Context Aware Sentiment Analysis

Anonymous
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY 14623

Abstract—Online service providers want to provide best services to their customers and users. To provide them with high quality of services they rely on the feedback they get from the customers, in the form of reviews. These reviews about their services can be found on the provider’s website or on the online communities like Yelp. For a popular product or service, it is normal that there could be thousands of reviews, with so many reviews to read and understand what customers are feeling, could be a daunting task for the providers. So, researchers are trying to find ways that could automatically process these huge number of customer reviews and provide insight to the providers, about the customers’ opinions or sentiments about their service or product. Moreover, finding just the sentiments of users is not enough, as most of the algorithms do not consider the context of the review. This paper focuses on describing various methods that do the task of finding sentiments based on the context and aspect.

Index Terms—Tags; Word2Vec; Doc2Vec; NLP; Latent Dirichlet Allocation

I. INTRODUCTION

Users express their opinion or their satisfaction online about products, services, places like restaurants etc. The service providers take advantage of this fact to analyze the sentiments of the users or customers and improve their services. The reviews are in the form of plain text and there is no global standard way defined yet, that a user has to follow to express his opinion, so users express their satisfaction or dissatisfaction in many ways. For example, "this service is not that good as food". To humans, this is easy to understand and figure out that customer is expressing his dissatisfaction with the service but is totally satisfied with the food. Many algorithms which do not consider the context or the aspect of the sentiment could classify this review as a positive review or a negative review. Algorithms that mine the opinion about the specific entities and the related aspect can help in understanding that "service is bad" and "food is good". In this paper, we will discuss algorithms that do take aspects into consideration to decide the sentiment of text.

II. BACKGROUND

Finding the true sentiments considering the aspect and context of a text is basically done in three phases. The opinion words are first, this phase is called as aspect detection. The second phase is the aspect classification phase, where it is decided that the detected aspect falls into which possible topics in the domain. Third phase, context detection phase, where the context of the aspect is decided, and the final, fourth phase, sentiment estimation, in this phase the sentiment final sentiment is decided. This approach is helpful because the four phases consider the aspects and context to decide the final sentiment of the text. This approach is useful only when all the phases are taken into account if any of the phases is not considered then results will vary. The rest of the paper discusses the various algorithms that take either aspect and context both into consideration or just consider one of them (Context or Aspect). The papers [1] [2] [3] [4] [5] represent various algorithms in this field.

III. RELATED WORK

A. Domain-specific Sentiment Analysis using Contextual Feature Generation

This paper suggests building a domain-specific sentiment lexicon. Firstly, contextual features are generated then a domain-specific sentiment classifier is build. Contextual feature generation is nothing but finding out the clues with respect to the domain. A domain is something that contains the text. A car is a domain if we have sentences that talk about, brakes, steering, head lights. So, from a sentence "unpredictable steering" for an instance can be used to learn that "unpredictable" is a clue and car is the domain, as steering is related to a car. So, the clue "unpredictable" should be treated as "negative" only when the domain is related to cars. To extract domain-specific contextual clues a Bootstrapping algorithm is used. Bootstrapping algorithm extract new clues from the text by comparing the old clues and their domain with the new clue word. If the new clue word falls in the same category of the domain it’s polarity is then decided with respect to the domain. The bootstrapping algorithm works as follows:

a. Preprocessing: Training examples, each example has a sentence and polarity, a set of clues.

Authors map the sentiment clue candidates in training examples to clues found in SentiWordNet lexicon and update their lexicon.

b. Sentiment topic identification: then they try to find out the topic based on the set of clues found in above step.

c. Sentiment clue generation: gathering new sentiment clue candidates and determining the polarity and accepting or rejecting each candidate as a new clue and adding it to the current set of clues.

d. Learning domain-specific classifier: After the first iteration, a new set of training examples was required so they used K-
means clustering to find examples automatically and then use this classifier to find the sentiment of the expression.

The step of determining the new clues stops when no more new clues are detected. Once Contextual feature generation is done, then comes the next step, Constructing domain-specific Classifier. A classifier is trained using the lexicon built above. The classifier can automatically classify the clue’s polarity as the classifier is built using the domain-specific lexicon.

The paper [2] focuses more on how to automatically discover what aspects are evaluated in reviews and how sentiments for different aspects are expressed. The paper [3] uses a different approach that talks about the task of finding a similarity between the sentiment and the language used for expressing the opinion. The paper [4] works along the same line as the present paper and group together the aspect and the opinion and assign a score to the pair. The paper [5] also works on the same line as this approach [1], the paper builds a domain-specific sentiment lexicon assuming the fact that the customer reviews are always divided into two parts, Pros, and Cons.

B. Aspect and Sentiment Unification Model for Online Review

This paper talks about how a system can automatically detect the aspect and also to detect the sentiments related to these aspects. There are two ways to automatically detect the aspects, one is to use supervised algorithms that require labeled dataset and second using unsupervised algorithms such as LDA (Latent Dirichlet Allocation). LDA is a technique which finds the topics from the given documents, in this case, LDA finds aspects from reviews. LDA assume a sentence is a mixture of topics and then determine the topic using probabilities. In simple words, LDA is a technique that is used to determine or guess the topics a particular text or a group of sentences contain. Also, LDA does not consider the positions of the words that occur in a sentence. The algorithm called as SLDA (Sentence LDA) is developed over the basic LDA. The assumption of SLDA is that position of the words matter too, i.e. words about the aspect tend to occur in the close proximity. Also, one constraint that SLDA believe is that all the words in the sentence are generated from one aspect. Although, this assumption is not true all the time, however, this works well in practice. SLDA is a generative process, which means that all the aspects, contexts and words representing them are generated automatically by the SLDA itself. First, for each aspect (in a sentence in a review of the document) a multinomial distribution of words is found. Second, for each review form a multinomial distribution of aspects, then for each aspect choose the multinomial distribution of words. Multinomial distribution of words is nothing a document(which is a sentence in this case) is represented by a feature vector with integer elements whose value is the frequency of that word in the document or sentence. With the assumptions made by SLDA, the probability of finding an aspect in the review and then finding a corresponding word in an aspect is calculated. The probability of finding an aspect in the review is calculated by dividing, the number of sentences that are assigned a particular aspect in a review by the total number of sentences that are assigned all the aspect. The probability of finding a word in a particular aspect is calculated by the number of words that are assigned a particular aspect by the total number of words in all the aspects. The SLDA only consider the aspect in a review, it doesn’t consider the sentiments yet. The algorithm called as ASUM, is an extension to SLDA. So, ASUM is more advanced than an LDA. Unlike SLDA, ASUM considers both, i.e. aspect and sentiment. ASUM uses the sentiment prior information. It assumes that if there is a word good or great then ASUM knows its prior polarity as positive, similarly, word awful and bad has prior information related to them. ASUM uses this prior information to feed it to the LDA. This approach of feeding prior information to LDA is called as an Asymmetric approach which is different from the normal LDA which do not consider the prior information of words. The paper then calculates the probability of sentiment in a review and probability of an aspect in a sentiment and probability of the word in an aspect-sentiment pair. The final result of the ASUM is a pair of aspect and sentiment.

The paper [1] focuses to build a domain-specific sentiment lexicon. To build a domain-specific sentiment lexicon, the process goes through two steps which are contextual features are generated then a domain-specific sentiment classifier. The paper [3] uses a different approach that talks about the task of finding a similarity between the sentiment and the language used for expressing the opinion. The paper [4] works along the same line as the present paper and group together the aspect and the opinion and assign a score to the pair. The paper [5] also works on the same line as paper [1], the paper builds a domain-specific sentiment lexicon assuming the fact that the customer reviews are always divided into two parts, Pros (positive reviews) and Cons (negative reviews).

C. Mining Sentiment Terminology Through Time

This paper talks about finding the sentiment of a sentence using opinion words. The paper assumes that the old algorithms only consider the common opinion words, the paper stress on the fact that the new opinion words keep on generating the user-generated content (reviews). These new opinion words are also considered in determining the sentiment of a text. The paper assumes that the polarity of a word also changes with time and prior algorithms do not consider this fact and hence are vulnerable to make mistakes. The new algorithm suggested also takes time into consideration and collect the polarity of the opinion word with time. Then an overall polarity is calculated for the opinion word, this is called as a collective polarity of the new opinion words. The find or mine the new opinion words a context-aware approach is used. This approach works on the fundamental of interchangeability. This approach finds words which are interchangeable. To decide if two words are interchangeable, few assumptions
are made. The first assumption, is two interchangeable words have low co-occurrence. For example, the words "recommend" and "suggest" implies the same meaning so these words are rarely used together in a single sentence. The second and third assumption is, the words appearing to the right and left of the target word has high overlap. If right neighboring words are considered then the effect of left occurring words is discarded. To decide the interchangeability of two words, it is assumed that a review is written in an almost same time span. To calculate the interchangeability of two words PMI is calculated. PMI is a way of calculating the association of a word with another. First for a target word its PMI is calculated and then the PMI of its left(right) occurring words is calculated. Only a few top words are then picked whose PMI is positive with respect to the target word, other words are then discarded. Now, a word (W_{old}) from old-time span is picked and the target word (W_{new}) is picked. Now, the contextual similarity is calculated between the old and new word using a mathematical formula given in the paper. The contextual similarity then assigns scores to the neighboring left or right words which has low co-occurrence. Now, an interchangeability score is calculated using the mathematical formula given in the paper. If the score is positive the words are interchangeable else not.

The paper [1] focuses to build a domain-specific sentiment lexicon. To build a domain-specific sentiment lexicon, the process goes through two steps which are contextual features are generated then a domain-specific sentiment classifier. The paper [2] focuses more on how to automatically discover what aspects are evaluated in reviews and how sentiments for different aspects are expressed. The paper [4] works along the same line as the present paper and group together the aspect and the opinion and assign a score to the pair. The paper [5] also works on the same line as paper [1], the paper builds a domain-specific sentiment lexicon assuming the fact that the customer reviews are always divided into two parts, Pros (positive reviews) and Cons (negative reviews).

### D. Automatic construction of a context-aware sentiment lexicon: an optimization approach

This paper talks about the idea of creating a sentiment lexicon for the specific domain, which is similar to other approaches. What makes this approach more feasible and better then other approaches is that this algorithm considers the fact that even in same domain the words can reflect different sentiments. For example, the word "large" is negative for battery if the domain is considered as "Computers" but the same word expresses a positive sentiment if it comes with "screen". The paper introduces an algorithm that not only creates the domain-specific sentiment lexicon but also aspect-based lexicon at the same time. The algorithm works for unlabeled corpus too. The algorithm collects the data from different resources to gain the context of the word, and assign a score to the word with respect to context, and finally use these scores to decide the polarity of the word that is context based and aspect based.

The first step in this algorithm is to create a context-based sentiment lexicon. The algorithm uses general purpose sentiment lexicon as a seed and then build their lexicon over it. The algorithm defines Aspect as the group of words characterizing a topic in a domain. To build a context-dependent sentiment lexicon the algorithm uses several resources. The first resource is a General-Purpose-Sentiment-Lexicon, this lexicon contains general opinion words which always point to same polarity irrespective of the domain. For example, excellent or bad. The second resource is Overall-Sentiment-Rating, the algorithm uses reviews from websites where overall sentiment is known to some degree, for example, TripAdvisor, the TripAdvisor asks for reviews in a specific format which make easy to interpret the sentiment of the review. The third resource is a Thesaurus, this resource helps in determining the polarity of a word. For example, the word large is positive or negative is not obvious. So, to find the polarity the algorithm makes use of synonyms and antonyms to decide the polarity like word big is similar to large or dissimilar to tiny. Now, if there are pieces of evidence about the polarity of the word big or tiny we can easily relate the polarity of word large. The fourth resource is linguistic heuristics, the and rule, the algorithm believes if there are clauses that are connected using a conjunction like and usually express the same polarity. The but rule, the algorithm believes that clauses are connected using but in conjunction usually express different polarity. The not rule, the algorithm believes that words prefixed with not, no, never expresses the opposite polarity, for example not happy express the negative sentiment like the word bad would do. Using the above resources the context-dependent-sentiment-lexicon entries are generated. The algorithm use clauses instead of a whole sentence as a tagging unit, since a sentence could comprise of many sub-sentences which could express different polarities. So, to split a sentence into and determine the clauses the algorithm uses Stanford Parser, this parser parses the sentences into a syntactical tree structure. The subtrees in the tree structure represent potential target clauses to work with. Now, the clauses represented by the subtrees are tagged with an Aspect, to do this the words in the clause are matched against the words in the general purpose sentiment lexicon, the Aspect is then assigned to the clause. For example, [The (check-in):SERVICE is very smooth] and [the (restaurant):FOOD is the best] represent two clauses with an aspect. SERVICE and FOOD are the aspects in the above example. Now, each clause is then analyzed to check if a new aspect-opinion pair can be found or not. In above example the pair could be [SERVICE, Smooth] and [FOOD, best]. In this way, the entries for context-dependent aspect-dependent sentiment lexicon is generated. The above steps mentioned above for different resources is then translated into code, this code is then called an Objective Function. This Objective function is the core of this algorithm. Once, this lexic is developed then comes the task of assigning sentiment polarity to each entry in the lexicon. To calculate the score for each entry the algorithm first check if the target opinion word’s prior information is available or not
(checks in general purpose sentiment lexicon). If the prior information is available then the same score is assigned, if the aspect of the generated lexicon and general lexicon matches else, the algorithm looks for clues about the sentiment in different resources (like Overall-Sentiment-Rating, General-Purpose lexicon, Thesaurus, linguistic heuristics, as mentioned above) to assign the score.

The paper [1] focuses to build a domain-specific sentiment lexicon. To build a domain-specific sentiment lexicon, the process goes through two steps which are contextual features generated then a domain-specific sentiment classifier. The paper [2] focuses more on how to automatically discover what aspects are evaluated in reviews and how sentiments for different aspects are expressed. The paper [3] uses a different approach that talks about the task of finding a similarity between the sentiment and the language used for expressing the opinion. The paper [5] also works on the same line as paper [1], the paper builds a domain-specific sentiment lexicon assuming the fact that the customer reviews are always divided into two parts, Pros (positive reviews) and Cons (negative reviews).

E. Automatic construction of domain and aspect specific sentiment lexicons for customer review mining

This paper talks about the construction of a sentiment lexicon that is based on content and aspect both. Consider an example, the food was lousy, too sweet or too salty and the portions tiny. The paper suggests that much previous work would fail to create a lexicon that could tell that food and portions are the aspect and opinion is negative. The system proposed in this paper works in 4 phases. The first phase is aspect-detection where the aspect is mined from the text, the second phase is aspect classification the aspect found in the first step is then classified into one category of the several possible categories in the domain. The third phase is context detection module, which determines the context related to the aspect. The final phase of the system is sentiment estimation module, which realizes the sentiment analysis of aspect and the opinion.

The aspect detection module is an open source project [6] which can be used to find the aspect from the text. The open source project goes by the name of IXA Pipeline. The IXA pipeline module is a system that is formed using other submodules. The modules are ixa-pipe-tok, ixa-pipe-pos, isa-pipe-nerc. The module ixa-pipe-tok tokenizes and segment the text and produces the output. This module takes text as an input and produces text as output. It is built over the Stanford English Penn Treebank tokenizer. The second submodule is ixa-pipe-pos which is used for POS (Parts-of-Speech) tagging and lemmatization. The lemmatization is done by using three different dictionaries. The third module ixa-pipe-nerc is the module that is used to find the Named Entity Recognition and Classification. The module can identify the entities which are one of the four types, one person, the second location, third organization and fourth is the one that identifies other entities which do not belong to above mentioned three categories. The modules are called as pipes because they output of the first module is fed as input to the second and the put of second is fed as input to the third module. The final output is the collection of words that give us the aspect of the opinionated text.

The second phase of the system is aspect-classification. To perform the task a Support Vector Machine with a linear kernel was used. The SVM is a technique to classify input into a category out of several categories, the idea is to learn or draw a plane in 2-dimensional space such that the data is separated into two planes, then any input given to SVM will fall in either one of the categories. To decide the number of planes in SVM a kernel is used, a linear kernel will try to fit a line in the 2-D space to separate the data in 2 planes. Now, for each sentence from the input, a feature vector is created and then fed to the SVM to classify the category. The limitation of using this method is, if a new word is encountered, i.e. a word that was not seen during the training of the SVM, then the results could go wrong. To avoid this scenario, the paper has combined WordNet and corpus (the dataset) together to develop a new hybrid model. This model has to combine word embedding model and semantic similarity model together. The word embedding model is the nothing but a method of representing the words in a text format to their equivalent representation in real numbers. The semantic similarity can be calculated by using the Sim function provided by Word2Vec [give a link to it]. The similarity between two words is calculated using Cosine similarity. The Word2Vec looks for the co-occurrence of words, but this approach could lead to the wrong result as there could be a sentence which talks about two different aspects, as the words in a sentence are co-occurring it could assign them same similarity. To avoid this semantic similarity WordNet is used. WordNet uses structural knowledge combined with Word2Vec to correctly classify the aspect. This completes the second phase of the system.

The third phase is aspect-context detection, the task of this module is to find out the text fragment in the entire text which represent the opinion about the aspect. To perform this task paper has introduced an algorithm. NOTE: I am trying to understand the algorithm, before expressing anything about it.

The fourth phase of the system is aspect-based-sentiment-analysis. The paper has combined two different techniques together to get the work done. These techniques are an ensemble of classifiers and ensemble of features. An ensemble of classifiers: uses the output of participating algorithms and combine them to get the final prediction of the sentiment. Two ensemble technique is used, which are Majority voting and Meta-learning technique. The meta-learning technique is nothing but a Random Forest classifier. An ensemble of Features: combines the word vectors or features that have been extracted differently. This completes the fourth phase of the system.

The paper [1] focuses to build a domain-specific sentiment lexicon. To build a domain-specific sentiment lexicon, the
process goes through two steps which are contextual features are generated then a domain-specific sentiment classifier. The paper [2] focuses more on how to automatically discover what aspects are evaluated in reviews and how sentiments for different aspects are expressed. The paper [3] uses a different approach that talks about the task of finding a similarity between the sentiment and the language used for expressing the opinion. The paper [4] works along the same line as the present paper and group together the aspect and the opinion and assign a score to the pair. The paper [5] also works on the same line as paper [1], the paper builds a domain-specific sentiment lexicon assuming the fact that the customer reviews are always divided into two parts, Pros (positive reviews) and Cons (negative reviews).

IV. SUMMARY OF ABOVE APPROACHES

As discussed above there are many approaches, algorithms, and systems that take care of sentiment analysis based on the context. There are two ways to perform context-aware sentiment analysis, one is Dictionary based approach and the second one is the corpus-based approach.

The Dictionary based approaches: a general purpose lexicon is used as a seed to start the algorithm and then building their own lexicon. To build a lexicon, the algorithm usually finds new target words and then match them in the general purpose lexicon for similarity or dissimilarity. If similarity or dissimilarity is found, then the polarity of the word in the general purpose lexicon is assigned to the new target word. One of the major disadvantages of the Dictionary based approach is that they are not domain and aspect specific, which calls for approaches that could take this into account while constructing their lexicon.

The Corpus based approaches: the basic idea of this kind of approach is to find the similar relatedness of the words and then decide the sentiment for the target word. There are many ways to find the similar relatedness such as word co-occurrences, contextual similarity. However, this approach could fail if a sentence with a conjunction is considered, which implies different sentiments. It is not always true that conjunctions like "and", "or" always reflect the same sentiments for the connected sentence. For example, "the camera is good and the price too.", here the conjunction "and" connect two aspects of similar sentiment polarity, consider the example "the car is nice and requires high maintenance". So, the systems or algorithm using this approach could fail.

Also, many approaches also depend on machine learning techniques. Machine learning techniques when compared to other approaches performs good, however, the limitation is that most of the machine learning techniques require labeled dataset to work with and these techniques are called as supervised learning techniques. The supervised learning techniques first learn from the data set and build a model from it and then can classify any sentence or text accordingly. The data set used for these techniques has to be a labeled data set. That is each sentence or text is labeled with a sentiment, this labeling is usually time-consuming and costly, as it is done manually. So, we must opt for an approach that overcomes all the difficulties or limitation mentioned above.

The most promising and effective way to analyze the sentiment based on the context is to use probabilistic topic models such as LDA (Latent Dirichlet Allocation). Unlike supervised machine learning techniques, LDA is an unsupervised technique which does not require a labeled dataset or corpus to work with. LDA is a technique which assumes that each document in the corpus is a collection of topics and each topic is a collection of words. LDA also assumes that each word is dependent or found from the underlying topic.

This project will use LDA as the base technique to perform the task of context-based sentiment analysis. To perform the task the research paper [2] will be taken as a reference. The paper [2] uses LDA to do sentiment analysis. The paper has proposed a version of LDA which is termed as SLDA (Sentence Latent Dirichlet Allocation) and ASUM (Aspect Sentiment Unification Model) as another technique that is based on SLDA. Although, the performance of this model is good as compared to other approaches [1] [2] [3] [4] [5].

V. HYPOTHESIS

A. Word2Vec: Simple & Elegant Approach

The technique, LDA seems to be promising to handle the task we are interested in. However, there are certain limitation of this approach too. Following are the limitations.

- As LDA is a bag of words model, the position of words is not considered.
- Bag of words model have limited usability if the context and semantics of the words have to be considered.

With these limitation again LDA seems to be unfit for the task.

With so many methods and techniques present today for sentiment analysis, let it be supervised or unsupervised or maybe semi-supervised techniques, all of them uses all sorts of combinations of techniques to get the work done which make these techniques either too complex to understand and handle or it is difficult to generalize them so that it can be used by everyone.

With all these shortcomings or cons such as complexity, not easy to use, a novel approach called as word2Vec was developed by Google in 2013-14. This word2vec [7] technique is so simple and an elegant solution for sentiment analysis. It doesn’t depend on any other information to build or learn the context, such as domain specific lexicons, etc. word2Vec is technique based on Neural Networks. This Neural network is a shallow network with just three layers and 100 neurons. word2Vec considers both the context and the semantics of the text to exactly determine the true sentiment. Although, this technique had some limitations in the sense that it was not possible to use it for practical purpose. So, the Google then proposed the new version [8] which has these limitations fixed and now can be used for practical purposes. The following figure represents the working of word2Vec technique. To
understand how word2Vec works please read the following Preliminary Section.

![Sentiment Analysis Flow Chart](image)

VI. PRELIMINARY

A. Why Word2Vec?

Sentiment classification, of freely available text through customer reviews, is of utmost importance to the online service providers or vendors who sell their products, to gain the insight of how well their product performed. There are many techniques to perform sentiment classification, but almost all of them suffers from a basic problem of context. A context-aware SA is important to really understand the hidden sentiment of any user or customer. Mostly, two types of techniques are used one is supervised learning and second is unsupervised learning, both techniques either use some statistical techniques or they don’t. Some advanced techniques such as Latent Dirichlet Allocation, which is a topic modeling technique is used in unsupervised learning. A technique such as a dictionary based takes the average of all the words polarity in a sentence and gives the final sentiment by averaging the scores of each word. This method fails for a simple sentence “not good”, the true sentiment is negative but this technique leads to the output sentiment as Neutral with score zero because of positive and negative polarity word together. A technique such as bag-of-words considers the text as a 1 X N vector where N is the number of words in the vocabulary, each column in the row represents a word and also a number which represents how many times the word has occurred in the text. This method is much advanced than previous but the major disadvantage is that then context is not considered. Also, the size of the vector could increase dramatically if the number of words in vocabulary is huge. All the techniques so far, either don’t consider the context or don’t consider the semantic. To overcome these problems a new technique is invented called as word2Vec technique in 2013 [7] [8]. The following discusses in detail about this technique. This technique uses Neural Networks. This technique is also called as Shallow Neural Network technique. As there are only 3 layers in the network. One is the hidden layer and other is the output Softmax layer.

B. Word2Vec Explanation

Below is the representation of the NN that Google has used everywhere to explain the concept of word2Vec.

![Word2Vec NN](image)

This technique is an advanced technique and makes use of Artificial NN as the classification algorithm. This technique has 2 different ways to perform the task. One is called as CBOW and the other is called as skip-gram. As we know that an input to an NN cant be text or a word or a sentence directly, we need to represent the word as a vector. This vector is build using the simple technique called as One-Hot-Encoding.

One-Hot-Encoding: To represent a word in a vocabulary we count the total number of words in the text. Then a vector (as a simple array) is created whose length is equal to the size of the total number of vocab words. So, a vocab with 10,000 words will represent a single word with a vector of length 10,000. This vector will then contain a 0 or a 1. The vector will contain the value 1 at the index where the word is present in the vocab and all other places will represent the zero. It is important to note the assumption made. The assumption is that the vocab will only contain the distinct words, i.e. no words in the vocab are repeated. So, the input to the word2Vec NN is the one-hot vector as explained above.

There are 2 layers in the NN as explained in the section Why word2Vec? The second layer is the Hidden Layer where the input is given. This hidden layer has 300 neurons. The paper assumes that they are using the 10,000 words vocab. This means that each neuron has to adjust its weight 300 X 10,000 times i.e. 3 million times. Which will turn out to be a huge network to train. If we want to train the neural network with a huge amount of text these 3 Million weight updates will increase further.

The paper [8] also presented by Google, discusses few techniques to handle the update of 3 Million weight.

Continuous Bag of words (CBOW): The CBOW method predicts a word when several other words
are given as the input. The following figure shows the overview of the CBOW. In this report, I will focus more on the Skip-Gram model, not CBOW.

**Skip-Gram model:** The skip-gram does the work the other way round. It predicts the group of words given input a single word.

Initially, the input to the NN is an N-dimensional vector but during training, the NN learns the optimal vector for each word. The advantage of using this method is that the feature space reduces to very low dimensions as compared to other techniques. This technique also preserves some context of the surrounding words. So, to classify a whole document we might take the average of all the word vectors and then classify the document. With this method, we are considering the context of surrounding words but we lose the word order, which again could lead to the wrong classification.

The Skip-Gram is an NN and our objective is to train an NN for predicting the words if an input word is given. There are 2 things to understand here. First, the output produced by the output layer is not the final output we are interested in. Second, our objective here is to find the final learned weights on the hidden layer after training is done. To train the NN we give the input as a vector as explained above. The vector is a one-hot-encoding vector representing the word as a vector.

As explained above we want to train an NN that could help us in finding the probability of nearby words gives a single word from a sentence. The near-by term means the number of nearby words we want to find. The nearby term is called as window size. If the window size is two, then NN will output four words, i.e., two words prior the input word and two words after the input word. We train the NN by feeding words in the pair of two if the window size is set as two. For example, consider the text, "The dog is too lazy to play", then we will feed the following pairs to the NN.

- The: (the, dog), (the, is)
- Dog: (dog, the), (dog, is)
- Is: (is, the), (is, dog), (is, too), (is, lazy)
- Too: so on and so forth

The network learns statistics from the number of times each pairing is present. The input to the NN is a vector of 10,000 lengths and the output is also a vector of length 10,000 length representing the probability of all the words in the vocabulary. Also, there is no activation function used in word2Vec in the hidden layer. Although, the output layer uses a Softmax. The hidden layer I am talking about is nothing but a weight matrix of size 300 X 10,000. So, if we multiply the one-hot vector representing the input word with our weight matrix we will find the row corresponding to the input word in the output. This is just the simple matrix representation.

\[
\begin{bmatrix}
0 & 0 & 0 & 1 & 0
\end{bmatrix}
\times
\begin{bmatrix}
17 & 24 & 1 \\
23 & 5 & 7 \\
4 & 6 & 13 \\
10 & 12 & 19 \\
11 & 18 & 25
\end{bmatrix}
= \begin{bmatrix}
10 & 12 & 19
\end{bmatrix}
\]

To understand, consider two matrix, one is 1 x 10,000 size and second is 300 x 10,000 size. Then the output of this matrix multiplication is 1 x 10,000 size vector.

**The Output Softmax Layer:** The output layer is Softmax regression layer, the final out of each 10,000 neurons is between 0 and 1 and the sum of all the output must be 1. The output of each neuron is then get multiplied to the weight vector as explained in above figure. The output of each neuron is then converted to exp(x) form. To make sure that the sum of all neurons output sums to 1. A simple formula
is used, the output of the neuron is divided by the sum of all the neurons converted to \( \exp(x) \) form.

\[
\text{Softmax} = \frac{e^x}{\sum e^x}
\]

The usefulness of word2Vec comes to picture when we talk about the context. The context words are assumed to be reflecting the same meaning as the other word in the same context. For example, consider the word smart, then the contextual word intelligent is similar to the word smart. The word2Vec takes care of these contextual similarities inherently as the words are synonyms or similar so the word vector of these two words is almost similar. This is a huge advantage of word2Vec, as we can see that this technique could handle the Stemming process by default. So, the words car and cars will have the similar word vector as they are just the same words.

With all the explanation above we can see the useful application of word2Vec for context-based sentiment analysis. Although, we can clearly see that the NN network can grow huge if the vocabulary is very large which can ultimately take more time to train the model which makes it slow. Also, as explained above that while creating the input vector for hidden layer there are words like "the" which do not provide any useful information for the other word in the pair. Like the pair (the, dog). The word "the" reveals no useful information about the "dog". So, the paper [7] has few disadvantages. So, summing up the disadvantages below:

- The words like "the", "is" are not useful in this NN.
- The NN could take long time to get trained if the input text is huge.

The two disadvantages could be said in technical jargon as:
1. Subsampling the frequent words to decrease the training examples
2. Negative sampling: this technique will modify only the small percentage of the weights in the NN hence decreasing the training time substantially.

The paper [8] also proposed by the Google takes care of the above-mentioned problems and improve the word2Vec even more. The above two points are helpful to make the NN train practically. So, we must have to embed these points in the discussion.

As we have already seen that the hidden layer has to handle 3 Million weights for the word vector of 300 components and 10,000 vocab size, this makes the NN slow and time-consuming.

**Sub-sampling:** As we discussed that the words such as the are irrelevant as they do not convey any useful information to NN while training and also their frequency is very high so to make the NN learn about this word is not good. To avoid this condition, we use sub-sampling. During the training phase, we remove the word "the". Removing this word helps in reducing the samples we have to feed to the hidden layer and also the word "the" will not appear in any of the contexts. Let's say for a window size of 10 and removing the "the" word following would be the effect on NN.

- We will see that we have 10 fewer samples where the input word is "the".
- There would be no "the" appearing in context of any word. The paper suggests doing the sub-sampling using the Sampling Rate.

**Sampling Rate:** The sampling rate is a mathematical formula to calculate the probability of keeping a word in the vocabulary. The formula is given below.

\[
P(W_i) = (\sqrt{W_i/0.001} + 1) \times 0.001/ZW_i
\]

**Negative Sampling:** To train an NN means we take a training sample and adjust it weight for all the neurons in the network. This means that for even a single word weight is going to be adjusted for all the 10,000 neurons if we consider our old example. To avoid this Negative sampling is used. In Negative Sampling, instead of updating the 10,000 neurons only a few of them is updated. As discussed above, input to the hidden layer is a one-hot-vector containing 1 for the word and all values as zero and the output layer is also a one-hot-vector representing the probability of the word and all values as zero. So, instead of giving all the remaining values as 0, we will give 2-5 zero in the one-hot-vector as zero. We will randomly pick any 5 words. The paper [8] suggests that for large dataset value of 2-5 is most optimal. In this case, our negative words are those 5 random words whose value in the vector is going to be zero and the positive word has the value 1. To understand how the computations will be reduced, let’s see the example with 5 negative words and 1 positive word. So, while training the Neural Network we will update it weight for the positive word plus 5 other weight updates for the 5 negative words. So, total 6 updates. For a word vector with 300 components, we have now only 1800 updates which are way below the previous weight updates, which were 3 Million in the count.

By using the negative sampling, we can reduce the weight update exponentially and still get the same results and but better performance. This technique really made the word2Vec method practical to use for huge datasets.

**How to select Negative Words:** The Negative sampling made the word2Vec approach practical to use for huge datasets, but how to select the negative words from the corpus? This is a real question to make negative sampling works. The paper [8] suggest an approach for doing this. The paper [8] suggests a simple mathematical formula for doing this. A word has a higher probability of being chosen as the negative word if the frequency of the word is higher is the corpus. The probability of selecting a negative word is nothing but the simple division of two values, one is the weight of the word
and second is the sum of all the weights. These two parameters i.e. the numerator and denominator are also raised to the three-fourths power. There is no clear explanation in the paper [8] explaining why three-fourth powers are chosen. Following is the formula.

\[ P(W_i) = \frac{f(W_i)^{3/4}}{\sum_{j=0}^{n} f(W_i)^{3/4}} \]

\( P(W_i) \): is the probability of choosing a word as negative word.
\( f(W_i) \): weight of the positive word.

C. doc2Vec: Advanced word2Vec

As our end goal is to analyze a corpus, we need something more than word2Vec. doc2Vec is exactly same as word2Vec but with a minor change. With all the functionalities of word2Vec there is an addition of one field called as the paragraph id. This paragraph id is used to identify the paragraph to which a sentence belongs. Like word2Vec, doc2Vec uses two methods called as Distributed memory (DM) and second Distinguished-bag-words method (DBOW). The approach is same as word2Vec. The DM predicts the single word when a group of previous words and a paragraph vector are given as input and DBOW predicts a group of random words when a paragraph vector is given as input. Refer the below figure to understand how doc2Vec looks like.

D. word2Vec Results

With all the concepts explained above, we must see how word2Vec performed on real data. The below-mentioned results are taken from [8] This section covers the evaluation of the word2Vec technique. The authors of the paper tested their proposed word2Vec technique with four other previous methods. To evaluate the performance the authors have chosen to use the skip-gram technique in word2Vec. To evaluate they tried to find the analogous terms for a given word. Following are the results snippet. For example, the input word given was “Redmond”.

- The first technique proposed by Collobert and Weston gave the following terms as the analogous words:
  - Redmond: conyers lubbock keene
  - Time take by this algorithm to train was 2 Months days.
- The second technique proposed by Turian gave the following terms as the analogous words:
  - Redmond: McCarthy Alston Cousins
  - Time take by this algorithm to train was few weeks days.
- The third technique proposed by Mnih and Hinton gave the following terms as the analogous words:
  - Redmond: Podhurst Harlang Agarwal
  - Time take by this algorithm to train was 7 days.
- The fourth technique proposed by Mikolov gave the following terms as the analogous words:
  - Time take by this algorithm to train was 1 day.

The amount of data used by this algorithm was two to three orders of magnitudes more than all the above algorithms. Also, the time complexity of this approach was about the fraction of all other techniques, as a result, the time taken to train the data was very small for about 30 billion words.

VII. Sentiment Analysis using Doc2Vec

The best way to understand word2Vec or it’s variants such as doc2Vec, we expect to see this algorithm in action. This section is sub-divided into 3 sections that explain the word2Vec functionality in detail. There are 3 sub-sections which are as follows:

1) The System Setup
2) Training and Testing the Model
3) In Action: Classifying sentences

A. The System Setup

To see word2Vec in action, I would be using the Python library called as “Gensim”. The information about Gensim can be found here [https://en.wikipedia.org/wiki/Gensim] and use this link to get the details about Gensim library [https://radimrehurek.com/gensim/models/doc2vec.html].
The original code for word2Vec is written in C. The code is highly optimized and difficult to understand. The Gensim library has done a commendable job by developing a wrapper in Python. The Python implementation is pretty easy to understand. As we know that word2Vec or doc2Vec is a neural network, so, the input to the Neural Network can’t be a raw sentence, so first, we need to perform some pre-processing to make the dataset reusable.

1) Preprocessing: As the dataset contains text from users it is highly likely that the text contains all the special characters and repeated symbols such as periods. We first removed these characters. A python code handles this by using a regular expression, RegEx. As discussed above in the Preliminary section that stop words are automatically not considered by the word2Vec algorithm we didn’t have to remove any stop words. Although, removing the stop words could help in decreasing the size of the pre-processed text and nothing more.
Using computing resource such as the processor is costlier than memory eaten up by stop words. So, stop words are not removed.

\[\text{regex} = [\text{\&} - z A - Z 0 - 9]\]

The regex means we are escaping alphabets from A to Z lower and upper case implied and numbers with 0 to 9 are escaped and everything else is removed from. The result of this preprocessing is to get 4 different text data files which contain the data for movie review. The four files are described in the dataset section. Each review is separated by a newline. Each file has 12500 movie reviews. This format of the dataset is of utmost importance to us.

2) Input to Doc2Vec: As doc2Vec considers the whole paragraph as an input, unlike word2Vec, word2Vec converts a word to a vector. To work with doc2Vec we need a label to represent a single review. The doc2Vec uses this label as a special word to represent a single line. To understand consider a line "star cast of movie is lame". word2Vec want this sentence in the following format. [ ['star', 'cast', 'of', 'movie', 'is', 'lame'], ['somelabel'] ]. To input any sentence in this format we use LabeledLineSentence class constructor to do that. But, this class LabeledLineSentence has a limitation, that it can do the work just for a document, not for a set of documents, which is the case we are dealing with. So, we input a dictionary with keys as the file names and values are the unique labels for all the input files. For instance, the call would look like this in code.

```python
sources = { 'test-neg.txt': 'TEST_NEG', 'test-pos.txt': 'TEST_POS', 'train-neg.txt': 'TRAIN_NEG', 'train-pos.txt': 'TRAIN_POS', 'train-unsp.txt': 'TRAIN_UNSP' }

sentences = LabeledLineSentence(sources)
```

Read the DATASET section for more information on the input dataset files.

B. Training and Testing the Model

1) Building the vocabulary: There is a prerequisite for building a Model using doc2Vec, we first need to build a vocabulary. This vocabulary would contain the unique words. We will then perform some mathematical operations on these words in the vocab such counting the words. To do that we need the input in form of an array of labels. So, we have already prepared this array in above step. Input to doc2Vec. So we will first create the object of doc2Vec class using the API doc2Vec() with some arguments. There are 5 mandatory arguments, which are as follows:

- **min_count**: 1, this parameter ignore the words whose count is less than min_count

- **window**: 10, the maximum distance between the current and predicted word within a sentence

- **size**: 100, dimensionality of the feature vectors in output

- **sample**: 1e-4, threshold for configuring which higher-frequency words are randomly down-sampled

- **negative**: 5, count for negative sampling

- **workers**: 7, this parameter is nothing but the number of threads to train the model

2) Training Model: To train the word2Vec model we use the API called as model.train() given by the Gensim [9]. While training we will give the input sentences in a random fashion. This is randomization is needed because it is observed that if sentences are fed in an order the training accuracy of the model tends to come very low. We train the model for 20 epochs. Once, the model is trained we can use it to test our model using the testing dataset present in the directory.

3) Saving & Loading the Model: There is one good thing the Gensim [9] has provided that is, the model what we have trained can be saved for the future purpose, i.e. if at a later point of time we want to test we don’t have to build and train the model again. To do that Gensim [9] has provided two APIs model.save() and model.load()

C. In Action: Classifying Sentences

Here we will test the model with our testing dataset. To test we must first extract the training vectors. As we know that we trained our model using 25000 positive and negative reviews. So, we have a training vector of size (25000, 100). Also, we would be requiring the labels vector, which is a vector of length 25000. We will create a numpy array of the same sizes. We will do the same thing for our testing data, i.e. we create vectors out of it. Now, we have two vectors with us containing the probabilities of word and sentences. To test we can simply train a Logistic Regression model using the training vectors. To train a Logistic Regression model we can use scikit-Learn Logistic Regression. Once, the model is trained we can test our testing dataset and find out the accuracy of the results using the score() API from skLearn.

![Logistic Regression from Sklearn](image)

![Classification Score on Testing Dataset](image)

VIII. DATASET

We will be using few datasets for the sentiment analysis task. As word2Vec or doc2Vec approach is a supervised technique we have four datasets which are labeled. There are four datasets that are used, refer following. [10]. The dataset is prepared by Stanford University. Each row in the dataset represent all the reviews about a single movie. Rows are not
tagged with any labels, the whole document is considered as positive or negative itself. Below is the snippet of the positive and negative dataset. The preprocessed dataset used can be directly downloaded from here [11] http://ai.stanford.edu/~amaas/data/sentiment/. The other datasets can be find here [12]

Below is the information about each type of dataset.

1) Training Data
   a) train-[name]-neg.txt: is the dataset that contains the negative reviews about the subject.
   b) train-[name]-pos.txt: is the dataset that contains the positive reviews about the subject.

2) Testing Data
   a) test-[name]-neg.txt: dataset is dataset that contains the negative reviews about the subject.
   b) test-[name]-pos.txt: dataset is dataset that contains the positive reviews about the subject.

where the [name] tag represents one of the names of the dataset.

IX. RESULT

To evaluate the performance of the doc2Vec algorithm, few data sets were selected. There are few datasets that has been used in this project. These datasets were used in this paper [12]. The tests were performed in 3 datasets. These datasets are as follows:

1) Yelp Dataset
2) GroupOn Dataset
3) IMDB Dataset
4) LinkedIn

The following table summarizes the tests results of the datasets.

<table>
<thead>
<tr>
<th>Sr.No</th>
<th>Dataset</th>
<th>Epochs</th>
<th>Accuracy</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IMDB</td>
<td>50</td>
<td>85%</td>
<td>31.2MB</td>
</tr>
<tr>
<td>2</td>
<td>GroupOn</td>
<td>500</td>
<td>87.8%</td>
<td>350KB</td>
</tr>
<tr>
<td>3</td>
<td>Yelp</td>
<td>60</td>
<td>81%</td>
<td>31MB</td>
</tr>
<tr>
<td>4</td>
<td>Linkedin</td>
<td>200</td>
<td>84%</td>
<td>15KB</td>
</tr>
</tbody>
</table>

We achieved an accuracy of almost 85% with our testing dataset, which is a good result if we consider that we have used a shallow Neural Network with just 50 epochs of training and a simple Logistic Regression for classification. To simply test the trained model we can use the saved model. To test that we have to load the model using model.load('model saved path'). We can input some word and check for most similar words. Tested the model with the input word "Lame". Following is the result.

To obtain the results for different datasets, there were few changes that were required in the code to get the results. The code for doc2Vec Sentiment Analysis is taken from here [13]. The default dataset used in this contains the equal number of positive and negative samples so to create the labels for training and testing purpose for the final accuracy.

1) GroupOn Dataset: The code to create the training and testing lables to be given as input to the logistic Regression Classifier were obtained by changing the iterative for loop for training and testing as shown below. These changes were necessary as the number of positive and negative samples were different.

These figures are after the preprocessing was done on datasets to make a compatible input for doc2Vec.

<table>
<thead>
<tr>
<th>Sr.No</th>
<th>Dataset</th>
<th>Type</th>
<th>+ve</th>
<th>-ve</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GroupOn</td>
<td>Train</td>
<td>466</td>
<td>1190</td>
</tr>
<tr>
<td>2</td>
<td>GroupOn</td>
<td>Test</td>
<td>87</td>
<td>242</td>
</tr>
</tbody>
</table>

2) Yelp Dataset: same sort of changes were done to analyze the sentiment of the Yelp dataset. Details are below:

<table>
<thead>
<tr>
<th>Sr.No</th>
<th>Dataset</th>
<th>Type</th>
<th>+ve</th>
<th>-ve</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>test-yelp-neg.txt</td>
<td>1256</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>test-yelp-pos.txt</td>
<td>2734</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>train-yelp-neg.txt</td>
<td>51136</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>train-yelp-pos.txt</td>
<td>97456</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A. Result Comparison

The results achieved by the doc2Vec technique above seems promising but it is no guarantee that doc2Vec has performed well until we have some baseline comparisons to make. For this purpose a paper [14] was chosen, since this paper [14] has used the above datasets but with a different approach. The following figure shows the performance of the method used in the paper [14]. Following are the results obtained using the same dataset but different technique.

<table>
<thead>
<tr>
<th>Sr. No</th>
<th>Dataset</th>
<th>Epochs</th>
<th>Accuracy</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GroupOn</td>
<td>500</td>
<td>87.8%</td>
<td>350KB</td>
</tr>
<tr>
<td>2</td>
<td>Y elp</td>
<td>60</td>
<td>81%</td>
<td>31MB</td>
</tr>
<tr>
<td>3</td>
<td>LinkedIn</td>
<td>200</td>
<td>84%</td>
<td>15KB</td>
</tr>
</tbody>
</table>

Table above shows the results for the datasets used by the paper [14], to really validate and make sure that doc2Vec has performed well. To do the full validation the state-of-the-arts algorithm results were compared for the same datasets. The results of the state-of-the-art algorithms are not that promising. The following table represents the performance of the SOA algorithms.

GROUPON DATASET RESULTS

<table>
<thead>
<tr>
<th>Sr. No</th>
<th>Dataset</th>
<th>Algorithm</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GroupOn</td>
<td>TF-IDF</td>
<td>83%</td>
</tr>
<tr>
<td>2</td>
<td>GroupOn</td>
<td>LDA</td>
<td>78.1%</td>
</tr>
<tr>
<td>3</td>
<td>GroupOn</td>
<td>Doc2Vec</td>
<td>87%</td>
</tr>
</tbody>
</table>

The TF-IDF algorithm used is Naive Bayes.

LINKEDIN DATASET RESULTS

<table>
<thead>
<tr>
<th>Sr. No</th>
<th>Dataset</th>
<th>Algorithm</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LinkedIn</td>
<td>TF-IDF</td>
<td>58%</td>
</tr>
<tr>
<td>2</td>
<td>LinkedIn</td>
<td>LDA</td>
<td>78%</td>
</tr>
<tr>
<td>3</td>
<td>LinkedIn</td>
<td>Doc2Vec</td>
<td>84%</td>
</tr>
</tbody>
</table>

The TF-IDF algorithm used is Naive Bayes.

YELP DATASET RESULTS

<table>
<thead>
<tr>
<th>Sr. No</th>
<th>Dataset</th>
<th>Algorithm</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Y elp</td>
<td>TF-IDF</td>
<td>70%</td>
</tr>
<tr>
<td>2</td>
<td>Y elp</td>
<td>LDA</td>
<td>76.9%</td>
</tr>
<tr>
<td>3</td>
<td>Y elp</td>
<td>Doc2Vec</td>
<td>81%</td>
</tr>
</tbody>
</table>

The TF-IDF algorithm used is Naive Bayes.
For making the SOA algorithm work, few changes were done in the code. The following snippet shows a glimpse of the change code. The code was taken from the GitHub [15]

```python

def para_corpus():
    # Function to process the corpus
    pass
```

REFERENCES


