tim Menzies. 2016. What is Wrong with Topic Modeling? (and How to Fix it Using Search-based Software Engineering). (2016). http://arxiv.org/abs/1608.08176

[2] Muhammad Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K Roy, and Kevin A Schneider. 2016. Mining Duplicate Questions of Stack Overflow. In *Mining Software Repositories (MSR), 2016 IEEE/ACM 13th Working Conference on*. IEEE, 402–412.

[3] Fouad Nasser A Al Omran and Christoph Treude. 2017. Choosing an NLP library for analyzing software documentation: a systematic literature review and a series of experiments. In *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE Press, 187–197.

[4] Vincenzo Ambriola and Vincenzo Gervasi. 1997. Processing natural language requirements. In *Automated Software Engineering, 1997. Proceedings., 12th IEEE International Conference*. IEEE, 36–45.

[5] Anton Barua, Stephen W Thomas, and Ahmed E Hassan. 2014. What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering* 19, 3 (2014), 619–654.

[6] Amiangshu Bosu, Michaela Greiler, and Christian Bird. 2015. Characteristics of useful code reviews: An empirical study at microsoft. In *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*. IEEE, 146–156.

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

[8] Daviti Gachechiladze, Filippo Lanubile, Nicole Novielli, and Alexander Serebrenik. 2017. Anger and its direction in collaborative software development. In *Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Results Track*. IEEE Press, 11–14.

[9] gensim 2016. Topic modeling for humans. https://radimrehurek.com/gensim/index.html. (2016). [last accessed 26-Jan-2018].

[10] Zellig S Harris. 1954. Distributional structure. *Word* 10, 2-3 (1954), 146–162.

[11] Robbert Jongeling, Proshanta Sarkar, Subhajit Datta, and Alexander Serebrenik. 2017. On negative results when using sentiment analysis tools for software engineering research. *Empirical Software Engineering* (2017), 1–42.

[12] Ninus Khamis, René Witte, and Juergen Rilling. 2010. Automatic Quality Assessment of Source Code Comments: The JavadocMiner.. In *NLDB*. Springer, 68–79.

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

[14] Everton Maldonado, Emad Shihab, and Nikolaos Tsantalis. 2017. Using natural language processing to automatically detect self-admitted technical debt. *IEEE Transactions on Software Engineering* (2017).

[15] Mika Mäntylä, Bram Adams, Giuseppe Destefanis, Daniel Graziotin, and Marco Ortu. 2016. Mining valence, arousal, and dominance: possibilities for detecting burnout and productivity?. In *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 247–258.

[16] Mika V Mäntylä, Nicole Novielli, Filippo Lanubile, Maëlick Claes, and Miikka Kuutila. 2017. Bootstrapping a lexicon for emotional arousal in software engineering. In *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE Press, 198–202.

[17] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. (2013).

[18] George A Miller. 1995. WordNet: a lexical database for English. *Commun. ACM* 38, 11 (1995), 39–41.

[19] J Natt och Dag, Björn Regnell, Vincenzo Gervasi, and Sjaak Brinkkemper. 2005. A linguistic-engineering approach to large-scale requirements management. *IEEE software* 22, 1 (2005), 32–39.

[20] Marco Ortu, Bram Adams, Giuseppe Destefanis, Parastou Tourani, Michele Marchesi, and Roberto Tonelli. 2015. Are bullies more productive?: empirical study of affectiveness vs. issue fixing time. In *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE Press, 303–313.

[21] Marco Ortu, Alessandro Murgia, Giuseppe Destefanis, Parastou Tourani, Roberto Tonelli, Michele Marchesi, and Bram Adams. 2016. The emotional side of software developers in JIRA. In *Mining Software Repositories (MSR), 2016 IEEE/ACM 13th Working Conference on*. IEEE, 480–483.

[22] Annibale Panichella, Bogdan Dit, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. 2013. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 522–531.

[23] Mohammad Masudur Rahman, Chanchal K Roy, and Raula G Kula. 2017. Predicting usefulness of code review comments using textual features and developer experience. In *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE Press, 215–226.

[24] Radim Řehůřek and Petr Sojka. 2010. Software framework for topic modelling with large corpora. In *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Citeseer.

[25] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. 2007. Detection of duplicate defect reports using natural language processing. In *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society, 499–510.

[26] Américo Sampaio, Ruzanna Chitchyan, Awais Rashid, and Paul Rayson. 2005. EA-Miner: a tool for automating aspect-oriented requirements identification. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. ACM, 352–355.

[27] Tanmay Sinha, Wei Wei, and Kathleen Carley. 2015. Modeling similarity in incentivized interaction: A longitudinal case study of stackoverflow. In *NIPS 2015 Workshop on Social and Information Networks, 29th Annual International Conference on Neural Information and Processing Systems*.

[28] Yuan Tian, David Lo, and Julia Lawall. 2014. Automated construction of a software-specific word similarity database. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*. IEEE, 44–53.

[29] Stefan Wagner. 2016. Natural language processing is no free lunch. In *Perspectives on Data Science for Software Engineering*. Elsevier, 175–179.

[30] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. 2008. An approach to detecting duplicate bug reports using natural language and execution information. In *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*. IEEE, 461–470.

[31] Bowen Xu, Deheng Ye, Zhenchang Xing, Xin Xia, Guibin Chen, and Shanping Li. 2016. 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*. ACM, 51–62.

[32] Di Yang, Aftab Hussain, and Cristina Videira Lopes. 2016. From query to usable code: An analysis of Stack Overflow code snippets. In *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 391–402.

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