Ross Abramson

Ross.Abramson@NYU.edu

Professor Meyers

May 15, 2018

“I’ve Never Heard that Word Before!”

Using Word2Vec and Doc2Vec in Text Classification

**1. Abstract**

This paper demonstrates that highly accurate text classification may be obtained through word embedding representations of documents in combination with a K-Nearest Neighbor algorithm. Additionally, this study compares and examines the differences of word embeddings vs. document embeddings in classification. This study found that through TF-IDF word extraction and Word2Vec and cosine similarity scoring to category name vectors, the model was able to achieve roughly 60% accuracy in correctly labeling a BBC news category as one of five topics. This percent, while 40% higher than the Naïve Bayes approach, was undermined by debatable human annotated categorization, similarity of news documents, and unexpected weights of connections by the Word Embedding model. Conversely, when utilizing document embeddings with a Distributed Bag of Words model in combination with a K-Nearest Neighbor classification algorithm, document vectors not only scored higher than other TF-IDF vector KNN approaches, but rivaled the dominant SVM approach by averaging 95% accuracy in classification.

*Keyterms: Keyword Extraction, Text Classification, Word Embedding, Word2Vec, Term Frequency – Inverse Document Frequency, Doc2Vec, K-Nearest Neighbor*

**2. Introduction**

Classifying elements has existed since the stone age when cavemen understood the difference between fruits and rocks. Classification has been a vital component of human history from simple areas, such as assessing safety (*harmless* or *dangerous*) or identifying food sustenance (*poisonous* or *healthy*), to even more complex fields such as identifying authors of books or composers of music. Classification is the task of outputting a correct label or category given an input. One use in particular is to classify news documents for a given set of categories. News categories are typically assigned one category, but a reader could easily attribute several areas per article. For example, given the headline, "Amazon Develops New Supermarket System," one could find an invariable amount of possible labels including: Technology, Business, Food & Health, Real Estate, Shopping, and more. It begs one to ask: how is it possible to assess a given article and assign it a proper news category?

Many methods already exist for text classification and there have been several examples of achieving an accurate categorization system. One new approach proposed in this paper is the utilization of word and document embeddings under a Bag-of-Word model with K-Nearest Neighbor. This acts as an alternative to the more traditional TF-IDF vectorization. Word embeddings capture many semantic relationships and meanings from text. The logic follows that if word vectors can capture multiple connections amongst words in a vector space, then it may be possible to use these connections to find similarities amongst words and thus identify a shared class of categories.

Unfortunately, word vectors do not capture the specific context of the word at the instance of usage. For example, without proper context, *chicken* could refer to the literal animal, a coward, or even dinner. But through the use of *document embeddings*, or *paragraph embeddings*, the entire document may be represented as a word embedding defined by all the words within the document. This would mean a word embedding with a specific and narrow context and setting as opposed to a general word embedding capturing all possible relationships.[[1]](#endnote-1) [[2]](#endnote-2)

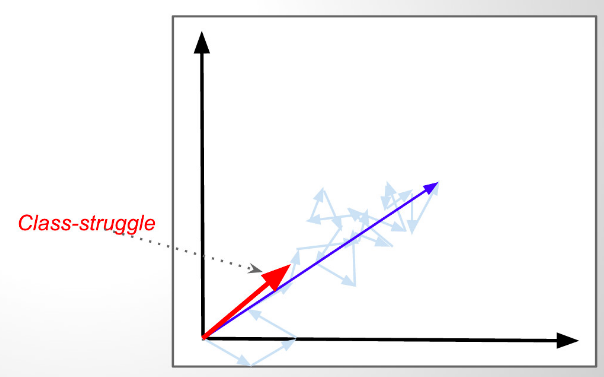


Figure 1: The above graph shows several word embedded vectors found in "The Tale of Two Cities." Each individual word has its own individual vector direction. When averaged together, we get one singular vector representative of the entire document. In this case, that average vector resembles “Class Struggle.” [[3]](#endnote-3)

The aim of the paper is to highlight the faults and issues found within word embeddings for the purpose of text classification as well as showcasing its powerful potential and successful implementation when used with a KNN approach combined with document embeddings. The paper first examines extraction of keywords and comparing their relative cosine similarity to categories. The results will show that in comparison to traditional TF-IDF vectors with KNN topic modeling, using document vectors with KNN scores significantly better.

**2. Related Work**

Many authors have already incorporated and explored the use of word embeddings for text classification. In "Bag-Of-Embeddings for Text Classification," the authors were able to implement a multi-prototype word embedding based on text classes on the Reuters news dataset and were able to achieve accuracy scores relatively close to Support Vector Machine models (SVM).[[4]](#endnote-4) Specific to this dataset, few authors have published results. In "Text Classification" by Nura Kawa, Kawa developed logistic regression, SVM, and KNN models for text classification using TF-IDF vectors, bag-of-words, and N-grams.[[5]](#endnote-5) Kawa was able to achieve above 90% accuracy for most of his models, with the exception of KNN which wasn't able to bypass 50% accuracy. In "Comparison of Text Classifiers on News Articles," the authors used the BBC News set and achieved 97% with SVM, but only 88% with KNN-- Again utilizing TF-IDF vectorization only this time with more data preprocessing and feature selection techniques such as Chi-Square selection. This paper uses both approaches as a baseline comparison for the KNN approach presented near the end of the document.

**3. Dataset: BBC**

The dataset used was the BBC news dataset. This dataset consists of 2225 short news articles for five categories from 2004-2005: ‘Business,’ ‘Sport,’ ‘Politics,’ ‘Entertainment,’ and ‘Tech.’ The dataset was chosen for its simplicity of only five categories and because the the categories ‘naturally’ feel different (i.e. sport and technology typically wouldn’t be clustered together by a human annotator). Additionally, this dataset has been relatively underused. It should be noted that the articles are not evenly distributed (Ex: sport and business articles form roughly 46% of the dataset. This is relevant more so for when introducing KNN because if a certain article type is represented more, it may skew the algorithm to label articles as sport or business.

****

Figure TF-IDF + Word2Vec Model

**4. First Approach: TF-IDF + Word2Vec**

***4.1 Preprocessing Phase***

First the user will receive all data from the BBC news articles. The articles are labeled in separate folders. The user goes through and labels all folders as falling under one of the five pre-assigned categories: Business, Sports, Politics, Entertainment, and Tech. The data is then preprocessed by removing all stop words, utilizing Named Entity Recognition, stemming words, and sending all letters to lowercase. Lastly, the words are POS tagged.

These preprocessing rules are done for two reasons. First, the goal of TFIDF, is to recognize words which are important for the article and less so for other articles, in a way, identifying its unique key terms. Removing stop words would cut down on preprocessing time. Additionally, in order to better grasp word usage, we stem words so the model would rank words like ‘cats’ and ‘cat’ as two of the same term because plurality and tense are not important—only the root of the word. Secondly, the Word2Vec model has an N-Sliding-Window. Removing stop words would diminish the distance between words in sentences and better capture word connections. Stemming would also capture more word connections by not differentiating words based on tense or plurality.

***4.2 TF-IDF Keyword Extraction***

Term Frequency-Inverse Document Frequency is applied to the preprocessed texts to get keywords. Each document is passed to the vectorizer. Emphasis was given to words labeled as adjectives and nouns. The vectorizer ignores elements found to be determinants and prepositions because these type of words tend to denote little information about the main content of the article itself. Elements which were found to be abundant in one article and not as abundant in other articles are considered highly informative. TFIDF was used as opposed to other algorithms like RAKE because Word2Vec functions off of words and not phrases. For example, “The uneducated political speech” is more informative to a reader than “speech,” but the word2vec model cannot infer a vector from the former. Thus, TF-IDF was used because of its simplicity. The top K keywords are selected based on the highest TF-IDF values. Due to the short length of the documents, 10 keywords were extracted.

***4.3 Word2Vec Model***

The word embedding models were trained using DeepLearning4Java’s Word2Vec vectorizer. The articles were passed as one text file to the Word2Vec model and word embeddings were established accordingly. Three models were generated for comparison and an already pretrained model based off of Google News documents was used. The dimensional size of the model, the size of the vector, was kept at 100. Generally, larger dimensions may add information and make a more accurate model, but this can lead to adverse effects if the given dataset is small.[[6]](#endnote-6) The amount of minimum words needed to be considered valuable towards the model and the window size, for seeing connections between words in a sentence, was altered per each model. Additionally, model 3 incorporated stemming. Once the word model was done training, each of the keywords obtained in the TF-IDF step and the Word2Vec model are passed to the classification step.

*Word Embedding Model Overview*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Min Word Feq.** | **Window Size** | **Dimensions** | **Additional Features:** |
| **Model 1** | 15 | 15 | 100 | - |
| **Model 2** | 2 | 5 | 100 | - |
| **Model 3** | 2 | 10 | 100 | Added Stemming |
| **GoogleNews** | 15 | 10 | 300 | Utilized Google News Corpus as training data |

***4.4 Text Classification***

Once we have both the top N keywords and a Word2Vec model, the algorithm scans each keyword, finds it’s most similar category judged by cosine similarity, and then ranks the most likely category for the document. Two algorithms were utilized to asses the likelihood of the likely category.

The first is what’s defined as “Total.”

***Procedure*** *Total(Keywords, Categories)*

***Input****: List of keywords, List of categories*

***Output****: Cumulative highest cosine similarity ranked category*

*A = array of size 5 {corresponding to the five categories}*

For each keyword in Keywords

For each category in Categories

A[Category] = A[Category] + CosineSimilarity(keyword, category)

End**\_**For

End\_For

Return *MaxiumCategory(A)*

***End\_Procedure***

The algorithm finds the most similar category per word. The highest cumulative total score for a category amongst the keywords is the documents category. The logic of Total is that a keyword which has a stronger match to a category should have greater weight. For example, if within three keywords, two have very low scoring similarities to all categories and the remaining keyword has a 80% similarity score to “Sports,” it would make more sense to go with the clearest connection.

The second method is known as “Fair Vote.”

***Procedure*** *FairVote(Keywords, Categories)*

***Input****: List of keywords, List of categories*

***Output****: Category based on what the majority keywords have the highest similarity.*

A = array of size 5 *{corresponding to the five categories}*

For each keyword in Keywords

B = array of size five {corresponding to the five categories}

For each category in Categories

B[Category] = CosineSimilarity(keyword, category)

End**\_**For

TopCategory = MostSimilarCategory(B) {Returns the highest similar category}

A[TopCategory]= A[TopCategory]+1 {Store most similar category as one vote}

End\_For

Return *MaxiumCategory(A) {Returns the highest category}*

***End\_Procedure***

In this approach, each keyword has a vote. If keyword ‘A’ matches category “Entertainment” the most, then there is one “vote” casted towards Entertainment, and zero votes for the other categories. We repeat the process for the other keywords. At the end, the highest scoring category is the answer. The reasoning behind this approach is to remove outliers and normalization. For example, if the majority of words’ highest corresponding scores is for category “Politics,” but one word has a larger cumulative similarity with another category, that word is considered an outlier. It may not be indicative at all of the document and the other lower scoring words, as a group, are more explanatory.

**5 Results of First Approach**

The results show that on average, Fair Vote with a combination of higher number of keywords performed the best. Total performed slightly worse, performing better when given the top 2 keywords. Of all the models, Model2 performed the best, breaking through to the 60% correct rate under Fair Vote. Surprisingly, GoogleNews, despite a larger corpus performed worsed. This is likely due to the fact that GoogleNews has a much larger dictionary and thus more relationships represented within the vector space (This is further explained in the error analysis section).

***Best Results Per Model - Total***

|  |  |  |
| --- | --- | --- |
| ***Name of Model*** | ***Top N Keywords*** | ***% Correct*** |
| Model 1 | 2 | *.5452* |
| **Model 2** | ***2*** | ***.5735*** |
| Model 3 | *3* | *.5243* |
| GoogleNews | *7* | *.3281* |

***Best Results Per Model – Fair Vote***

|  |  |  |
| --- | --- | --- |
| ***Name of Model*** | ***Top N Keywords*** | ***% Correct*** |
| Model 1 | 3 | *.5142* |
| ***Model 2*** | ***6*** | ***.6049*** |
| Model 3 | *10* | *.5888* |
| GoogleNews | *9* | *.5299* |

**6 Evaluating Results: The Issues of Word Embeddings**

As shown, word embedding’s can be incredibly effective, yet they still perform signifigantly worst than the typical SVM and LDA models which can achieve in some cases more than 90% correctness. The reasons and issues further explained are: the algorithm may pick up on connections not considered with human annotators and the topic of a documents can be ambiguous or have multiple categories.

**6.1** ***Facebook is business, technology, and entertainment:* *Most News Documents aren’t just one category.***

The issue with pinpointing the correct category is that a category could have a justifiably equal connection to multiple categories. Take for example this title from one document: “Ink helps drive democracy in Asia.” The remainder of that article talked about the relationship of ink and the coming election in a certain area of Asia. The majority of readers would inherently think the category of the article is politics, but the article is actually talking about the technology of ink and thus is labeled as “tech.” Esoteric category labeling was found abundantly within the news corpus. To visualize the space of these documents, Gensim’s Doc2Vec library was used along with PCA and K-Means. The results show that it may be possible to use a KNN approach of labeling new incoming documents which place on the outer rims of the cluster. But for those documents which fall closer to the center, the model would struggle to solve.

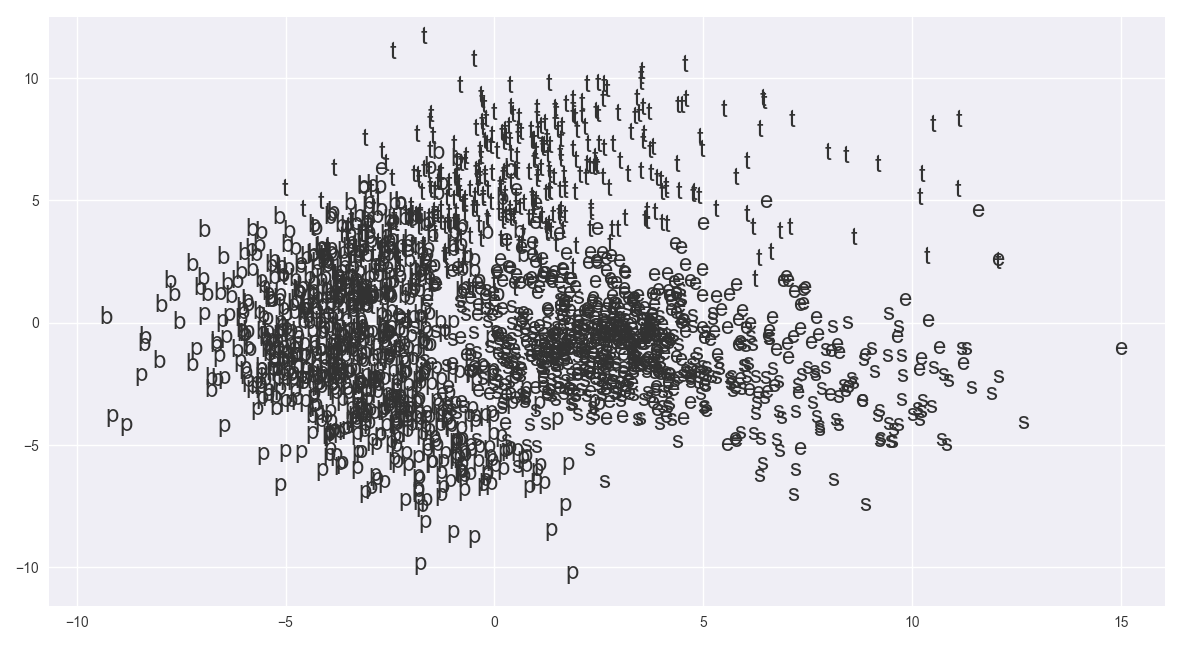


Figure 3: PCA Representation with single character labels. Notice the central overlap.

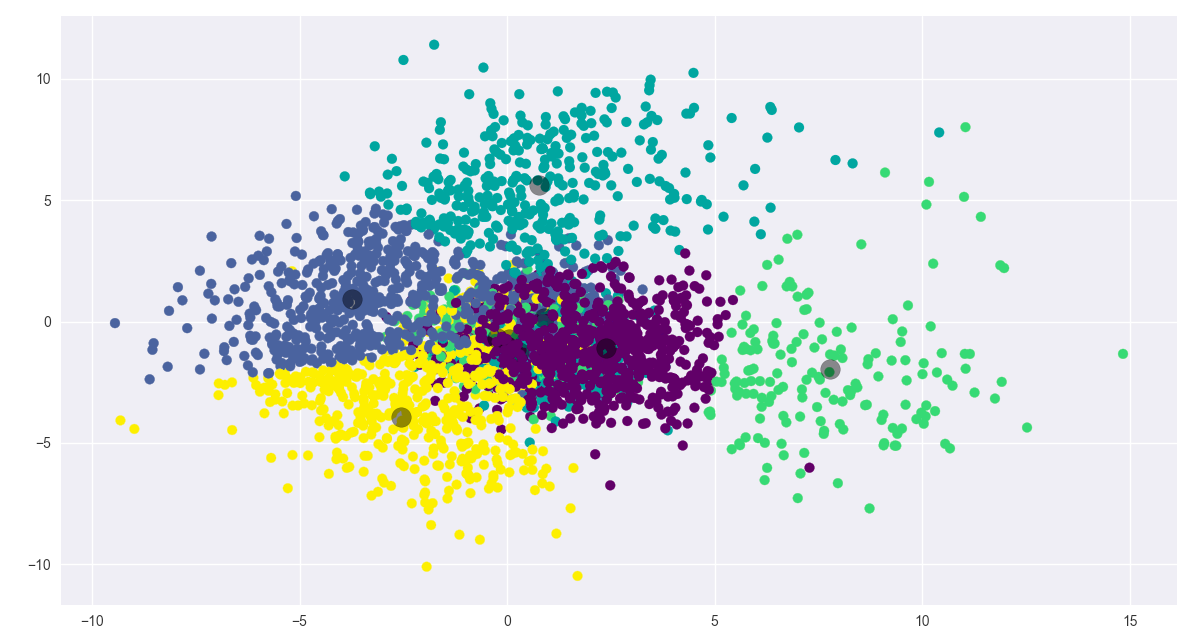
**

Figure 4: K-means of PCA Representation. Centroids are darkened circular centers. Notice how the center overlaps.

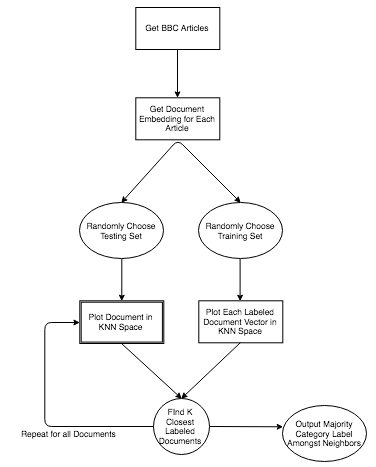
* 1. ***Word2Vec Sees Indirect Connections***

The average reader, especially consisting of a factual source of information like news, sees the most immediate connection first, and *only then* considers less obvious constructs. For example, consider the first word that comes to mind when reading “hot.” Trained on the Gutenberg Text, a Word2Vec model outputted: [cold, warm, cool, heat, wet, weather, fresh, chilly, dry, damp]. The answers listed are not things which would be considered ‘hot,’ like volcanoes, fire, sun, or things that are considered hot. Instead, the model placed greater weight on descriptions of weather than things closer associated with the word. This explains why GoogleNews performed the worst as its dictionary and corpus size are several thousands times larger than the BBC trained models—meaning, it has a much larger scope of relationship inference contained within each word vector.

**7. Improved Model: Document Embedding + KNN**

The primary issues found within the first approach is the lack of context and ambiguity of word vectors. This may be resolved by vectorizing the entire document into a word embedding space, thus capturing the context and theme of the entire document as opposed to individual word vectors. The unlabeled document embedded vector is then mapped to a graph along with other pre-categorized documents. The unlabeled document takes on the label of the K nearest labeled documents. This is the k-nearest neighbor model with document vectors as inputs.

The results show not only a 35% improvement in classification but also a viable KNN classification approach that beats the traditional KNN utilizing TF-IDF vector algorithm on the same BBC dataset.

***7.1 KNN Overview.***

KNN functions by first plotting several pre-labeled inputs onto an N-dimensional graph. An unlabeled input will be plotted within the graph. We check for its closest K neighbors. The label of the unmarked input will be whatever is found to be the majority category amongst the closest K neighbors.

***7.2 Improved Second Approach:* Doc2Vec + KNN Model**

First the documents are vectorized using Gensim's Doc2Vec with a Bag-Of-Word model. After modeling of the documents is finished, 90 percent of the articles are randomly chosen to be used as the training set and the remaining 10 percent are the test set. The test set is then labeled according to the KNN algorithm. The document model uses a vector size of 100 dimensions, a window size of 10, and is trained over 100 iterations with a minimum word frequency of 1 to capture all words.

**7.3 Computing Average Top Accuracy**

The program took 1000 randomly chosen training and test sets and computed the average of the accuracy given a k. The best results were found to be around K=35 and K=105 with an accuracy score, Precision, Recall, and F1-Score averaging about 95%

Figure 5: Doc2Vec KNN Algorithm

6 ♂ 
96 ♂ 

Figure 6: Graph of average accuracy score per given k. Average accuracy was computed over a 1000 iterations of randomly selected testing and train sets. Accuracy is the worst when searching for the first closest neighbors at K=1 and peaks at K =35 and 105.

**8. Results of KNN Approach**

The KNN scored on average more than 94% accuracy amongst the top 120 K scores. The best K values were found to be 35 and 105, both achieving more than 95% accuracy, and, in some cases, even 98%. The highest scoring Recall and F1-Score was found to be Sports documents. This would make sense as sports seems, on face value, to share the least in common with the other four categories. This complements the related TF-IDF approaches as well; Sports was also found to be the highest scoring category within those articles. Technology conversely generally scored the worst in recall and F1-Score.

*Word Embedding KNN Model Compared to Others*

|  |  |
| --- | --- |
| **Model** | **Accuracy** |
| TF-IDF + N-Gram + BoW KNN | 49% |
| TF-IDF + Feature Selection KNN | 88.5509% |
| **Document Embedding BoW + KNN** | **95.5157%** |
| SVM | 97.6744% |

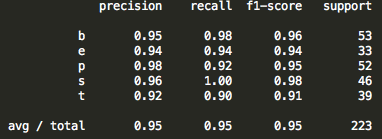


Figure 7: Precision, Recall, and F1 Score per each article type. The left most column is the first character of each category. Support is the number of article types per test set. The bottom row shows an average of 95% amongst the three.

**9. KNN Error Analysis**

As mentioned before, the core issue with document classification is that some articles are highly ambiguous. It may be that an article shares similarities to both business and politics. The KNN approach has removed most of that error but not entirely.

T o τ τε 0] 
:aaoos Koean»e  Additionally, the algorithm is highly dependent upon the initial training set. Take for example the following two confusion matrices. Both tables utilize a K value of 35, but one scored much better than the other, nearing a 98% correct rate. The reason is solely due to the randomness of what documents are chosen to be the training and test set.

=35, accuracy score: ø. 
[ e 38 ø ø 
[1 e 53 ø ell 

**10. Conclusion and Future Work**

This paper has shown the issues involved with relying solely on Word Embeddings for text classification as well as the advantage of using KNN and Document Embeddings as opposed to the traditional TF-IDF vector approach. While not yet outperforming SVM, this approach still comes within striking distance. In the future, this paper will drop the TF-IDF + Word2Vec approach and instead entirely focus on the Doc2Vec + KNN method (the reasoning of including the first approach is to highlight the process of discovery for the purposes of this course). The next phases of this algorithm will incorporate other datasets commonly used, specifically the Reuters dataset, for comparison with other mainstream algorithms. Additionally, I will introduce another metric of comparison between neighbors found and the test article, such as weighing closer neighbors as being more important than farther away neighboring articles.

**11. Work Cited**

1. Kusner, Sun, Kolkin, and Weinberger. “From Word Embeddings to Document Distances” Washington University in St. Louis. <http://proceedings.mlr.press/v37/kusnerb15.pdf> [↑](#endnote-ref-1)
2. Le, Quoc and Mikilov, Tomas. “Distributed Representations of Sentences and Documents” Google. <https://cs.stanford.edu/~quocle/paragraph_vector.pdf> [↑](#endnote-ref-2)
3. Tamir, Mike Phd. “Short Introduction to Using Word2Vec for Text Classification. LinkedIn. <https://www.linkedin.com/pulse/short-introduction-using-word2vec-text-classification-mike/> [↑](#endnote-ref-3)
4. Jin, Zhang, Chen, and Xia. “Bag-of-Embeddings for Text Classification”. Joint Conferce on Artificial Intelligence. <https://www.ijcai.org/Proceedings/16/Papers/401.pdf> [↑](#endnote-ref-4)
5. Kawa, Nura. “Text Classification” University of California, Berkley. <https://www.stat.berkeley.edu/~aldous/Research/Ugrad/Nura_Kawa_report.pdf> [↑](#endnote-ref-5)
6. Le, Quoc and Mikilov, Tomas. “Distributed Representations of Sentences and Documents” Google. <https://cs.stanford.edu/~quocle/paragraph_vector.pdf> [↑](#endnote-ref-6)