Natural Language Processing: Sentiment Classification using Document Embeddings

Thongtan, Tan, and Tanasanee Phienthrakul. "Sentiment Classification Using Document Embeddings Trained with Cosine Similarity." ACLWeb, Association for Computational Linguistics, 1 July 2019, www.aclweb.org/anthology/P19-2057/.

STATS 302 Final Project Report

gr90@duke.edu, sj288@duke.edu, zjs5@duke.edu

1 Introduction

As a venture into the methods of generating and processing human language with Natural Language Processing, sentiment analysis presents a statistical model to "understand" what a given line of text makes us feel- angry, sad, happy, cheerful- essentially to perform subjective perception with textual data. Binary sentiment classification is a further simplification of the emotional compass into two states: positive and negative. In sentiment classification models, there are several distinct steps which allow for a guided inquiry into what insights we are looking for including the pre-processing step involving textual cleaning, attribute selection for document representation and finally the form of model in which we treat words, paragraphs or other text corpus.

Each step has substantial inferential power and particularly the method in which we encode and compare text in the attribute selection part has significant influence in the performance of the algorithm. A part of this task is that of document processing and representation. Document representation is carried out with the help of document embedding models that map each documentation to a dense real-valued vector and fall into two main categories: bag of words models and neural embedding models.

Subsequently, we must devise a similarity measure for us to compare the document embeddings.A dot product between the input vector and the output vector is commonly used to calculate the similarity measure between the two vectors in learning neural n-gram and text embeddings, i.e. 'similar' vectors should have a high dot product. The use of cosine similarity instead of the dot product in computing the similarity measure between the input and output vectors is investigated in this paper. Modifications to the PV-DBOW and related DV-ngram models are the subject of this paper. Over the training set, the cosine similarity between a paragraph vector and vectors of n-grams in the paragraph is maximised.

2 Algorithm Definition

The algorithm proposed by the authors takes a string of text, creates a document embedding based on an n-gram approach then compares similarity to produce a vector embedding for documents. The document embedding is then used with a soft max function to do a sentiment analysis binary classification task.

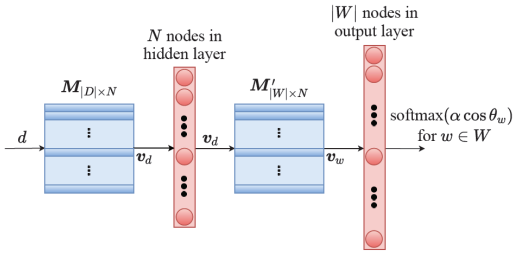

Figure 1. Proposed Architecture

Illustrated above is the proposed model architecture by the authors of the paper. There is a hidden layer with N nodes corresponding to the dimensionality of the paragraph vectors and an output layer with W nodes corresponding to the number of distinct n-grams found in the dataset. M_0 is a collection of D document vectors each having N dimensions, and $M_1$is a collection of W n-gram vectors each also having N dimensions. These are the matrices we attempt to learn through running our neural network. An input document id d is used to select its vector representation v_d which is exactly output through the N nodes of the first hidden layer. The output of each node in the output layer represents the probability $p(w|d)$ of its corresponding n-gram w which can be calculated using the soft-max function. Since we restrict the case to uni-grams our results are even more interpret-able.

3 Data Sources

The paper we have selected aims to train and test on the IMDB Dataset of Movie reviews. This is a binary sentiment classification data set that is composed of 50,000 movie reviews with binary labels. The data is composed of 25,000 highly polar movie reviews for training, and 25,000 for testing with an equal split in positive and negative reviews, such that the first 12,500

reviews are highly positive, the next 12,500 reviews are highly negative in the train and test sets respectively. While there is additional "unlabelled" data as well provided in the data set, we ignore this for now as this does not pertain to our model which is based on supervised learning rather than unsupervised learning. Besides the IMDB dataset on which our paper is based, we

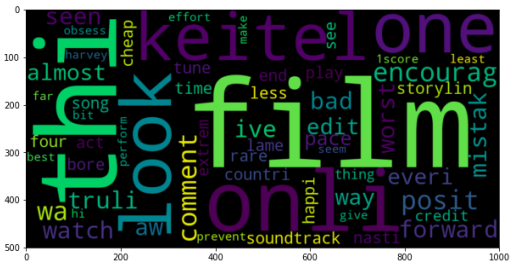

Figure 2. IMDB Dataset Word Cloud

also make use of two alternate datasets, carefully chosen from a larger list of datasets after some exploratory data analysis, on which we will train and test the model, propose improvements and illustrate overall results. As is consistent with the data science process, we started out with several other datasets but as we proceeded with the model we found flaws in these datasets so we abandoned them after partial analysis. We will include this process in the report as this allows us to better explain the model, and we later propose alternate approaches to treat the data sets we abandoned after partial analysis in the future scope of this work.

The first alternate data source is the Trip-Advisor Data Set , which consists of over 20,000 Hotel Reviews and a rating on a scale of 1 to 5 given by customers. Since our goal is still to carry out binary sentiment classification, we pre-processed this data to fit the problem formulation as explained in the Data Pre-processing section.

The second data set we used is the Twitter Data set. This data set contains 1,62,980 Tweets on political opinions towards the current political party in power in India taken from Twitter along with labels of 1 0 and -1 to depict neutral positive or negative. These labels are generated by user classification (and are evidently susceptible to human error). However, it is worth noting that lots of the tweets in this choice of dataset are not in English but are written as transliterations of some regional languages of India. This choice of dataset is interesting as it illustrates an important aspect of NLP- which is that in general, embedding models that do not rely on linguistic features such as order of sentence construction, Subject verb agreement, and are primarily formulated as statistical problems are language agnostic. It also allows us to note that to make more robust models for a specific language it is therefore worth considering linguistic features such as word order, construction etc.(More on this later!)

Besides these data sets which we fully analyse, we also worked with a data set on wine reviews. This is not as effective for reasons we explain in the next section.

3.1 Data Pre-processing

The stable implementation of the project written by the authors of the paper in Java relies on a .txt file input where each review is placed on a different line.The code treats each review as a document and creates document embeddings.The code (which used the IMDB Dataset) relies also on the fact that the first 12,500 sentiments are positive followed by the next 12,500 being negative in assigning the target vector for the model. In order to replicate this process, we took the alternate data sets we had and organised them to fit this formulation.

3.1.1 Trip Advisor Dataset

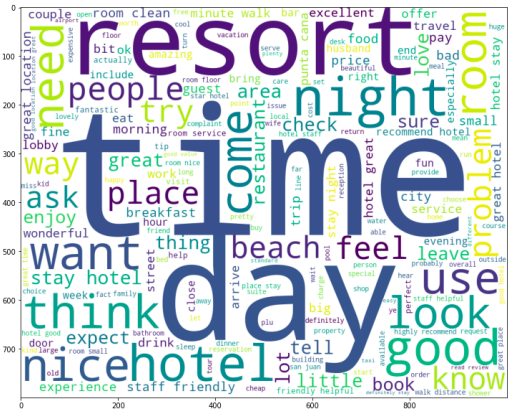

Figure 3. Tripadvisor Dataset Word Cloud

Originally, this dataset was composed of text corpus reviews and a rating on a scale of 1 to 5. The distribution of this is showcased on the figure below. Since we want to do a binary classification we can reformulate this simply as treating all '5' ratings as positive and all others as not positive. This also gives us a roughly equal spread of the data for each class. We then sort the data sequentially and make train and test vectors with an equal split of positive and non positive reviews to train our model

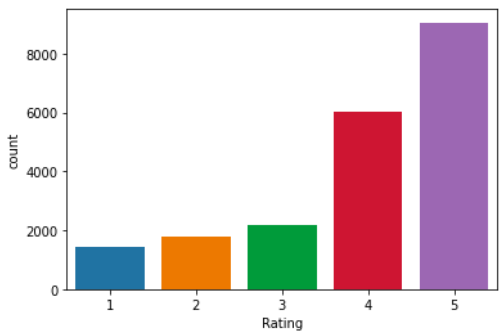

Figure 4. Distribution of Tripadvisor Reviews

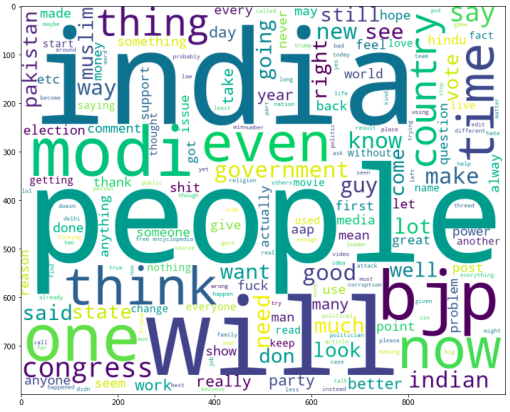

Figure 5. Twitter Dataset Word Cloud

3.1.2 Twitter Dataset

Originally, this dataset was composed of text corpus tweets and a classification of -1,0 and 1 for negative, neutral and positive. We want to take polar views and the dataset already is quite large so we simply forego the neutral reviews. This also gives us a roughly equal spread of the data for each class. We then sort the data sequentially and make train and test vectors with an equal split of positive and non positive reviews to train our model.

3.1.3 Wine Dataset

While this dataset has a wide variety of data, including geographic origin, price, name, type of grape we ignore all this data. We then simply look at the text reviews. Each wine is also rated on a scale of 100 but all reviews are between 80 and 100. However,there is one major problem with this data, even to a human reading the text it is impossible to suggest what would the wine be rated as since the evaluations are very subjective. Our conclusion: "Wine tasters are somewhat snobbish, unlike our algorithm which is therefore unable to classify this data with any acceptable level of accuracy"

3.2 External Libraries

For most of the analysis we relied on a stable implementation of the project written in Java by the authors of the paper. We also used NLTK, spaCy as well as standard Python Libraries for writing some improvements, and making visualisations. For any java implementations of the LR classifier we use the LIBLINEAR library following the authors.

4 Model Development

As explained in the algorithm definition section, the algorithm proposed by the authors takes a string of text and creates a document embedding based on an n-gram approach. An n-gram is a set of words treated as a unique entity, a uni-gram being each word treated as a unique abstract entity, a bi-gram as a pair of words forming one unique entity and so on, and being transformed into a unique vector form. Each word in the entire set of documents is assigned a real valued vector defined over D dimensions. In the original paper, the authors use 500 dimensions which are all real valued (to put this in perspective, if the authors would have simply chosen a binary vector embedding then 500 dimensions would have allowed for 2^{500} unique embeddings). Real valued vectors allow for a non-finite embedding set and provide very high flexibility. However, with our limited computational capacity we will restrict this dimension to 50, a tenth of the original value without any noticeable loss in the model flexibility and accuracy. The way a word embedding to a vector may be formed is as follows. All words have initially been assigned a unique numerical value. Let us hypothesise that we simply used a sequential approach to do this, i.e every time the model saw a unique word, it gave it a new numeric value based on the bag of words approach. Now given this structure, each sentence or document becomes a collection of numerical values. [1] Now we want to see which numbers are often seen side by side, or with some fixed gaps etc (i.e. we want to find a pattern in the structure) and then we want to classify this as some aspect of similitude. [2] Here is the part that we will use our neural network which is attempting to train words that are similar (i.e appear closer in the text, or appear after fixed gaps to be closer on some similarity measure such as cosine similarity, dot product etc). [3]First we initialise a random array of weight and bias terms. We then train the neural network by evaluating at each forward step the network parameters and then checking against the loss function as described in the paper. The loss function in essence is a depiction of the misclassification at a given point of time. In the backpropagation we then fix the weights and biases using a Gradient Descent with a learning rate determined by hyper-parameter tuning to minimize the loss function. Our gradient descent an take a host of approaches,but this is irrelevant in terms of the objective of the model except for affecting run-time. The number of iterations or passes of the neural network we do is set by hyper parameter tuning. Thus we have obtained an "understanding of words" in a relational sense.

The idea of this relational understanding is vital.What this is saying is that given the words we have been tested on we know how the are related to each other, how "near" or close they are based on our chosen similarity measure. The model has no idea in an absolute sense what is a 'car' for example, just that it is somehow related in some sense to a 'road', a 'vehicle' and an 'automobile' and not related to 'cricket' or 'Bollywood'.

Note that instead of repeating all the mathematical formulations, we summarise the intuition here as the paper we choose explains these mathematical underpinnings well and we guide the reader to see the paper if they want a full mathematical treatment. Reproducing the mathematical equations adds no additional insights.

4.0.1 Dot Product

Given any two vectors $\mathbf{u}$ and $\mathbf{v}$ such that each entry corresponds to the same attribute $u_i \triangleq v_i$, we can take their dot products to get a numerical interpretation of similarity. The dot product

between $\mathbf{u}$ and $\mathbf{v}$, $\mathbf{u} \cdot \mathbf{v}$, can be interpreted as projecting $\mathbf{u}$ onto $\mathbf{v}$, and then multiplying by the product of projected length of $\|\mathbf{u}\|$ with length of $\|\mathbf{v}\|$.

$$\mathbf{u^T v} = \|\mathbf{u}\|\|\mathbf{v}\| \cos\theta$$

Geometrically speaking, When u is orthogonal to v, the projection of u onto v is the zero vector, corresponding to a notion of zero similarity. When they are not orthogonal to each other, the largest value of $\mathbf{u} \cdot \mathbf{v}$ is observed when $\mathbf{u}$ and $\mathbf{v}$ point in the same direction. Similarly, the lowest value of $\mathbf{u} \cdot \mathbf{v}$ is observed when $\mathbf{u}$ and $\mathbf{v}$ point in opposite directions.

The objective function to be minimised is

$$\sum_{d \epsilon D} \sum_{w_0 \epsilon W_d} \left[-\log \sigma(v_d^T v_{w_0}) - \sum_{w_n \epsilon W_{neg}} \log \sigma(v_d^T v_{w_n}) \right]$$

4.0.2 L2R Dot Product

L2R Dot Product measures similarity in the same way as the Dot Product with the addition of L2 Norm regularization. The regularization appends the minimisation of the objective function by adding $\frac{\lambda}{2}\|v_{w_0}\|^2$ where λ is the strength of regularization. To the loss function, the squared magnitude of the coefficients is added as a penalty expression. As a result, the weights do not get too large.

The objective function to be minimised is

$$\sum_{d \epsilon D} \sum_{w_0 \epsilon W_d} \left(-\log \sigma(v_d^T v_{w_0}) + \frac{\lambda}{2}\|v_d\|^2 + \frac{\lambda}{2}\|v_{w_0}\|^2 \right)$$

$$-\sum_{d \epsilon D} \sum_{w_0 \epsilon W_d} \sum_{w_n \epsilon W_{neg}} \left(\log \sigma(v_d^T v_{w_n}) + \frac{\lambda}{2}\|v_{w_n}\|^2 \right)$$

4.0.3 Cosine Similarity

Cosine similarity is an improved version of dot product similarity. By dividing the $\mathbf{u} \cdot \mathbf{v}$ by the magnitudes of each vector $\|\mathbf{u}\|$ and $\|\mathbf{v}\|$, the range becomes limited.

$$\cos\theta = \frac{\mathbf{u^T v}}{\|\mathbf{u}\|\|\mathbf{v}\|}$$

This limit is the trigonometric function cosine, and since we know $\cos\theta \in [-1, 1]$, this tells us that or measure of similarity is now scale invariant, and that the arbitrary size of any one of the attributes nor the magnitude of the entire vector will not impact similarity. Instead direction, encapsulated by angle proximity, holds all of the determination in weather or not two vectors are considered similar.

The objective function to be minimised is

$$\sum_{d \epsilon D} \sum_{w_0 \epsilon W_d} \left[-\log \sigma(\alpha \cos\theta_{w_0}) - \sum_{w_n \epsilon W_{neg}} \log \sigma(\alpha \cos\theta_{w_n}) \right]$$

4.0.4 Hyper-parameters

There were a variety of hyper-parameters that could be tuned using the model.

This first set of hyper-parameters were fixed at 1 100 and 50 respectively for all data sets:

- Gram. The gram refers to the n in n-gram, so how many words we looked at at a time when assigning id's to the words. We used 1-grams for each model, so every word was viewed distinctly without the context of an adjacent word.For the purpose of our model we set this to one so as to aim for simplicity and to save computational cost.
- Batch Size. Since the program uses the steepest gradient descent algorithm for optimisation and not mini-batch gradient descent, this determines how many documents (paragraph vectors) were assigned to each thread. This does not impact the results of the model, but is related to computational power of the computer. For each case on my computer, the batch size was set to 100.
- Dimensionality. This corresponds to the number of attributes in the paragraphs vector, which is simply the vectors size. A larger number of attributes will lead to a more intricately trained model, but will run slower. We used 50 dimensional vectors when generating the models.

This following set of hyper-parameters were fine tuned by data set to improve the accuracy of the model.

- Learning Rate. This is related to the size of the jumps made by the gradient descent algorithm. Larger learning rates mean that there are more change per iteration of the optimisation problem, leading to a faster but potentially more inaccurate solution. Likewise, a smaller learning rate leads to smaller jumps in descent and thus a slower algorithm, but potentially more accurate solution.
- Iterations. This is how many times the data was iterated through, one forward and backward pass, through the neural network.
- LR Annealing. This is a binary parameter that can either be true or false. When set to true, it decreases the learning rate of the SGD as we get closer and closer to the minimum. It is only set to true for dot product. As the learning rate for the cosine similarity and L2R dot product is already small to begin with, convergence itself requires a large number of epochs. So, linear annealing is not used.

In addition to these parameters, there were two additional hyper-parameters that were mode specific: alpha and lambda.

- Alpha was used only for cosine similarity. Since cosine similarity ranges from -1 to 1, using the cosine similarity term alone as an input to the soft-max function may not

be enough in modelling the conditional probability distribution. Therefore, it is a weighting added to the cosine similarity value $\cos\theta$ which increases the possible range of possible probability for each conditional term.

- Lambda was used only for L2R dot product. Since this is a regularised dot product, the lambda value is the parameter dictating the strength of regularisation.

The paper had determined the best set of hyper-parameters that give the best accuracy on the IMDB data set. They used a cross validation technique where a range of 5 different learning rates, 5 different iterations quantities, 3 different alpha values (only applicable to cosine similarity) and both linear annealing cases were done. This would effectively mean that the model was trained 50 times with the model giving the best accuracy being selected. (150 times if its mode was set to cosine similarity instead of dot product or l2r dot product). Thus when we ran the model ourselves on a subset of the IMDB data, we used the same ideal hyper-parameters as described in the paper.

As for the other data sets, we began with the assumption that these hyper-parameters would give the best results as well on the other data sets, then varied one parameter at a time. For example we would change the learning rates and hold all the other parameters constant. This meant that we trained the model 5 times. Then picking the best, we then held that one constant and repeated this procedure but instead by varying the iterations parameters and so on. This method resulted in us needing to train fewer models since training 50 or 150 was not computationally feasible on our own computers. This time saving procedure was also based off the assumption that each variable is independent, which will be discussed further in a later section.

4.1 Final model and prediction

Based on the procedure illustrated above following the architecture showcased in the Algorithm Definition section, we have realised a unique vector representing with some relational meaning in higher dimensional space that can now be applied to a soft-max function to carry out the binary sentiment classification.

Based on the procedure explained above, the final

5 Results

We now summarise our results based on running the algorithm for the dataset of the authors as well as our own datasets. Our results match our expectations and support the papers findings. Moreover, we are able to get a sense of the run-time and accuracy trade-off first hand which we will talk about with more detail in the discussion section.

5.1 IMDB Data Set

The IMBD dataset performed with the highest accuracy at around 89 percent for all methods, cosine similarity, dot product and L2R Dot Product, with Cosine similarity being marginally better than the other two methods and Dot Product being marginally worse. This was expected as it corroborates the results of the paper.

	Cosine Similarity	Dot Product	L2R Dot Product
Gram	1	1	1
Batch Size	100	100	100
Dimensionality	50	50	50
Learning Rate	0.001	0.25	0.025
Iterations	120	10	20
Linear Annealing	false	true	false
Alpha	6	-	-
Lambda	-	-	0.01
Accuracy	89.45	89.0	89.1

Table 1. IMDB Data set Optimal Hyper-parameters and Results

5.2 Alternate Data Sets

5.2.1 Tripadvisor Data Set

	Cosine Similarity	Dot Product	L2R Dot Product
Gram	1	1	1
Batch Size	100	100	100
Dimensionality	50	50	50
Learning Rate	0.001	0.25	0.025
Iterations	120	10	20
Linear Annealing	false	true	false
Alpha	6	-	-
Lambda	-	-	0.01
Accuracy	79.36	79.22	79.22

Table 2. Tripadvisor Data set Optimal Hyper-parameters and Results

Hotel Reviews are a neat way to explore the efficacy of our model. They are also easy to find and tend to use words that are good and conveying emotion since reviews tend to be written from either an excellent experience or a terrible one that motivated the desire to write one. This data set received around 79 percent accuracy for all methods with cosine similarity also performing marginally better over the other two methods. However, this data set performed worse than the IMBD set, but the primary reason for this may be due to the fact that the data set used was smaller than the IMDB data set.

5.2.2 Twitter Data Set

	Cosine Similarity	Dot Product	L2R Dot Product
Gram	1	1	1
Batch Size	100	100	100
Dimensionality	50	50	50
Learning Rate	0.001	0.25	0.025
Iterations	120	10	20
Linear Annealing	false	true	false
Alpha	6	-	-
Lambda	-	-	0.01
accuracy	70.38	68.35	67.18

Table 3. Twitter Data set Optimal Hyper-parameters and Results

The Twitter data set was taken from a collection of tweets sent from India around the time of its elections, so each data point is relatively short, capped at Twitter's 280 character post

limit, and unlike something more relatively formal such as a trip advisor or movie review, these contained spelling mistakes and a mix of Hindi and English. Because of this the program which is not expected to handle a spelling mistakes, treats these as new and separate words. Taking into account the fact that there is less data encodable per document, less chance of repeat words because of spelling errors and an increased vocabulary for the neural network to deal with (Hindi and English), we expected the model to be relatively inaccurate since the frequency of same words it sees is not great. However, the accuracy was near 70 percent for cosine similarity and the high 60s for the other two methods, going beyond our expectations given the conditions under which the raw data was in, and also conforming to the results of the paper, claiming that using cosine similarity would give the best accuracy in the model.

6 Conclusion

We concur with the authors results that the cosine similarity measure is the best individual model for getting highest accuracy. As summarised in the tables above we see that the highest accuracy for each data set is achieved on the cosine similarity model. A possible explanation could be that cosine similarity can aid in the reduction of overfitting to the embedding task, resulting in more useful embeddings. [3][4]

To begin with, cosine similarity acts as a regularization mechanism; by ignoring vector magnitudes, there is less reason to increase the magnitudes of the input and output vectors, while in the case of dot product, vectors of frequently occurring document-n-gram pairs can be rendered to have a high dot product simply by increasing their magnitudes.Overall, the weights learned should be lower. Large numbers of document vectors which make it more difficult for the end classifier to fit in a way that generalises well, which could explain why cosine similarity and L2R dot product outperform dot product on the IMDB dataset.

After getting the results for the singular similarity measures, we also decided to test our uni-gram approach in combination with the Naive Bayes Weighted Neural Bag of ngrams (BON) to introduce a better accuracy in the system.

7 Comparisons

We now compare these approaches with some possible modifications proposed by the authors but not sufficiently explored in the paper. We have carried out training and testing on these ideas to see if the propositions of the authors are sound. [5] [6]

7.1 + Naive Bayes-weighted-BON

Naive Bayes-weighted BON uses an objective function to train paragraph vectors to predict not only the words in the paragraph but also the ngrams. Each log probability term is weighted using a method that is similar to calculating the absolute values of Naive Bayes weights.

After training n-gram embeddings, we used NB weighting to train the neural models to concentrate on important terms . The methods can be thought of as distributed (dense) counterparts of sparse bag-of-n-grams in Naive Base Support Vector Machine (NBSVM), which uses the Naive Bayes (NB) feature to weight sparse bag-of-n-grams representation but suffers from sparsity issues, unlike NB-Weighted-BON.[2][6][6]

Our results for Cosine Similarity + NB-weighted-BON and L2R Dot product for IMDB dataset and the alternate datasets , not Tripadvisor Dataset, are consistent with the trends presented in the paper. The inclusion of NB-weighted-BON increased the accuracy relative to the singular similarity measures on the IMDB and Twitter Datasets where accuracy of cosine similarity + NB-Weighted BON was greater than L2R Dot Product + NB-Weighted BON.

In terms of the TripAdvisor dataset, we saw that the addition of the NB Weighted BON substantially decreased the accuracy. We postulate that this was due to the nature of the dataset. For Naive Bayes, words that have uneven distributions over classes are given more weights. The cutoff for the positive and negative distribution in the reviews as a rating of greater than 4 and less than 4 respectively. In general terms, a rating of 4 should be closer to a favouring review. This leads us to believe that the distribution of the positive implicating words is split between both positive and negative classes which may have caused this decrease in accuracy once more weights are assigned to these unevenly distributed words.

8 Discussion

In this section, we will briefly summarise how the results we obtain are consistent with our expectations of the model.

8.1 Accuracy

We see consistently albeit marginally better accuracy with cosine similarity on all the data sets. We also see that the longer a document or paragraph, or the more polar its contents are, the richer the amount of information that we have available leading to better classification. With the Twitter Dataset for example the 280 character limit might actually be limiting information leading to a weaker model with lower accuracy. We also see obvious shortcomings of a the model such as its inability to ignore misspelled words, leading to further loss in accuracy on the Twitter Dataset. [1][6][6]

8.2 Run-time

One of the major drawbacks of our model is its time complexity which is non polynomial complex. As a result of hyper-parameter tuning through cross validation we further scale up time by a power for each additional parameter we tune. One of the shortcomings of the way we have carried out hyper-parameter tuning is assuming that they are independent, we have tuned one hyper-parameter at a time and sequentially set each hyper-parameter to its optimal value. This approach may not hold if the hyper-parameters are not independent.

9 Improvements

We now propose improvements to the model. The first major improvement we propose is that of improving the data prepossessing.We will do this by considering two well known techniques in Natural Language Processing called Stemming and Lemmatization. We do this with a dual objective, reducing the dimensionality required to do the processing by reducing the number of words to consider and also to make the model resilient to noise in textual data. The overall results we should see are higher computational efficiency and higher accuracy, which is what we observe.

9.1 Stemming

Stemming is a prepossessing technique for NLP where suffixes are cut off such that works like running and run are considered the same. This is beneficial because then when word2vec or doc2vec procedures are run on a document, so ”run” and ”running” are treated as the same word instead of different words. This reduces the size of the vocabulary helping in two ways. Firstly, places for variation are reduced and thus there is more consistency across the model because ”run” and ”running” now get scored the same. Secondly, the run-time of the algorithm is sped up since less calculations are needed to be made for these essentially duplicate words.

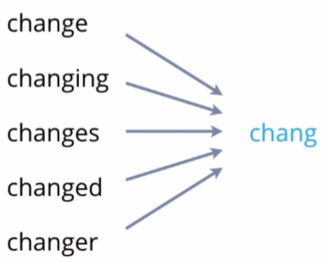

Figure 6. Stemming Example

9.2 Lemmatization

Lematization is an improved prepossessing technique for NLP where conjugations or other verb tenses suffixes and plurals are morphed into a words lemma. A lemma is the root word, and the word that shows up in the dictionary. This in some cases gives the same result as stemming, but in many cases does not. For example, ”running” can be easily stemmed into ”run”, and the lemma of ”running” is simply ”run”, but for a word like ”houses”, stemming would prune the s making it singular returning ”house”, whereas lemmatization can morph ”houses” ”house” ”homes” and ”home” all into one word that gets treated as the same. Another example is the word ”was”. This is the past tense of ”be”, but the stemming technique does not work here as there is no suffix for it to cut.

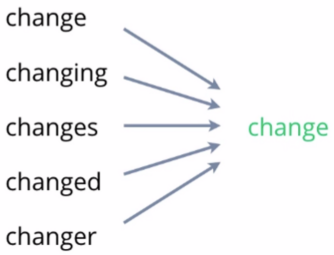

Figure 7. Lemmatization Example

9.3 Noise

One final approach we can take to improve our model is to rid it of ”noise”. Textual noise may comprise of words such as ”the”, ”or”, ”and” which we may completely forego in our data set without any loss of meaning. This makes our data set smaller, and serves as a pre-processing step that helps us map our data onto a lower dimensional set, which we can subsequently train and test.

10 Extensions

This work can be extended by improving the way the document embeddings are done via the aforementioned techniques of Stemming or Lemmatization. (Preferably Lemmatization as it is an improved and more encompassing form of Stemming). This approach leads to a decrease in the number of words classified and an increase in the frequency of words. This simplification allows the neural network to be able to make a stronger and more certain judgement on predicting sentiment. We can now see if these lead to any improvements in the accuracy using NLTK packages to carry out this process. We observe that the model leads to a marginally higher accuracy on the IMDB and the Trip-advisor data set. However, we see a decrease in accuracy for the Twitter Dataset. These results make sense, as the Twitter Dataset contains ”foreign” language words the process of stemming and lemmatization may have corrupted some of the richness in words since it might have treated them as English words. Further, by accounting for noise by simply creating a dictionary of words which we think the model can ignore we are also able to shrink our training data substantially. Overall, the key gain through this process we have made is that we have cut down the time for running our neural network substantially, while upholding (or mildly ameliorating) the accuracy.

11 Applications

The apparatus we develop has substantial industrial, commercial and educational applications beyond the immediate scope of the datasets we observe:

- Customer Relationship Management: The use of sentiment analysis to observe customer feedback based on customer emails, messages or even transcriptions of customer calls is common in call centres and is a huge way

in which large companies maintain quality of their products and gauge customer satisfaction levels.

- Product Feedback: Similar to the last point, the use of sentiment analysis is a huge driver for e-commerce business which relies on customer reviews and feedback text corpus to identify good products. There are well known datasets such as the Amazon Product feedback dataset containing several million paragraphs of product reviews.
- Election Analysis : Similar to the Twitter dataset we used, sentiment analysis algorithms are useful in determining election outcomes, often this can be dangerous as they can also end up influencing elections giving some big firms unprecedented power over society.

12 Future Work

For future inquiry, there are several directions we can explore. Work can be done with regards to better including grammar and linguistic context into the model. This can allow us to build models that are not language agnostic but are better at a specific language. It is less common to have "bilingual" data so such a model should be useful in attempting a pure single language treatment of the corpus.

Also, our current model only looks at n-grams (and in our testing 1-grams) and cannot look at two different numbered chunks of words at a time. For example, if I had the sentence "wow... this really is some great party...", a 1 gram would look at the ellipsis as individual periods and view it as an indicator of neutrality since periods appear at the end of every sentence. Only a 3-gram may pick up on the fact that the ellipsis is used to imply sarcasm and would then learn that it is potentially an indicator of sarcasm. The current method for addressing this would involve prepossessing the data to generate all consecutive groups of 3 characters, running the model on the 1-gram set, running the model on the 3-gram set, and then averaging. The problem with this is that averaging is less of an improvement and more of a compromise. An ideal method would be one that can pick the best of both worlds. That is to say, look at 1-grams when appropriate and 3-grams when appropriate. More generally, since the structure of grammar is quite dynamic, another approach might be to try develop a model that can dynamically look at different sized n-grams when appropriate.

References

[1] J. Alammar, "The illustrated word2vec," 03 2019.

[2] S. Palachy, "Document embedding techniques," 10 2019.

[3] M. Hazoom, "Word2vec for phrases—learning embeddings for more than one word," 12 2018.

[4] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," 2014.

[5] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, (Portland, Oregon, USA), pp. 142–150, Association for Computational Linguistics, June 2011.

[6] B. Li, Z. Zhao, T. Liu, P. Wang, and X. Du, "Weighted neural bag-of-n-grams model: New baselines for text classification," 2016.