REPUBLIC OF TURKEY

FIRAT UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

SENTIMENT ANALYSIS OF SOCIAL NETWORK DATA USING MACHINE LEARNING

MASTER THESIS

ALI ABBAS ALBABAWAT
Supervisor: Assoc. Prof. Dr. Galip AYDIN

September-2017

REPUBLIC OF TURKEY

FIRAT UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

SENTIMENT ANALYSIS OF SOCIAL NETWORK DATA USING MACHINE LEARNING

MASTER THESIS

ALI ABBAS ALBABAWAT

142129110

Department of Computer Engineering

Submission Date to the Institute: 18 August 2017

Thesis Presentation Date: 13 September 2017

Supervisor: Assoc. Prof. Dr. Galip AYDIN **/ Firat Uni. / Comp. Eng.**

Other Jury Members: Asst. Prof. Erkan DUMAN **/ Firat Uni. / Comp. Eng.**

Other Jury Members: Asst. Prof. Ozal YILDIRIM **/ Munzur Uni. / Comp. Eng.**

September - 2017

ACKNOWLEDGEMENT

Firstly, I would like to express my sincere gratitude to my supervisor Assoc. Prof. Dr. Galip AYDIN for the continuous support of my Master's Degree study and related research, for his patience, immense knowledge and most importantly motivation. His guidance helped me in all the time of research and writing of this paper. I appreciated your passion and the way you delivered the lessons. Thank you for the structure and consistency you demonstrated in each meeting. The product of this research paper would not be possible without you!

I would also like to thank Mr. Ibrahim Hallac for his valuable help and the knowledge I got from him, and all family members and friends who were there when I needed them regarding this work.

May all my work hours and dedication be a humble contribution to the sake of science and developing a brighter future.

Ali Abbas Albabawat
August 7, 2017

TABLE OF CONTENTS

ABSTRACT

Opinion mining have shown to be a source of great value when it comes to collecting people's feedback about a product or an event as an example. The World Wide Web recently became very rich of people's opinions and thoughts so instead of filling some surveys for gathering feedback, due to the advance of sentiment analysis, we can now harvest thousands, millions or thousands of millions of opinions about what we are trying to develop, or thoughts about what we did recently. By the improvements that has occurred in the machine learning field we can now build a model that does some mathematical equations on large sets of data to give us the thoughts of millions of people about our work. With the many microblogging services online nowadays, these websites tend to have huge amounts of valuable information such as reviews, feedback, opinions and experience with specific products or events that can be benefited and gathered from by machines that work according to some machine learning model or algorithm. This is where sentiment analysis comes to existence, it is the field of Natural Language Processing (NLP) that uses machine learning models and techniques to collect subjective information from real world text (corpuses).

In this research we aim to benefit from the opinions-rich social media network data by extracting the sentiments from them. We will use multiple sentiment analysis techniques (Paragraph Vector, Deep Learning for Java and Naïve Bayes-SVM) on multiple platforms (DL4J, Gensim and regular Machine Learning) all that on two different tasks; IMDB Movie Review Dataset and 1.6 million tweets, and change parameters (like epochs, text manipulation etc.) to see what affects the results of those models, thus ending up with a better model. We also compare the results of all models on different datasets in the Result chapter to find out which model scores higher accuracy in prediction plus the computation power and time needed for each of them. We try to use a distributed computing platform, Apache Spark, to achieve parallel and distributed computing power which really comes in handy when using repetitive calculation on big amounts of records (Big Data).

Keywords: Big Data, Deep Learning, Sentiment Analysis, Spark, Paragraph Vector

ÖZET

SOSYAL AĞ VERİLERİ KULLANARAK MAKİNE GÖRÜŞ ANALİZİ ÖĞRENME

Görüş analizi, bir ürün ya da olay hakkında örnek olarak insanların geri bildirimlerini toplama konusunda çok değerli bir kaynak olduğu kanıtlanmıştır. World Wide Web son zamanlarda insanların düşüncelerinden çok zengindi; bu nedenle, geri bildirim toplamak için bazı anketler doldurmak yerine, duygu analizinin ilerlemesiyle binlerce veya milyonlarca düşünmeye çalıştığımız görüşleri toplayabiliriz. Makine öğrenme alanındaki gelişmeler sayesinde, artık çalışmalarımız hakkında milyonlarca insanın düşüncelerini vermek için geniş veri setleri üzerinde bazı matematik denklemleri üreten bir model oluşturabiliriz. Günümüzde birçok mikroblogging hizmeti ile bu web siteleri, bazı makine öğrenme modeline göre çalışan makinelerden yararlanılabilecek ve toplanabilecek belirli ürünler veya olaylarla ilgili incelemeler, geribildirim, görüş ve deneyim gibi değerli bilgilerin büyük miktarlarına sahip olma eğilimindedir. İşte bu noktada duyarlılık analizi var: Doğal dil işleme (NLP), gerçek dünya metinlerinden öznel bilgi toplamak için makine öğrenme modelleri ve teknikleri kullanıyor (derlemeler).

Bu araştırmada, görüşlerin zengin olduğu sosyal medya ağ verilerinden onlardan duygular çıkararak yararlanmayı amaçlıyoruz. Çoklu platformlarda (DL4J, Gensim ve Normal Makine Öğrenimi) çoklu duyarlılık analizi tekniklerini (Paragraf Vektör, Java ve Naïve Bayes-SVM için Derin Öğrenme) iki farklı görevi kullanarak kullanacağız; IMDB Movie Review Dataset ve 1,6 milyon tweet'ler ve bu modellerin sonuçlarını neyin etkilediğini görmek için parametreleri (çağlar, metin işleme vb.) Değiştirerek daha iyi bir modelle sonuçlanır. Ayrıca, Sonuç bölümündeki farklı veri kümelerindeki tüm modellerin sonuçlarını karşılaştırarak hangi modelin tahminde daha yüksek doğruluğu, artı her biri için gereken hesaplama gücü ve zamanı bulduğunu buluyoruz. Büyük miktardaki kayıtların tekrarlanan hesaplamasını (Big Data) kullanırken gerçekten kullanışlı olan paralel ve dağıtılmış bilgi işlem gücü elde etmek için dağıtılmış bilgi işlem platformu Apache Spark'ı kullanmaya çalıştık.

Kelime Anahtarı: Big Data, Derin Öğrenme, Görüş Analizi, Spark, Paragraf vektörü.

LIST OF FIGURES

Page No

LIST OF TABLES

ABBREVIATIONS

ANN : Artificial Neural Networks
API : Application Programming Interface
BOW : Bag of Words
CBOW : Continues Bag of Words
CNN : Convolutional Neural Networks
CPU : Central Processing Unit
CRM : Customer Relationship Management
CSV : Comma Separated Values
DL4J : Deep Learning for Java
DNN : Deep Neural Networks
ETL : Extract, Transform and Load
GFS : Google File System
GPU : Graphical Processing Unit
GRU : Gated Recurrent Unit
HDFS : Hadoop Distributed File System
IM : Instant Messaging
IMDB : International Movie Database
LSTM : Long Short-Term Memory
ML : Machine Learning
MVRNN : Matrix Vector Recursive Neural Networks
NB : Naïve Bayes
NBSVM : Naïve Bayes Support Vector Machines
NCE : Noise Contrastive Estimation
NLP : Natural Language Processing
NN : Neural Networks
NNLM : Neural Network Language Models
PB : Peta Byte
PCA : Principal Component Analysis
POS : Part of Speech Tagging
RAM : Random Access Memory
RDD : Resilient Distributed Datasets

RNN : Recurrent Neural Networks

SIMD : Single Instruction Multiple Data

SQL : Structured Query Language

SVM : Support Vector Machines

TCP : Transmission Control Protocol

YARN : Yet another Resource Negotiator

1. INTRODUCTION

We live in a digital age, which is rapidly changing communication management in astronomical ways [1]. Microblogging, a relatively new phenomenon, defined as "a form of blogging that lets you write brief text updates (usually less than 200 characters) about your life on the go and send them to friends and interested observers via text messaging, instant messaging (IM), email or the web" [2], provides an easy form of communication that enable users to share information, opinions and statuses directly with each other or on a public platform. Online services which provide microblogging tools, like Twitter, Tumbler, Facebook etc., are seeing millions of messages appearing daily. As more and more users share their views on products and services they use, and express their political, social, economic and religious sentiments, microblogging websites become valuable sources of people's subjective opinions.

1.1. Overview

Although there have been quite some studies on how opinions are expressed in genres such as news articles and online surveys, the sentiment analysis of informal languages and microblog posts with message-length constraints has been much less studied [1]. Twitter, one such microblogging websites, is a valuable corpus for opinion mining and which can be used to answer interesting subjective questions like, is the product opinion positive or negative? How are people responding to latest ads, products, campaigns? This project aims to provide an automated solution to answer such question. This project encompasses broad areas of research pertaining to fields of Natural Language Processing and Deep Neural Networks, to create an accurate and high performance real time sentiment analysis service. Most sentiment analysis techniques use the bag-of-words approach to determine sentiments, which ignore the sentence structure, making it oblivious to complex linguistic features like negations [11]. This project uses a Recursive Tensor Neural Network model for sentiment analysis, which handles such complexities and can accurately score negations at all levels in a complicated sentence. This project also involves implementing a web application which consumes the Sentiment Analysis service via restful interface, and integrates with Twitter Streaming API to provide sentiment analysis for trending topics in real time.

1.2. Sentiment Analysis and Opinion Mining

Sentiment Analysis is the method and procedure for extracting subjective information from a source text. The process typically involves the use of Computational Linguistics and Natural Language Processing. It is the task of defining the attitude, which in its simplest forms can be viewed as a text classification. In its more complex form, it is able to recognize the holder or the source of the attitude, plus the aspects of the attitude i.e. the thing for which the attitude is about, and the type of attitude, be it love, hate, desire etc. It has many other names like to extract opinions, intellectual judgment, mood analysis and subjectivity analysis. The analysis of opinions enables us to answer all types of subjective questions in an automated way, such as [1, 8]:

- Movies: Is this movies review positive or negative?
- Products: how was the opinion of people about the new Galaxy S8, positive or negative?
- Public: how confidant is the consumer?
- Politics: what do the people thing about our new president?
- Prediction: how satisfied are the customers about our market trend?
- Campaigns: people's response about the new online campaign?

While Sentiment Analysis methods and procedures are formally well defined, sentiments all by themselves are hard to define or quantify. Merriam-Webster defines sentiments as an attitude, thought, or judgement prompted by feeling [10]. Subjective views on such emotions can be difficult for non-quantity data such as like or dislike, but objectively this can be approximately classified using polarity weighting polarities such as positive, negative or neutral along with the strength. For the purpose of this project, we will be classifying sentiments into 5 classes, very positive, positive, neutral, negative and very negative. Each of the sentiment classes is represented as a value in the implemented software, ranging from 0 to 4 inclusive. The sentiment classes and their corresponding sentiment value are listed in Table 1.1

Table 1.1. Sentiment classes and sentiment values

Sentiment Cass	Sentiment Vlaue
Very psitive	4
Positive	3
Neutral	2
Negative	1
Very negative	0

1.3. Natural Language Processing (NLP)

The topic of Natural Language Processing lies in the field of Artificial Intelligence, which is concerned with making computers understand naturally taking place texts. Natural text can be in any language or genre, it can be written orally spoken, but used by people in their everyday communicating.

The goal of NLP is to perform human-like language processing [6]. Texts ambiguity make NLP very difficult. Like most other languages, ambiguity, an important occurrence, is widespread in the English language [7]. For example, the syntactic ambiguity in the text "Teacher strikes idle kids" can be understood differently based on what is considered the main verb. Here the author intended to make the main verb "idle", i.e. the strikes forced children to become idle, but another understanding would be to consider the "strike" as a verb, which would mean that the teacher hits children. Another important kind of ambiguity is ambiguity in the sense of the word. For example, in the text "The red tape is holding up the Bridges", the writer wanted the "holding up" words to mean a delay, but another understanding would be a different interpreting of the phrase "holding up" as support. Modern NLP process is made up of several tasks which are algorithms based on statistical Machine Learning. These tasks can be pipelined in different configurations for domain specific needs. This project uses the open source Stanford CoreNLP Natural Language Processing Toolkit, which is a Java library for core NLP tasks described in this section. This toolkit and its usage in this project is described in more details in section? Some the major NLP tasks used by this project are:

Tokenization

When processing a large document or long sentences, it is necessary to subdivide it into small parts, or tokens. This kind of slicing can also include the discarding of not very useful parts, such as punctuation, quotes, etc. For an example, for the input sentence,

“John, Sid and William, come near me”, the result of tokenizing the output will be single words with no punctuation marks, “John”, “Sid”, “William”, “come”, “near”, “me”.

Part-of-Speech Tagging (PoS)

Part-of-Speech (PoS) tagging includes the tagging of verbs, nouns adjectives and other parts of speech in the sentence. These tags belong to the category of words with related grammatical characteristics. For example, "Water" can be understood as a noun, as in "he is drinking water," or it can be a verb, as in "watering the plants." The English language has many of such ambiguities, and PoS helps to solve this problem.

Parsing

The analysis of a sentence grammatically is called Parsing. It includes the generation of a parse tree, similar to the parse trees generated by compilers for programming languages. Nonetheless, the grammar of natural languages has ambiguity, which can have multiple understandings and, therefore, several parse trees.

1.4. Twitter Data for Sentiment Analysis

Twitter is a social network and a microblogging service which provides a wealth of unstructured text. Registered users of this service can post short messages called tweets which are limited to 140 characters. These tweets are broadcasted publicly by default and the followers of the user are also notified [7, 8]. Twitter users can also interact with each other and share each other’s tweets which helps propagate the message to wider audiences and thus creating a viral effect. The messages or tweets have become a medium for people to share statuses, post interesting resource links, or to express their opinion on social, political and economic status quo. Table 1.2 lists few examples of twitter posts with expressed user’s opinions. The nature of such propagation of ideas has been seen by advertisers, political. Moreover, exploiting the real time nature of this service, information found on Twitter has also been shown useful for monitoring earthquakes [8] and predicting flu [1]. Twitter by 2015 will have more than 500 million active users [5] with more than 350 million tweets posted per day [15]. Twitter is basically mainstream, but it does not substitutes existing communications and media, but it complements them. It is used along with television as an "active audience" as a backchannel, through which social activity is maintained and becomes more broadly visible [3].

Table 1.2. Examples of tweets containing opinions. The "@" signs represent usernames

@richardebaker no. it is too big. I'm quite happy with the Kindle2.
House Correspondents dinner was last night whoopi, barbara & sherri went, Obama got a standing ovation
how can you not love Obama? he makes jokes about himself.
@stellargirl I loooooooovvvvvveee my Kindle2. Not that the DX is cool, but the 2 is fantastic in its own right.

This project uses Twitter as a corpus for sentiment analysis because of the richness and diversity of this platform. Twitter is mainstream and its adoption is growing ever so rapidly as an electronic word of mouth medium.

2. SENTIMENT ANALYSIS

Resources like Natural language processing, computational linguistics and text analysis all form great resources for extracting or identifying subjective information, this is generally known as sentiment analysis (or opinion mining). Many applications like marketing or customer service nowadays can sufficiently benefit from sentiment analysis that enables them to review people's or groups' opinions which can be extracted from social media or found in reviews.

On the other hand sentiment analysis uses neural networks and it needs a lot of text data to be build, trained and tested on, data like tweets on twitter or blog posts, any text that can contain a person's or a group's opinion about a product or an event, a huge amount of data that needs to be fed to the network, it also goes under a lot of processing, in the proposed study we aim to combine the great use of sentiment analysis with the processing capabilities of distributed systems in a well-coordinated manner.

2.1. Overview

With the increase in user reviews on the web, there has been a huge demand for opinion mining techniques which facilitate effective summarization of huge volumes of opinions. This goal can be achieved by identifying specific aspects of a product or service being reviewed and determining the sentiment expressed about these aspects. This task is popularly referred to as aspect specific sentiment analysis in literature [6].

In order to illustrate the task at hand, let us consider a text snippet expressing a customer's opinion about a particular beer. "This beer is tasty and leaves a thick lacing around the glass" This snippet discusses multiple aspects such as the taste of the beer and its appearance. The review expresses positive sentiments about both the aspects. It is interesting to note that the word "tasty" serves both as an aspect as well as a sentiment word in this case [10]. The phrase "leaves a thick lacing" suggests that the snippet is discussing about the appearance of the beer and usage of "thick lacing" can be attributed to positive sentiment. This example demonstrates the intricacies involved in the task of aspect specific sentiment analysis.

In order to tackle the problem at hand, several approaches ranging from heuristic based methods to sophisticated topic models have been proposed. However, there are two

major drawbacks with most of the proposed approaches. Firstly, a chunk of them [6, 12] treat the tasks of aspect extraction and sentiment analysis as two separate phases. The process of interleaving these two phases in a more tightly coupled manner allows us to capture subtle dependencies. Secondly, though there exist approaches which consider joint modeling of aspects and sentiments [8, 7, 11], they constrain the way these phases interleave by making rigid modeling assumptions. In order to address the aforementioned drawbacks, we propose a novel deep learning based framework for solving the problem at hand. The major distinguishing factor of this framework is that the joint modeling of aspects and sentiments is carried out without making strict modeling assumptions about the interleaving of aspect and sentiment extraction phases.

2.2. Sentiment Analysis on Big Data (Social Media)

Sentiment analysis field of study focuses on the public attitude, evaluation, emotion or any sentimental information in general regarding a product or an event real world text. It is one of the most efficient and most widely used area in natural language processing (NLP) and broadly benefited from in other fields like text mining, Web mining and data mining in general. Actually, this field of research, and duo to the importance that it provides to costumer-opinion-based businesses and society in general, it has spread to cover not only computer science but it has ranged outside to be used in other sciences like social sciences and management sciences. Duo to the growth in the use of social media like discussion forums, blogs, reviews, Twitter and other social medias, sentiment analysis has also shown great importance as the mentioned resources have evolved became more broadly used in the daily life to express opinions and thoughts, all of these resources form a huge digital data deposit for people's opinions to profited from when analyzed in the correct way [8].

Systems has been developed to harvest opinions and reviews over the web, in fact they are being used in most social and business fields because of the vital information that can be mined from opinions. Social media today tend to contain our thoughts, observations of reality, beliefs, and even choices that we make in our daily lives, all of these can be considered as indicators on how we evaluate the world around us [13]. We often need to consider other people's opinions regarding our personal life and behavior, the importance of others opinions is not only essential to individuals but for organizations also.

2.3. Vector Space Modeling

Vector Space Model is a successful model for the formal text representation of documents that can be understood by computers for manipulation which is based on algebra. In this algebra based model, text documents are represented in the form of vectors with dimensions strict by some different terms, which can be a sentence or a word, depending on the application. This powerful form of representation gives machines the ability of manipulation and querying of these documents by using vector operations. A set of these vectors, represented in the form of a mathematical structure, is called vector space [8, 16].

Words exist as a means of communication between people, but they are just labels people who speak the same language have agreed upon to communicate meaning. They offer nothing in so far as denoting what they describe. Words are dense representations in that each symbol of the word offers nothing in isolation, they are atomic units. In the alternative, a distributed representation, some of the dimensions may be lost and still information about the object is present. For example "cat" and "lion" trigger commonalities in the mind as they trigger internal representations, but are useless to computers for describing the objects they denote. What these systems need are its own distributed semantic vector representations. These semantic vectors describe, with each dimension, some feature of the input that is useful for solving meaningful problems.

2.4. Neural Word Embeddings

Neural word embedding translates a word to numbers, it simply does that but not like a normal translation. It is more like an auto-encoder that encodes each word and the result will be a numerical vector, it does that by training and examining each word with its relatives and neighbors in the inputted text (corpus) [3].

As mentioned earlier, two methods can be used in predicting the targeted result in neural word embeddings, first a method called continues bag of words (CBOW) which examines the contexts to predict the targeted word, and skip-gram that in fact does the opposite and uses the word to predict the context. In this paper we use the skip-gram because of its efficiency to predict more accurate results when it comes to huge datasets [17].

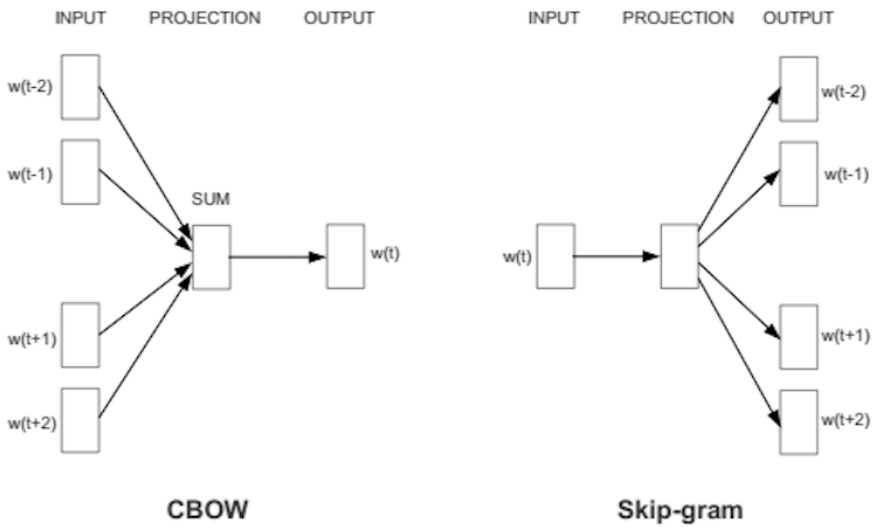

Figure 2.1. The skip-gram model architecture versus the continuous bag-of-words model architecture

When the word is encoded to a vector and that vector doesn't predicts the word's actual context accurately, the vectors components will be adjusted until it can do so, context are send back to be adjust until the highest accuracy is achieved. Words that have similar context, their vector's numbers are toggled until they are closer to each other.

For instance, a vector with arranged 500 numbers can be a representation of a one word or a group of words. Each word is located as point by those numbers in a 500 dimensional vectorspace. An average human mind finds difficulty in visualizing a three dimensional space. (when Geoff Hinton was teaching some people to try imagining a 13 dimensional space, he advised them to try imagining 3 dimensional space and then keep repeating to themselves: "Thirteen, thirteen, thirteen."

In order to call a set of word vectors a well-trained one, similar words in that space that have related context must be placed adjacent to each other. For instance, words like father, mother and family should be group in one cluster, whereas computer, technology and datamining would clustered in one place.

The relative meanings of similar ideas and things are put close, and have been encoded to values that can actually be measured. In another word, algorithms can be applied after representing qualities by quantities. Similarity forms the basis of the many relationships which Word2vec can learn.

Word2vec can achieve highly accurate prediction of the words meanings, but it depends on the data fed to the network, usage and context. By achieving highly accurate predictions we can present relationships that associates words to one another (e.g. “man” association to “king” is what “woman” is to “queen”), or group text documents and establish topic classification among them. This technique can be benefited from and become the basis for many search recommendation fields like customer relationship managing, scientific research and e-commerce [17].

Word2vec neural network outputs a set of items with their translated vectors attached to them, called vocabulary, and then it can be used to find relationships between words or fed to a deep-learning neural net.

Cosine similarity measurement is not expressed as a 90 degree angle, rather the total similarity of 1 is represented as a 0 degree angle or complete overlap; for example the word “Sweden“ equals “Sweden“, itself, whereas the word “Norway“ has a 0.760124 cosine distance from the word “Sweden“, which is the highest of any other country name.

Below is a list of the words associated with the word “Sweden” using Word2vec, in order of relationship:

Table 2.1. Words and their cosine distance in a word2vec model [1]

Word	**Cosine Distance**
Norway	0.760124
Denmark	0.715460
Finland	0.620022
Switzerland	0.588132
Belgium	0.585835
Netherlands	0.574631
Iceland	0.562368
Estonia	0.547621
Slovenia	0.531408

The results show that the cosine similarity actually represents the relativeness among those countries and Sweden, as it showed Scandinavian countries and some wealthy northern European countries.

These vectors can show more broad subjectivity between words, it can even show implicit semantics, for example, Berlin, Moscow, Ankara and Paris tend to be cluster under

one corner, but not only that, they also show that each of their cosine distance in the vectorspace to the countries they are capitals to, is the same distance; i.e. Ankara – Turkey is equal to Germany – Berlin and so on, and if you know that the capital of Turkey is Ankara and you wanted to know what is to Germany as in Ankara to Turkey, you can simply find the result in (Turkey - Ankara) + Germany, and you get Berlin as a result. Imagine the possibilities.

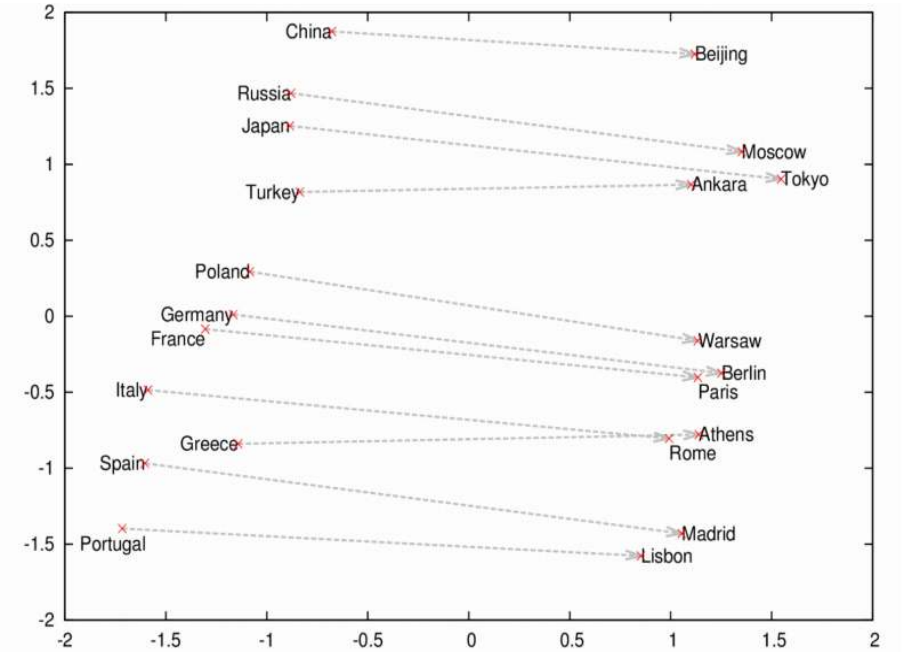

Figure 2.2. The projection of two-dimensional pca of a 1000-dimension skip-gram vector representation of countries and capital cities [1]

After training the model, we need to investigate the results, so we try to find word that is the relevant word to some user-specified words, by using the distance tool.

The model contains various NLP semantics that can be performed on word tasks. Some of those are built-in

```
>>> model.most_similar(positive=['man', 'queen'],
negative=['woman'])
[('king', 0.50882526), ...]
>>> model.doesnt_match("breakfast dog dinner lunch".split())
'dog'
>>> model.similarity('king', 'queen')
0.72723527
>>> model['computer']  #numpy vector of a word
```

```
array([-0.00549447, -0.00320097,  0.02424786, ...],
dtype=float32)
```

2.4.1. Skip-gram Models

Representation of words is limited to the fact that individual words cannot represent non-singular idiomatic expressions. For example, «Boston Globe» - This newspaper is not a natural combination of "Boston" values and "Globe" for this reason. Consequently, for having more expressive meaning we use skip-gram for vectors representations to represent the sentences. For example, the iterative autoencoders [17], other methods used to represent the meaning of the cues by drawing their vectors will also benefit from the use of the word vector instead of the vector. The Extension is relatively simple, from word based to phrase based on expression models. First define a large number of expressions using a learner-based approach and then treat these expressions as separate coins during training. To evaluate the quality of phrasal vectors, we developed a test set of similar problems with reasoning, which includes words and phrases. A typical pair of test kits are a typical pair "Montreal" "Montreal Canadians": "Toronto" "Toronto Maple Leafs". The closest representative is vec («Montreal Canadians») if the vec («Montréal») + vec («Toronto») («Toronto Maple Leafs») is believed.

Find textual representations that are useful for predicting words that surround a sentence or document - the aim of educational model skip-gram. More formal w1, w2, w3, . . . , wT , words given learning sequence, the goal of the model is to maximize the mean logarithmic probability of "skip -gram"

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

Here c - size of the scope of the training (which can also represent the center word function wt). Larger C can lead to more training samples and therefore achieve higher accuracy work, after training it. With using the SoftMax function, skip-gram base formulation will be p(wt+j |wt):

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^{\top} v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left({v'_w}^{\top} v_{w_I}\right)}$$

Where vw and v ′ w represent the “input” and “output” vector representations of w, and W is the number of words that the vocabulary contains. As a result of the cost of computing ∇ log p(wO|wI) is proportional to W, which is often large (105–107 terms), this formulation is impractical.

As mentioned earlier, the sentences have a value that is not the simple composition of the values of many individual words. We are in the first other context to find words that are seldom often displayed together and to find a vector representation for expressions. For example, some unique tokens are used to represent and replace "New York Times" and "Toronto Maple Leafs", whereas in Bigrams "this is" remains unchanged.

Thus, without significantly increasing the lexicon size, reasonable sentence becomes easier to form; theoretically, it will be possible to use all n-grams in the skip-gram training task, but it will be very heavy in terms of memory. Many methods have previously been developed for the purpose of identifying textual sentences in the text data; however, we are not able to compare them, it is out of the thesis coverage. The approach which we decided to use is more focused on simple data-driven, in which either unigram or bigram is used for sentences are based upon.

Finally, we reveal another interesting feature of the model skip-gram here. We have found that the simple addition of vectors often can yield significant results. For instance, these two vectors are very close vec («Russia») + vec («river») to vec («Volga River») and we can also get a close value from vec («Germany») and vec («Capital») to vec («Berlin»), can be obtained by using basic mathematical operations on vector representations of words at the explicit level.

2.4.2. Continuous Bag-Of-Words (CBOW)

News articles and research articles, as well as inquiries or suggestions can be made up of documents, of course, words. Documents are represented in most text mining methods in term-document matrices as column vectors, terms (more likely, words) represent the rows in those matrices. More precisely, an equivalent corresponding bag of words is representing the document vector.

The bag order is not important in the bag of words model, as for the structure, there is no structure. As in humans, we know that the language (and its structure) is set as word order, but neither is important here. The “cat on a hat” sentence (or query) cannot have the same assumed value, like a “hat on a cat” sentence. In addition, most of the time stopping words are ignored: High frequency words that have relatively low information content, such as function words (for example, the) and scripts or articles (for example, a). However, it is also clear that “cat on a hat” and “cat hat” sentences can be different, as they are interpreted by humans. This potential difference is shown in Figure 2.3.

(a) Example of a cat on a hat. (b) Example of a hat on a cat. (c) Example of cat hat.

Figure 2.3. Different sentences that have different possible illustrations

In addition to querying such context, there are other simple and practical situations where the language structure can be of great impact on the meaning. Most obviously, like in Machine Translation. Other difficulty - Classification of texts, such as opinion mining (or sentiment analysis).

Most previous models of a real language models, as well as a representation of words and the predication of the next word, gave the context in which they are studied as a probability distribution and representation of words. Learning the representations of words being the first. In this research we focus only on two models that care only about the word vectors: the continuous skip-gram model and the continuous bag of words model (CBOW) [3, 17].

As with CBOW architecture it is very similar to that of feedforward Neural Network Language Models (NNLM), but it doesn’t have non-linear hidden. The projection layer (not for matrix projection) is shared among all words; In this case, all words are estimated at the same location (the vectors are averaged). In this context, it implies that the words

order is irrelevant in the context, as the name of bag of words implies. It is important to note that contrary to the traditional language context model, future words are also contained in the context. The model cannot be tried to learn to guess the next word, but it makes sense, but to learn how to represent a word and limiting the context of the past words like in the natural language model is of no significant. In fact, this model can be regarded as a prediction of the current learning words according to context. Figure 2.4 shows the architecture of CBOW.

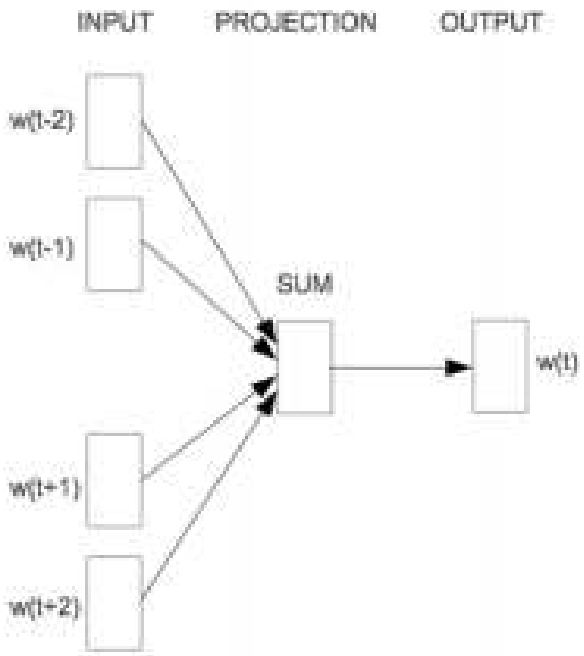

Figure 2.4. Continuous bag of words model [3]

As mentioned earlier in this chapter, The Skip-gram model uses these words to predict the sentence from them, making the words with distance sampled and with less weight.

Skip-gram model training is an effective way to simplify the negative sample, referred to as Noise Contrastive Estimation (NCE).

In the CBOW model, the input vector of the sentence word is not directly copied, but instead the average of the input vector of the context words is taken, and the product of input→hidden weight matrix with the vector of average is used when the output of the hidden layer is being computed.

2.5. Word Vectors (Word2vec)

Word2vec is text processing neural network with two layers. It takes a corpus (real world text) as an input and it outputs a set of vectors, each vector is represent features extracted for words in the corpus. Word2vec converts real world text into numerical for deep networks to handle, it is not considered as a deep learning neural network.

The applications of Word2vec' not only cover the sentences in the real world but they go beyond that to. It can be practical just as well to code, genes, playlists, likes, social media graphs and other verbal or figurative chains wherein patterns may be distinguished.

Word2vec is usefulness and purpose is mostly due to its grouping of similar and related words in a vectorspace. And this similarity between words is calculated mathematically. Vectors contain word Features, like the context of an individual word represented numerically in a distributed manner, and doing that autonomously without any intervention from humans [18].

Word2vec - a two-layer neural network that processes text. It takes a text corpus as input and its output is in this case the set of vectors, sets vectors for the words in that corpus. Although Word2vec is not considered a deep neural network, it can transform the text into a numeric values that the deep networks can understand.

Word2vec applications go beyond just parsing random words. It can be applied on any symbolic or verbal sequence where in which patterns can be distinguished, such as genes, social media or playlists.

Word2vec can make very accurate predictions about the meaning of the word if given enough contexts, usage and data, based on past occurrences o that word. The prediction later can be used to find word association between that word and another one (e.g. the word "boy" is to "man", as the word "girl" is to "woman"), that or, clustering documents to classify them based on topics. This type of clustering can form a base for search and recommendations in various field, such as customer relationship management (CRM), scientific research, e-commerce and legal discovery.

A Word2vec neural network output is a vocabulary in which you can find the relationships between words, each word bound to a numerical vector representation, the result can be simply queried to observe word relationships or can be fed to a deep learning algorithm for further analysis.

The cosine similarity in Word2vec, no similarity is expressed as an angle of 90 degrees, the exact similarity measurement of 1 is an angel of 0 degrees, which is a complete overlap; for example the similarity between the word Sweden and the word Sweden is an estimate of 1, while Norway to Sweden is 0.760124, which is the highest of any other "country" [19].

To cut a long story short: Word2Vec builds and N-dimensional latent space for word projections, in which N represents the size of the word vectors which is obtained. The coordinates of the words in the N-dimensional space are represented with float values. Latent space projection's basic idea is that objects are put in different space and continuous dimensional space, by representing the base objects by vectors who have more interesting properties to calculate than base objects.

To find similar words, you find words with maximum cosine similarity (equivalent to finding words with minimum Cosine distance after unit normalizing the vectors, which *most_similar* function is doing.

To find analogies, we can simply use the difference (or direction) vector between raw vector representations of word-vectors. For example,

- v('Paris') - v('France') ~ v('Rome') - v('Italy')`
- v('good') - v('bad') ~ v(happy) - v('sad')

In genism:

```
model = gensim.models.Word2Vec. load_word2vec_format (
'GoogleNews-vectors-negative300.bin' , binary=True )
model.most_similar( positive=['good', 'sad'],
negative=['bad'])
[(u'wonderful', 0.6414928436279297),
 (u'happy', 0.6154338121414185),
 (u'great', 0.5803680419921875),
 (u'nice', 0.5683973431587219),
 (u'saddening', 0.5588893294334412),
 (u'bittersweet', 0.5544661283493042),
 (u'glad', 0.5512036681175232),
 (u'fantastic', 0.5471092462539673),
 (u'proud', 0.530515193939209),
 (u'saddened', 0.5293528437614441)]
```

2.6. Paragraph Vectors (Doc2vec)

Text classification and clustering plays an important role in many applications for example, document search and retrieval, searching the web, filtering spams. Machine algorithms lays at the core of those applications, such as logistic regression or k-means [17, 18]. These algorithms usually take fixed length vectors as input so the text should be represented as such. Bag of words or bag of n-grams are (by Harris, 1954) the representations of fixed length vectors most commonly used, due to the high efficiency, simplicity and accuracy the offer [21].

But the bag-words (BOW) however, have some notable disadvantages. The word order is not considered and you can have the exact same representation from different sentences, if both contain the same words. Although the bag of n-grams take into account the order of the words in the short context, the high dimensionality and data sparsity are the factors that it suffers from. Only a little meaning in the semantics of the words and distances between is held by both the bag of words and bag of n-grams. This means that despite the fact that the word "strong" should be closer to the word “powerful” from the word “Paris”, the all have the same distance. In this research, we use Paragraph Vector [22], which is an unsupervised framework that represents text fragments into continuous distributed representations. The texts may vary in length from a sentence to a document. As the name Paragraph Vector emphasizes, varying lengths of text can be applied on, from a simple phrase to a large document.

It is possible to create representations of input strings in variable length by Vector Paragraph. Paragraph Vector, unlike some of the other approaches, it is common and applies to texts of any length: phrases, text paragraphs or a document. This does not require a special configuration of the weighting function for the words or the parsing tree. Later in the research, we present experiments on multiple databases that show Paragraph Vector benefits. For example, in the results of the work of Mikolov shows in terms of error rate more than 16% relative advance more than other state-of-art complex methods, complex methods and it beats the common method of bag of words by 30% improvement [17].

For more understanding the concept of Paragraph Vector we compare it to its successor Word2vec, a Word2Vec algorithms simply do this:

Imagine that you have this sentence:

The man went ___ for a walk.

You obviously want to fill the blank with the word "*outside*" but you could also have "*out*". The w2v algorithms are inspired by this idea. You'd like all words that fill in the blanks near, because they belong together. Therefore the words "*out*" and "*outside*" will be closer together whereas a word like "*carrot*" would be farther away.

As for paragraph vectors, they do exactly the same thing as in Word2vec but instead of using the sum, they average the word vectors and concatenate it with the paragraph vector, as if you had two orthogonal dimensions (x, y) where x is the word vector (of N dimensions) and y the paragraph vector (of M dimensions).

The paragraph vector and word vectors are averaged or concatenated to predict the next word in a context. In the experiments, we use concatenation as the method to combine the vectors. By re-propagating the gradient they get "a sense" of what's missing, bringing paragraph with the same words/topic "missing" close together. A paragraph token can be viewed as another word. It tends to act as a memorial factor that holds the current context's missing - or the paragraph topic [22].

Both Doc2vec and Word2vec convert a generic block of text into a vector similarly to how word2vec converts a word to vector. Paragraph vectors don't need to refer to paragraphs as they are traditionally laid out in text. They can theoretically be applied to phrases, sentences, paragraphs, or even larger blocks of text.

Doc2vec model gets its algorithm from word2vec. In word2vec there is no need to label the words, because every word has their own semantic meaning in the vocabulary. But in case of doc2vec, there is a need to specify that how many number of words or sentences convey a semantic meaning, so that the algorithm could identify it as a single entity. For this reason, we are specifying labels or tags to sentence or paragraph depending on the level of semantic meaning conveyed.

If we specify a single label to multiple sentences in a paragraph, it means that all the sentences in the paragraph are required to convey the meaning. On the other hand, if we specify variable labels to all the sentences in a paragraph, it means that each conveys a semantic meaning and they may or may not have similarity among them. In simple terms, a label means semantic meaning of something.

When it comes to distributional representation versus distributed representation, we would say that word2vec algorithms are based on both. When people say distributional representation, they usually mean the linguistic aspect: meaning is context, know the word by its company and other famous quotes.

But when talking about distributed representation, it mostly doesn't have anything to do with linguistics. It is more about computer science aspect. As in the works of Mikolov and others, the word distributed in their papers means that each single component of a vector representation does not have any meaning of its own. The interpretable features (for example, word contexts in case of word2vec) are hidden and distributed among uninterpretable vector components: each component is responsible for several interpretable features, and each interpretable feature is bound to several components.

Accordingly, word2vec (and doc2vec) uses distributed representations technically, as a way to represent lexical semantics. And at the same time it is conceptually based on distributional hypothesis: it works only because distributional hypothesis is true (word meanings do correlate with their typical contexts).

In paragraph vector, the vector tries to grasp the semantic meaning of all the words in the context by placing the vector itself in each and every context. Thus finally, the paragraph vector contains the semantic meaning of all the words in the context trained.

When we compare this to word2vec, each word in word2vec preserves its own semantic meaning. Thus summing up all the vectors or averaging them will result in a vector which could have all the semantics preserved. This is sensible, because when we add the vectors (transport + water) the result nearly equals ship or boat, which means summing the vectors sums up the semantics.

Before the paragraph vector paper got published, people used averaged word vectors as sentence vectors.

Furthermore, it is very important to note that word2vec (hence, doc2vec) can be used with both word vector representation techniques mentioned in the previous chapter, continues skip-gram and continues bag of word (CBOW) depending on the task that is intended for.

Results from our technique and some other baseline methods are listed in Table 2.2 for long documents that has multiple sentences, bag-of-words models are problematic to develop upon them when using word vectors. In 2012, by the work of [60] is where the most important development occurred, they managed to achieve enhancement in terms of error rate and they reached a minimum percentage of 10.77% by establishing a significant combination between bag-of-words and Restricted Boltzmann Machines models (WRRBM + BOW (bnc)). On the other hand, another enhancement occurred in the same year by the work of Wang & Manning in 2012 [59] and the work of Mikolov in 2014 [3], their work was done with bigram features-Naïve Bayes-Support Vector Machines (NBSVM-bi) and it achieved an even lower error rate percentage (8.78%). While the Paragraph Vector method which is used in this paper, it not only managed to go under 10% error rate, it also went beyond what both the mentioned methods could achieve, it was 15% (relatively) better than the best method out there (NBSVM-bi), it yielded 7.42% in terms of error rate.

Table 2.2. Paragraph vector performance on the IMDB dataset compared to other techniques [3, 59]

Model	**Error Rate**
NBSVM-uni (Wang & Manning, 2012)	11.71%
NBSVM-bi (Wang & Manning, 2012)	8.78%
Paragraph Vector	7.42%

2.7. Regular Machine Learning

2.7.1. Support Vector Machines and Naïve Bayes

Most of the research done in the field of sentiment analysis focus on the use of different ML algorithms that use different sets of features and compare their efficiency. In a study conducted by Rushdi-Saleh [24] for classifying movie reviews as negative or positive, they used two different weighing schemes during the process validation: the frequency of the term frequency-inverse of the document and the frequency of the term, as well as testing the interrupt effect during the pre-processing of the text. They reached 90% in terms of accuracy using SVM while 84% accuracy was achieved with NB, using somewhat the same weighing scheme and the same model of n-gram. The achieved close results to the results also achieved by Pang [25], whom also used frequency-inverse

frequency scheme for document weighting, using the SVM classifier, without applying any stemming during the preprocessing operation.

2.7.2. Classifiers Foundations of SVM and Naïve Bayes

To build a model that will be used in the problem of classifying of any unlabeled or invisible data, there must be a set of labeled data with the targeted class. If we only have two targeted classes, then we have a binary classification problem in this case; otherwise, this is the problem of classifying multi-classes. The construction of this model involves the selection of sets of functions that are thought to be related to the target class. These features sets will be extracted from the sentences to create a vector representation of the features for all sentences, with each feature attribute having an appropriate value. Each classifier has a function of: f (x): Rd → R, which assigns a sentence to its class [23].

Depending on the function result. For example, a binary classifier assigns the sentence to a positive class if the function value is greater than or equal to zero and assigns it to a negative class if the value of its function is less than zero. This chapter briefly describes the theoretical foundations of the Naïve Bayes and SVM classifiers, as they have been used in the literature to classify sentiments, and they will be used in our research.

2.7.2.1. Classifier Foundation of SVM

The basic idea of SVM is to define the boundaries of the decision that are based on the concept of plane decisions. These decision planes are defined as those that divided between a set of objects that have memberships in different classes. A special rule is created, called a linear classifier, in which its function can be written as:

$$\mathbf{f(x; w, b) = < w, x > + b}$$

Where w and b are the parameters of the function, and the signs "<", ">" are the inner product of two vectors.

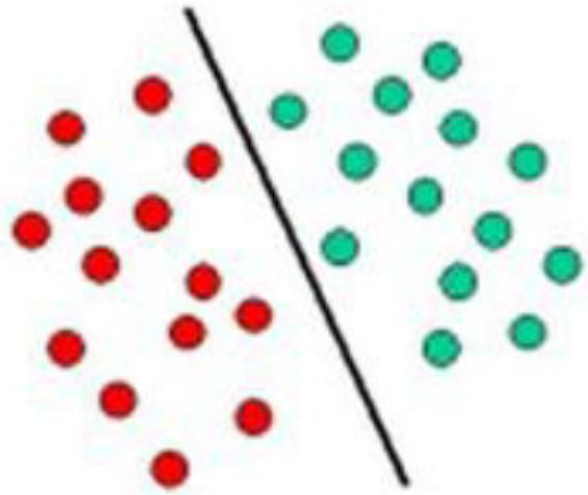

Figure 2.5. Support vector machines classifier

The data points used to prepare the training of the classifier are shown in the figure above. The figure shows that the main goal of the classifier is to find the best hyperplane that splits the data points of the negative class from the data points of the positive class with the maximum possible margin for each of the point's sets from the hyperplane [26, 27, 28]. Data points in the margin fields are called "support vectors". The most important property of training the data in SVM classifier is linear separation, where:

$$y_i f(x_i) > 0, \forall i = 1, \ldots, l$$

This means that the two hyperplanes of both field of margin can be chosen so that there are no data points between them [28].

2.7.2.2. Classifier Foundation of Naïve Bayes

The basic idea of Naïve Bayes is that the model is Bayesian theorem based, that works very well with high dimensionality inputs. The model has the word "naïve", because, given the class, the model assumes that conditionally the attributes are independent from one another. This assumption allows the model to calculate the probabilities of the Bayes formula from a training set which considered relatively small. The NB is based on conditional probabilities. The product of two probabilities which generates what is so called "posterior probability" which the final classification is based on. The prior probability is actually the first probability, and then the likelihood which is

the second probability. The prior probability is based on previous experience and it's an unconditional probability. In other words, that state of knowledge before the data is experimented. For each class, it is calculated as follows:

$$P(C = i) = \frac{\text{\# data points in class i}}{\text{Total Number of Points}}$$

Since objects are well clustered, we can undertake that with more points of data points in the neighborhood class to the class of X, the more likely that new points also belong to this particular class. For the likelihood to be measured, we draw a circle around X that includes several points (chosen a priori) regardless to their class labels. Then, the number of points in the circle belonging to each label of the class is calculated. The probability of likelihood is calculated as follows:

$$P(X \mid C = i) = \frac{\text{\# of i in vicinity of X}}{\text{Total \# of i cases}}$$

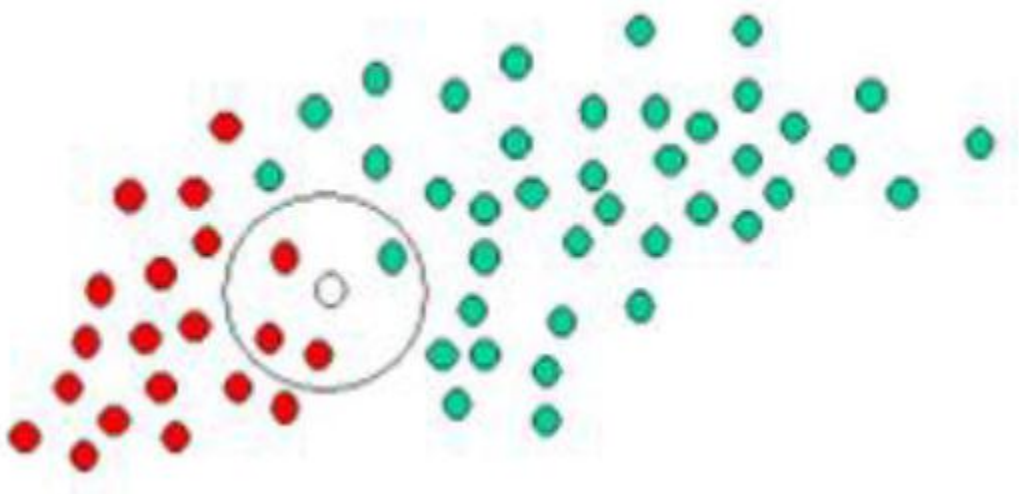

Figure 2.6. Naïve bayes classifier

The diagram above, demonstrates the plane of the data points used for the classifier training, and the new object neighborhood to be classified [28]. After obtaining the two probabilities, the likelihood and the prior, the result of their products is the final classification for the formation of a posterior probability using the so-called Bayesian rule:

$$f_i^{NB}(\mathbf{x}) = \Pi_{j=1}^{n} P(X_j = x_j | C = i) P(C = i)$$

Which is calculated for each class, and the new object will belong to the class that achieves the highest value. Xia & Zong [29] showed that unigrams perform better, in the SVM case, however in the Naïve Bayes case, dependency relationships and n-grams that are higher worked better. This is due to the algorithms nature themselves. A discriminating model such as SVM, can capture the complexity and independency of the relevant features, which is more present in the unigrams than in n-grams of a higher order. While a generative model like Naïve Bayes, can capture the assumptions about the independence of the features that are present in the bigrams and dependencies relation [29].

3. DEEP LEARNING

It is a broad field of research and this project has focused on its application to sentiment analysis, as set forth in [31]. Sentiment analysis is the determination of how positively or negatively an author of a piece of text feels about the subject of the text. This has applications in fields such as predictions of stock market prices and corporate brand evaluation.

Deep learning has led to state of the art innovation in the areas of Automatic Speech Recognition [30], Image Recognition [17] and Natural Language Processing [18]. This project specifically deals with its applications in sentiment analysis through the creation of semantic vector representations. A brief introduction is provided in this chapter to the technologies that have led to the state of the art results in sentiment analysis of sentences of varying length [31].

Deep learning is the process of learning successive semantic vector representations of data in a hierarchical manner. Each vector feeds into the next, more abstract vector to get more and more abstract representations. For example, given an input of an image on a network tasked to find faces, the first layer could detect edges, the next layer eyes, ears and noses and then a third could detect faces. Presented with raw image data at the input, it generates successively more abstract vectors to eventually carry out a task at the output.

To return to the cat analogy, one feature of an abstract layer would be ’is it a cat?’, followed on an even more abstract layer with ’is it a lion?’ and these would take cue from the lower layers of ’has it four legs?’ or ’has is it got fur?’. This is useful for simplifying the system as at each layer information is concentrated, abstracted and discarded. The process of learning is by determining how much of an error is in the network at the output. Functional units that produce the output are modified and optimized to reduce the error and eventually generalize to produce the correct output for unseen inputs.

3.1. Overview

Deep training is a form of machine learning that attempts to simulate the effectiveness and reliability of the presentation and learning of information by the human brain [32]. The human brain operates as a network of neurons that are interconnected with each other, the activation of which determines the path of linear recognition [33].

Proceeding from this, deep learning profound methods use artificial neural networks (ANNs) that are inspired by the biological neural networks, which are able of accomplishing machine learning tasks.

For example, a face recognition neural network can be well-defined as a neurons set that can be stimulated by the input image pixels. These inputs are transferred on the network from one neuron point to another, after it is worked on by some specific functions. This is recurrently done until the activation of the output neuron, which regulates the recognized object. Deep learning has its applications in many fields of computer science such as speech recognition, computer vision, robotics and NLP [31]. For a specific task, different architectures or models are used by deep learning neural networks.

3.2. General Idea of Deep Learning

Deep learning is a subfield which involves the use of neural networks. These neural networks try to learn the underlying distribution by modifying the weights between the layers. Now, consider the case of image recognition using deep learning: a neural network model is divided among layers, these layers are connected by links called weights, as the training process begins, these layers adjust the weights such that each layer tries to detect some feature and help the next layer for its processing [31, 32, 36]. The key point to note is we don't explicitly tell the layer to learn to detect edges, or eyes, nose or faces. The model learns to do that itself. Unlike classical machine learning models.

(a)Data flow in traditional machine learning

(b)Data flow in deep learning

Figure 3.1. The data flow in both machine learning and deep learning algorithms

In machine learning algorithms, such as linear regression or random forest you give the algorithms a set of features and the target and then it tries to minimize the cost function, so no it doesn't learn any new features, it just learns the weights. Now when you

come to deep learning, you have at least one, (almost always more) hidden layer with a set number of units, these are the features that are being talked about. So a deep learning algorithm doesn't just learn the sets of weights, in that process it also learns the values for hidden units which are complex high level features of the trivial data that you have given. Hence while practicing vanilla machine learning a lot of expertise lies in your ability to engineer features because the algorithm isn't learning any by itself [34].

Here are some applications of machine learning that you see ***very*** often in Silicon Valley:

- **Advertising**: what to show whom and when? The data comes fast and furious so the engineering part is quite sophisticated.
- **Recommendation**: similar to the advertising problem, except there are no advertisers, and the volume is slower, so you can afford to use more sophisticated machine algorithms.
- **Security**: did somebody outside penetrate your system? Did somebody inside abuse your system?
- **Text/audio/video processing**: information retrieval on media. Requires finding good representations of the items using machine learning; e.g., fetch all pictures similar to the input, where the similarity can be construed in various ways, such as similarity of texture, style, color, content, etc.

Conceptually, the first main difference between "traditional" (or "shallow") Machine Learning and Deep Learning is Unsupervised Feature Learning.

As you already know, successfully training a "traditional" Machine Learning model (ex: SVM, xgboost...) Is only possible after suitable pre-processing and judicious feature extraction to select meaningful information from the data. That is, good feature vectors contain features distinctive between data points with different labels and consistent among data points with the same label. Feature Engineering is thus the process of manual feature selection from experts. This is a very important but tedious takes to perform!

Unsupervised Feature Learning is a process where the model itself selects features automatically through training. The topology of a Neural Network organized in layers connected to each other have the nice property of mapping a low-level representation of

the data to a higher-level representation. Through training, the network can thus "decide" what part of the data matters and what part of the data doesn't. This is particularly interesting in Computer Vision or Natural Language Processing where it is quite hard to manually select or engineer robust features.

Figure 3.2. The feature extraction in machine learning

As an example, let's assume we want to classify cat pictures. Using a Deep Neural Net, we can feed in the raw pixel values that will be mapped to a set of weights by the first layer, then these weights will be mapped to other weights by the second layer, until the last layer allows some weights to be mapped to numbers representing your problem. (Ex: in this case the probability of the picture containing a cat).

Even though deep neural networks can perform unsupervised feature learning, it doesn't prevent you from doing feature engineering yourself to better represent your problem.

Deep neural networks can solve some non-linearly separable problems by bending the feature space such that features become linearly separable [35]. Once again, this is possible thanks to the network topology organized in layers mapping inputs to new representations of the data.

Lastly, deep neural networks are surpassing traditional machine learning algorithms in some domains because some architectures are showcasing statistical invariance (ex: spatial statistical invariance with convolutional neural networks and temporal statistical invariance with recurrent neural networks).

Deep learning is generally not well suited for domains that have only very small training datasets. This includes fields like medicine where, for example, patient data can

only be obtained through expensive and time consuming clinical trials. Clinical trials will often contain only several hundred records.

By design, deep learning models have many parameters (e.g., millions), and like with any machine learning algorithm, you're likely to see overfitting when the number of parameters greatly exceeds the number of training records.

Deep learning may be hot now but some variants of it or something new altogether may emerge later. These are some points that should be considered when talking about deep learning:

- Slow Learner

Slow learner as in, it converges to an optimal solution slowly but with GPU acceleration the training speed can be improved dramatically. The slowness is affected by the learning rate.

Adjusting the learning rate affects the reliability of the resulting deep neural net.

- Huge Training Data Requirement

Deep learning requires very huge training data to achieve good performance. The presence of a huge number of parameters to adjust requires some huge example sets. The fact that deep learning requires such a large number of training examples makes it ***dull*** in some way, despite such huge training set requirement deep neural nets do have error rates of about 10%.

- Overfitting

Deep learning tends to over fit easily but with the new dropout algorithm, this problem can be avoided but with some consequences such as an increase in error rates.

- Poor initial state

Deep neural nets have parameters that need to be initialized. The most used method is random initialization, this results in neural nets with very poor initial states. Compare this to a mammalian brain, the brain is born with some rigid instincts such as basic survival behavioral patterns.

- Sensory data transformations

For example in visual object recognition tasks, images undergo various geometric and photometric transformations that need to be modeled by a recognition system to rectify new image observations. Deep learning as used today does not take such transformations into account, this is one of the reasons why deep neural nets still suffer from relatively high error rates (relative to a human).

Some other reason can be the slow transformation, algorithms maybe emerge together in future, structure and the approaches [34, 36]. These are some pin points from my side. You can also go through these posts, they're also on it and to the point.

3.3. Feature Engineering

Traditional machine learning depends on feature engineering. Feature engineering is the practice of hand producing a set of functions that transform an input into a semantic or feature vector. The goal is to make the feature vector a good representation of the input. Whether that be by lowering the dimensionality and increasing the signal to noise ratio, or to perform nonlinear operations to transform a dimension of the output vector making it linearly separable from the other vectors of a different class. The engineer tries to define a semantic distributed representation by hand.

This feature engineering is time consuming, depends heavily on the expertise of the engineer and may miss important features of the data that would be useful in carrying out the task at hand. Also the degree of complexity to feed the output of a layer of features into the input of another layer grows with each successive layer of abstraction. This means truly powerful hierarchical representations of the input are difficult to manually engineer. Instead of defining these feature vectors by hand, they can be learned. These feature vectors may then reveal information about the data useful to carrying out the task that is less apparent, but still contributes to the robustness of the system. As the outputs of one layer of features are used as the inputs into the next layer of features, these features become more robust as each would be required to influence multiple features in the next layer. The forced generalization helps stop the features from overfitting. This can be used through multiple layers of features, creating complex hierarchical representations of the input. Further analysis can be found in [13].

3.4. Artificial Feed-Forward Neural Network

Networks are commonly trained via supervised learning, this is a method of training where the network has a dataset of inputs x, be they text, image, video or some other raw data, and corresponding labelled outputs y which describe correct answers for the each input on the task at hand. The dataset is then split into the training set and the testing set randomly. The network is then trained on the training set and its ability is assessed on the testing set.

One disadvantage of neural networks is they don't benefit from the head start most machine learning techniques receive from feature engineering. When engineering features the engineer can use intuition to rapidly develop meaningful features. An artificial neural network may take time to develop these features. It has to slowly move down a gradient in high dimensional pace, but given enough training data and computational resources they are capable of surpassing even the most complex handmade features. Without this foresight they can also get stuck in local minima in the gradient space [36]. This can be helped by using other optimization strategies such as simulated annealing.

3.5. Recurrent neural networks (RNN)

Recurrent neural networks are deep learning networks in which history is considered in by the neurons in the recurrent connections between them, unlike in regular feedforward neural net, where history is limited and represented by the context of N−1 words.

History length is unlimited. In addition, recurrent neural networks can learn to compress the whole history in a space with low dimensionality, while just a single word is compressed (projected) in the feedforward neural nets. Recurrent neural nets can create short-term memories, so they can cope with a network of positional invariants which feedforward nets cannot do.

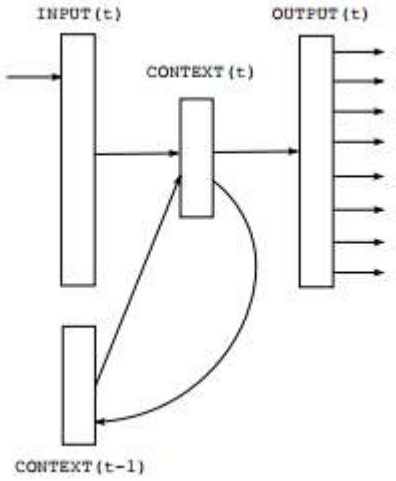

Figure 3.4. RNN-based language model

The recurrent neural networks have three layers, input layer referred to as x, hidden layer referred to as s (it is also referred to as the context or state layer) and the output layer referred to as y. By concatenating the vector w (that represents the current word) and the output from the context layer neurons s at time t −1, the input vector x (t) is formed. Word with frequent occurrence are merged into a single token for the sake of performance improvement. In RNNs computations are occurred with a relatively high cost, because the usually the hidden layer needs to be huge and the network will be evaluated for each character.

The hidden layer may keep a lot of information that are not be required to calculate the probability distribution of the **next word**, as it stores a huge amount of history information.

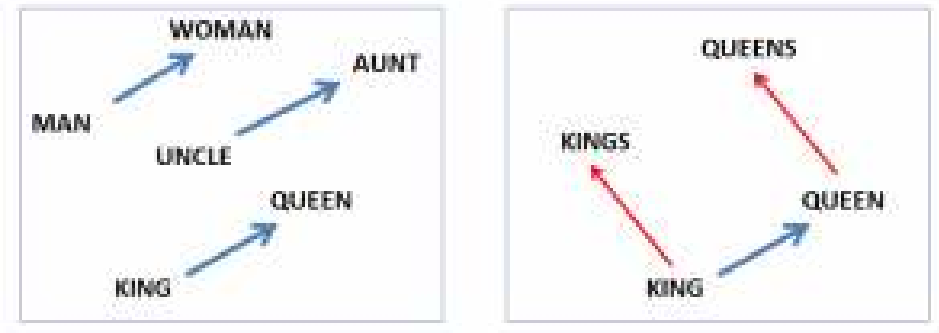

Figure 4.5. Left figure: the relationship between genders. Right figure: plural relationship between two words. Multiple relationships can be produced for a single word in high-dimensional space

3.6. Convolutional Neural Networks (CNN)

Deep learning models have accomplished noteworthy computer vision [37] and speech recognition [38] results in recently. The biggest part of the work in natural language processing with deep learning techniques has involved using neural language model for learning the word vector representations [39], and also using the represented word vectors for performing compositions for text classification and sentiment analysis [40].

Convolutional neural network (CNN) are used in conjunction with convolving filter applied to local features to utilize layers in the net [41]. Initially invented for the computer vision, the CNN model has been shown to be effective for NLP later and in semantic parsing by achieved excellent results [42], search engine queries, sentence modeling, and other traditional NLP tasks [40].

We find the high quality of word vector representations led to by the many language models on a pool of syntactic and semantic language problems as in the works of Tomas Mikolov. We noted that it is possible to raise high quality word vector representations using very simple architectural model when compared to the popular models of the neural network (both recurrent and feedforward nets). Because of a much lower computational complexity, a much larger set of data can be computed with very accurate and a large dimensionality word vectors.

Word vectors representations can also be efficiently applied to automatic extensions of the knowledge bases facts as shown in the studies of Mikolov, and as well as for the corrections verification of existing facts. The results of the machine translation experiments are also very promising. We believe that extensive testing will help the scientific community to develop methods for improving the representations of word vectors in the future. We also expect high-quality word vectors to be an important building block for future NLP implementations.

3.7. Convolutional vs. Recurrent Neural Networks in General

Natural language processing (NLP) has benefited greatly from the resurgence of deep neural networks (DNNs), due to their high performance with less need of engineered features. There are two main DNN architectures: convolutional neural network (CNN) [43] and recurrent neural network (RNN). Gating mechanisms have been developed to alleviate

some limitations of the basic RNN, resulting in two prevailing RNN types: long short-term memory (LSTM) [44] and gated recurrent unit (GRU) [45]. Generally speaking, CNNs are hierarchical and RNNs sequential architectures. How should we choose between them for processing language? Based on the characterization “hierarchical (CNN) vs. sequential (RNN)”, it is tempting to choose a CNN for classification tasks like sentiment classification since sentiment is usually determined by some key phrases; and to choose RNNs for a sequence modeling task like language modeling as it requires flexible modeling of context dependencies. But current NLP literature does not support such a clear conclusion.

For example, RNNs perform well on document-level sentiment classification Tang [46] and Dauphin [47] recently showed that gated CNNs outperform LSTMs on language modeling tasks, even though LSTMs had long been seen as better suited.

In summary, there is no consensus on DNN selection for any particular NLP problem. We can compare CNNs, GRUs and LSTMs systematically on a broad array of NLP tasks: sentiment/ relation classification, textual entailment, answer selection, question-relation matching in Freebase, Freebase path query answering and part-of-speech tagging.

These investigations support two key findings:

- CNNs and RNNs provide complementary information for text classification tasks. Which architecture performs better depends on how important it is to semantically understand the whole sequence.
- Learning rate changes performance relatively smoothly, while changes to hidden size and batch size result in large fluctuations.

A CNN will learn to recognize patterns across space. So, likewise, a CNN will learn to recognize components of an image (e.g., lines, curves, etc.) and then learn to combine these components to recognize larger structures (e.g., faces, objects, etc.).

So when comparing the two most widely used RNN and CNN – in representative sample of NLP tasks. We found that RNNs perform well and robust in a broad range of tasks except when the task is essentially a key-phrase recognition task as in some sentiment detection and question-answer matching settings. In addition, hidden size and batch size

can make the models performance vary dramatically. This suggests that optimization of these two parameters is crucial to good performance of both CNNs and RNNs.

We could say, in a very general way, that a RNN will similarly learn to recognize patterns across time. So a RNN that is trained to translate text might learn that "dog" should be **translated differently** if preceded by the word "hot".

The mechanism by which the two kinds of NNs represent these patterns is different, however. In the case of a CNN, it is looking for the same patterns on all the different subfields of the image. In the case of a RNN it is (in the simplest case) feeding the hidden layers from the previous step as an additional input into the next step. While the RNN builds up memory in this process, it is not looking for the same patterns over different slices of time in the same way that a CNN is looking for the same patterns over different regions of space.

We should also note that when we talk about "time" and "space" here, it shouldn't be taken too literally. You could run a RNN on a single image for image captioning, for instance, and the meaning of "time" would simply be the order in which different parts of the image are processed. So objects initially processed will inform the captioning of later objects processed.

3.8. Learning Semantic Vectors

Semantic vectors can be learned by a neural network being exposed to problems it can only solve by building layered hierarchical vector representations of the input that are necessary in solving the problem. The input can be any raw data and the network can learn these vectors and transformations via back-propagating the error on the output and adjusting the parameters to minimize the output of a cost function.

These vectors hold enough semantic meaning to attempt to solve any problem the network has seen in the past, so if the question “is it a cat?” is proposed to a network trained on words in the English language, inputs such as "lion" and "cat" will have probably have some dimension close to one, whereas the input “chair” will have a number closer to zero in that dimension. It is worth noting however that unless this information has been useful to the network in solving problem it has seen in the past, it would not learn this information and thereby a diverse input is essential for complete, robust learning. For example the network could have learned that a sufficient commonality to identify cats is

that they have four legs, but then it could confuse a chair for a cat given it has four legs. This is not a failure of the network, but more a failure of the training data. In practice vector dimensions are rarely this quantifiable, this example is contrived and for illustrative purposes. Getting this kind of information from the generated vectors requires learning a linear transformation of the vector space The idea of learning semantic vectors to represent words in the English language is introduced in [42], and developed further in [47], with the unsupervised generation of vector representations of words. The methods set out in the latter paper generate a language matrix over the words found on Wikipedia. This matrix in initialized to Gaussian noise and after training represents relations between words as approximately linear relationships in this high dimensional space, with each column vector representing a word across the language. Each word in the language indexes a vector from the matrix and provides it as input to the system. The network is trained to come up with meaningful representations of the input so it can carry out the task at hand, which in [44], is to give correct English sentences a higher score than incorrect English sentences.

The use of this method purely for mapping relationships to a linear space has been improved upon, with the current state of the art set out in [44]. The evaluation of this technology found that if the vector representing the word "Man" is subtracted from the vector representing the word "King", and this resultant vector is added to the vector for "Woman", we end up with a vector that is close to "Queen".

3.9. Composition of Semantic Vectors

The idea of composing semantic vectors is proposed in [41]. The idea of composing word vectors together to enable representations of varying length sentences is useful as it allows the semantic vectors of any sentence length to be compared. This paper re-introduces the Recursive Neural Network (RNN) set out in [43]. A RNN is a network that the output vector is the same in length as each of the two input vectors. This means the output can then be provided to the same network as an input at the next stage. Each input/output mapping uses the same weight matrix. This is useful for applying over a structure, such as an abstract syntax tree. Where the child nodes can be composed into an output vector representation at the parent node. This can then act as a child for its parent, until a single vector representing the tree is composed from bottom-up. This works under the assumption that the composition function for the children into a parent is constant for all child parent relations.

A problem with the basic RNN is the limitation on how one word can transform the other. The vectors are concatenated before being multiplied by the weight matrix and they cannot multiplicatively affect one another, they can only additively affect the output vector. Words like ‘not’ are inherently unary operators and have no real standalone meaning. They could be viewed more as matrices than vectors that should transform the semantic vector of another word. For example, ‘excited’ would have a specific meaning, ‘not excited’ would then be linearly transformed across the vector space by the word ‘not’. This can conceptually be achieved through the use of Matrix-Vector Recursive Neural Networks (MV-RNNs). MV-RNNs [43] seek to solve this issue with assigning each word a matrix, as well as a vector, that can transform the sibling vector. These matrices can be learned in the same way as the vectors and can enable words like "not" transform other words into a different area of the space before combining the two, allowing more flexible representations to be learned. However, the number of parameters gets squared by the introduction of the matrices and so does the dimensionality of the gradient space, so these matrices are difficult to learn. A lower parameter version of this approach is required.

3.10. Transformation to a Sentiment Space

Carrying the proposition that the semantic space contains all the semantic information relevant to the network during its training, then it follows that with relevancy of sentiment during training, the sentiment information must be contained in the semantic space. A projection can be performed into a sentiment space to reveal the sentiment information. A sentiment space with five interdependent dimensions is proposed, very negative, negative, neutral, positive and very positive. A vector can be transformed from the semantic space to this space by a matrix learned in the same way as the vectors and is trained on labeled data. So given a vector representing “this movie is terrible” in the semantic vector space, a matrix would be trained to output close to a 1 in the very negative dimension and zeros in all the other dimensions.

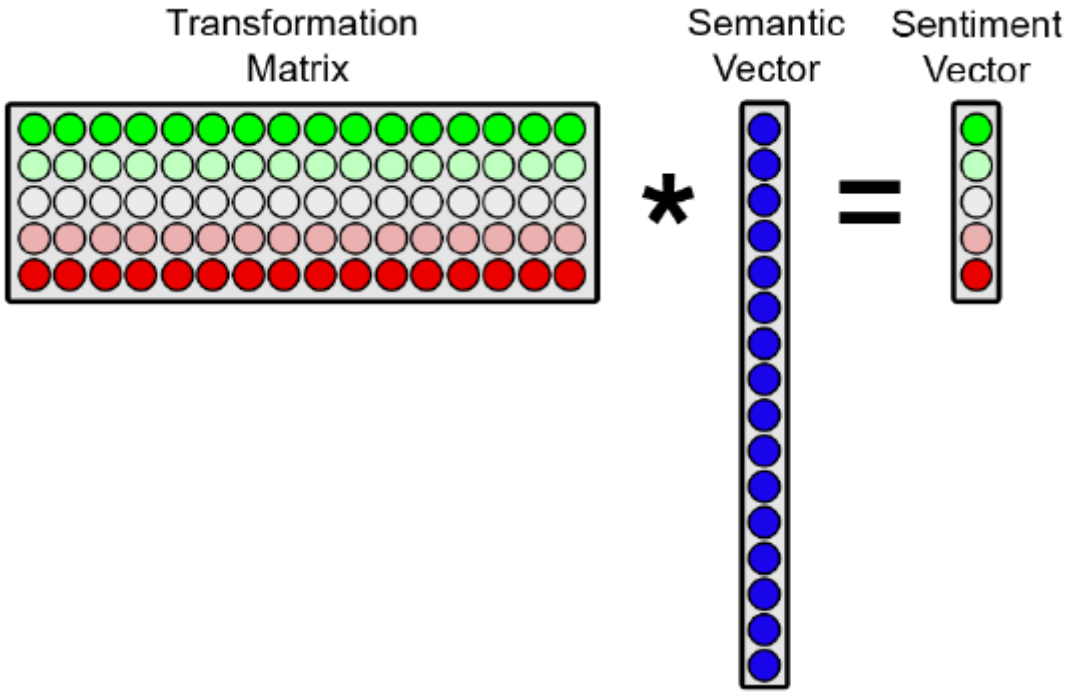

Figure 3.6. Example of the transformation of a semantic vector into the sentiment space

3.11. Sentiment Analysis and Semantic Vectors

The research in this project is about learning semantic vector representations and how to gain useful information from them. As the semantic vectors' axes are undefined, a transformation must be performed to known axes to gain insight into the information contained within the space. Sentiment is a poorly defined concept as it is not always objectively clear what the sentiment of a sentence is, for example "I love when people die in war" gets a sentiment rating of 0.54. This sentence contains sentiment, but in multiple, separate dimensions. When this space gets projected to a one dimensional space all the sentiment inherent in the sentence is lost.

If clear objective axes are defined, and a projection is learnt to them, the network could yield a more useful insight into the semantic meaning of the vector representations. If a multidimensional sentiment model could be proposed such as the sentiment of the writer on one axis and the perceived sentiment of the reader on the other, the network may be trained to produce a more robust, context free sentiment analyzer than the ones that have been seen so far. A model such as this is hard to define, the proposal above may not be a valid model, as the sentiment of a reader is a subjective concept and thus may require a different transformation based on the political and moral viewpoint of the reader [17].

It would be interesting to investigate what other meaningful projections could be achieved from the semantic vectors. A model where sentiment is decomposed into two dimensions may just be the beginning. The field of sentiment analysis could be generalized into one of semantic analysis with multiple dimensions revealing their own information along with sentiment. Multiple transformation matrices could encode different viewpoints and encode personal biases into the vectors and allow for insight into the personal semantic information a reader would receive from a piece of text.

3.12. Conclusions

In conclusion deep learning is an effective means of determining sentiment analysis, as sentiment is one projection on semantic meaning. The technology set out in [41] is the only technology that takes the entire semantic meaning of a sentence into account before applying this projection. Words alone cannot determine sentiment without first taking into account what they mean in the context of the sentence and the ordering they appear in. Without this composition of semantic vectors further progress in sentiment analysis is unlikely.

This powers two novel implementations of sentiment analysis, one on any piece of text visited by the browser, the other temporally mapping tweets across a map during a given scenario. Implementations of this type are effective uses of this technology and warrant further investigation in the future.

4. DISTRIBUTED COMPUTING

The terms "parallel computing" and "distributed computing" certainly have a large overlap, but can be differentiated further. We can consider "distributed computing" as the more general term that involves "distributed processing" as well as, for example, "distributed storage". The common term, "distributed", usually refers to some sort of message passing over a network, between machines that are physically separated.

The term "parallel computing" is also in the process of being further defined, e.g. by explicitly differentiating between the terms "parallel" and "concurrent", where - roughly - the first one refers data parallelism and the latter to task parallelism, although there are hardly really strict and binding definitions [50].

So one could say that "distributed processing" usually (although not necessarily) means that it also is "parallel processing", "distributed computing" is more general, and also covers aspects that are not related to parallelism, and obviously, "parallel computing"/"parallel processing" does not imply that it is "distributed"

Considering parallel processes, the code executed is the same, regardless of the level of parallelism (instruction, data, and task). You write a single code, and it will be executed by different threads/processors, e.g., while computing matrices products, or generating permutations.

On the other hand, distributed computing involves the execution of different algorithms/programs at the same time in different processors (from one or more machines). Such computations are later merged into an intermediate/final results by using the available means of data communication/synchronization (shared memory, network). Further, distributed computing is very appealing for Big Data processing, as it allows for exploiting disk parallelism (usually the bottleneck for large databases) [49].

Finally, for the level of parallelism, it may be taken rather as a constraint on the synchronization. For example, in GPU, which is single-instruction multiple-data (SIMD), the parallelism occurs by having different inputs for a single instruction, each pair (data_i, instruction) being executed by a different thread. Such is the restraint that, in case of divergent branches, it is necessary to discard lots of unnecessary computations, until the threads re-converge [49]. For CPU threads, though, they commonly diverge; yet, one may

use synchronization structures to grant concurrent execution of specific sections of the code.

4.1. Distributed Systems

Distributed systems can consist of sets of computers and/or a set of networks that are designed for the same purpose. "Concurrent computing", "Distributed computing" and "Parallel computing", similar and overlapping meanings can be observed from the three of these terms, and no clear difference between them can be really found. A given system can poses the characteristics of a "parallel" and "distributed" system characteristics at the same time; Processors in a common distributed system process at the same time (simultaneously) [48, 49, 51]. Parallel computing and distributed computing can be viewed as a special pair of close ones, but in some other cases they seem loosely related.

Distributed systems offer resource sharing. The importance and benefits of sharing resources are well understood to users [50]. The hardware shared by users includes, for example, soft resources in the form of data files, printers, and sometimes search engines (resources with a higher value and a very specific task).

4.2. Hadoop Distributed File System

The most powerful characteristic of Hadoop is that it works both in distributed (divides data and calculations among thousands of machines) and parallel (the calculations are performed simultaneously in parallel and near the data) environments simultaneously [51]. Hadoop clusters machines for storage, computation and I / O bandwidth purposes are usually commodity machines. It is deployed by Yahoo!, it has 25,000 servers and a of 25 PB storage capacity for application data only, with a 3500 cluster of servers, which is considered the largest in the system. Many other organizations and companies (one hundred worldwide) have made use of Hadoop deployment. Hadoop is a project of Apache; Yahoo has contributed as much as 80% and all components are available for free use by Apache all components (Ghemawat et al., 2003).

HDFS is a distributed component of the Hadoop project file system. In HDFS, data replication and metadata are stored individually. NameNode (which is a dedicated server) is used for storing the metadata of the system (like in GFS and Luster, other distributed file systems), which is the master server. While the other servers (the so-called DataNodes) are

used as data storage for the application. To connect all servers to each other TCP-protocols are used to ensure full communication between them. DataNodes hold application data replications for reliability, as in GFS [52].

4.3. Apache Spark

Apache Spark is an open source clustered computing environment, originally developed at the University of California at Berkeley in the AMPLab, but later transferred to the Apache Software Foundation as a donation, where it continues to this day [53]. By comparing it to the MapReduce paradigm, Hadoop has a paradigm of two stages and disk based, whereas the spark instead uses a multi-stage in-memory paradigm with assessed performance in some applications, 100 times faster [51]. When data is loaded into cluster memory in Spark, data requests will be faster than in Hadoop's. Spark provides a well-suited machine as well. MapReduce mechanism is shown in Figure 4.1.

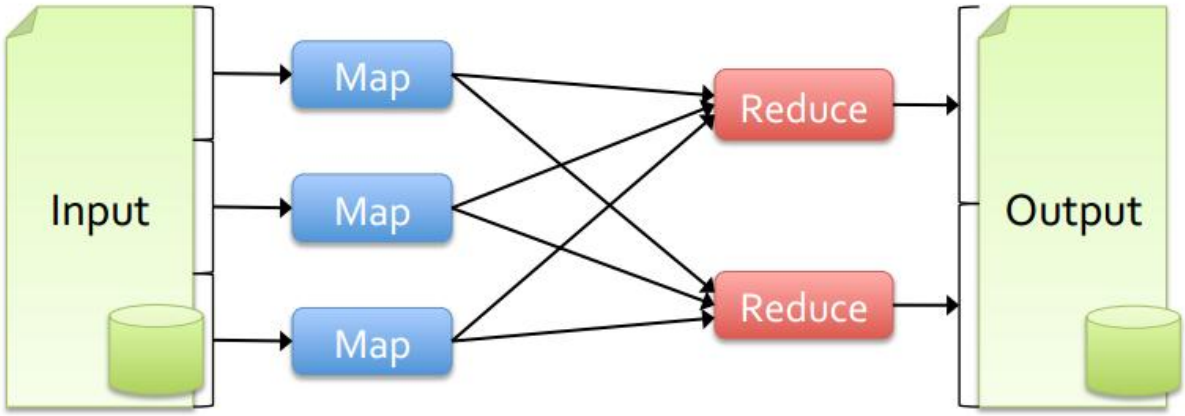

Figure 4.1. Mapreduce

To run Spark, you need some requirements, such as a distributed system sometimes (for example, OpenStack, Amazon S3, Kudu or the Hadoop HDFS) and a cluster management system (for example, its own cluster, Apache Moses or Hadoop YARN) [53]. Spark can also work in a custom implemented environment and can support a semi-distributed environment for testing purposes. It can use the local system instead of using a distributed system where the number of CPU cores are represented as the number of executers.

Spark is thought-out to be one of the most widely used projects with large source code, that is widely and not only in the Apache family, since its contributions have been estimated to overpass 450 in 2014 [54]. Turning it into one of the most widely used projects that deal with big data and datamining.

Apache Spark [51] - is a fast mechanism that processes in memory with elegant and impressive development of API, allowing workers to perform efficient data flow, machine learning or SQL workload that require fast and repeatable access to datasets.

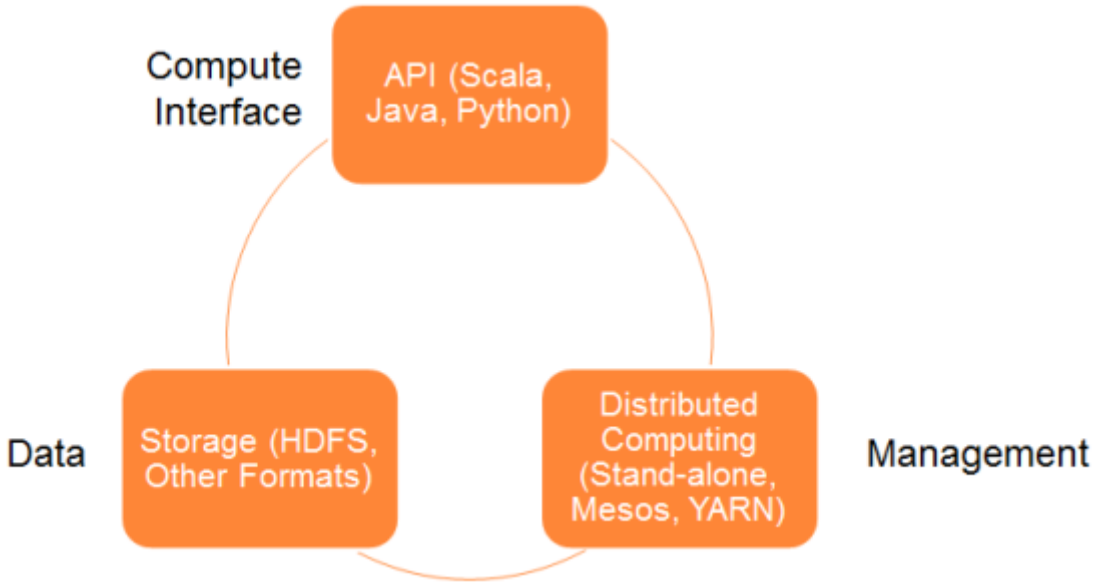

Figure 4.2. Spark architecture create it with word

Spark's architecture as shown in the figure above, consists of computing interface (APIs) that supports Scala, Java or Python programming, and data storages like HDFS or even a local system. Finally, the process management which can be Moses, YARN or stand-alone.

Apache Spark consists of a Core and a series of supporting libraries. Spark Core – provides a distributed execution engine and API mechanism for Java, Scala and Python providing a platform for the development of distributed ETL-applications (extract, transform and load).

Additional libraries built on the Spark Core allow the developer to perform workloads in various SQL, streaming, graphic and machine learning.

Spark is developed for big data analysis and its abstraction data is facilitating big data analysis. Machine learning is widely used by data scientists, machine learning algorithms tend to be iterative and the Spark acceleration for such iterative processing of

the data mechanisms makes it ideal for implementing such algorithms in large measures and datasets.

Spark also includes a library that provides a growing set of algorithms for general science research methods in machine learning: classification, regression, collaborative filtering, clustering and reduction in dimensionality, which is called Spark's MLlib [54].

ML Pipeline API in Spark - a high level abstraction that allows you to model the whole process of scientific knowledge. It is a level of abstraction that makes researchers more productive.

Spark endures some efficient characteristics:

- Speed – by reducing the number of read/write processes, spark can perform 10 times faster when it is working on the dick and 100 times faster when running on memory, because it tend to store the intermediate data in-memory.
- Multi-language support - Spark provides Java, Scala and Python built-in API. Thus, various languages can be used to develop applications. Spark presents interactive querying with 80 high level operators.
- Advanced Analytics: not only "map" and "reduce" operations are supported by Spark. SQL queries, machine learning, data streaming, and graphical algorithms are also supported by Spark.

Fortunately, there are alternative frameworks that can both - use data locality and keep data in memory between stages. Probably, the most notable of them is Apache Spark. Spark is complete replacement for Hadoop's MapReduce that uses its own runtime and exposes pretty rich API for manipulating your distributed dataset. Spark has several sub-projects, closely related to data science

4.4. Spark Components

4.4.1. Apache Spark Core

Spark Core is the main mechanism for implementing a platform in which all other functions of a spark are built. Where in-memory computing is provided and the reference to datasets that are stored externally.

4.4.2. Spark SQL

Spark provides a new data abstraction called SchemaRDD, which provides support for structured and semi-structured data, which is built on top of the Spark Core.

4.4.3. Spark Streaming

Spark Streaming is the streaming analytics performer that utilizes the quick scheduling of Spark Core. It absorbs data in mini packs data and perform RDD (Resilient Distributed Datasets) transformation on the mini data packets.

4.4.4. MLlib (Machine Learning Library)

Due to Spark's distributed memory-based architecture, MLlib – a distributed framework for machine learning that lays above Spark. In line with these criteria, developers have implemented MLlib against Alternative Least Squares (ALS) applications. Nine times faster than the Hadoop Apache Mahout which is disk-based.

4.4.5. GraphX

GraphX - A graphic processing distributed framework on top of Spark. This framework provides an API for graphical operations of the user-defined graphical computation, it uses Pregel abstraction. At the same time, it provides optimized runtime time for the abstraction.

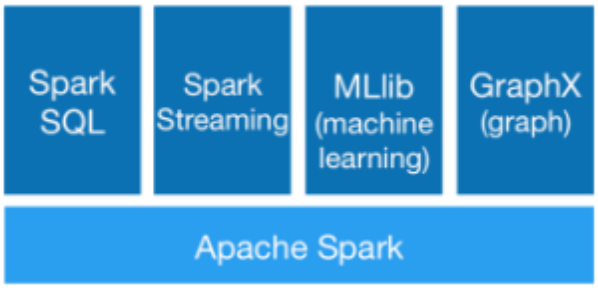

Figure 4.3. The basic components of spark

So there's pretty large set of data science problems that you can solve with Hadoop and related projects.

4.5. MapReduce

It is a framework for running computations on nodes with data. MapReduce heavily uses *data locality* exposed by HDFS: when possible, data is not transferred between nodes, but instead code is copied to the nodes with data.

So basically any problem (including data science tasks) that doesn't break data locality principle may be efficiently implemented using MapReduce (and a number of other problems may be solved not that efficiently, but still simply enough).

Let's take some examples. Very often analyst only needs some simple statistics over his tabular data. In this case Hive, which is basically SQL engine over MapReduce, works pretty well (there are also Impala, Shark and others, but they don't use Hadoop's MapReduce, so more on them later).

In other cases analyst (or developer) may want to work with previously unstructured data. Pure MapReduce is pretty good for **transforming** and **standardizing** data that is being our research case.

MapReduce (map/reduce) is most appropriate for parallelizable offline computations. To be more precise, it works best when the result can be found from the result of some function of a partition of the input. Averaging is a trivial example; you can do this with map/reduce by summing each partition, returning the sum and the number of elements in the partition, then computing the overall mean using these intermediate results. It is less appropriate when the intermediate steps depend on the state of the other partitions.

Some people are used to exploratory statistics and visualization using tools like R. It's possible to apply this approach to big data amounts using RHadoop package.

And when it comes to MapReduce-based machine learning Apache Mahout is the first to mention.

There's, however, one type of algorithms that work pretty slowly on Hadoop [55] even in presence of data locality, namely, iterative algorithms. Iterative algorithms tend to have multiple Map and Reduce stages. Hadoop's MR framework **reads** and **writes** data to disk on each stage (and sometimes in between), which makes iterative (as well as any multi-stage) tasks terribly slow.

4.6. Resilient Distributed Datasets

Resilient Distributed Datasets (RDD) are a basic data structure in Spark. This is an absolute collection of distributed objects. RDDs are divided into logical sections that can be computed on different nodes in the cluster. RDD can contain any type of Java, Python and Scala objects and it may include user-defined classes.

More formally, RDD - a collection of partitioned, read-only records. RDD data can be generated either from deterministic data stored in a stable storage or can be built from another RDD. It is a collection of elements which are fault-tolerant and can be dealt with in a parallel manner.

The developer has two ways of creating RDD – parallelizing an a collection of data that exists in the driver program, or setting a reference to a dataset which is stored in an external storage system, like a distributed file system, HDFS or HBase [56].

Spark uses the RDD concept to perform MapReduce operations faster and more efficiently. Let's talk about MapReduce operation first, and why they tend to be not so effective sometimes.

4.7. Using Spark for Data Sharing

Repetitive data replications, serializations and the slow disk I/O makes sharing the data slow in MapReduce. In most Hadoop applications 90% of the time is usually spent on performing the read and write in the HDFS.

Having realized this problem, researchers have developed a framework with a special structure called Apache Spark. The main idea about Spark is the Resilient Distributed Datasets (RDDs); it supports computation within memory. This means that the state of the memory is recorded as the object across all the works and the object is shared among all tasks. In memory data sharing is 10-100 times faster than when the data is stored on a disk and on the network.

Now let's try to understand the iterative and interactive process in Spark RDD.

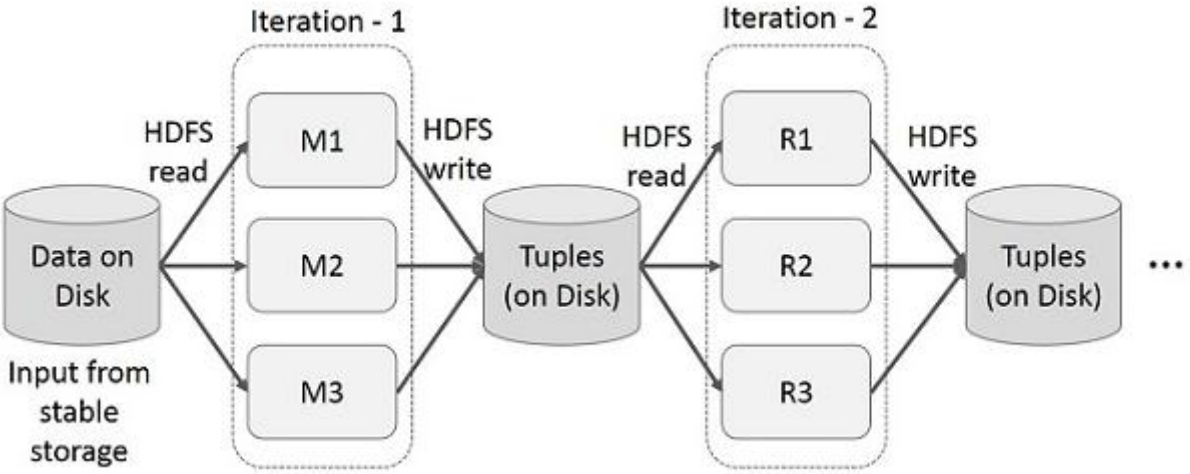

Figure 4.5. Regular mapreduce architecture

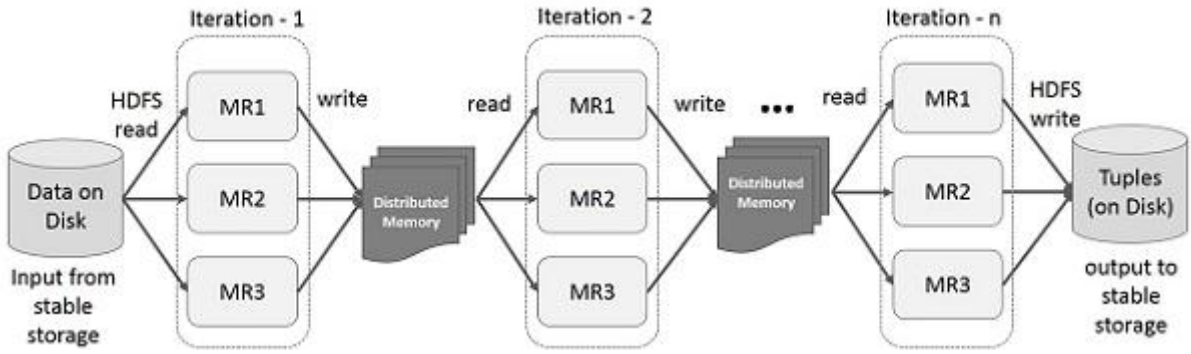

Figure 4.6. RDD architecture

Instead of storing the intermediate results in a stable disk storage, it stores the results in a distributed memory, thus making the system run faster, as the architecture of both MapReduce and RDDs are shown in Figure 4.5 and Figure 4.6.

Note. If the shared memory (distributed RAM) is not so reliable to store the intermediate results (the state of the current job), then the results are saved on the disk.

5. IMPLEMENTATION AND METHODS

In this chapter we show the most rival models and techniques in the state-of-art subject (sentiment analysis using deep learning) and show way we implemented each technique. We used 3 techniques for 2 different task (datasets) each, the tasks were two different dataset first the IMDB movie review dataset, which contained 100,000 critics reviews about movies; 50,000 for vocabulary, 12,000 negative reviews and 12,000 positive reviews, for both for train and test. The other task was 1,600,000 tweets collected from Twitter API, that contained half positive tweets and the half, negative tweets, we ran all the three models; Paragraph Vector, DL4J, and the NBSVM on both tasks.

5.1. Twitter Dataset: Preprocessing

Our dataset contained 1,600,000 tweets collected by Stanford university using Twitter API, the data is subdivided into two polarities, negative tweets and positive tweets [57]. The data was originally stored in CSV file format with 6 fields (columns) as follows:

1. The polarity of the tweet: 0 for negative and 1 for positive
2. Id of the tweet
3. The long data and time of the tweet
4. The query of the tweet: the subject, what is it talking about (i.e. Obama)
5. The user how posted the tweet
6. The text of the tweet

Table 5.1. Sample of 4 rows from the twitter dataset before preprocessing

4	2044	Mon May 11 03:19:04 UTC 2009	kindle2	SIX15	@kenburbary You'll love your Kindle2. I've had mine for a few months and never looked back. The new big one is huge! No need for remorse! :)
4	3148	Mon May 11 03:21:41 UTC 2009	kindle2	yamarama	@mikefish Fair enough. But i have the Kindle2 and I think it's perfect :)
0	3223	Mon May 11 03:32:48 UTC 2009	obama	kylesellers	@Karoli I firmly believe that Obama/Pelosi have ZERO desire to be civil. It's a charade and a slogan, but they want to destroy conservatism
0	3244	Mon May 11 05:06:22 UTC 2009	nike	vincentx24x	dear nike, stop with the flywire. that shit is a waste of science. and ugly. love, @vincentx24x

As observed from the above table, the data is not ready to be analyzed yet, it contains some characters and words that will affect the results of the algorithms. Thus we used some preprocessing steps for stripping the dataset from any unwanted data [58]. Words like tags or hashtags and such unwanted data can be stripped out using regular expressions, and it could be done by using any programing language, so I preferred VB.NET, since I am very familiar with it.

Table 5.2. The regular expressions and their use for preprocessing

Porpuse of the regular expression	The regular expression
Hashtags	(?:\#+[\w_]+[\w\'_\-]*[\w_]+)
Urls	((mailto\:|(news|(ht|f)tp(s?))\://){1}\S+)
	^[a-zA-Z0-9\- \.]+\.(com|org|net|mil|edu|COM|ORG|NET|MIL|EDU)$
	(?<=url\?q=).*?(?=&amp)
	http[s]?://(?:[a-z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-f][0-9a-f]))+
Emoticons	([oO0]*)([|:;=X^])([-']*)([)(oO0\]\[DPp*>X^@])
User tags (user names)	(?:@[\w_]+)
Emails	^[\w\.=-]+@[\w\.-]+\.[\w]{2,3}$
	^([\w\-\.]+)@((\[([0-9]{1,3}\.){3}[0-9]{1,3}\])|(([\w\-]+\.)+)([a-zA-Z]{2,4}))$
	^\w+[\w-\.]*\@\w+((-\w+)|(\w*))\.[a-z]{2,3}$
Others (unwanted spaces, numbers, unwanted characters)	&\w+
	[^A-Za-z]+
	[ ]{2,}

After the stripping (preprocessing) operation, the tweet that used to look like this “I like you Joash @joash338 ☺” ended up looking like this “i like you joash”. We generated different sub-datasets from the original dataset (1.6 M tweets, 800K positive and 800K negative) to further test the algorithms and models used upon data, and see what happens when different amounts of data are fed to the selected model. Finally the data was stored as pure text (instead of the CSV format), in some case (DL4J) the records (tweets) were required to be store separately (each record in a different txt file).

5.2. Doc2vec on Spark Using Gensim Library

Doc2vec is an extension from Word2vec, Doc2vec doesn’t just converts or translates words to vectors in the vectorspace, like in Word2vec, it rather also groups all the sentence’s words under one vector, and it does that by taking under consideration not only the sentence’s words but also the label of the sentence and treating it as a special word.

The reason behind using gensim to implement Doc2vec is that it is designed to extract subjective topics automatically from documents in a computer (efficient) and

human (painless) wise manner. It processes plain text (unstructured data) by applying unsupervised algorithms on plain text, which means no human interaction is required, only a lot of real world text documents. Once the data is processed and the patterns are found, we can query the outputted semantic representation for similarities between the documents.

Gensim provides many readable implementations of both Word2vec and Doc2vec, we also make advantage of general array manipulation techniques provided by numpy, and then we apply Logistic Regression using sklearn [59].

The proposed system uses Spark's RDD (Resilient Distributed Datasets), for loading the plain text file in RDDs, as RDD represent the data as a fault tolerant collection of elements that can be worked on in a parallel manner. Spark's MapReduce provides a well-suited environment for machine learning algorithms in general, by giving the user programs the ability to work with the cluster's memory and load and query data repeatedly from it.

5.3. Task 1: Implementing the IMDB Movie Review Classification

The reviews must first be cleaned by removing all punctuation and converting the text into lower case before being processed.

So the input data of the model will consist of five documents:

- Test data: 12500 negative movie reviews and 12500 positive movie reviews
- Training data: 12500 negative movie reviews and 12500 positive movie reviews
- Unlabeled data: 50000 movie reviews

5.3.1. Paragraph Vector

The model will first be trained on the two test data files (12500 negative movie reviews and 12500 positive movie reviews), and then we tested the model on another two test files (25000 movie reviews combined) to test the accuracy of the model, which gave an 87% accuracy.

In the Paragraph Vector framework, words and paragraphs are represented by unique vectors, each paragraph has its own unique vector and each word has its own unique vector. The vectors of paragraphs are then combined and averaged to predict the following word in the sentence's context. One paragraph vector is the same for all the contexts that are generated from the same paragraph, but it is not the same across all paragraphs.

However that is not the same for word vectors, a word vector the same and shared across all paragraphs for example, the vector for the word “strong” is identical for all paragraphs.

After creating a unique vector from the context of the paragraph for each paragraph, those vectors can be treated as the features of the paragraphs. Many conventional machine learning algorithms can be applied on the paragraph vectors like, K-means, support vector machine or logistic regression.

The used method, also according to the work of [60] can represent long documents with many sentences, so the input data doesn’t need to be parsed and divided into separated sentences, and this makes the method more reliable and useful in more cases compared to other methods. The IMDB dataset experiment validates this approach. It is first suggested that IMDB dataset to be a pint of reference regarding sentiment analysis. This dataset contains 100,000 reviews about movies where each review has multiple sentences which makes it more significant and diverse. The dataset holds a total of 100,000 movie reviews and it is separated into three sets: 50,000 labeled (25,000 training reviews and 25,000 test reviews) and 50,000 unlabeled training reviews, and the labeled reviews (both training and test reviews) are also subdivided into an equal number (12,500 each) of positive and negative reviews.

Word vectors are learned by training the model over 75,000 text documents, 25,000 of them are labeled and the 50,000 are unlabeled. A neural net that has 50 units feeds on the paragraph vectors of the 25,000 labeled reviews, this neural net has one hidden layer, the for prediction of the sentiment we use a logistic classifier.

5.3.2. Deep Learning for Java (DL4J)

This application runs on combination of both Word2vec and Recurrent Neural Networks (RNN), first it converts all words into vector representations and feeds the vectors to the recurrent net to find the polarity of the review. The dataset is "Large Movie Review Dataset" from Stanford University, and as this model requires, the data is stored separately meaning that, each record (**review**) is stored in a txt file format, so 25,000 files for training and another 25,000 files for testing, plus the 50,000 files for vocabulary. The code was based on “DL4J Examples” from Google.

5.3.3. Naïve Bayes Support Vector Machines (NBSVM)

This application uses Naïve Bayes-SVM and runs it on the IMDB movie review dataset (same as the later example) but it doesn't need to store each record in a separated file, it stores all the data that are associated together in a single file making them 4 files (training-negative, training-positive, test-negative and test-positive) plus the vocabulary file, it is based on a research by Wang and Manning [59]. Unlike the previous example which was in Java, this application is in python. In experiments based in the work of Wang and Manning [59] Support Vector Machines don't seem to work very well for long documents, so by combining SVM with Naïve Bayes we manage to set a very sophisticated supervised features that tend to score high results on terms of accuracy as we will show in the results chapter.

5.4. Task 2: Twitter Data for Sentiment Analysis

5.4.1. Paragraph Vector

Word2vec, as mentioned earlier gives vector representations of words (word embedding), in a form that words in with related meanings have vectors that are close to each other for example, the words "man" and "women" have a very close relationship to that of "king" and "queen" in a way that the word vector of "king" is to "queen" as "man" is to "women", that's if the model was fed correctly. However, in our case of study, only representing words with vectors just isn't enough, because we are dealing not only with words, but with sentences (paragraph), so adding the capabilities of Paragraph Vector to such word embedding can be used to represent paragraphs too. Therefore, by representing paragraphs with numerical vectors, we can run our analysis algorithms on paragraphs.

In this research, we used Gensim (a python library) since it presents many readable implementation of Paragraph Vector (Doc2vec) and thus Word2vec too. We also use Numpy for multidimensional array management, and for logistic regression we used SKLearn.

For achieving varying results, and after we ran the preprocessing operations, we sub-divided the Twitter dataset which consisted of 1,600,000 tweets (which had equal number of negative and positive tweets) into multiple datasets of varying sizes for all the training, testing and the vocabulary. And we also tried different parameter for the code (like epochs

and minimum count of words in the paragraphs) which we will get to in the results chapter. And finally we used logistic regression as a deep learning classifier by training it on the training data, and finding the accuracy upon the test data.

5.4.2. Deep Learning for Java (DL4J)

DJ4J provides and environment for deep learning in Java and some interesting implementation of deep learning algorithms and models, so with some changes on the code already provided by Google we managed to develop an application that uses Recurrent Neural Networks (RNN) with Word2vec for sentiment analysis on a big dataset .As mentioned earlier, DL4J needs the data to be stored, each record in a separated file. We prepared some datasets with various amounts of records to see how the accuracy is affected over data.

5.4.3. Naïve Bayes Support Vector Machines (NBSVM)

As the other two techniques that we used, we also prepared some datasets with various numbers of records, and we test this approach on the datasets three times each; for uni-gram, bi-gram, and tri-gram and observed the results in accuracy. It gave different accuracies with each approach.

6. RESULTS AND FINDINGS

6.1. Datasets Observations

At the beginning we should pay attention to a point that was observed in the implementing process of the Twitter dataset, Twitter posts (tweets) tend to be of special type English language, it contains many miss spelled words (like “Loove”) or even some special words that aren’t found in well grammar English (like “hahaha” or “hmm”) not mentioning the abbreviations (like “lol” or “OMG”) that are used and the characters that symbols specific context. Plus the sarcasm that appears a lot in the context of the sentences, which takes an experienced person to understand, such as these random sentences that we found in the dataset; “me toooo” and “aww thats sad”, both of these sentences were found in the negative part of the dataset, for the second sentence we can easily say that it is a negative sentence, but the first one doesn’t give a complete context, because it was originally associated with a sad emoticon before the preprocessing operation, it used to look like this “me toooo ☹”.

All of these exceptions tend to affect the decision of the machine to classify it as negative or positive. Also noting that some words like “aww” or “awww” may occur only once or twice per 10,000 tweets. And as so, both “who ever thought of twitter” and “wow good to hear” sentences were found in the positive section of the dataset, realizing the fact that Twitter data may not be of strict context as data with formal grammar (like IMDB data).

Another factor that can affect the results of the models is that, as observed from the implementation, Doc2vec (hence, Word2vec) relies not only on the number of records fed to the model, but also the count of words in each records, so it can give better representation to words and discover or enhance relationships between words. The Twitter sentences (records) that were used in this research, were found to have 12 as an average count of words per paragraph (record), while the IMDB movie review dataset had 238 (that almost **20 times more data** per record) words in a paragraph, and as observed by the techniques we used, the result can be affected by that factor.

6.2. Task 1: Implementing the IMDB Movie Review Classification

When running the different approaches on the IMDB dataset, all the approaches showed interesting results in terms of accuracy, wither by using RNN with Wor2vec like in DL4J, Paragraph Vectors or even regular machine learning, and they all took different time periods to accomplish the task.

Table 6.1. The accuracy and time spent for the three approaches

Method	Accuracy	Time elapsed on a single node
DL4J	75%	2 days
Paragraph Vector	87.03%	3 days
NBSVM-bi	91.55%	10 minutes
NBSVM-tri	**91.82**%	12 minutes

As the table above shows, NBSVM produced better results both in terms of accuracy and the time spent for the model to finish. As a machine learning algorithm, Naïve Bayes simply relies on just extracting features from the dataset and thus deciding wither the paragraph is considered to be positive or negative.

Table 6.2. The accuracy results from NBSVM on IMDB movie reviews datasets

Technique		Correctly classified	Accuracy
12,500 records	NBSVM-uni	22685/25000	90.74%
	NBSVM-bi	228875/25000	91.55%
	NBSVM-tri	22955/25000	**91.82**%
10,000 records	NBSVM-uni	17486/20000	87.43%
	NBSVM-bi	18104/20000	90.52%
	NBSVM-tri	18183/20000	90.915%
5,000 records	NBSVM-uni	8694/10000	86.94%
	NBSVM-bi	8955/10000	89.55%
	NBSVM-tri	8970/10000	89.7%
1,000 records	NBSVM-uni	1687/2000	84.35%
	NBSVM-bi	1753/2000	87.65%
	NBSVM-tri	1767/2000	88.35%

When we tested NBSVM on the IMDB movie reviews dataset, we tried to examine the performance of the approach on different sub-datasets with varying amounts of records to observe the influence of the factor of record numbers on the accuracy and speed of the approach, as the table above shows we used different numbers of records, the first column represents the number of reviews used for each of the test negative, test positive, train negative and train positive files which were fed to the application, and by knowing that NBSVM does not need a vocabulary dataset, the total number over records fed in the second row are 40,000 records (20,000 for testing and 20,000 for training) which the application managed to classify 18,183 of them correctly (90.915%) when using NBSVM-tri.

As for the other two Word2vec based deep learning approaches, most of the time is spent on word embedding process, and giving each unique word a unique numerical vector representation. Although they gave less sentimental accuracy than regular machine learning, they showed to be promising if their environment where to be more supportive (more data and more working nodes) as we are going to see in the coming sections.

6.3. Task 2: Twitter Data for Sentiment Analysis

Twitter and other social media form a great source for people's opinions, we have been trying to harvest and benefit from it for the past years using different techniques of computer science. On the other hand, twitter posts tend to be of special language, it doesn't look like the English we see in books and formal English writings, and it contains special symbols, words, and combination of characters (emoticons), plus the fact that the length of a post is usually short (12 average count of words in our case) which means it comprises less context: the major factor that our approaches rely on. These exclusions disturb the accuracy of a sentiment analysis model that depend mostly on the contextual meaning and the relationships of the words and the sentences.

We had varying results of the various techniques that we used. First to begin talking about are the results of Deep Learning for Java (DL4J).

Table 6.3. The accuracy of DL4J on three twitter datasets with varying sizes

Number of Records for Each of the Training And Testing Data	Number of Records for Vocabulary	Accuracy
5,000	20,000	64.82%
50,000	200,000	66.93%
100,000	400,000	**67.82%**

Note: the 5,000 in the first row of the above table means that we used 5,000 records for positive training, 5,000 records for negative training, 5,000 records for positive testing and 5,000 records for negative testing, and as such in the tables. And due to the long time the DL4J took to finish using this dataset, we ran the application on only 3 datasets which each of them took days to finish.

Being the approach with the least accuracy achieved, DL4J was tested on two datasets although it achieved a humble accuracy for the RNN and Word2vec (~68%), it showed some promising results as it shows improvement in the accuracy when size of the dataset was increased.

As for the Doc2vec (Paragraph Vector) using Gensim library, we tried different datasets and also different parameters to see what affects the accuracy and what way, so we prepared 7 datasets of Twitter records, some with minimum count of words of 12 words, and some with different number of epochs used for the neural network.

Table 6.4. The accuracy of paragraph vector on different datasets and parameters

Number of Records for Each of the Training And Testing Data	Number of Records for Vocabulary	Accuracy	Parameters changed
1,000 negative	20,000	65.25%	
5,000	20,000	67.07%	
40,000	180,000	67.51375%	
50,000	100,000	69.631%	
100,000	400,000	70.8%	15 epochs
100,000	400,000	71.015%	12 min word count
100,000	400,000	69.331%	
50,000	500,000	71.15%	
12,000	800,000	**72.0708333333**%	
1,000 test 100,000 train	400,000	67.15%	

We used Paragraph Vector model with 10 epochs, it gave better results than the DL4J platform gave (Table 6.3 and Table 6.4) and by increasing the amount of records for the dataset it also showed improvement in the accuracy (as DL4J). Also it showed that we can get better results not only by increasing the number of records but the amount of context that the paragraphs held in (count of words) and the increasing number of epochs (hence, more time needed to finish).

As for the regular machine learning approach (Naïve Bayes SVM), we also used different sized datasets and different gram-features (uni-gram, bi-gram and tri-gram), the bi-gram gave the best results. This approach showed way better results than the two deep learning approaches in both the accuracy achieved and the time needed to run, as it only extracts features from the training data and applies them on the test data (no vocabulary need), instead of the word embedding process that both the above methods do and which took most of the overall time needed to finish the operation. Below is a table of results for the NBSVM with different grams and datasets.

Table 6.5. The accuracy results from NBSVM on twitter datasets

Technique		Correctly classified	Accuracy	Time consumed
250,000 records	NBSVM-uni	392535/500000	78.50%	3 min
	NBSVM-bi	400443/500000	80.09%	10 min
	NBSVM-tri	402764/500000	80.5528%	18 min
100,000 records	NBSVM-uni	157836/200000	78.918%	1.5 min
	NBSVM-bi	160375/200000	81.19%	3 min
	NBSVM-tri	161228/200000	80.614%	7 min
50,000 records	NBSVM-uni	78463/100000	78.463%	1 min
	NBSVM-bi	79683/100000	79.683%	2 min
	NBSVM-tri	80046/100000	80.046%	2 min
10,000 records	NBSVM-uni	15156/20000	75.78%	< 10 sec
	NBSVM-bi	15376/20000	76.88%	< 10 sec
	NBSVM-tri	15434/20000	77.17%	< 10 sec

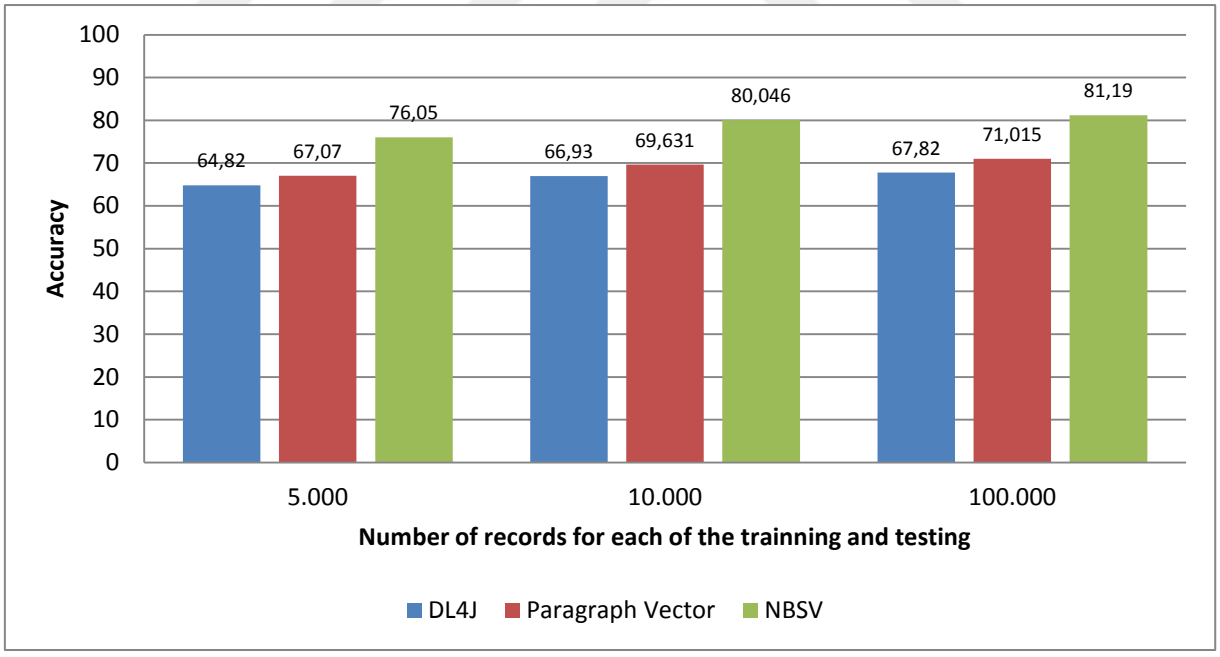

Figure 6.1. Accuracy vs. data on all three approaches

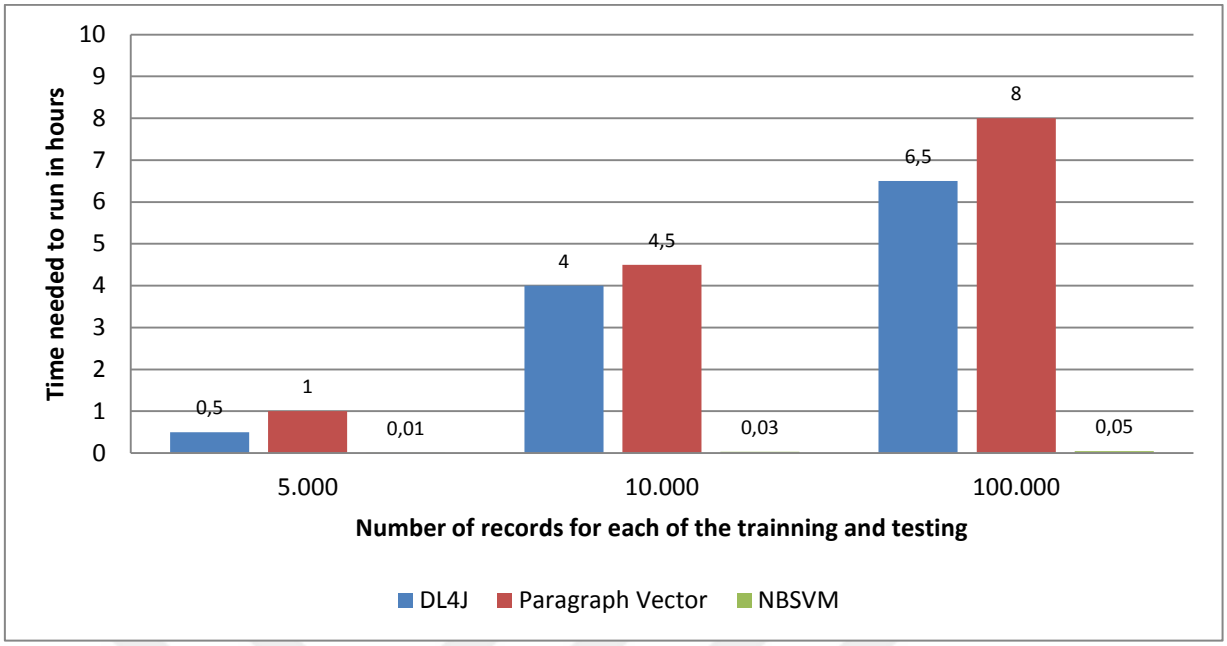

Figure 6.2. Time spent vs. data on all three approaches

In figures 6.1 and 6.2 we used three Twitter datasets with different numbers of records per file (5000, 10000 and 100000 records, for each of the test-pos, test-neg, train-pos and traing-neg files), in both DL4J and paragraph vector models the datasets are also associated with one extra file which is used for vocabulary (4 times the number of records in each file) but in the case of NBSVM the model doesn't take a vocabulary. When comparing the approaches in terms of data vs. accuracy, all three show increase in the accuracy when the data contains more information (records and words per record), but in the case of NBSVM the model stopped showing significant increase in the accuracy when the data passed the 100 thousands record bar, unlike the deep learning models (DL4J and paragraph vector).

As for the time taken for each approach to finish the task, the deep learning approaches took a lot more time compared to the regular machine learning approach, NBSVM, the later took much shorter time that when designing the chart (figure 6.2) we found it hard to compare the results, NBSVM took fractions of an hour (seconds), while the other two took several hours to run on a single machine.

As a summery for the Twitter dataset using all three approaches, we will compare the performance of the approaches on the same amount of records used (100,000 tweets) in terms of accuracy and time needed to accomplish the task. When using DL4J (as shown in the table below) it showed to be the model with least accuracy and it also took a long

period of time (several hours), while the paragraph vector using genism accomplished a relatively higher accuracy but in a longer time period, and finally in the NBSVM it not only showed a high accuracy of ~81% but also accomplished the task much faster (7 minutes in NBSVM-tri). In another point of view, these experiments showed that using deep learning (DL4J and paragraph vector in our case) can show promising results when used in a suitable environment with multiple powerful nodes and a dataset with more text per record.

Table 6.6. Different approaches ran on the same dataset

Approach	Number of Records for Each of the Training and Testing Data	Number of Records for Vocabulary	Accuracy
DL4J	100,000	400,000	67.82%
Paragraph Vector	100,000	400,000	69.331%
NBVM-tri	100,000	No vocabulary	**80.614%**

7. CONCLUSION

In this paper we applied an unsupervised learning algorithm (Paragraph Vector) to use for representing words and paragraphs with varying sizes by vectors. Those representations of vectors are then used to predict missing or potentially related words in the paragraph, relying on the context. And by using a distributed computing platform for processing the huge amount of real world text needed as input for the models we minimize the effort and time needed for the learning and testing procedures. We also used some other competitive and widely used methods in the same area of study for comparison reasons, the paragraph vector method showed less relatively high accuracy for predicting the sentiments of the paragraphs. Although NBSVM gave better results, but our method showed some promising factors like the increase of accuracy over data which NBSVM didn't really show. The case study that we used supports that the used method can be competitive and reliable compared to other state-of-art methods and it also shows that paragraph Vector can overcome some of the downsides of bag-of-words models in terms of error rate.

Although the proposed approach takes time to accomplish more than regular machine learning, but most of the time is actually taken to represent the words in vectors (digitally understand the context of words) and built its own set of feature (major feature of deep learning) unlike regular machine learning which comes with a predefined set of features (supervised or semi-supervised).

REFERENCES

[1] **Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J.** (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111-3119).

[2] **Cocuzzo, D., & Wu, S.** (2013). Hit or Flop: Box Office Prediction for Feature Films. Stanford University.

[3] **Le, Q. V., & Mikolov, T.** (2014, June). Distributed Representations of Sentences and Documents. In ICML (Vol. 14, pp. 1188-1196).

[4] **Rong, X.** (2014). word2vec parameter learning explained. arXiv preprint arXiv:1411.2738.

[5] **Bezáková, I.** Analyzing Sentiments of Twitter Trends Using Deep Learning (Doctoral dissertation, Rochester Institute of Technology).

[6] **Xu, X., Liang, H., & Baldwin, T.** (2016). UNIMELB at SemEval-2016 Tasks 4A and 4B: An Ensemble of Neural Networks and a Word2Vec Based Model for Sentiment Classification. Proceedings of SemEval, 183-189.

[7] **Lakkaraju, H., Socher, R., & Manning, C.** (2014). Aspect specific sentiment analysis using hierarchical deep learning. In NIPS Workshop on Deep Learning and Representation Learning.

[8] **Abdelwahab, O., & Elmaghraby, A.** (2016). UofL at SemEval-2016 Task 4: Multi domain word2vec for Twitter sentiment classification. Proceedings of SemEval, 164-170.

[9] **Brady, C.** On Using Deep Learning For Sentiment Analysis.

[10] **Ghiyasian, B., & Guo, Y. F.** (2014). Sentiment Analysis Using SemiSupervised Recursive Autoencoders and Support Vector Machines.

[11] **Srividya, K., Sowjanya, A. M., & Kumar, T. A.** (2017). Sentiment analysis of facebook data using naïve bayes classifier. International Journal of Computer Science and Information Security, 15(1), 179.

[12] **Cambria, E., Schuller, B., Xia, Y., & Havasi, C.** (2013). New avenues in opinion mining and sentiment analysis. IEEE Intelligent Systems, 28(2), 15-21.

[13] **Pang, B., & Lee, L.** (2008). Opinion mining and sentiment analysis. Foundations and Trends® in Information Retrieval, 2(1–2), 1-135.

[14] **Liu, B.** (2012). Sentiment analysis and opinion mining. Synthesis lectures on human language technologies, 5(1), 1-167.

[15] **Collobert, R., & Weston, J.** (2008, July). A unified architecture for natural language processing: Deep neural networks with multitask learning. In Proceedings of the 25th international conference on Machine learning (pp. 160-167). ACM.

[16] **Mikolov, T., Chen, K., Corrado, G., & Dean, J.** (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

[17] **Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J.** (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111-3119).

[18] **Levy, O., & Goldberg, Y.** (2014, June). Dependency-Based Word Embeddings. In ACL (2) (pp. 302-308).

[19] **LeCun, Y., Bengio, Y., & Hinton, G.** (2015). Deep learning. Nature, 521(7553), 436-444.

[20] **Dai, A. M., Olah, C., & Le, Q. V.** (2015). Document embedding with paragraph vectors. arXiv preprint arXiv:1507.07998.

[21] **Harris, Z. S.** (1954). Distributional structure. Word, 10(2-3), 146-162.

[22] **Le, Q. V., & Mikolov, T.** (2014, June). Distributed Representations of Sentences and Documents. In ICML (Vol. 14, pp. 1188-1196).

[23] **Moraes, R., Valiati, J. F., & Neto, W. P. G.** (2013). Document-level sentiment classification: An empirical comparison between SVM and ANN. Expert Systems with Applications, 40(2), 621-633.

[24] **Kaleh, M. R., Martín-Valdivia, M. T., Montejo-Ráez, A., & Ureña-López, L. A.** (2011). Experiments with SVM to classify opinions in different domains. Expert Systems with Applications, 38(12), 14799-14804.

[25] **Pang, B., Lee, L., & Vaithyanathan, S.** (2002, July). Thumbs up?: sentiment classification using machine learning techniques. In Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10 (pp. 79-86). Association for Computational Linguistics.

[26] **Gamon, M.** (2004, August). Sentiment classification on customer feedback data: noisy data, large feature vectors, and the role of linguistic analysis. In Proceedings of the 20th international conference on Computational Linguistics (p. 841). Association for Computational Linguistics.

[27] **Bifet, A., & Frank, E.** (2010, October). Sentiment knowledge discovery in twitter streaming data. In International Conference on Discovery Science (pp. 1-15). Springer Berlin Heidelberg.

[28] **Fradkin, D., & Muchnik, I.** (2006). Support vector machines for classification. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 70, 13-20.

[29] **Xia, R., & Zong, C.** (2010, August). Exploring the use of word relation features for sentiment classification. In Proceedings of the 23rd International Conference on Computational Linguistics: Posters (pp. 1336-1344). Association for Computational Linguistics.

[30] **Bezáková, I.** Analyzing Sentiments of Twitter Trends Using Deep Learning (Doctoral dissertation, Rochester Institute of Technology).

[31] **Schlünz, G. I.** (2014). Advanced natural language processing for improved prosody in text–to–speech synthesis (Doctoral dissertation, North-West University).

[32] **Kim, Y.** (2014). Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882.

[33] **Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., & Khudanpur, S.** (2010, September). Recurrent neural network based language model. In Interspeech (Vol. 2, p. 3).

[34] Weigel**, V. B.** (2002). Deep Learning for a Digital Age: Technology's Untapped Potential to Enrich Higher Education. Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741.

[35] **Mikolov, T., Kombrink, S., Burget, L., Černocký, J., & Khudanpur, S.** (2011, May). Extensions of recurrent neural network language model. In Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on (pp. 5528-5531). IEEE.

[36] **Mikolov, T.** (2012). Statistical language models based on neural networks. Presentation at Google, Mountain View, 2nd April.

[37] **Krizhevsky, A., Sutskever, I., & Hinton, G. E.** (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[38] **Graves, A., Mohamed, A. R., & Hinton, G.** (2013, May). Speech recognition with deep recurrent neural networks. In Acoustics, speech and signal processing (icassp), 2013 ieee international conference on (pp. 6645-6649). IEEE.

[39] **Mikolov, T., Yih, W. T., & Zweig, G.** (2013, June). Linguistic Regularities in Continuous Space Word Representations. In Hlt-naacl (Vol. 13, pp. 746-751).

[40] **Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P.** (2011). Natural language processing (almost) from scratch. Journal of Machine Learning Research, 12(Aug), 2493-2537.

[41] **LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P.** (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[42] **Galon, J., Mlecnik, B., Bindea, G., Angell, H. K., Berger, A., Lagorce, C., ... & Nagtegaal, I. D.** (2014). Towards the introduction of the 'Immunoscore'in the classification of malignant tumours. The Journal of pathology, 232(2), 199-209.

[43] **LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P.** (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[44] **Hochreiter, S., & Schmidhuber, J.** (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[45] **Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y.** (2014). On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint arXiv:1409.1259.

[46] **Tang, D., Qin, B., & Liu, T.** (2015, September). Document Modeling with Gated Recurrent Neural Network for Sentiment Classification. In EMNLP (pp. 1422-1432).

[47] **Kahou, S. E., Bouthillier, X., Lamblin, P., Gulcehre, C., Michalski, V., Konda, K., ... & Ferrari, R. C.** (2016). Emonets: Multimodal deep learning approaches for emotion recognition in video. Journal on Multimodal User Interfaces, 10(2), 99-111.

[48] **Krzyzanowski, P.** (2000). Distributed File Systems Design.

[49] **Coulouris, G. F., Dollimore, J., & Kindberg, T.** (2005). Distributed systems: concepts and design. pearson education.

[50] **Andrews, Gregory R.** (2000), Foundations of Multithreaded, Parallel, and Distributed Programming, Addison–Wesley, ISBN 0-201-35752-6.

[51] **Kumari, R., Singh, M. K., Jha, R., & Singh, N. K.** (2016, March). Anomaly detection in network traffic using K-mean clustering. In Recent Advances in Information Technology (RAIT), 2016 3rd International Conference on (pp. 387-393). IEEE.

[52] **Ghemawat, S., Gobioff, H., & Leung, S. T.** (2003, October). The Google file system. In ACM SIGOPS operating systems review (Vol. 37, No. 5, pp. 29-43). ACM.

[53] **Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R. H., ... & Stoica, I.** (2011, March). Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In NSDI (Vol. 11, No. 2011, pp. 22-22).

[54] **Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., ... & Xin, D.** (2016). Mllib: Machine learning in apache spark. Journal of Machine Learning Research, 17(34), 1-7.

[55] **Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I.** (2010). Spark: Cluster Computing with Working Sets. HotCloud, 10(10-10), 95.

[56] **Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., ... & Stoica, I.** (2012, April). Resilient distributed datasets: A fault-tolerant abstraction for in-memory

cluster computing. In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (pp. 2-2). USENIX Association.

[57] **Go, A., Bhayani, R., & Huang, L.** (2009). Twitter sentiment classification using distant supervision. CS224N Project Report, Stanford, 1(12).

[58] **Kouloumpis, E., Wilson, T., & Moore, J. D.** (2011). Twitter sentiment analysis: The good the bad and the omg!. Icwsm, 11(538-541), 164.

[59] **Wang, S., & Manning, C. D.** (2012, July). Baselines and bigrams: Simple, good sentiment and topic classification. In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2 (pp. 90-94). Association for Computational Linguistics.

[60] **Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A. R., Jaitly, N., ... & Kingsbury, B.** (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine, 29(6), 82-97.

CV

Ali Abas Albabawat

ali.b1989.12@gmail.com

+9647504238891

Malta, Duhok 42001, Kurdistan Region, Iraq

PERSONAL STATEMENT

Self-improving, hardworking, good timekeeper, team worker, teaching and coding experience, strong communication skills as well as the ability to convert client needs into an exciting software or website.

CAREER & ACHIEVEMENTS

- International English Language Test System (IELTS), total score of 6.
- BMRS: Bone marrow patient registry system at the hematology department - Azadi hospital – Duhok.
- SRS: Developed a student affairs management system and online student registration for the University of Duhok (5 colleges), still improving and managing it.
- CDC: Designing a website-based system for Career Development Center at the University of Duhok.
- MobiCoffe: Mobile (Android) application that gives the users the ability to order for themselves in coffees and restaurants.

EMPLOYMENT HISTORY

- Teaching Assistant at the University of Duhok – Computer Science Department, currently since December-2012.
- Web Designer & Software Developer, currently since October-2011.

KEY SKILLS

Programming Languages: PHP, Visual Basic (VB.NET, VB6), C, C++, C#, Java, Java Android and python

Software Packages: Microsoft Office collection, database management systems (Oracle, MS SQL Server, MySQL, SQLite), networking, 3D design, photo editing and video editing

EDUCATION

- Bachelor Degree: Computer Science (C) - University of Duhok, 1st in class rank.
- Studying Masters in Computer Engineering at Firat University – Turkey

LANGUAGES

Kurdish | English | Arabic.

PERSONAL INTERESTS AND ACTIVITIES

Classical Music | Traveling | Chess | Gym.

REFERENCES

References are available upon request.