Mini-Project 4: Improved Baselines for Sentence and Document Representations

Jean-Sébastien Grondin
McGill Id:260345519
jean-sebastien.grondin@mail.mcgill.ca

Jonathan El Baze
McGill Id:260893222
jonathan.elbaze@mail.mcgill.ca

Zhourong Li
McGill Id:260674414
zhourong.li@mail.mcgill.ca

Abstract

The goal of this project was to revisit the statements made in the work of Le & al [1] with regard to the performance of *Paragraph Vectors* in natural language processing applications. The authors of this study, claimed that *Paragraph Vectors* achieved state-of-the-art results on text classification and sentiment analysis tasks. To verify this statement, this project first started with reproducing the best baseline referenced in this report (i.e. Naïve-Bayes-SVM), and exploring the hyper-parameter space to improve it. All comparisons were made on a sentiment classification task using the IMDB sentiment dataset [4]. The best NB-SVM baseline developed in this study achieved an accuracy of **92.096%** on the test set. This is +0.876% above the baseline reported in the article. Then, another baseline was created using a word embeddings approach. The word2Vec technique [2][3] was used to create a word vector representation, which was then used to train a logistic regression classifier. The new baseline developed for this project with the word2vec technique obtained an accuracy of **94.234%**. When compared to the performance of *Paragraph Vectors*, which obtained 92.58%, the NB-SVM model is under-performing by 0.484% but the latter outperforms *Paragraph Vectors* by 1.654%.

1 Introduction

In 2014, the work of Le & al [1] has introduced *Paragraph Vectors* which are fixed-length vector representations of text that can be of variable length, going from sentences to paragraphs and even whole documents. The vector representation is obtained from running an unsupervised neural network algorithm on text with the goal of predicting the surrounding words in contexts of sampled from paragraphs. The authors of this research claim that this new type of text representation has the potential to overcome the weakness of the more traditional bag-of-words models, which may not be able to fully capture the semantics and contextual relations in text. To demonstrate the effectiveness of this novel technique, Le & al compared the performance of *Paragraph Vectors* to other baseline approaches on sentiment analysis as well as information retrieval tasks.

The goal of this project was to reproduce and improve the baselines used and referenced in this paper and evaluate whether the benefits of *Paragraph Vectors* are truly as good as the authors claim them to be. As a first step, a Naïve-Bayes Support Vector Machine (NB-SVM) baseline was reproduced, then extensively evaluated and compared against the *Paragraph Vectors* performance on the IMDB sentiment analysis task. As a second step, a new baseline using word embeddings was created to evaluate the performance of *Paragraph Vectors*. This second baseline used word2Vec, developed by Mikolov & al. [2] and now an open source project [3]. Word2Vec has become one of the most popular technique to compute vector representations of words and is now used in many natural language processing applications.

The best NB-SVM baseline developed in this study achieved an accuracy of **92.096%** on the test set using a combination of uni-, bi- and tri-grams. This accuracy is +0.876% above the baseline reported in the article. The new baseline developed for this project with the word2vec technique obtained an accuracy of **94.234%** on the test set. When compared to the performance of *Paragraph Vectors*, which obtained 92.58%, the former is under-performing by only 0.484% but the latter outperforms *Paragraph Vectors* by 1.654%.

2 Related Work

For many natural language processing (NLP) applications, selecting the right word representations is critical. A very common representation for texts is the bag-of-words and bag-of-n-grams. The disadvantage of using bag-of-words is that this representation does not take into account the word order and the context of the neighbouring words, and thus fails at capturing

the semantics of the words. The bag-of-n-grams does consider some word order but rarely in the context of more than two or three words. This difficulty has motivated many research in the field of NLP to use neural networks to learn vector representations for words [6][7][8] and more recently extend these models to phrase-level or sentence level representations [1][2][9]. Among the various options for word vectors, perhaps the most popular is Word2vec [2][3]. It is a neural network based algorithm that creates distributed vector representations for words. One of the advantage of Word2vec is that it is better suited for capturing the semantics and relationships between words. Another advantage is that the creation of the vector representations does not require label data.

Simple models like Naive Bayes (NB) and Support Vector Machines (SVM) have been considered by many NLP researchers as relevant baselines for text classification and sentiment analysis tasks. Wang & Manning [5] have demonstrated these simple models can in fact outperform the state-of-the-art methods on many if the model is selected carefully and the right features are utilized (e.g. unigrams, bi-grams, frequency word count, binary word count, TF-IDF, etc.). Wang & Manning have also proposed a novel NB-SVM combination, that is a SVM model variant, which uses NB log-count ratios as input features, and demonstrated that it is a robust performer for many NLP tasks.

3 Dataset and setup

Le & al compared the performance of *Paragraph Vectors* with simpler methods on several datasets. One of the benefit of the method they are proposing is that it is not restricted to work on sentences but can also be used on paragraphs or documents. To validate this, they used the IMDB dataset, which was first proposed by Maas & al [6]. This dataset consists in 100,000 movie reviews, where each review is composed of several sentences. We will thus focus our attention on this dataset, and verify that the performance of *Paragraph Vectors* can indeed supplant that of simpler methods like the NB-SVM approach.

This dataset includes 50,000 labelled instances, divided in two subsets: 25000 training and 25000 test instances with an equal amount of positive (50%) and negative (50%) comments. It also includes 50,000 unlabelled instances, that can be used to train vector representations.

The test data was left aside and 10% of the training data was randomly split and left aside for validation. The remaining instances (90%) were used to train the two baseline models.

4 Proposed approach

4.1 Baseline #1: Naïve Bayes Support Vector Machines

A NB-SVM machine learning pipeline was implemented based on the model description from Wang & Manning [5]. This model starts with a feature count vector $\mathrm{f}^{(i)} \in \mathbb{R}^{|V|}$, for each training case i, along with its label $y^{(i)} \in \{-1, 1\}$. V is the set of features and $\mathrm{f}_j^{(i)}$ represents the number of occurences for the jth feature in instance i. We can then compute the total count for all features in positive instances, $p = \alpha + \sum_{i:y^{(i)}=1} \mathrm{f}^{(i)}$, and negative instances, $q = \alpha + \sum_{i:y^{(i)}=-1} \mathrm{f}^{(i)}$. Using the following equation, we can obtain the log-count ratio:

$$r = log\left(\frac{p/||p||_1}{q/||q||_1}\right)$$

The SVM classifier is a linear classifier, where the prediction for test case k is:

$$y^{(k)} = \mathrm{sign}(\mathrm{w}^T x^{(k)} + b)$$

We use $x^{(k)} = \hat{r} \circ \hat{\mathrm{f}}^{(k)}$. For the element-wise product, we use the binarized version of $\mathrm{f}^{(k)}$ instead, i.e. $\hat{\mathrm{f}}^{(k)} = 1\{\hat{f}^{(k)} > 0\}$, where 1 is the indicator function. The training of this model is achieved by finding w and b through minimizing:

$$\mathrm{w}^T\mathrm{w} + C\sum_i \max(0, 1 - y^{(i)}(\mathrm{w}^T\hat{\mathrm{f}}^{(i)} + b))^2$$

Finally, an interpolation between a multinomial naïve bayes (MNB) and SVM is achieved using:

$$\mathrm{w}' = (1 - \beta)\bar{w} + \beta\mathrm{w}$$

where $\bar{w} = ||\mathrm{w}||_1/|V|$, is the average magnitude of w and $\beta \in [0, 1]$ is the interpolation parameter.

A starting model was obtained from the github of Joshua Chin [10] and SKLearn [11] was used to create the machine learning pipeline.

A thorough exploration of the NB-SVM model was achieved by a) searching for the best combination of pre-processing steps (see list below) and b) thoroughly exploring the hyperparameter space (e.g. α, β, C).

- **URLs:** removed / not removed
- **email addresses:** removed / not removed
- **capital letters:** lowercase / unmodified
- **token patterns for count vectorizer:** alphabetical only, alphanumeric only, alphanumeric & punctuation, alphanumeric & punctuation & creation of tokens for scores out of 10 in text (e.g. "10/10", "9/10", etc).
- **lemmatization or stemming:** various inflected forms of a word are regrouped into a lemma (i.e.

one base word) or reduced to their stems. These two techniques were used interchangeably. Not using either option was also tested.

- **stop words:** removal of "in", "of", "at", "a", "the" / not removed.
- **n-grams:** the benefit of introducing bi-grams and tri-grams was evaluated, together with the various possible combinations.

4.2 Baseline #2: word2vec embeddings

The word2Vec tutorial [12] was used as a starting point. From this starting point, different improvements have been tested out. The following steps were implemented:

- **HTML tags:** removed from the raw review comments
- **Non-letters:** removed from the raw review comments
- **Capital letters:** converted to lowercase
- **Stop words:** removing them was tested to verify if it improves accuracy
- **Number of words in cluster:** the number of words per cluster in K-means clustering
- **word2Vec size:** the size (number of features) included in word2vec vectors
- **Hyper parameters tuning:** Hyperparameters in the logistic regression model

5 Results

5.1 Baseline #1: Naïve-Bayes Support Vector Machines

As a first step, the baseline accuracy reported in Le & al. [1] was reproduced. The author of the original paper [5] conveniently indicated that they used $\alpha = 1$, $C = 1$, $\beta = 0.25$. It was not possible to exactly reproduce the results but the accuracy obtained was within 0.25% for both the unigram and bigram solutions. The differences could be due to the fact that the accuracy reported in that article comes from the test set while the accuracy reported for this project is on the validation set.

Table 1 shows the incremental improvements that were subsequently made to the original NB-SVM baseline. The incremental improvements that were retained are highlighted in bold. Some other options were tested but not retained, and these are not highlighted in bold.

Different token patterns for the count vectorizer were tested, going from using alphabetical characters (ID1), to alphanumeric characters (ID2), then including punctuation (ID3), and finally including scores out of 10 (ID4) mentioned in text (e.g. "9/10", "10/10").

Then, different variants of text pre-processing were tested in steps ID5 to ID10. In the end, only the removal of capital letters (ID5), URLs (ID6) and stop words (ID10) were kept.

Figure 1: Accuracy on validation set for different β

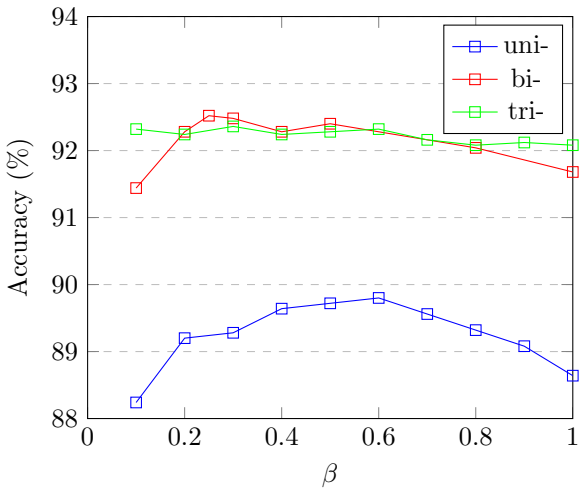


Table 1: Incremental improvements made to NB-SVM baseline accuracy (%)

ID	Description	uni-	bi-	tri-
0	**Baseline**[1][5]	**88.290**	**91.220**	**NA**
1	[a-zA-Z]+	88.360	91.000	91.628
2	[a-zA-Z0-9]+	88.400	91.300	91.924
3	[a-zA-Z0-9]+ + punctuation	88.600	91.396	92.032
4	**[a-zA-Z0-9]+ punctuation & % score**	89.080	91.460	92.044
5	**lowercase**	89.080	91.460	92.044
6	**removing URLs**	89.080	91.464	92.048
7	removing emails	89.080	91.460	92.044
8	stemming	86.720	91.276	91.960
9	lemmatizing	89.400	91.468	92.036
10	**removing stop words**	89.160	91.692	92.092

Subsequently, the hyperparameter space of the NB-SVM was fully explored. Figure 1 shows how the accuracy changes as the NB-SVM interpolation parameter β is varied. The optimal β for the unigram, bigram and trigram solutions are found to be 0.6, 0.25 and 0.3 respectively. It is interesting to note that when using uni-gram only, we rely more on the SVM component and as we introduce the bi-gram and tri-gram we add more weight to the NB component.

Different values of the penalty parameter C were also tested. C is the penalty parmater of the error term, which controls the trade off between smooth decision boundary and classifying training points correctly. High values of C could lead to overfitting. In retrospect, it

Figure 2: Accuracy on validation set for different C

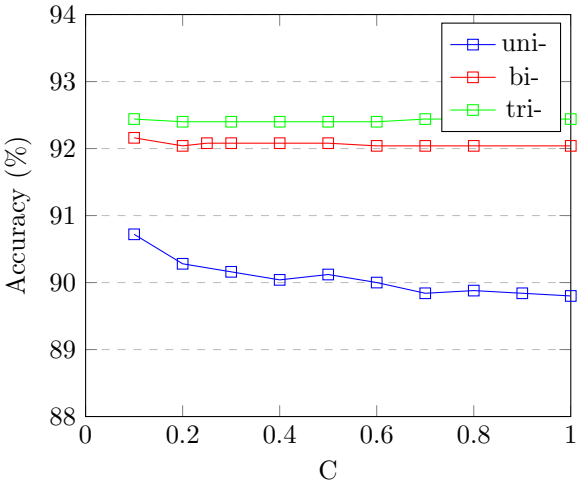
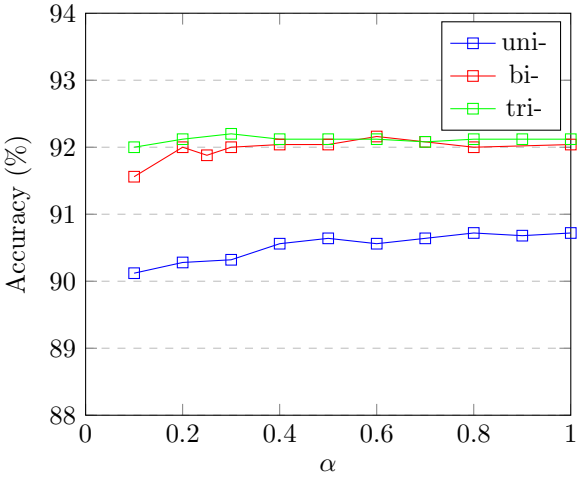


Figure 3: Accuracy on validation set for different α

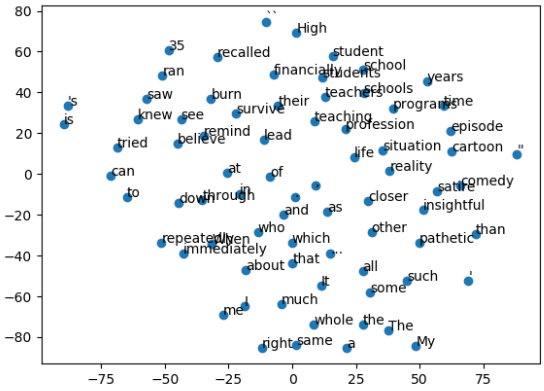


appears like higher values of C could have been tested. Figure 2 shows the results obtained for the uni-, bi- and tri-gram solutions. The optimal C penalty parameter for the unigram, bigram and trigram solutions are found to be 0.1, 0.1 and 1.0 respectively.

Several values of the smoothing parameter α were also investigated. This parameter in the NB-SVM is analogous to the Laplace smoothing parameter used in Naive Bayes model and will prevent posterior probabilities suddenly dropping to zero. Figure 3 shows that the optimal α smoothing parameter for the unigram, bigram and trigram solutions are 1.0, 0.6 and 1.0 respectively.

The best performing NB-SVM baseline model performance is shown in Table 2. The best NB-SVM baseline achieved an accuracy of **92.096%** on the test set using a combination of uni-, bi- and tri-grams. This accuracy is +0.876% above the baseline reported in the article from

Figure 4: t-SNE of words



Le & al, and only -0.484% under-performing compared to *Paragraph Vectors*.

Table 2: Test set accuracy for NB-SVM (%)

Description	uni-	bi-	tri-
Baseline[1][5]	88.290	91.220	NA
Final model	**88.372**	**91.744**	**92.096**

5.2 Baseline #2: word2vec embeddings

Visualization

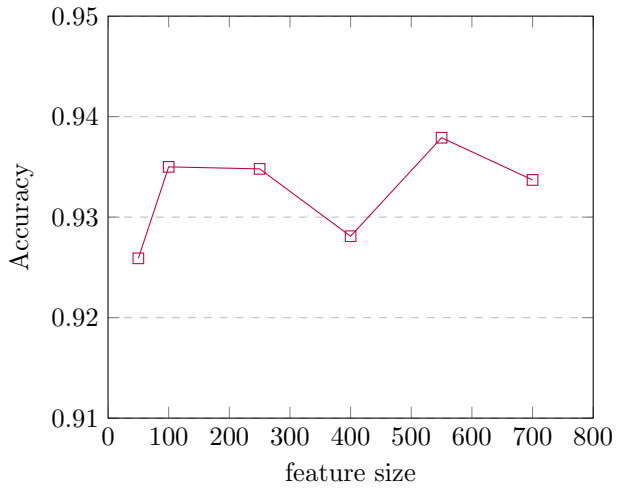
Word2vec has a very high dimension as the number of features are large. Thus, in order to visualize the relationship between different words, the t-SNE function is being used. t-SNE reduces the dimension of features to 2, which is human friendly and maps the words to a 2-D graph.

From Figure 4, it is possible to see how similar words are close to each other in euclidean distance, for example "student" and "school" are close to each other, and the stop-words, "the" and "a" are also close to each other, which shows that they have similar meanings.

Baseline and Improvements

The baseline model was created with no pre-processing, using a K value initialized as 5, a number of features fixed at 250, and $C = 100$ for logistic regression. Incremental improvements are listed in Table 3. A baseline accuracy of 92.48% is obtained in the validation set. Secondly, the HTML tags are removed using the html parser from the package *beautifulsoup*[13], and the

Figure 5: Accuracy on validation set for different number of features



validation accuracy increases to 93.64%. Removing non-letter characters then increases the accuracy to 93.87%. Removing capital letters also benefits the validation accuracy. However, removing stop-words results in lower performance. this suggests the stop-words can actually bring information (semantics and meaning of stop-word itself) to the word2vec model, which can be useful for the semantics analysis. Thus stop words are retained.

Table 3: Incremental improvements made to word2vec baseline accuracy (%)

ID	Description	Val Acc
0	**Baseline**	**0.9348**
1	removing html tags	0.9364
2	removing non-letters	**0.9387**
3	lowercase	0.9256
4	removing stop-words	0.9255

The number of features used in in the word2Vec representations was found to have an effect on the accuracy. The following number of features were tested: 100, 250(baseline), 400, 550 and 700 and their corresponding accuracy are reported in Figure 5.

As the number of features increases, the validation accuracy tends to increase up to around 550 features and then starts decreasing. The highest validation accuracy is found to be 93.79% with 550 features. One should note that the more features are considered, the longer it takes to construct the word2Vec model.

Word2Vec creates clusters of semantically related words. The K-means clustering is being used as a clustering technique. Different number of words in each cluster were tested. The results are shown in Figure 6. The optimal number of words per cluster was found to be 5,

Figure 6: Accuracy for number of words per cluster

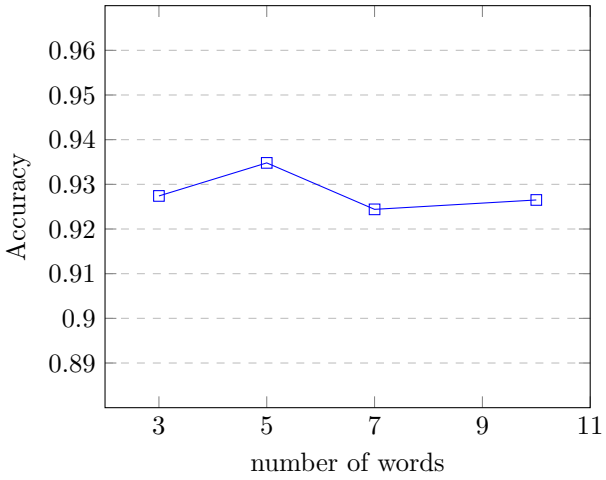
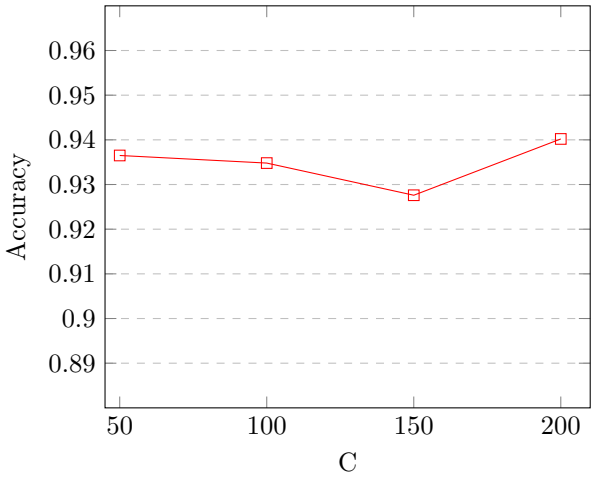


Figure 7: Accuracy on validation set for different C



which was the original baseline.

Hyper-parameter tuning for the logistic regression was also explored. Different values for the penalty parameter C was investigated for the logistic regression classifier. The results are show in Figure 7. The optimal value for the penalty parameter C was found to be 50.

The best performing logistic regression model using the word2vec technique delivered a performance of **94.234%**. Results are summarized in Table 4. This accuracy is +1.654% above the performance of *Paragraph Vectors*, reported in Le & al.

6 Discussion and Conclusion

The best NB-SVM baseline model implemented in this study used a combination of uni-, bi- and tri-grams

Table 4: Test set accuracy (%)

Description	Accuracy (%)
NB-SVM Baseline[1][5]	91.220
Best NB-SVM	**92.096**
Paragraph Vectors[1]	92.580
Best log. reg. with word2Vec	**94.234**

and was found to achieve an accuracy of **92.096%** on the test set. This is +0.876% above the best baseline reported in Le & al [1] and the original paper [5]. Since the latter used a combination of uni- and bi-grams only, we can also compare the performance of our solution with uni- and bi-grams. We were able to achieve an accuracy of 91.744% which is still +0.524% above the accuracy reported in the original article. The new baseline created for this project was a logistic regression classifier using the word2vec technique. It obtained an accuracy of 94.234% on the test set, which corresponds to an improvement of +1.654% above the performance of *Paragraph Vectors*

One of the interesting findings of this project was to realize that by doing a deep exploration of the hyperparameter space, it is possible to achieve significant improvements to the accuracy and even beat the performance of similar models reported in scientific papers. For example, just with a more clever usage of token patterns, we were able to achieve an improvement of +0.460%. With the usage of trigrams, we were also able to improve the performance of +0.352%. Having said that, our best baseline NB-SVM model is now just slightly under-performing compared to *Paragraph Vectors* with -0.484%. Having done a more thorough exploration of the NB-SVM baseline solution, we can now confirm that the *Paragraph Vectors* do perform better than simpler state-of-the art models that do not consider vector representations but not by a significant amount. To put things in perspective, the incremental improvement achieved by selecting the right token pattern for the count vectorizer is similar to the performance gap between the NB-SVM baseline and the *Paragraph Vectors* solution.

Theoretically, paragraph vector should have better performance than word2vec. The reason is while making the average vector and centroid matrices in word2vec, the order of words were lost. While the paragraph vector preserves word order information. However, with carefully preprocessing setting and optimized parameters, word2vec attains better score of 94.234, which outperforms the 92.58 of paragraph vector in paper. This indicates paragraph vector still has potential to achieve better performance with fully explore parameters and carefully selected preprocessing.

Another key realization is that as we get closer to the maximum theoretical performance of one type of classifier, it becomes increasingly more difficult to achieve performance gains. The marginal amount of effort required to improve the accuracy becomes higher and higher.

After a lot of experimentation, there is now a strong sentiment of curiosity and appetite for going further and testing more features and functionalities, as well as exploring word embedding approaches. Here are some steps that would be interesting to investigate and that could help improve further the results:

- **A)** Investigate GloVe, which is another unsupervised learning algorithm for obtaining vector representations for words. Compare the performance with that of word2Vec.
- **B)** Create an ensemble method to combine both baselines developed and assess performance gains.
- **C)** Reproduce the *Paragraph Vectors* solution from Le & al., and see if improvements can be made to the classification accuracy.

7 Statement of Contributions

- **J-S Grondin** Primary role was to implement and evaluate the NB-SVM model as a baseline. Also contributed to the writing of the report.
- **J El Baze** Primary role was to build data loading routine and explore the hyperparameter space for the NB-SVM baseline solution. Also contributed to writing the report.
- **Zhourong Li** Primary role was to implement and evaluate a baseline solution that uses word2vec to create word vector representations. Also contributed to writing of the report.

References

[1] Le & al. *Distributed Representations of Sentence and Documents*. International Conference on Machine Learning (2014), Google Inc., 2014

[2] Mikolov & al. *Distributed Representations of Words and Phrases and their Compositionality*. Neural and Information Processing System (NIPS), 2013

[3] *Word2Vec*, July 2013, https://code.google.com/archive/p/word2vec/

[4] *Sentiment Analysis: Large Movie Review Dataset*, https://ai.stanford.edu/ amaas/data/sentiment/

[5] Wang & Manning. *Baselines and Bigrams: Simple, Good Sentiment and Topic Classification*. Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, July 2012

[6] Maas & al. *Learning Word Vectors for Sentiment Analysis*. Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, June 2011

[7] Bengio & al. *Neural probabilistic language models*. Innovations in Machine Learning, pp. 137-186, Springer, 2006

[8] Collobert & al. *A unified architecture for natural language processing: Deep neural networks with multitask learning.* Proceedings of the 25th International Conference on Machine Learning, pp. 160-167, ACM, 2008

[9] Grefenstette & al. *Multi-step regression learning for compositional distributional semantics.* Conference on Empirical Methods in Natural Language Processing, 2013

[10] Joshua Chin *NBSVM implementation*, https://github.com/Joshua-Chin

[11] Pedregosa & al. *Scikit-learn: Machine Learning in Python.* JMLR 12, pp. 2825-2830, 2011.

[12] Angela Chapman *Bag of Words Meets Bags of Popcorn*, https://www.kaggle.com/kyen89/2-sentiment-analysis-word2vec/data?fbclid=IwAR1pywAjPu3N6lYiATTZouVpOvcNSYeOdYGgZl6zOhEJehioZ-$_K9CCrsp8$

[13] Leonard Richardson *Beautiful Soup 4.7.1*, https://www.crummy.com/software/BeautifulSoup/#Download