Efficient Use of Word Embeddings for Short Text Classification

Thomas Beucher, Gil Katz

Abstract

Text representation is an important step in many Natural Language Processing (NLP) tasks. Leveraging word embeddings, a new approach for text representation is proposed, which can be considered as a generalization of count-vector representation, while allowing for the consideration of syntactic and semantic similarities between words. Having this new representation be similar in format to count-vectors allows for its use with simple and computationally efficient down-stream algorithms, enhancing their performance. It is shown that the proposed representation can be created efficiently in run-time, thus allowing for its use in real-time applications, a feature that distinguishes it from the state of the art.

Using intent classification as downstream task, the Naïve Bayes and Logistic Regression algorithms are chosen to visualize and demonstrate possible performance enhancements. It is shown that while adapting the algorithm's hyper-parameters to each dataset is beneficial, making a global choice for all datasets still improves results significantly, in relation to traditional NB/LR classification over count-vector representation.

1 Introduction

Text classification is a central task of many natural language processing systems. For typically short text, it becomes essential to maximise the use of all available information in each input in order to correctly understand and classify it. In this paper, we concentrate on intent classification, a task in which typical inputs are short and quick responses are required.

In recent years, both classical machine learning and deep-learning based approaches have been tried in order to face the challenge of text classification. While neural network based approaches including Feed Forward Networks (Wiener et al., 1995), Convolutional Neural Networks (CNN) (Kim, 2014), Recurrent Neural Networks (RNN) (Dai and Le, 2015) and Hierarchical Attention Networks (HAN) (Yang et al., 2016) often offer performance gains, they may require more training data, resources and time than traditional approaches. Algorithms such as decision trees (Lewis and Ringuette, 1994), k-nearest neighbors (kNN)(Cover and Hart, 1967; Tan, 2005; Yang and Chute, 1994) and support vector machines (SVM) (Joachims, 1998) may perform well enough in some cases, and offer advantages in those aspects. In this paper, we will focus on improving the performance of simple classification algorithms through efficient text representation.

In (Feng and Wu, 2016), the authors show a significant gain in classification performance through the use of Word Embeddings (Bengio et al., 2003). Using the Naïve Bayes (NB) algorithm (Lewis, 1998; McCallum et al., 1998) for classification, when an out-of-vocabulary word is encounted during inference, the proposed solution searches in the embedding space for a close in-vocabulary word, in order to replace it. Thus, the significant weakness of NB, whereby unseen features, encountered during inference, are ignored, is mitigated. Ignoring a word, for example due to a typo, may occur frequently, especialy when considering typically short and conversational texts. By leveraging the embedding space, the authors present significant performance gains. However, as the solution is based on the calculation

of distances in a high-dimensional space, it may be hard to scale when the known vocabulary grows, especially for real time applications.

In order to solve this problem and leverage any unknown words viewed during inference in real-time applications, we present in this paper a new efficient approach to text representation using word embeddings, through the creation of surfaces in the word-embedding space. Indeed, as word embeddings are created in order to model information about linguistic relationships between words, including both semantic and syntactic relations (Senel et al., 2018), we show that the proposed text representation approach may lead to significant performance gains in text classification tasks over different datasets.

Starting from a basic representation of a text segment as a bag of words, a vector of the size of the known vocabulary can be created for each document, and populated by counting the number of times each word appears in the text. Leveraging the power of word embeddings, we show that by using embedding surfaces (ES) this vector can be populated efficiently also with information derived from unknown words, by implicitly considering their similarity with all known words. While we use NB as the classification algorithm of choice throughout this paper, we emphasize that the proposed text representation approach can be used with most classification algorithms. We demonstrate this in Section 5 by also considering the Logistic Regression (LR) algorithm.

The rest of this paper is organized as follows: Section 2 presents the Naïve Bayes classifier and related work done to introduce word embeddings in text classification. Section 3 describes our proposed approach conceptually, while Section 4 discusses implementation aspects. Experimental results are presented in Section 5. Finally, concluding remarks are given in Section 6.

2 Preliminaries

2.1 Notation

Throughout this paper, vectors are denoted by $\vec{u} = (u_1, u_2, ..., u_n)$ where u_i denotes the i-th component. Bold letters represent matrices as well as tensors, with their dimensions as a superscript when necessary $\boldsymbol{A}^{n \times m} = \{a_{i,j}\}_{i \in \{1,...,n\}, j \in \{1,...,m\}}$. $\boldsymbol{I}$ denotes the identity matrix. The probability of an event A is denoted by $P(A)$, while the conditional probability of event A given event B is denoted by $P(A|B)$.

2.2 Naïve Bayes Classification

A Naïve Bayes classifier is a supervised learning algorithm based on the use of Bayes' theorem with the assumption of conditional independence between features (Duda and Hart, 1973) (Lewis, 1998). NB can be considered to be theoretically optimal when the independence assumption holds and a-priori class probabilities, as well as feature probabilities per class, are precise. However, even in some cases where the independence condition is not respected, classification performance is not necessarily degraded, as shown in (Domingos and Pazzani, 1996).

Trying to find the class c of a document $\vec{d}$ out of a set of $\mathcal{C}$ possible classes under the assumption of variable independence, the NB decision rule can be expressed as follows:

$$\hat{c} = \arg\max_c P(c|\vec{d}) = \arg\max_c \prod_{i=1}^{n} P(d_i|c)P(c) \ . \tag{1}$$

In this paper, we use the Multinomial Naïve Bayes Classifier (MNB) (McCallum et al., 1998), which assumes that $P(d_i|c)$ follow multinomial distributions. This is a popular choice when the data can be represented through feature counts, such as in the case of text classification, where documents can be represented as bags of words, as discussed above. In MNB the distribution $P(c|\vec{d})$ can thus be expressed through the vector $\vec{\theta_c} = (\theta_{c1}, ..., \theta_{cn})$, where n is the number of features (distinct words in the vocabulary) and $\theta_{ci} = P(d_i|c)$.

In run-time, MNB classification can be thought of as a multiplication of a vector representing the document with a coefficient-vector per each possible class. The decision rule can thus be expressed as :

$$\hat{c} = \arg\max_c \vec{E} \cdot \vec{\theta_c} = \arg\max_c \sum_i E_i \theta_{ci} \ , \tag{2}$$

where $\vec{E}$ is the count vector representing the document $\vec{d}$ by counting the number of apparitions of each feature (i.e., token) d_i. In this paper, we start by using token count-vectors as our text representation, and propose an efficient way to transform them, in order to also take information about unseen words into consideration.

2.3 Naïve Bayes and Word Embeddings

Using NB classification with token count-vector representation, coefficients are only learned for words that are present in the training dataset. This is a severe limitation of the approach, as users of intent classification systems are likely to express themselves in ways that are not present in training. Indeed, if a user uses a word that was not seen in training, whether because of a typo, choosing to use a synonym to a known word, or being grammatically or orthographically incorrect, the user utterance would contain words that will not be leveraged to find what the user intended.

To address this problem, a new feature filtering that uses Word2Vec word embeddings (Mikolov et al., 2013) was proposed in (Ge and Moh, 2017). By performing feature reduction through the clustering of similar features, it was shown that performance gains can be achieved for different types of text classification algorithms. Furthermore, by using the embedding space, it is possible to leverage any new word viewed during inference. Specifically, (Feng and Wu, 2016) introduces a way to improve the calculation of the probability that a word belongs to a class c by using the words closest to it in the embedding space, out of all known words. The conditional probability that the existence of an unknown word w is the result of the text belonging to class c can thus be expressed as:

$$P(w|c) = \sum_{k=1}^{|V|} f(sim(w, w_k))\theta_{ck} \ , \tag{3}$$

where $sim(w, w_k)$ is a measure of the similarity between the word w and each word in the known vocabulary w_k, $k \in \{1, \ldots, |V|\}$. $|V|$ is the size of the vocabulary V and $f(x)$ is a threshold function used to filter out words with weak relationships.

Computing this function requires the calculation of $|V|$ similarity measures per each unknown word in an input document. Considering that the vocabulary can contain thousands of known words, this calculation may require significant computational power. In order to contend with this challenge, the authors proposed a parallelized architecture of Naive Bayes based on Hadoop.

3 Text Classification through Embedding Surfaces

Considering the results achieved by (Feng and Wu, 2016), it is clear that using word embeddings to improve Naïve Bayes text classification is beneficial to performance. The operation of the Naïve Bayes approach over a document represented through its count vector can be visualized as in Figure 1a. Each word in the vocabulary contributes with a different value for each class, while words not seen during training are ignored. Thus, known words, combined with token probability coefficients, can be thought of as delta functions in the word-embedding space, when considering NB classification (Figure 1a presents a simplified illustration over a one-dimensional word-embedding space).

Per each class, we propose to construct a function over the embedding space, using a seed function ϕ, which we choose arbitrarily. This function is superimposed over each of the points representing known words in the space. The sum over all of the known words gives the resulting surface. As an example, consider the Gaussian seed function. For each class, by summing the Gaussians superimposed over all the words in the vocabulary, each with its respective coefficient, we obtain a surface in the embedding space, representing the class. This process can be visualized as in Figure 1b.

Another example for an interesting seed function can be based on the sigmoid function:

$$\phi(d) = 1 - sigmoid(d) = \frac{1}{1 + e^{\lambda(d-\sigma)}} \ , \tag{4}$$

where d is the distance between two words in the embedding space, and σ is a hyper-parameter that designates the function threshold. Manipulating the parameter λ, this function can be brought to be arbitrarily close to a step function, which simulates the work in (Feng and Wu, 2016).

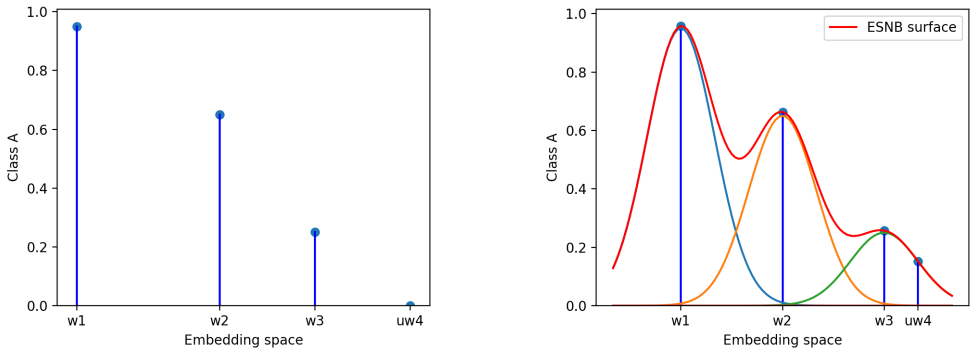

Figure 1: Simplified visualization of Naïve Bayes classification rule with count-vector (a) and Embedding Surface (b) representations. Words w1, w2 and w3 were seen in training, word uw4 is unknown.

Contrary to the sigmoid function, the Gaussian seed function allows for a smoother use of the learned coefficients, which may be beneficial, when compared to a hard limit. Using the multivariate Gaussian, the expression for computing the influence of a word on a given training dataset is:

$$\vec{E}_w = e^{-\frac{1}{2}(\vec{x}-\boldsymbol{\mu})^t \boldsymbol{\Sigma}^{-1}(\vec{x}-\boldsymbol{\mu})} \,, \tag{5}$$

where $\boldsymbol{\mu}^{|V|\times m_e}$ is constructed from the word embeddings, of size m_e, of each of the words in the known vocabulary by stacking them as columns. $\vec{x}$ is the embedding of the current word being evaluated w, and $\boldsymbol{\Sigma}$ corresponds to the chosen covariance matrix of the Gaussian seed function. The operation $\vec{x} - \boldsymbol{\mu}$ can be practically performed through a simple broadcasting, consisting of the repetition of vector $\vec{x}$ along the missing dimension to match the dimension of $\boldsymbol{\mu}$. The result of the calculation on the right-hand side of Equation 5 is a vector of size $|V|$, which represents the influence of the word w on the document representation $\vec{E}$. Choosing the covariance matrix $\boldsymbol{\Sigma}$, it seems apparent that different words should influence their environment differently. However, it is hard to arbitrarily discriminate between words, and even harder to discriminate between different dimensions of the same word. We thus arbitrarily choose to define the covariance matrix for all words to be $\boldsymbol{\Sigma} = diag(\sigma^2)$, where σ is a scalar hyper-parameter of the algorithm.

In order to calculate the value of the word w on the surface of class c, we simply calculate the dot production $\vec{E}_w \cdot \vec{\theta}_c$. In practice, in order to classify a full entry, it is more efficient to first sum over the words in the entry $\vec{E} = \sum_w \vec{E}_w$ and then multiply by the coefficient vectors θ_c, as seen in Equation 2. This is the process we follow subsequently in Section 4.

Note also that cosine similarity is considered to be more significant than the Euclidean distance to determine the similarity between different words based on their embeddings. Thus, we chose to normalize all embeddings, making the distance between two words inversely proportional to their cosine similarity, and thus equivalent to it (Levy et al., 2015). This choice is consistent with the results in (Wilson and Schakel, 2015), which found that for similarity and word relation tasks, using normalized word vectors improves performance.

4 Implementation

Implementing the solution proposed in Section 3, it turns out that all steps of computing the representation can be done efficiently in run-time, leaving the training process of MNB unchanged. Another advantage of this approach, as will be seen subsequently, is that the resulting representation is not dependent on the MNB algorithm. As the final text representation can be seen as a generalization of count vectors, it can be used to enhance the performance of any approach for classification over them, such as SVM, LR and more.

Algorithm 1 Embedding Surface Naïve Bayes

Input: Text document τ, Word embedding $\vec{\mu}_i$ for each word $i \in V$, MNB coefficient vectors $\vec{\theta}_c$ for each $c \in \mathcal{C}$, Variance hyper-parameter σ

Output: $\hat{c} \in \mathcal{C}$, the predicted class for the document τ

1: Build the representation matrix $\boldsymbol{\mu} = \{\mu_{i,j}\}$ by stacking the known vocabulary's word embeddings as columns
2: **for** word w in τ **do**
3: Retrieve $\vec{x}$, the embedding representation of word w
4: Define $\boldsymbol{N} = \vec{x} - \boldsymbol{\mu}$
5: Calculate $\vec{E}_w = e^{\frac{-diag(\boldsymbol{N}^t \boldsymbol{N})}{2\sigma^2}}$
6: **end for**
7: Calculate the representation vector for document τ, $\vec{E} = \sum_{w=1}^{n} \vec{E}_w$
8: Find the class of τ, $\hat{c} = \arg\max_c \sum_i E_i \theta_{ci}$
9: Return $\hat{c}$

Algorithm 1 presents the classification process, assuming no change was done to the training of MNB. We start by creating the matrix $\boldsymbol{N}$ through broadcasting, as described in Section 3. We next calculate the influence of the current word on each of the locations in the representation vector, each referring to a known vocabulary word, as in count vectors. In this instance, we do so by using Gaussians as a the seed function, although other seed functions can also be chosen. As we choose the covariance matrix of each of the Gaussians in the process to be $\Sigma = \sigma^2 \boldsymbol{I}$ as to not unjustly discriminate between words, the resulting calculation of Equation 5 can be simplified as seen in step 5 of Algorithm 1. The final step of the algorithm returns the predicted class through multiplication with the learned MNB coefficients. Note that all previous steps create a general representation to the input text τ, and are not specific to MNB. Moreover, seeing that σ corresponds to the width of the Gaussians, setting σ close to 0 recreates the basic tokens count vector representation.

Remark 1 *For readability, Algorithm 1 operates over the tokens in the input document τ separately, summing their influence on the final representation at the end. It is however possible to improve efficiency by directly considering the entire input document through tensor operations. Thus, instead of looping over the words in the document, we define $\boldsymbol{X}^{m_e \times 1 \times |\tau|}$ to be the stacking of the word-embeddings over all input words in τ. Here, m_e is considered to be the number of dimensions in each word embedding vector and $|\tau|$ the length of the input document, in number of tokens. By subtracting the vocabulary matrix $\boldsymbol{\mu}^{m_e \times |V| \times 1}$ as is done in step 4, both $\boldsymbol{X}$ and $\boldsymbol{\mu}$ are broadcasted, each in the necessary dimension, in order to create the tensor $\boldsymbol{N}^{m_e \times |V| \times |\tau|}$. Thus, a single computation of step 5 is now required in order to compute the resulting text representation for the document τ.*

Furthermore, note that the operation to extract the diagonal $diag(\boldsymbol{N}^t \boldsymbol{N})$ can be efficiently accomplished by taking the Hadamard product of $\boldsymbol{N}$ with itself and summing over the first dimension.

5 Results

We use fastText word embeddings (Bojanowski et al., 2016) with 300 dimensions. Using fastText presents the advantage of being able to produce representations for words that were never seen during the embedding training phase by combining n-gram representations to create word representations. We benchmark our proposed approach for text representation through the task of intent classification on short text entries, comparing the same NB model while using regular count vectors and the embedding-surface approach. Four different datasets are considered, of which three are standard benchmarking datasets (Braun et al., 2017) and the forth is a starter-bot available on the SAP Conversational AI platform (SAP-Conversational-AI, 2019). The number of intents per dataset ranges from 2 (ChatbotCorpus) to 26 (starter-skills). Vocabulary size as well as the number of total expressions in each dataset are in the hundreds, where the average number of tokens per expression is in the single digits.

Dataset	Count	ES (Gaussian, $\sigma = 0.33$)	ES (Gaussian)	ES (Sigmoid)
AskUbuntuCorpus	0.843	0.858	0.858 ($\sigma = 0.33$)	**0.859** ($\sigma = 0.74$)
starter-skills	0.731	0.742	**0.745** ($\sigma = 0.34$)	0.727 ($\sigma = 0.51$)
WebApplicationsCorpus	0.646	0.694	**0.776** ($\sigma = 0.37$)	0.703 ($\sigma = 0.50$)
ChatbotCorpus	0.976	0.99	**0.995** ($\sigma = 0.32$)	**0.995** ($\sigma = 0.56$)

Table 1: F1-scores obtained on benchmarks for the different datasets, with NB as the classification algorithm.

Dataset	Count	ES (Gaussian)	ES (Sigmoid)	WE-Mean	FT
AskUbuntuCorpus	0.86	0.89	0.88	0.87	**0.91**
starter-skills	0.76	**0.84**	0.78	0.83	0.73
WebApplicationsCorpus	0.74	0.82	**0.83**	0.82	0.78
ChatbotCorpus	0.99	**1**	0.99	0.99	**1**

Table 2: F1-scores obtained on benchmarks for the different datasets, with LR as the classification algorithm, compared to fastText results

To gather the benchmark results, we remove any class with less than 5 expressions and perform a stratified 10-fold $90/10$ cross-validation. We test several versions of the proposed ES approach: For both seed functions discussed in this paper, we search for a well-performing σ hyper-parameter per dataset. In the case of the Gaussian seed function, we also compare results under a stricter constraint, which demands a fixed σ across all datasets. In Table 1, the F-1 scores of all experiments are compared. While both seed functions present improvements over count-vector representation, experiments show that the smooth nature of the Gaussian function offers performance gains even when a single σ hyperparameter is choosen across all datasets.

Remark 2 *Choosing $\lambda \gg 0$ (in our experiments $\lambda = 100$) makes for a steep Sigmoid function, which approaches a step function. Thus, the solution proposed in this paper becomes similar, but not identical, to the one proposed in (Feng and Wu, 2016). Unlike the work done in (Feng and Wu, 2016), we do not choose the closest vocabulary word in order to represent an unknown word. Thus, an unknown word can be represented by a non one-hot vector. Moreover, even a known word can influence the vector in several different locations, as discussed above. However, these effects are improbable for steep and narrow Sigmoids.*

The benchmarks seen below were written in python and run on a regular personal computer (Intel(R) Core(TM) i7-7820HQ CPU @ 2.90GHz). With this configuration, using ES vectorization makes for an increase in prediction time of about one order of magnitude with relation to traditional count-vector representation, while staying fast enough to perform real-time predictions (on average $9ms$ with the smallest vocabulary size dataset and $30ms$ for the largest). Furthermore, prediction time increases linearly with the vocabulary size.

Another popular approach to simple text classification using word embeddings is through the averaging of all embeddings in the entry and the usage of a Logistic Regression classification algorithm. In fact, this is the approach taken by the built-in classifier of the fastText package, with some modifications (Joulin et al., 2016). In order to compare ES to this approach, we choose to combine it with a LR classifier. We compare the results to "vanilla" LR over averaged word embeddings, as well as the fastText classifier. The main difference between these two classifiers is that the fastText approach allows for fine-tuning of the word embeddings during the classifier training phase. This aspect can also be considered as future work for classifier training over ES representation. Table 2 presents benchmark results over the same datasets, with LR as the classification algorithm. LR classification results over count-vectors are also added as a baseline. Results for the fastText classifier were averaged over 5 runs, with hyperparameter search over the learning rate, the number of n-grams and the number of epochs for each dataset.

It can be seen that the ES approach, combined with LR classification, outperforms the vanilla LR over averaged word embeddings approach, and is on-par with the fastText classifier. This is despite the fact that the fastText classifier fine-tunes word representations as a part of the training process. Considering other types of text classification tasks, where both the vocabulary size and average document size may be significantly bigger than in the case of intent classification, we found good results using ES (e.g., a 26% improvement with relation to the fastText classifier on the AG-News dataset, provided by Dataturks (Dataturks, 2019).

6 Concluding Remarks

In this paper, Embedding Surfaces were presented as an efficient way to create text representations for intent-based classification tasks. Using NB as an example to a simple and quick classification approach, our proposed approach can be visualized as a surface construction, for each class, over the word embedding space, which allows to quicky compare any new document words to the known vocabulary, creating an enriched sentence representation. Implementing the proposed vectorization scheme with tensor operations allows for fast and efficient computations. Using several open text classification datasets, performance enhancements were shown for both the NB and LR classification algorithms. Moreover, the computational efficiency of our approach allows for real time applications of our algorithm.

Given a choice for the hyper-parameter σ, the same value is used across all words and all dimensions. Thus, an identical seed is artificially constructed around each known word in the vocabulary. In future work, intelligent discrimination between words and between dimensions can be considered, specifically using recent work on distributional word embeddings (Vilnis and McCallum, 2014; Athiwaratkun and Wilson, 2017; Athiwaratkun et al., 2018). In addition, training the classifier in conjunction to fine-tuning the word embeddings of vocabulary words can be considered using ES representation.

References

Ben Athiwaratkun and Andrew Gordon Wilson. 2017. Multimodal word distributions. *arXiv preprint arXiv:1704.08424*.

Ben Athiwaratkun, Andrew Gordon Wilson, and Anima Anandkumar. 2018. Probabilistic fasttext for multi-sense word embeddings. *arXiv preprint arXiv:1806.02901*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *CoRR*, abs/1607.04606.

Daniel Braun, Adrian Hernandez-Mendez, Florian Matthes, and Manfred Langen. 2017. Evaluating natural language understanding services for conversational question answering systems. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 174–185, Saarbrücken, Germany, August. Association for Computational Linguistics.

Thomas Cover and Peter Hart. 1967. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27.

Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. In *Advances in neural information processing systems*, pages 3079–3087.

Dataturks. 2019. News classification dataset.

Pedro Domingos and Michael Pazzani. 1996. Simple bayesian classifiers do not assume independence. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1386–1386.

Richard O Duda and Peter E Hart. 1973. Pattern classification and scene analysis. *A Wiley-Interscience Publication, New York: Wiley, 1973*.

Mengke Feng and Guoshi Wu. 2016. A distributed chinese naive bayes classifier based on word embedding. In *International Conference on Machinery, Materials and Computing Technology (ICMMCT 2016)*, pages 1121–1127.

Lihao Ge and Teng-Sheng Moh. 2017. Improving text classification with word embedding. In *Big Data (Big Data), 2017 IEEE International Conference on*, pages 1796–1805. IEEE.

Thorsten Joachims. 1998. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

David D Lewis and Marc Ringuette. 1994. A comparison of two learning algorithms for text categorization. In *Third annual symposium on document analysis and information retrieval*, volume 33, pages 81–93.

David D Lewis. 1998. Naive (bayes) at forty: The independence assumption in information retrieval. In *European conference on machine learning*, pages 4–15. Springer.

Andrew McCallum, Kamal Nigam, et al. 1998. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

SAP-Conversational-AI. 2019. Bot starter skills.

Lutfi Kerem Senel, Ihsan Utlu, Veysel Yucesoy, Aykut Koc, and Tolga Cukur. 2018. Semantic structure and interpretability of word embeddings. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*.

Songbo Tan. 2005. Neighbor-weighted k-nearest neighbor for unbalanced text corpus. *Expert Systems with Applications*, 28(4):667–671.

Luke Vilnis and Andrew McCallum. 2014. Word representations via gaussian embedding. *arXiv preprint arXiv:1412.6623*.

Erik Wiener, Jan O Pedersen, Andreas S Weigend, et al. 1995. A neural network approach to topic spotting. In *Proceedings of SDAIR-95, 4th annual symposium on document analysis and information retrieval*, volume 317, page 332. Las Vegas, NV.

Benjamin J Wilson and Adriaan MJ Schakel. 2015. Controlled experiments for word embeddings. *arXiv preprint arXiv:1510.02675*.

Yiming Yang and Christopher G Chute. 1994. An application of expert network to clinical classification and medline indexing. In *Proceedings of the Annual Symposium on Computer Application in Medical Care*, page 157. American Medical Informatics Association.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489.