Exploring Word Embedding Techniques to Improve Sentiment Analysis of Software Engineering Texts

Eeshita Biswas, K. Vijay-Shanker, Lori Pollock
Computer and Information Sciences
University of Delaware
Newark, DE 19716 United States
{biswas, vijay, pollock}@udel.edu

***Abstract*—Sentiment analysis (SA) of text-based software artifacts is increasingly used to extract information for various tasks including providing code suggestions, improving development team productivity, giving recommendations of software packages and libraries, and recommending comments on defects in source code, code quality, possibilities for improvement of applications. Studies of state-of-the-art sentiment analysis tools applied to software-related texts have shown varying results based on the techniques and training approaches. In this paper, we investigate the impact of two potential opportunities to improve the training for sentiment analysis of SE artifacts in the context of the use of neural networks customized using the Stack Overflow data developed by Lin et al.**

We customize the process of sentiment analysis to the software domain, using software domain-specific word embeddings learned from Stack Overflow (SO) posts, and study the impact of software domain-specific word embeddings on the performance of the sentiment analysis tool, as compared to generic word embeddings learned from Google News. We find that the word embeddings learned from the Google News data performs mostly similar and in some cases better than the word embeddings learned from SO posts. We also study the impact of two machine learning techniques, oversampling and undersampling of data, on the training of a sentiment classifier for handling small SE datasets with a skewed distribution. We find that oversampling alone, as well as the combination of oversampling and undersampling together, helps in improving the performance of a sentiment classifier.

***Index Terms*—Sentiment Analysis, Software Engineering, Word Embeddings**

I. Introduction

Software engineers often search online for code examples, libraries and packages to help them in their software development and maintenance activities. They search through code repositories, developer question and answer (Q&A) forums, blog posts and tutorials online. In developer communications such as developer Q&A forums and chat forums, they use the sentiments of contributors to guide them to good posed solutions. Sentiment Analysis (SA) is an opinion mining technique in natural language processing (NLP) that analyzes a unit of text to determine whether the opinion being expressed is positive, negative, or neutral. Researchers have been exploring sentiment analysis as a way to automatically distinguish between good and bad information in these software developer communications. Sentiment analysis of software engineering (SE) textual artifacts has been used to benefit software engineers by improving code suggestions and recommending better software packages and libraries [22], [10], [24], [21], [9]. Sentiment analysis of Q&A sites such as Stack Overflow (SO) has been used to recommend insightful comments on quality, deficiencies or possibilities for further improvement of source code [29] and to extract problematic API design features [37].

Given the many potential uses of sentiment analysis for SE, researchers have conducted studies of various off-the-shelf sentiment analysis tools applied to the SE domain [15], [26], [11]. Most of the sentiment analysis for SE involve feature-based machine learning (ML) techniques [7], [1] such as SVM. However, more recently, Lin et al. [19] used a neural network method using an off-the-shelf recursive neural network [31]. They, among others (e.g., [1, 14, 7, 4]) have considered the adaptation of off-the-shelf sentiment analysis tools to the software domain. Thus, they built a training set using text from Stack Overflow posts that were annotated with a sentiment score. Despite using a state-of-the-art tool for sentiment analysis and customizing this sentiment analysis tool by training on Stack Overflow data, they obtained results that were noted to be "unacceptable accuracy levels in classifying positive/negative opinions".

This paper investigates two additional issues in the context of the use of neural networks customized using the Stack Overflow data developed by Lin et al. [19], henceforth called the LinSOData. First, as noted by Lin et al. [19] and others, sentiment analysis data usually exhibit data imbalance with a preponderance of neutral sentiment text and relatively low amounts of positive/negative text. This phenomena is true for LinSOData as well. We investigate a common technique of using under- and over-sampling [18, 28, 5, 20] to address the imbalance issue. Second, while Lin et al. customized the SA tool to SE domain by using appropriate training data, we consider an additional form of customization to the domain. Lexical information is a fundamental source for successful sentiment analysis, and neural network methods often use word embedding as their representation for input words. There are well-known techniques such as Word2Vec [34] that take a large corpus of text as input and produce a dense high-dimensional vector representation for the words in the corpus. It has been suggested that these word embedding methods capture the semantics underlying the word usage. Our second question investigates additional domain adaptation in the form of word embedding information induced from Stack Overflow content.

Thus, specifically, we investigate the following two research questions:

1) How much is the imbalance in the dataset distribution a factor in the unsatisfactory results of sentiment analysis applied to SE artifacts?
2) How much does additional customization of the sentiment classifier to the software domain by using domain-specific word embedding impact the sentiment analysis process?

To answer research question 1 (RQ1), we developed a study to explore the impact of standard machine learning techniques involving oversampling and undersampling, to address the skewed distribution of the dataset in LinSOData [19]. While the imbalance issue has been brought up in several works, to our knowledge, the impact of sampling methods has not been addressed in SA for SE. It can be noted that Calefato et al. [4] achieve good performance by starting with a balanced dataset. To answer research question 2 (RQ2), we explored the use of software-specific word embeddings learned from Stack Overflow (SO) posts since word embeddings capture the semantics underlying the words and sentiment analysis approaches rely extensively on the words used while trying to assign a sentiment.

Our evaluation shows that using oversampling and undersampling increased the effectiveness of sentiment analysis on the same software engineering dataset used by Lin et al. [19] with recall up to 56% and 46% for negative and positive sentences respectively, which is an improvement on their results, which achieved the highest recall of 36.5% and 14.5% for negative and positive sentences, respectively. Unfortunately, our results indicate that customizing the sentiment classifier to the software domain using software-specific word embeddings does not surpass the results achieved by the generic word embeddings for most combinations of oversampled and undersampled data that we tested, except for one dataset that achieves a balanced distribution of all three classes of data through oversampling.

II. State of the Art

We start by surveying the several studies of the use of off-the-shelf sentiment analyzers without any customization for SE artifacts. Tourani et. al [35] used SentiStrength (a mainstream off-the-shelf SA tool) to extract sentiment information from the user and developer mailing lists of two projects (Tomcat and Ant) from the Apache software foundation, and achieved a precision of 29.56% and 13.1% for positive and negative sentences, respectively. Novielli et.al [23] discussed the challenges of employing SA techniques to assess the affective states (ranging from personality traits to emotions expressed in the Stack Overflow question, answer, and comments). Jongeling et. al [16] conducted a comparison of four widely used SA tools: SentiStrength, NLTK, Stanford CoreNLP, and AlchemyAPI and found none of them provides accurate predictions of expressed sentiment in the SE domain. One potential cause for the low effectiveness could be the training of the SA tools on non-SE related data such as news articles and movie reviews. Another possible cause can be the presence of an imbalance in most datasets with respect to positive, negative, and neutral opinions.

To address the unreliable performance of general purpose SA tools when applied to SE text, several domain-specific SA tools have been customized for SE text. Blaz and Becker [3] presented a technique to evaluate the sentiment contained in tickets for Information Technology (IT) support by creating a domain dictionary that contains terms with the sentiment in the IT context. Ahmed et al. developed SentiCR [1], a customized SA tool for code review comments from open source projects, which uses NLTK, TF-IDF features, and the following supervised learning algorithms - Adaptive Boosting, Decision Tree, Gradient Boosting Tree (GBT), Naive Bayes, Random Forest, Multilayer Perceptron, SVM with Stochastic Gradient Descent, and SVC. Islam and Zibran developed SentiStrength-SE [14], a customized version of SentiStrength by changing the configuration parameters and adding heuristic rules. They evaluated SentiStrength-SE on issue comments from JIRA issue tracking and found an improvement over SentiStrength. In another recent work, Islam and Zibran presented DEVA [13], a dictionary-based lexical approach for automatic detection of emotions in both valence and arousal space, using JIRA issue comments.

In addition to the Lin et al. [19] study which considered adapting an off-the-shelf sentiment analysis tool based on recursive neural network as discussed in the introduction section, Ding et al. built SentiSW [7], an entity-level SA tool using feature vectorizations and supervised classification algorithms (Random Forest, Bagging, GBT, Ridge Regression, Naive Bayes, SVC) on a gold set of issue comments from GitHub projects. They use TF-IDF and Doc2Vec model for feature extractions, training their Doc2Vec model on approximately 231k issue comments, getting the best results from GBT using TF-IDF. Calefato et al. built Senti4SD [4], a sentiment classifier trained and validated using a gold standard of Stack Overflow questions, answers, and comments that were manually annotated and had a balanced distribution of all three classes. Senti4SD exploits lexicon-based, keyword-based and semantic features based on word embeddings. They use the CBOW architecture for training the word embedding model and train Senti4SD using Support Vector Machines. With respect to their baseline SentiStrength, Senti4SD reduces the misclassifications of neutral and positive posts as negative. Novielli et al [25] conducted a comparison of SA tools: SentiStrength, SentiStrength-SE, Senti4SD, SentiCR on SE data and concluded that SE-specific customization provides a boost in accuracy with respect to their baseline represented by SentiStrength and reliable sentiment analysis in SE is possible, but the manual annotation of gold standards should be inspired by theoretical models. None of these works studied the impacts of data imbalance.

Two other studies included Senti4SD. Imtiaz et al. [11] also conducted a study on existing sentiment and politeness detection tools: SentiStrength, Alchemy, NLTK, Stanford CoreNLP, Senti4SD, SentiCR and Politeness Tool on a dataset of 589 manually annotated Github comments and found the existing tools to be unreliable and also inconsistency amongst human annotators in identifying sentiment and politeness in the developer discussions. Islam and Zibran conducted a comparative study specifically on the software domain-specific SA tools (SentiStrength-SE, Senti4SD, EmoTxt) [12] on three datasets from JIRA issue comments, Stack Overflow posts and code review comments, and find the accuracy of the tools to vary across datasets.

Within the machine learning community, different sampling methods have been proposed to address the problem of imbalanced training data in sentiment analysis. One such approach is oversampling of the minority classes by either duplication, or by creating synthetic examples (e.g., SMOTE [5]). Xu et al. [36] presented a method called WEC-Mote that uses word embeddings and SMOTE to produce a fully balanced dataset; their evaluations on two datasets show improvement in imbalanced sentiment and emotion classification for both English and Chinese data. Studies [18, 28] also show how undersampling of minority classes as well as combination of undersampling and oversampling [20] improves the classification performance on sentiment analysis.

III. Word Embeddings & Software-specific Word Embeddings

Sentiment analysis techniques rely extensively on the words that are being used (in a specific sentence or a phrase or the individual words themselves) while trying to assign a sentiment. Hence, the most obvious way for domain adaptation is to use an appropriate representation for the lexical information. In neural networks, all texts have to be provided as vector of numbers. Word embeddings transform a word into a vector that captures the semantics underlying the words. More specifically, the word embeddings place related words close to each other in the representation space. Thus, using word embeddings as input to neural models allows the neural models to generalize beyond the specific words found in the input sentence/documents while assigning the sentiment class.

Bengio et al. coined the term word embeddings in 2003 [2] and trained them in a neural language model. In 2008, Collobert and Weston established word embeddings as a useful tool for downstream tasks and introduced a neural network architecture that forms the foundation for many current approaches [6], being the first to show the usefulness of pre-trained word embeddings.

In 2013, Tomas Mikolov created Word2Vec [34], which creates word embeddings or vectors based on the distributional hypothesis (words that occur in the same contexts tend to have similar meanings). Word2Vec takes as input a large corpus of text (e.g. English Wikipedia, Google News) and produces a high-dimensional vector space and assigns a corresponding vector for each unique word in the corpus. The output of Word2Vec is a word embedding matrix that contains vectors for every distinct word in the training corpus. Word2vec can obtain the embeddings using continuous bag-of-words (CBOW) or skip-gram approaches.

A software-specific word embedding model was created by Efstathiou et al. [8] using Word2Vec with skip-gram. They used all the Stack Overflow posts from August 2008 to December 2017 to derive the word embeddings. They preprocessed the SO posts to remove code snippets and HTML tags, perform conservative punctuation removal and convert the text into lower case. Their resulting word embedding model has a dimension of 200 features and a vocabulary of 1,787,145 words.

IV. Sentiment Classifier for Our Study

To assess how imbalance in dataset distribution impacts the accuracy of sentiment analysis applied to SE artifacts, we applied sampling techniques just prior to training as indicated in Figure 2 using the LinSOData [19]. To investigate how using software-specific word embeddings could impact sentiment analysis in the SE domain, we needed to be able to provide software-specific as well as generic word embeddings as input to the sentiment analysis tool. We originally intended to use the sentiment analysis tool of Stanford CoreNLP as in Lin et al. [19]. However, we could not find a way to change the word embeddings of this sentiment analysis tool. Therefore we implemented our own neural network based sentiment classifier using recurrent neural network (RNN). With this implementation, we are able to change the word embeddings and thus assess the impact of using different word embeddings for software engineering texts.

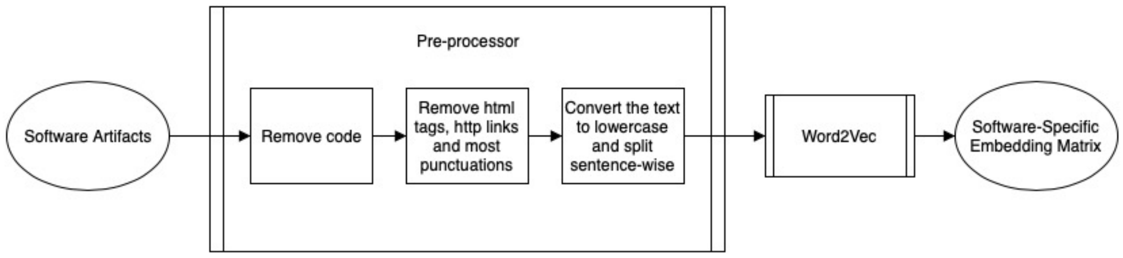
The following subsections describe our process to obtain the software-specific word embeddings and then the training of the sentiment classifier that we built and used for answering both RQ1 and RQ2.

A. Software-Specific Word Embeddings

For adaptation to the software domain, we used the word embeddings developed by Efstathiou et al. [8]. In the Efstathiou et al. [8] work, they remove all stop words from the SO posts, including words such as 'no' and 'not', before developing the word embeddings from the SO posts. However, the impact of eliminating stop words in the context of sentiment analysis is not so clear, since certain stop words, such as negations can be suggestive of sentiments. Also, eliminating all stop words from the sentences modifies the sentences themselves, bringing about change in the context of words, thereby affecting the word vector values.

With this in mind, we decided to experiment with two different software-specific word embeddings, one by Efstathiou et al. [8], with all stop words removed (hereafter SOwoStop word embeddings) and one with the inclusion of stop words that we create ourselves (hereafter SOwStop word embeddings).

To create software-specific word embeddings we followed the same overall process as Efstathiou et al. [8], training word embeddings using Word2Vec on a large dataset of Stack Overflow posts. Figure 1 shows the major steps of the process.



Pre-processor
Software Artifacts
Remove code
Remove html tags, http links and most punctuations
Convert the text to lowercase and split sentence-wise
Word2Vec
Software-Specific Embedding Matrix

Fig. 1. Creating a software-specific word embedding matrix as by Efstathiou et al.[8]

Like Efstathiou et al. [8], we used the software-related text from Stack Overflow (SO). We downloaded the SO data dump spanning from August 2008 to June 2018 and containing approximately 40 million posts from the Stack Exchange archive and used only the SO posts (questions and answers) for building the dataset.

Our pre-processor, written as a Python script using regular expressions and NLTK library first removes code segments and then removes HTML tags and HTTP links from the body of the posts. The pre-processor also removes most punctuations, keeping symbols such as '.', '?', '!', '+', '#', that often have a significant meaning in programming languages, for example, words such as 'C++', 'C#', etc, and symbols such as '.', '?', '!', that are used as delimiters. As the final step, the pre-processor then performs a sentence-wise splitting on the posts and converts the whole text into lowercase to avoid variations of the same word due to the use of different case, but does not remove any stop words from the text.

After preprocessing the SO post text, we used the open source Python library Gensim implementation of Word2Vec to create the word embeddings. We tokenized the sentences from the preprocessing and passed that list of tokenized sentences as input to Word2Vec. Word2Vec uses all of the tokens to build the vocabulary (a set of unique words). We trained the new set of word embeddings with the same parameters used by Efstathiou et al. [8]. Our resulting word embedding model has a dimension of 200 features and a vocabulary of 1,795,316 words.

B. Sentiment Classifier

To build the neural network architecture for our sentiment classifier, we have used RNN with LSTM (Long Short-Term Memory) as the RNN units. We decided to use RNN because sentiment analysis takes into consideration each word in a sentence and their sequence, and RNN takes into consideration this aspect of the data. Our RNN based classifier follows the design from the tutorial by Deshpande [27]. We used Tensorflow for our implementation.

Figure 2 provides an overview of how our sentiment classifiers were trained with the different word embedding matrices.

1) We load pre-trained word embeddings (described in the previous subsection), which contains the word/vocabulary list and the embedding matrix that holds all of the word vector values.
2) We take each of the input sentences from LinSOData, tokenize them and create an integerized representation of the data, which holds the index location of each of the tokens/words in the embedding matrix. From the integerized representation of each example, we construct their vector representation by performing an embedding lookup.
3) The final step of building the sentiment classifier is training. Using Tensorflow, we define hyperparameters, such as the number of LSTM units, dropout rate, number of epochs and the number of output classes. We feed in a batch of training data and labels and try to minimize the loss for each batch. For labeling the training examples, we use a one-hot representation using either [1, 0, 0], [0, 1, 0] or [0, 0, 1], to indicate whether each training example is negative, neutral or positive, respectively.

V. Study Methodology and Findings

The goal of this study is to analyze the effect of an imbalanced dataset on the accuracy of neural network-based sentiment analysis of software engineering artifacts and to investigate how additional customization of a sentiment classifier to the software domain by using domain-specific word embeddings can impact the sentiment analysis of software engineering artifacts.

A. RQ1 - How much is the imbalance in the dataset distribution a factor in the unsatisfactory results of sentiment analysis applied to SE artifacts?

The subject of our study is LinSOData, the gold standard used in Lin et al.'s [19]. Their gold standard consists of manually labeled 1,500 sentences from Stack Overflow. They extracted the list of all discussions that were tagged with Java and contained one of the following words - library/libraries, API(s). Doing this, they collected a total of 276,629 discussions from which 5,073,452 sentences were extracted using the Stanford CoreNLP toolkit. Finally, from the set of 5,073,452 sentences, they randomly selected 1,500 sentences and five of the authors manually labeled them by assigning a sentiment score. Sentences labeled with a sentiment score of '-1' are

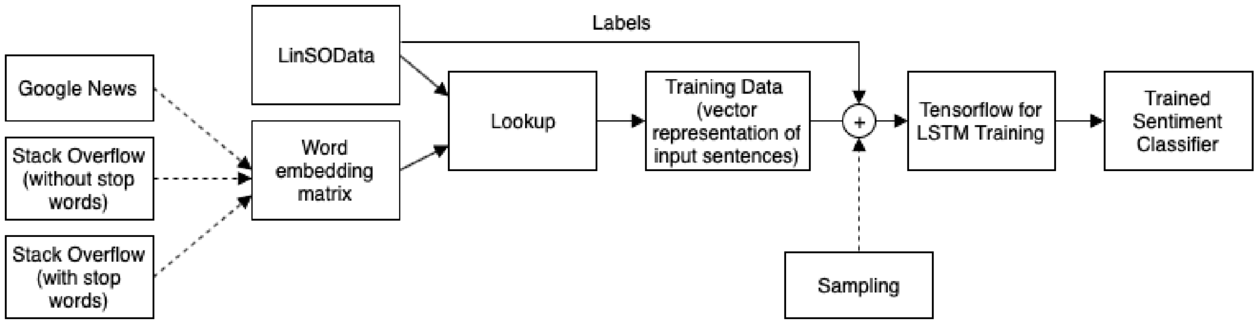

Labels
LinSOData
Google News
Stack Overflow (without stop words)
Stack Overflow (with stop words)
Word embedding matrix
Lookup
Training Data (vector representation of input sentences)
+
Tensorflow for LSTM Training
Trained Sentiment Classifier
Sampling

Fig. 2. Training the Sentiment Classifier.

considered negative, sentences labeled with '0' are considered neutral and sentences labeled with '1' are considered positive. Among the 1500 sentences, 178 sentences were annotated as negative, 1,191 as neutral, and 131 as positive. They performed ten-fold cross-validation and used all of the 1500 sentences in their evaluation.

In the results provided in Lin et al.'s paper [19], although the accuracy on the entire dataset is quite high, the precision and recall for the negative and positive classes are quite low. Since their dataset has a large concentration of neutral sentences, leading to a skewed distribution of negative and positive sentences, looking at the high accuracy would not be effective. Even a constant neutral classifier (ignoring the minority classes) would obtain high accuracy in this scenario. Skewed class distribution is a common phenomenon in NLP, and a common way to address this issue is to adjust the sampling rates of the majority and minority classes using two machine learning techniques, under- and over-sampling of data. These techniques are used to alleviate the problem caused by the imbalance in class distribution.

Over-sampling duplicates the minority class examples, while under-sampling discards the majority class examples in order to modify the class distribution. If we applied the technique of under-sampling on our dataset, we could resolve the class imbalance issue and enhance the sensitivity of the sentiment classifier, but, as it is the dataset is very small, under-sampling would reduce the dataset exceedingly. To balance the dataset using only under-sampling, about five-sixths of the majority class examples will have to be discarded thus throwing away much of the potentially useful information.

Using over-sampling, no information from the original dataset is lost since we keep all examples from the minority and majority classes, but we also risk a chance of the sentiment classifier leading to overfitting as the over-sampling method we adopted was duplication of the minority examples. Therefore, we decided to go with the sampling techniques of over-sampling both the positive and negative classes as well as both under-sampling (of majority class 'Neutral') and over-sampling (of both minority classes) done simultaneously, since the combination of both under- and over-sampling done simultaneously have often produced the best results.

As discussed in section IV, our sentiment classifier involves the use of word embeddings. For comparison to non-domain-specific word embeddings in our study, we provided a generic word embedding model, the Google News word embeddings (hereafter GN word embeddings) released by Mikolov et al. [34] as input to the sentiment classifier. We loaded the GN word embeddings into the sentiment classifier and performed ten-fold cross-validation on the LinSOData similar to the experiments in Lin et al.'s [19]. We divided the 1,500 sentences into ten different groups, each comprising of 150 sentences. All ten groups maintained an even distribution of the three classes of examples, avoiding the situation of one group containing all positive examples whereas the other 9 groups are left without any positive examples. For each of the folds, one group was assigned as the test set and the remaining 1,350 sentences as the training set, repeating the process ten times.

While creating the test and training sets, the test sets were kept as per the original data and all changes (for sampling) were made in the training sets only. During over-sampling, the minority classes (Negative and Positive) were replicated to achieve a balanced distribution with the majority class 'Neutral'. Whilst under-sampling, the majority class 'Neutral' examples were dropped to achieve a balanced distribution with the minority classes negative and positive. In this way, our training set had a balanced distribution of all the three classes of sentiments to train on, while our test set still consisted of the original dataset and did not contain any duplicate entries.

To get a valid understanding of how the sampling techniques would influence the results, we need to balance the distribution of the three classes in the training data. In the original dataset, the neutral class comprised 80% of the dataset, the negative class comprised approximately 12%, and the positive class constituted the remaining 8%. We tried out a number of sampling rates and here we report the results of the following four samplings. These samplings demonstrate both the techniques of oversampling as well as the combination of both under- and over-sampling done simultaneously while maintaining a balanced distribution of the negative, positive and neutral classes in the training set. The other sampling rates provided similar results.

- (1, 7, 9): Over-samples the negative examples by 7 and

the positive examples by 9 times to achieve an equal distribution of all classes. This sampling rate was based on the distribution of the negative, positive and neutral classes in the original data.
- (1, 5, 6): Over-samples the negative examples by 5 and the positive examples by 6 times to achieve a balanced distribution of the negative, positive and neutral classes in the training set, but not an equal distribution.
- (0.8, 6, 7): Uses the combination of both under- and over-sampling to achieve an equal distribution of all classes. Under-samples the neutral class to keep 80% of the examples, over-samples the negative examples by 6 and the positive examples by 7 times.
- (0.6, 4, 5): Achieves an equal distribution of all classes using a greater under-sampling rate. Under-samples the neutral class to keep 60% of the examples, over-samples the negative examples by 4 and the positive examples by 5 times.

Results for RQ1. Table I reports the results achieved by applying our sentiment classifier on LinSOData using the GN word embeddings for all the sampling rates, while Table II presents the results reported by Lin et al. [19]. The table shows the total number of correct predictions out of the 1500 sentences and the precision and recall values for each of the three categories of sentences - positive, negative and neutral. The first row in Table I presents the performance measures for the original dataset without any sampling. A point to note here would be that even for the original dataset, the precision and recall for the negative and positive sentences have increased more than the values reported by Lin et al. [19]; the recall of positive sentences has more than doubled for the original dataset.

The second and third row of Table I presents the results of over-sampling. Although the precision for both negative and positive sentences did not change much, the recall for both the positive and negative sentences increased by 10 - 15% from the recall obtained on the original data. The fourth and fifth row of Table I presents the results of combining under- and over-sampling simultaneously. The results have not seen much difference from the results obtained using over-sampling alone. The precision for both the negative and positive sentences did not change much, similar to the over-sampled data; however, the recall for both the positive and negative sentences increased by 10 - 15% from the recall obtained on the original dataset. The highest recall obtained is about 56% for negative sentences and 46% for positive sentences.

Even if not by a higher percentage, the results in Table I indicate that over-sampling and under-sampling positively enhanced the sensitivity of the sentiment classifier towards the minority classes. The classifier did better in recalling both positive and negative sentences. We believe that better values for both precision and recall could have been accomplished had the dataset been substantial in size.

Since our over-sampling method duplicates examples of the minority classes and does not create new examples, and in all the sampling rates, we over-sampled/duplicated the minority classes by quite a high rate (starting from a minimum of 4, 5 to a maximum of 7, 9), the resulting sentiment analysis models might have been more prone to overfitting, thereby reducing the chances of correct prediction for unseen examples. But given our dataset, it was not possible to further reduce the over-sampling rates or increase the under-sampling rates. Further reducing the over-sampling rates of the negative and positive classes would cause an imbalance in the training sets, negating our purpose of balancing the dataset. Similarly, if we increased the under-sampling rate and discarded more examples from the neutral class to achieve balance, the dataset would have turned to less than half of its original size, and it would not be a reasonable choice to train and test the sentiment classifier on such fewer sentences, making the above mentioned four sampling rates to be the optimal choices, given our dataset.

Based on the results on our smallish dataset, both the techniques of over-sampling and combination of under- and over-sampling performed similarly. The reason for this might be that in the samplings where we did under-sampling, we also reduced the oversample rates as compared to the sampling (1, 7, 9) which achieves an equal distribution of the three classes. Thus, the loss of information that happened due to under-sampling was balanced out by the reduction in overfitting caused due to higher over-sampling rates.

To provide some qualitative data, Table III lists a few examples of sentences belonging to the negative and positive class, that were predicted as neutral by the sentiment classifier on the original data but were correctly identified on the over-sampled and under-sampled dataset. Most of these sentences have several subphrases, resulting in more than one sentiment portrayed throughout the sentence. For example, the second example in Table III starts on a positive note, but the sentence in its entirety is a negative sentence. Similarly, in the fourth sentence in Table III, the words ‘more intuitive’ suggests that it is a positive sentence.

From these observations, we conclude that both the sampling techniques (over-sampling and under-sampling coupled with over-sampling) correctly identify a higher number of negative and positive sentences, producing highest recall values of 56% for negative sentences and 46% for positive sentences, which is a substantial improvement on the results obtained in Lin et al.’s [19], where the highest recall values achieved were 36.5% and 14.5% for negative and positive sentences, respectively.

B. RQ2 - How much does additional customization of the sentiment classifier to the software domain by using domain-specific word embedding impact the sentiment analysis effectiveness?

The goal of this study is to analyze the accuracy of the sentiment classifier tool when applied to software engineering datasets, with the purpose of investigating how the inclusion of software-specific word embeddings can impact the performance of the sentiment classifier as opposed to using generic word embedding trained on general English text. As

TABLE I
TESTING RESULTS OF OUR SENTIMENT CLASSIFIER WITH DIFFERENT SAMPLING TECHNIQUES FOR DATA IMBALANCE.

Sampling Rates	Correct Prediction	Negative		Neutral		Positive	
		Precision	Recall	Precision	Recall	Precision	Recall
Original	1200	0.453	0.433	0.881	0.908	0.402	0.313
1, 7, 9	1171	0.43	0.517	0.884	0.861	0.455	0.42
1, 5, 6	1137	0.47	0.512	0.888	0.832	0.32	**0.458**
0.8, 6, 7	1160	0.43	**0.562**	0.801	0.734	0.466	0.42
0.6, 4, 5	1134	0.4	0.53	0.88	0.821	0.37	0.413

TABLE II
TESTING RESULTS FROM 'SENTIMENT ANALYSIS FOR SOFTWARE ENGINEERING: HOW FAR CANWE GO?' [19]

Correct Prediction	Negative		Neutral		Positive	
	Precision	Recall	Precision	Recall	Precision	Recall
1139	0.365	0.365	0.836	0.886	0.317	0.145

TABLE III
EXAMPLES OF MINORITY CLASS SENTENCES CLASSIFIED NEUTRAL ON ORIGINAL DATASET BUT CLASSIFIED ACCURATELY ON THE RESAMPLED DATASET USING THE GOOGLE NEWS WORD EMBEDDINGS

Sentence	Class
A workaround might be to use reflection, however this isn't a generally recommended approach for a number of reasons (brittleness, performance problems, breaking encapsulation, etc).	Negative
I am using standout library, I added a videoView in it, Everything is running correctly, Except if i click on video then mediaController Is not Showing and i am getting null pointer exception.	Negative
Enabled and pointed my client to a secure port on my container and the last example (the one that causes the exception) works successfully.	Positive
If you are looking for a way to write to streams in a more intuitive way, try CODE_FRAGMENT.	Positive

discussed earlier (section IV A), initially, we used SOwoStop word embedding, the word embedding model released by Efstathiou et al. [8] for specialized software domain-specific knowledge. We replaced the GN word embeddings used in the sentiment classifier, with the SOwoStop word embeddings, and performed ten-fold cross-validation on LinSOData. To compare against a sentiment classifier without customization to the software domain, we used the results obtained from the experiments in RQ1 using the GN word embeddings.

Table IV reports the results achieved by applying the sentiment classifier on the LinSOData using the SOwoStop word embeddings. The results presented in Tables IV and I highlight that, despite using the software-specific word embeddings, the sentiment classifier does not achieve better performance than the generic word embeddings for analyzing the sentiment of software-related SO discussions. Instead, the performance deteriorates for the negative sentences. For each of the sampling rates, the precision for negative sentences decreases by 10 - 20% and the recall for negative sentences decreases by 15 - 20%.

In the instances of positive sentences, the SOwoStop word embeddings accomplished similar results to the GN word embeddings, but its performance deteriorated for the negative sentences. As discussed earlier, since the SOwoStop word embeddings by Efstathiou et al. [8] were trained after eliminating all stop words from the SO posts, the reason for the decrease in the precision and recall values for the negative sentences can be the absence of negations. To this end, we performed another set of experiments using the new set of word embeddings for the software domain developed without removing stop words, referred to as the SOwStop word embeddings (see section IV A).

Table V reports the results achieved by applying the sentiment classifier on LinSOData using the SOwStop word embeddings. Although the sentiment classifier using the SOwStop word embeddings performs very inadequately for the original dataset (which is heavily imbalanced), for the resampled datasets, sentiment analysis with the SOwStop word embeddings identifies negative sentences with a higher recall and precision than the previous SOwoStop word embeddings from Efstathiou et al. [8] with the exception being the sampling rate (1, 5, 6). For the positive sentences, we can say that both the models performed somewhat similarly since the SOwStop word embeddings achieved higher precision and recall measures for some sampling rates, whereas the SOwoStop word embeddings achieved higher values in other sampling rates. The best results achieved by the SOwStop word embeddings are for the (1, 7, 9) over-sampled dataset, which has an equal distribution of all three classes without any undersampling.

Comparisons of the results from Table I and Table V indicate that the overall performance of the sentiment classifier using the GN word embeddings and the SOwStop word embeddings are similar for the four resampled datasets combined. We have already noted the poor recall of negative and positive sentences by the SOwStop word embeddings on the original dataset with no sampling. An interesting point to note here would be that for the (1, 7, 9) over-sampled dataset, the SOwStop word embeddings achieve the best results with

TABLE IV
TESTING RESULTS OF OUR SENTIMENT CLASSIFIER USING SOWOSTOP WORD EMBEDDINGS (WITHOUT STOP WORDS)

Sampling Rates	**Correct Prediction**	**Negative**		**Neutral**		**Positive**	
		Precision	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
Original	1172	0.35	0.216	0.839	0.913	0.40	0.32
1, 7, 9	1126	0.333	0.37	0.858	0.854	0.352	0.32
1, 5, 6	1177	0.39	0.38	0.865	0.875	0.47	0.39
0.8, 6, 7	1042	0.23	0.41	0.87	0.831	0.41	0.35
0.6, 4, 5	1101	0.32	0.40	0.88	0.84	0.345	0.48

an improvement on the results achieved by the GN word embeddings. This might be an indication of the SOwStop word embeddings performing better on datasets that have an equal distribution, but again would need further investigation using a substantially larger dataset.

Although the results indicate that additional customizing of the sentiment classifier through software-specific word embeddings did not surpass the results achieved by the sentiment classifier using generic word embeddings, we cannot conclude that this kind of customizing of the sentiment classifier to the software domain does not impact sentiment analysis of software engineering texts. One primary reason for these results could be the conditions/parameters in training both the word embeddings such as the size and amount of the data. The GN word embeddings are trained on a much larger corpus of about 100 billion words and the trained model contains vectors for about 3 million words. Also, the quality of the GN data should be finer, as the Google News data is usually written by qualified professionals less prone to spelling and grammatical errors, thus probably enabling the GN word embeddings to yield better results.

Another reason for the better performance of the GN word embeddings could be the presence of generic sentences in the dataset. On analysis of the 1500 sentences, we found that the dataset contains many general English sentences rather than all software-specific sentences. Table VI provides a few examples of positive and negative sentences that do not get correct classifications from our sentiment classifier while using the SOwStop word embeddings but are correctly identified by when using the GN word embeddings. Although these sentences are from SO posts, they are not specific to the software domain and in fact are completely generic, thus should not be categorized as software engineering texts. It is expected for the sentiment classifier using the GN word embeddings to perform better than the SOwStop word embeddings with respect to these categories of sentences.

VI. THREATS TO VALIDITY

The first threat to validity is related to the human annotated gold set in Lin et al. [19]. The sentiment expressed in the text can be misconstrued by people. There is no guarantee that the manually assigned sentiments are always accurate. Also, in developing the human annotated gold set, an extra step of sentiment score mapping (five-scale to three-scale) was implemented by the authors. The sentiments expressed in the text were labeled using 5 different degrees, i.e., negative (-2), slightly negative (-1), neutral (0), slightly positive (+1) and positive (+2). The authors categorized the sentiments in the dataset into three levels by considering sentences labeled with '-2' and '-1' as negative sentences (-1), sentences labeled with '0' as neutral sentences (0) and sentences labeled with '+1' and '+2' as positive sentences (+1). As also mentioned in Lin et al. [19], predicting a slightly negative sentence as neutral would be a smaller mistake than predicting a very negative sentence as neutral.

Threats to internal validity concern internal factors we did not consider that could affect the predictions from the sentiment classifier. In our study, they would be due to the technique we used for building our sentiment classifier, the different hyperparameters we used and the training of the software-specific word embeddings. Our sentiment classifier is designed using RNN and LSTM as the RNN units, and we performed many experiments to select the best configuration of the hyperparameters to get the optimal results. Even then, there exist many more configurations for hyperparameters that remain to be tested. Further experiments may reveal that some parameters might be further tuned to increase the accuracy of the sentiment prediction. While training the software-specific word embeddings, experimenting with different parameters such as feature dimension, window size, minimum count, etc may yield better quality embeddings, enabling better predictions.

Threats to external validity correspond to the generalizability of our experiments and findings. In this paper, we designed our sentiment classifier using RNN, since the sequence of words in a particular sentence matter for sentiment analysis, and RNNs have turned out to be very useful in accounting for that dependency. Using a different architecture may impact the performance of the classifier.

VII. CONCLUSIONS

In this paper, we studied the impact of two machine-learning techniques, oversampling and undersampling to address the skewed distribution of a dataset and how adjusting the sampling rates can affect the sentiment analysis of software engineering texts. We also investigated how additional customization of a neural network-based sentiment classifier to the software domain by using domain-specific word embedding can impact the sentiment analysis effectiveness.

TABLE V
TESTING RESULTS OF OUR SENTIMENT CLASSIFIER USING SOWSTOP WORD EMBEDDINGS (WITH STOPWORDS)

Sampling Rates	**Correct Prediction**	**Negative**		**Neutral**		**Positive**	
		Precision	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
Original	1200	0.522	0.068	0.796	0.993	1	0.031
1, 7, 9	**1205**	**0.5**	**0.6**	**0.893**	**0.881**	**0.442**	**0.38**
1, 5, 6	1168	0.472	0.34	0.844	0.89	0.32	0.33
0.8, 6, 7	944	0.225	0.624	0.915	0.706	0.318	0.301
0.6, 4, 5	1038	0.41	0.53	0.92	0.746	0.226	0.54

TABLE VI
EXAMPLES OF MINORITY CLASS SENTENCES USED FOR SOWSTOP WORD EMBEDDINGS, BUT NOT SPECIFIC TO THE SOFTWARE DOMAIN

Sentence	Class
... but no such thing exists.	Negative
Problem solved.	Positive
so I'm not happy with it.	Negative
Solved, in case anyone stumbles upon similar problem.	Positive
but still nothing.	Negative
This is your problem.	Negative
On Facebook i ran in to a problem.	Negative
Now we're getting to the good part.	Positive

The application of oversampling and undersampling techniques in our study show that oversampling and oversampling coupled with undersampling achieve higher recall for the negative and positive sentences as compared to the original dataset. Thus, adjusting the sampling rates improves the sensitivity of the model towards the minority classes in terms of recall, without affecting the majority class predictions. We interpret the underlying reason for not achieving considerable higher precision and significantly higher values of recall, to be dependent on the size of the dataset. Both oversampling and undersampling had limitations, duplicating the small number of minority class examples (131 positives and 178 negatives) multiple times and discarding about half of the neutral (1191 in total), restrict the sentiment classifier beyond a certain point of improvement. Nonetheless, the apparent trend was that both the resampling techniques (oversampling and oversampling coupled with undersampling) were able to correctly identify a higher number of negative and positive sentences, achieving the highest recall of 56% and 46% for negative and positive sentences respectively, which is an improvement on the results obtained by Lin et al. [19], that achieved the highest recall of 36.5% and 14.5% for negative and positive sentences respectively. A much larger and balanced dataset of software engineering text, can increase the precision and recall for the negative and positive classes by manifolds, thus making sentiment analysis on software related texts much more reliable. Finally, neither oversampling alone nor undersampling and oversampling coupled together significantly outperforms one another for our dataset.

Outcomes from the evaluation of the sentiment classifier using software-specific word embeddings indicate that the performance of the sentiment classifier using the GN word embeddings is similar to the performance of the SOwStop word embeddings for the resampled datasets. However, the SOwStop word embeddings achieve extremely poor recall for negative and positive sentences on the original dataset as compared to the GN word embeddings. Although customizing the sentiment classifier to the software domain using software-specific word embeddings does not surpass the results achieved by the generic word embeddings, further investigation is needed to affirmatively conclude that customizing the sentiment classifier to the software domain does not impact sentiment analysis of software engineering texts. While the SOwStop word embeddings model is based on all of Stack Overflow data, it is still dwarfed by the size of the Google News corpus. The GN word embeddings are trained on a much larger corpus of about 100 billion words, thus probably enabling the GN word embeddings to yield better results than the SOwStop word embeddings.

VIII. FUTURE WORK

There are several avenues to further improve the results. Our primary focus would be to conduct a large-scale study using a much more extensive dataset and re-training the software-specific word embeddings by experimenting with different parameters such as feature dimension, window size, etc so as to capture a better-quality respective vector representation of words.

While the current software-specific word embeddings have been trained on all of the available SO posts, it is dwarfed by the size of the Google News corpus. Thus, we also plan to consider using a multi-channel model where we could combine the breadth of the generic word embeddings (obtained from Google News) and the specificity of the software-specific word embeddings (obtained from SO posts) and use the resulting embeddings for our sentiment classifier.

In our experiments, the oversampling of the minority classes has been done by replicating/duplicating the training examples. Oversampling with replication may not significantly improve minority class recognition since as we oversample the minority class, the model tries to be more and more specific, leading to overfitting. To prevent overfitting, we would consider using SMOTE [5], an oversampling approach in which the minority class is oversampled by creating synthetic examples rather than by oversampling with duplication.

Identifying the main words that contribute to the prediction can help to get better predictions from a sentiment classifier

[17, 30]. Thus, we also intend to look at convolutional neural networks specifically with techniques such as word attention that helps us to identify the main words that contribute to a classification/prediction. We also consider training sentiment-specific embeddings [33, 32], which would encode sentiment information in the vector representations. Word embeddings take into consideration the syntactic context of words ignoring the sentiment of text, and in the process also map words with similar syntactic context but opposite sentiment polarity to neighboring word vectors. Encoding sentiment information in the word vectors would enable us to customize the word embeddings for the sentiment analysis task.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 1422184.

References

[1] Ahmed, Toufique and Bosu, Amiangshu and Iqbal, Anindya and Rahimi, Shahram. "SentiCR: A Customized Sentiment Analysis Tool For Code Review Interactions". In: *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press. 2017, pp. 106–111.

[2] Bengio, Yoshua and Ducharme, Réjean and Vincent, Pascal and Janvin, Christian. "A Neural Probabilistic Language Model". In: *J. Mach. Learn. Res.* 3 (Mar. 2003), pp. 1137–1155.

[3] Blaz, Cássio Castaldi Araujo and Becker, Karin. "Sentiment Analysis In Tickets For IT Support". In: *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE. 2016, pp. 235–246.

[4] Calefato, Fabio and Lanubile, Filippo and Maiorano, Federico and Novielli, Nicole. "Sentiment Polarity Detection For Software Development". In: *Empirical Software Engineering* 23.3 (2018), pp. 1352–1382.

[5] Chawla, Nitesh V and Bowyer, Kevin W and Hall, Lawrence O and Kegelmeyer, W Philip. "SMOTE: Synthetic Minority Over-sampling Technique". In: *Journal of Artificial Intelligence Research* 16 (2002), pp. 321–357.

[6] Collobert, Ronan and Weston, Jason. "A Unified Architecture For Natural Language Processing: Deep Neural Networks With Multitask Learning". In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: ACM, 2008, pp. 160–167.

[7] Ding, Jin and Sun, Hailong and Wang, Xu and Liu, Xudong. "Entity-Level Sentiment Analysis Of Issue Comments". In: *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering*. ACM. 2018, pp. 7–13.

[8] Efstathiou, Vasiliki and Chatzilenas, Christos and Spinellis, Diomidis. "Word Embeddings For The Software Engineering Domain". In: *Proceedings Of The 15th International Conference On Mining Software Repositories*. MSR '18. Gothenburg, Sweden: ACM, 2018, pp. 38–41.

[9] Graziotin, Daniel and Wang, Xiaofeng and Abrahamsson, Pekka. "Do Feelings Matter? On The Correlation Of Affects And The Self-assessed Productivity In Software Engineering". In: *Journal of Software: Evolution and Process* 27.7 (2015), pp. 467–487.

[10] Guzman, Emitza and Bruegge, Bernd. "Towards Emotional Awareness In Software Development Teams". In: *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2013. Saint Petersburg, Russia: ACM, 2013, pp. 671–674.

[11] Imtiaz, Nasif and Middleton, Justin and Girouard, Peter and Murphy-Hill, Emerson. "Sentiment And Politeness Analysis Tools On Developer Discussions Are Unreliable, But So Are People". In: *2018 IEEE/ACM 3rd International Workshop on Emotion Awareness in Software Engineering (SEmotion)*. IEEE. 2018, pp. 55–61.

[12] Islam, Md Rakibul and Zibran, Minhaz F. "A Comparison Of Software Engineering Domain Specific Sentiment Analysis Tools". In: *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. 2018, pp. 487–491.

[13] Islam, Md Rakibul and Zibran, Minhaz F. "DEVA: Sensing Emotions In The Valence Arousal Space In Software Engineering Text". In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. ACM. 2018, pp. 1536–1543.

[14] Islam, Md Rakibul and Zibran, Minhaz F. "Leveraging Automated Sentiment Analysis In Software Engineering". In: *Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference*. IEEE. 2017, pp. 203–214.

[15] Jongeling, Robbert and Datta, Subhajit and Serebrenik, Alexander. "Choosing Your Weapons: On Sentiment Analysis Tools For Software Engineering Research". In: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2015, pp. 531–535.

[16] Jongeling, Robbert and Sarkar, Proshanta and Datta, Subhajit and Serebrenik, Alexander. "On Negative Results When Using Sentiment Analysis Tools For Software Engineering Research". In: *Empirical Software Engineering* 22.5 (2017), pp. 2543–2584.

[17] Lee, Gichang and Jeong, Jaeyun and Seo, Seungwan and Kim, CzangYeob and Kang, Pilsung. "Sentiment Classification With Word Attention Based On Weakly Supervised Leaning With A Convolutional Neural Network". In: *arXiv preprint arXiv:1709.09885* (2017).

[18] Li, Shoushan and Wang, Zhongqing and Zhou, Guodong and Lee, Sophia Yat Mei. "Semi-Supervised Learning For Imbalanced Sentiment Classification". In: *Twenty-Second International Joint Conference on Artificial Intelligence*. 2011.

[19] Lin, Bin and Zampetti, Fiorella and Bavota, Gabriele and Di Penta, Massimiliano and Lanza, Michele and Oliveto, Rocco. “Sentiment Analysis For Software Engineering: How Far Can We Go?” In: *Proceedings of the 40th International Conference on Software Engineering*. ICSE ’18. Gothenburg, Sweden: ACM, 2018, pp. 94–104.

[20] Mubarok, Mohamad Syahrul and Adiwijaya and Aldhi, Muhammad Dwi. “Aspect-Based Sentiment Analysis To Review Products Using Naïve Bayes”. In: *AIP Conference Proceedings*. Vol. 1867. 1. AIP Publishing. 2017, p. 020060.

[21] Sebastian C Müller and Thomas Fritz. “Stuck And Frustrated Or In Flow And Happy: Sensing Developers’ Emotions And Progress”. In: *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference*. Vol. 1. IEEE. 2015, pp. 688–699.

[22] Murgia, Alessandro and Tourani, Parastou and Adams, Bram and Ortu, Marco. “Do Developers Feel Emotions? An Exploratory Analysis Of Emotions In Software Artifacts”. In: *Proceedings of the 11th working conference on mining software repositories*. ACM. 2014, pp. 262–271.

[23] Novielli, Nicole and Calefato, Fabio and Lanubile, Filippo. “The Challenges Of Sentiment Detection In The Social Programmer Ecosystem”. In: *Proceedings of the 7th International Workshop on Social Software Engineering*. ACM. 2015, pp. 33–40.

[24] Novielli, Nicole and Calefato, Fabio and Lanubile, Filippo. “Towards Discovering The Role Of Emotions In Stack Overflow”. In: *Proceedings of the 6th International Workshop on Social Software Engineering*. SSE 2014. Hong Kong, China: ACM, 2014, pp. 33–36.

[25] Novielli, Nicole and Girardi, Daniela and Lanubile, Filippo. “A Benchmark Study On Sentiment Analysis For Software Engineering Research”. In: *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. IEEE. 2018, pp. 364–375.

[26] Ortu, Marco and Murgia, Alessandro and Destefanis, Giuseppe and Tourani, Parastou and Tonelli, Roberto and Marchesi, Michele and Adams, Bram. “The Emotional Side Of Software Developers In Jira”. In: *Proceedings of the 13th International Conference on Mining Software Repositories*. MSR ’16. Austin, Texas: ACM, 2016, pp. 480–483.

[27] *Perform Sentiment Analysis with LSTMs, using TensorFlow*. URL: https://www.oreilly.com/learning/perform-sentiment-analysis-with-lstms-using-tensorflow.

[28] Prusa, Joseph and Khoshgoftaar, Taghi M and Dittman, David J and Napolitano, Amri. “Using Random Undersampling To Alleviate Class Imbalance On Tweet Sentiment Data”. In: *2015 IEEE International Conference on Information Reuse and Integration*. IEEE. 2015, pp. 197–202.

[29] Rahman, Mohammad Masudur and Roy, Chanchal K and Keivanloo, Iman. “Recommending Insightful Comments For Source Code Using Crowdsourced Knowledge”. In: *Source Code Analysis and Manipulation (SCAM), 2015 IEEE 15th International Working Conference*. IEEE. 2015, pp. 81–90.

[30] Shin, Bonggun and Lee, Timothy and Choi, Jinho D. “Lexicon Integrated CNN Models With Attention For Sentiment Analysis”. In: *arXiv preprint arXiv:1610.06272* (2016).

[31] Socher, Richard and Perelygin, Alex and Wu, Jean and Chuang, Jason and Manning, Christopher D and Ng, Andrew and Potts, Christopher. “Recursive Deep Models For Semantic Compositionality Over A Sentiment Treebank”. In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. 2013, pp. 1631–1642.

[32] Tang, Duyu. “Sentiment-Specific Representation Learning For Document-level Sentiment Analysis”. In: *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*. WSDM ’15. Shanghai, China: ACM, 2015, pp. 447–452.

[33] Tang, Duyu and Wei, Furu and Yang, Nan and Zhou, Ming and Liu, Ting and Qin, Bing. “Learning Sentiment-Specific Word Embedding For Twitter Sentiment Classification”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vol. 1. 2014, pp. 1555–1565.

[34] Tomas Mikolov and Kai Chen and Greg Corrado and Jeffrey Dean. “Efficient Estimation Of Word Representations In Vector Space”. In: *CoRR* abs/1301.3781 (2013). arXiv: 1301.3781.

[35] Tourani, Parastou and Jiang, Yujuan and Adams, Bram. “Monitoring Sentiment In Open Source Mailing Lists: Exploratory Study On The Apache Ecosystem”. In: *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*. IBM Corp. 2014, pp. 34–44.

[36] Xu, Ruifeng and Chen, Tao and Xia, Yunqing and Lu, Qin and Liu, Bin and Wang, Xuan. “Word Embedding Composition For Data Imbalances In Sentiment And Emotion Classification”. In: *Cognitive Computation* 7.2 (2015), pp. 226–240.

[37] Zhang, Yingying and Hou, Daqing. “Extracting Problematic API Features From Forum Discussions”. In: *Program Comprehension (ICPC), 2013 IEEE 21st International Conference*. IEEE. 2013, pp. 142–151.