COMPUTER SCIENCE

# Empirical Survey On Text Representations In Sentiment Analysis

**Supervisors:**
Dhr. Dr. Adam Belloum
Johannes (Jan) C. Scholtes

**Candidate:**
Aimilios Voma
Std.No. 2594330

Amsterdam - 16/09/2019

# Abstract

Sentiment mining has been a popular and active NLP research topic for almost two decades since commercialized enterprise and data-driven marketing flourished along with the technological development of today. The main scope of this research is to argue about different text representations techniques regarding their performance in sentiment domains. Specifically, it comprises of two parts, (i) investigating TFIDF sparse model information capacity across sentiment domains , (ii) Investigating the performance of self-trained word embedding models with domain specificity(sentiment corpus) and pretrained word and contextual embedding models that were trained on generic data. Different preprocessing parameters were applied such as text normalization heuristics and n-grams combinations. Additional insights were provided regarding the intrinsic characteristics of the algorithms(sparse and dense) and how they reflect on sentiment domains.

# Contents

# List of Figures

# List of abbreviations and acronyms

| | |
|---|---|
| biLM | bi-directional Language Model |
| BN | Bayesian Network |
| BOW | Bag Of Words |
| CLM | Contextual Language Model |
| CVS | Continuous Vector Space |
| CBOW | Continuous Bag-of-Words |
| COD | Curse of Dimensionality |
| DRT | Dimensionality Reduction Technique |
| FFNN | Feed Forward Neural Network |
| FP | False Positive |
| FN | False Negative |
| JPD | Joint Probability Distribution |
| IR | Information Retrieval |
| LM | Language Model |
| LR | Logistic Regression |
| MNB | Multinomial Naive Bayes |
| NLM | Natural Language Model(ing) |
| NLP | Natural Language Processing |
| NNLM | Neural Network Language Model |
| Opinionated | Sentimentally Annotation(positive, negative) |
| PMI | Pointwise Mutual Information |
| POS | Part of Speech |
| RNN | Recurrent Neural Network |
| RF | Random Forest |
| SLM | Statistical Language Modeling |

| | |
|---|---|
| SVM | Support Vector Machine |
| TDM | Term-Document Matrix |
| USE | Universal Sentence Embeddings |
| TDM | Term-Document Matrix |
| TFIDF | Term Frequency Indiscreet Document Frequency |
| VSM(s) | Vector Space Model(s) |
| WE(s) | Word Embedding(s) |

# Chapter 1

# Introduction

Data Science is a delegate and highly operational field of science since the 21st century. Where the technological advancement of today set the necessary cornerstone in the deep foundations of the field, the data. While, digital media and platforms made the data accessible towards data preservation, extraction, and manipulation techniques. Sentiment analysis is one of the major data science fields, which is a broad meaning is oriented around peer's opinion, sentiment and emotional reflection of their viewpoints upon a subject. It is well known that the procedure of the opinion-making or formation, of a single individual inside a group, is a bidirectional relationship with the group itself. A Collective Behaviour is a phenomenon where the group's individual needs to choose whether to accept to participate or not in an aggregated group's behavior. With this in mind, it is very feasible to reproduce the subjective samples of each of the individuals. To statistically form a general pattern that will define the group's dynamics. Thus if we broaden this concept in our social, economic and political spectrum, we realize that many applications involve the opinion mining process. [3]

One of the first sentiment analysis applications was to capture and extract favorability information out of the product reviews [38]. By meaning favorable opinion we refer to a positive or negative sentiment statement that was captured inside a text fragment. Since performing sentiment analysis on the whole review was quite a costly operation that involved complex Natural Language Processing(NLP) mechanics. Especially if the sentences are quite long, or have a complex linguistic structure. Thus the early baseline approaches mostly relied on common syntactic patents which could signify a presence of opinion inference and its targets. A priory knowledge of natural language forms which could be used to detect emotion patterns inside text.

Nowadays sentiment analysis applications are rapidly spread around numerous scientific fields. From consumer-based opinion ranking and product reviews to political polls and social media analytics. Practically sentiment mining became a tool of predicting the polarity outcome out of different domain sources. Recent mentions of popular sentiment mining applications are directly linked to social media platforms and micro-blogging. Twitter data, for instance, was used to predict election results [55], stock market analysis [5] and many more applications that incorporate sentiment analysis to the core of modern analytics. In academia sentiment analysis is one of the challenging, active NLP research fields. Which flourished along with data mining and other techniques of extracting and scraping data from multiple sources such as web mining and Information Retrieval(IR).

## 1.1 Objectives of the Study

In our work, we empirically observe both high dimensional sparse and dense VSMsm, regarding their performance in sentiment domains. While applying at the same time rule-based preprocessing heuristics. Our work is comprised of two parts.

### 1.1.1 Sparse Text Representations Experiments

Specifically for sparse text representations, we examined the quality of the produced features in terms of transferable knowledge across different sentiment domains, given the proper amount of training data. Certain preprocessing steps were taken into consideration such as word form expansion, inflectional form reduction, and n-grams combinations.

**RQ1** *What is the capacity of TFIDF algorithm in terms of cross-domain information transfer*

**RQ1.1** *Does utilizing richer n-grams combinations always improve the TFIDF performance on sentiment domains?*

### 1.1.2 Dense Text Representations Experiments

Regarding self-trained models, we observed the word vectors quality of NNLMs(Doc2Vec, Word2Vec) and count-based statistical model(Glove) which were trained on domain-specific sentiment corpus(Sentiment140). The resulted in WE models contained domain-specific embeddings which were evaluated(ML models) against the same sentiment domain.

**RQ2** *Does domain specificity reflects on better word embeddings representations, regarding sentiment classification tasks?*

Regarding pretrained models, we observed the performance of word(Word2Vec,Glove) and contextual(FastText, USE, Elmo) LMs on sentiment domains

**RQ3** *What is the performance difference between TFIDF, pretrained contextual and word LMs on sentiment domains*

# Chapter 2

# Background

## 2.1 Sentiment Classification Overview

### 2.1.1 Sentiment Classification Models Categories

1. **Supervised Learning**
   Since it is about text classification with a certain number of classes(categories), for instance, two classes positive and negative. We practically can apply any of the available supervised learning models (logistic regression, SVM, Naive Bayes) to make a classification. By fitting a classification model with opinionated (sentiment labeled) training data.

2. **Semi Supervised Learning**
   With this approach, we use only little training data or a specific set of rules to weakly supervise our classification model. For instance, in this work [25], a manual seeding technique was used to sentiment lexicon(SentiWordNet) to extract synsets(synonyms) of the seeded words(positive and negative terms). Afterwards, the synsets were processed, feature information was extracted on multiple levels(POS, sentiment orientation, sentiment strength) with different NLP techniques and stored in a custom lexicon. Which later on was used to enhance the sentiment classification process of the SVM. Thus seeding could be interpreted here as weak supervision to create the sentiment dictionary which will be used in the final phase of the classification.

3. **Unsupervised Learning**
   In unsupervised learning, the data has not been opinionated or categorized. The purpose of the unlabelled data is to make the model to learn for itself any possible principles or connection between the data features. As we mentioned above the work of [38], where heuristic syntactic patterns with POS tags were used to extract

sentiment information from text fragments. Since there was a match in the heuristic patterns, the matched phrase(term1, term2) was extracted. PMI was used to determine the degree of the statistical freedom between the extracted terms, or else the co-occurrence probability of the terms. Sentiment orientation was calculated based on the PMI(term1, term2) score and was aggregated for all the extracted pairs inside a document. The aggregated result was the final sentiment outcome of the document.

### 2.1.2 Sentiment Classification Analysis Levels

1. **Document Level Analysis**
   In which the entire document opinion is being inferred as a positive or negative sentiment. A close example of this kind of analysis is analyzing the movie reviews. Since the morphology of the data usually resembles long sentences grouped in the form of a paragraph or else document. We can see relative work [39]. It is quite convenient to represent the whole document as a negative or positive review. Although in practice, to have satisfying results close to human assessment ones should synergistically use complex NLP mechanics, rule-based approaches or multiple feature extraction techniques. Considering that multiple sentences are present in a single document. These sentences could have a contradicted sentiment statement, thus an aggregation rule should be present to determine the overall emotional outcome of the document.

2. **Sentence Level Analysis**
   At this level of analysis, we examine each of the sentences and their sentiment outcome. It is also called sentence subjectivity since subjectivity is related to sentences that express a fragment of truth, personal beliefs or opinions. Thus the sentences are treated as data points and each operation of feature extraction is evolved around the sentences. In this work [20] the model combines both document level and sentence level analysis. Where a document is decomposed to sentences, feature extraction is performed individually for each of the sentences. Utilizing both syntactic and semantic extraction patterns(positive-negative terms frequency, special characters(?!) frequency, negation words). Sentiment orientation of the words of each of the sentences was determined used a custom sentiment lexicon. Lastly aggregated rule was applied to determine the overall sentiment outcome of the sentences.

3. **Entity and Aspect Level Analysis**
   This approach goes way beyond the structural dependencies of documents, sentences or phrases. The core idea is to directly focus on opinions and their targets. Since each opinion has its target, an opinion without a target offers zero to none information. For instance, we have a sentence “New Motorola X series has decent call quality but is a bit fragile”. Here we have to evaluate two aspects, the call quality and the quality

of the device. Our target is Motorola X series which is an entity. The call quality aspect has positive sentiment while the call quality aspect has a negative sentiment towards the target entity. This level of analysis is also called a fine-grained analysis since the model is trying to evaluate every possible aspect and its targets. Something that is highly challenging even for state of the art sentiment analysis techniques.

### 2.1.3 Major Problems Of Sentiment Classification

Sentiment Classification tasks, besides the different techniques and implementation methods, highly rely on sentiment polarity words. Sentiment polarity words, or else opinion words are the words that could be directly related to negative or positive sentiment. For instance, words like "great", "incredible" are related to positive emotions while words like "bad", "horrible" are referred to as negative sentiment. Nevertheless, there are many linguistic structures and word formations that complicate the proper sentiment expression of the opinion words. [30]

1. **Sentence types which express no sentiment**
   For instance, if we have an interrogative sentence(question mark present), or another form of text structure which don't involve directly any sentiment. For instance, "Is that camera any good?", is an interrogative sentence with opinion word present("good"), but expresses no sentiment. Another example is conditional sentences, for instance, "If I find something descent I will buy it". Opinion word is present("descent") but the sentence itself expresses no opinion of the present.

2. **Polysemy in word formations**
   For instance, a sentence "I like that car, quite speedy" contains a positive opinion word("like"). Another sentence with the same opinion word, "Can I also order an MCflurry, just like that one", expresses zero to none sentiment information about the subject. Since the specific word is being used in different speech contexts reflecting different meaning upon occasions.

3. **Sarcastic Sentences**
   Probably the hardest type of classification is to obtain a sarcastic behavior in text data. For the obvious reason that the whole point of sarcasm is to project the other side of the stated opinion. For instance, a sentence "What a good product, got rid of it after a week" could be detected as a highly positive statement. When it is the quite opposite, due to sarcastic elements which are present to satirize the same contradiction of the statement.

4. **Sentences without opinion words**
   Opinion words are not present in many of the sentences. Meaning that the sentence

itself could imply positive or negative sentiment opinion, without directly involving any of the opinion words. For instance, "this car consumes too much fuel" has no direct relation to opinion words but is implying a negative opinion about the fuel consumption of a car.

## 2.2 Sparse Vector Space Models

### 2.2.1 Sparse Models

One of the early contributions to the field of NLP and IR was indexing data with Vector Space Models(VSMs) [49]. Salton argued whether there was a way to approximate a distributional structure of the human language. More specifically he underlined a hypothesis, in which different parts of the flow of speech if identified properly could yield a certain distribution of their co-occurrences. Thus co-occurrences of different linguistic elements could provide insights about their semantic relatedness.

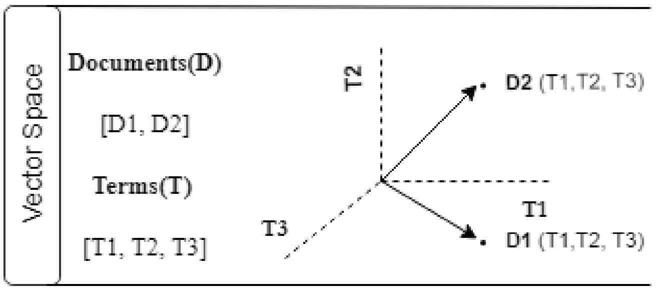


Figure 2.1: Documents are represented as points in three dimensional space

VSM was developed and introduced for the SMART information retrieval project [48]. The main concept behind the VSM was to represent a collection of documents in document space. A multidimensional vector space where each of the dimensions would represent the respected index terms of the documents. For instance in a three-dimensional vector space where we would only have three distinct terms. We could mathematically represent a collection of the documents in relation to these terms, **Figure 2.1**.

VSM's have good performance of NLP downstream tasks which involve measuring similarities between words, phrases, and documents. One of the reasons behind VSM's efficiency is the distributional hypothesis, which states that words that occur in similar contexts tend to have similar meanings [18]. Ignoring the structure of the word and capturing only their event of co-occurrence, this technique was called Bag Of Words(BOW). Of course, there are many exceptions to this assumption. Case scenarios which include polysemous words or differences in documents domain, tend to add statistical noise to vector space representation.

### 2.2.2 Sparse Models Implementations

One of the first VSMs implementations was a Term-Document Matrix(TDM). TDM is one way of representing a collection of documents as a matrix of unique terms frequencies. TF-IDF is one of the algorithms which utilize the TDM concept. Suppose that the total number of documents is n and our overall number of unique tokens is m. The TDM($m \times n$) then will be a matrix of m rows(one per each unique term) and n columns(one per each document). Thus an element $x_{ij}$ of a TDM matrix will represent a frequency of the $i$-th term in the $j$-th document. Since usually most of the documents will use a small fraction of the overall dictionary(unique token collection). Thus the produced TDMs are sparse matrices with most of its elements set to zero. Nevertheless having a raw term frequency matrix is not enough to add the proper representation to the term frequencies. Since the whole idea of weighting is to give more meaning to surprising events. To learn more about new linguistic terms rather than the usual ones which could stop words. Repeating word terms with high frequency but zero to none semantic relation [56]. Later findings and experiments with different weighting functions concluded that using different weighting techniques over raw frequencies significantly improves the quality of the VSM models such as TF-IDF [47].

### 2.2.3 Sparse Models Major Challenges

While the common VSM implementations are quite efficient in most of the NLP downstream tasks, they also bear weaknesses. Two major problems of VSMs are the high dimensionality and sparsity of the vector space representation [46].

Since sparce VSMs are singly relying on statistical evidence of the event that is happening inside a context (such as co-occurrences). The co-occurrence matrix will become significantly large in dimensions for any reasonable content of data. High dimensionality issues affect the model's scalability factors and computational efficiency. Thus if a resulted dictionary of unique tokens is quite large(several millions of tokens), the VSM model will suffer from high dimensionality of several millions of dimensions. Which makes the model itself more susceptible to noise, something that consequently leads to worst model performance in NLP tasks.

Numerous researches have been done to tackle this problem, specifically, a Dimensionality Reduction Technique(DRT) was quite popular when dealing with sparse VSMs. From early approaches where linear DRT models were used to map a high dimensional to low dimensional space. By clustering possible semantically correlated components and performing local dimensionality reduction. Utilizing mathematical linear analogies between the two dimensional spaces(high and low). Principal Components Analysis(PCA) and Latent Semantic Indexing are some of the popular techniques [52].

Although linear models of dimensionality reduction usually don't perform well on real-life problems with huge feature spaces. Non-linear models were introduced with different techniques rather than clustering methods. For instance, in [44] global coordinate system was used to map high dimensional input to low dimensional space. The mapping technique is a linear reconstructing process(optimization function) which maps each neighboured high dimensional feature to low dimensional space. There is also a cost function present which regulates the reconstruction error and calibrates the linear weights. After minimizing the reconstruction error cost function, adjusted linear weights are used for the final phase of mapping to low dimensional embedded space.

## 2.3 Dense Continuous Vector Space Models

### 2.3.1 Statistical Language Modeling

One of the first scientific approaches to tackle the matter of distributional representations of content was introduced by [23]. Where the idea of a hierarchical ecosystem of entities and their relations was introduced in distributional fashion. Instead of treating entities separately, join them in role groups and determine the group relations based on specific criteria. These criteria could serve as domain's regularities, or else patterns which could determine the relations between the role groups. In text-domain these regularities are expressed statistically and can determine the probability relations between different lexical terms. Precisely, the probability distribution over sequences of lexical terms(words) is the core aspect of every Statistical Language Model(SLM) implementation.

### 2.3.2 Neural Language Modeling

Neural language models(NLMs) became quite popular since the early 20's. The base principle of these models is to apply the SLM paradigm inside an NN architecture. Where the domain's regularities and lexical terms relations are internally expressed as 'weights' of the NN. When it comes to text-domain, it seems quite an appealing solution to utilize Baesyan models and conditional independence of terms to infer statistically linguistic relations. First NLM models were evolved around the Bayesian principle. Mapping local conditional dependencies of consecutive words and modeling its joint probability distribution(JPD).

In [7] we can see a relevant implementation where the JPD is expressed as dependency probabilities between consecutive lexical terms. Characterized as parents, or merely proceeding terms in grammatical order of text input. The input terms are transformed and modeled as discrete points in high dimensional vector space.

$$p(Z_1 \ldots Z_n) = \prod_{i=1}^{N} P(Z_i \mid Parents_i) = \prod_{i=1}^{N} P(Z_i \mid Z_1 \ldots Z_{i-1}) \tag{2.1}$$

Where $Parents_i$ are the proceeding random variables(terms) in the set which are called the parents of variable $i$.

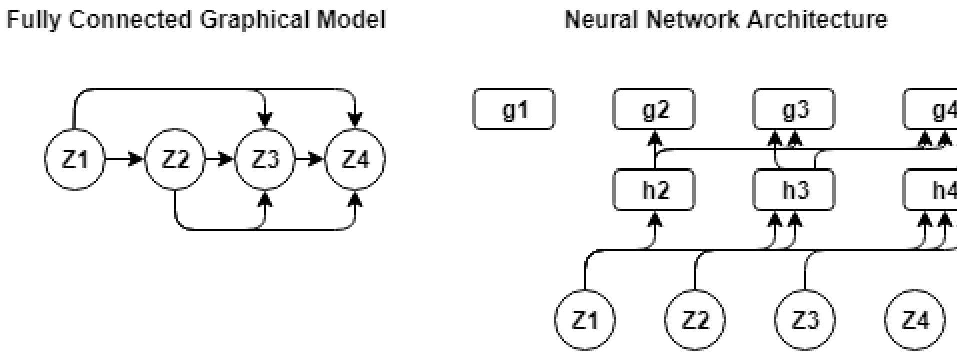


Figure 2.2: Bayes graphical model(left) and architecture of NN(right)

In **Figure 2.2**, the arrow relation expose the parental variables for the lexical term $Z_i$. For instance, the parents of the $Z_3$ are $Z_1$ and $Z_2$. The observed values $Z_i$ are encoded into an appropriate input group. The $h_1$ is the group of hidden units that perform linear transformations of the input in the network and $g_1$ is the group of the output units. The result of the output units is strongly depended on the input group $Z_1 ... Z_{i-1}$ and models the parameters outcome(probabilities) of a distribution over $Z_i$. These conditional probabilities are multiplied to represent the JPD, $P(Z_1 ... Z_N)$. Of course, the structure and the order of the input group influence the probability distribution.

When it comes to text context the grammatical order of words represents unique probability distribution each time. Thus a different combination of input groups represents the different context of words. Although this baseline approach introduced some new challenges, such as the curse of dimensionality(COD). Where high dimensional discrete data modeling was not scaling well with the quantity of the training data. With the immediate results of peaked computational complexity and model's inefficiency for a large number of training data vocabulary.

#### 2.3.2.1 Curse Of Dimensionality

In the above, early implementations of Bayesian logic, **Figure 2.2**. The lexical terms were mapped as multivariate discrete data in high dimensional space. The overall space dimensionality was based on the size of the vocabulary. Let suppose that one wants to model the joint distribution of $n$ consecutive words with the size of the vocabulary of $m$. The overall free parameters in the NN will be as many as $m^n$. If the size of the vocabulary is big enough as millions of unique tokens. The joint distribution modeling process would be incredibly slow and inefficient. Due to computation overhead induced by a gigantic number of free parameters inside the NN.

For discrete spaces, the generalization of the estimation function is not that obvious. Even small changes in the discrete values may bear a huge impact on the estimation function. On the contrary, in continuous spaces, the generalization of the estimation function is easily observed since usually continuous spaces imply local smoothness [8]. Since the generalization is impacted in high dimensional discrete spaces, the training capacity of the model is also compromised. The model tends to “overfit”, while its space dimensionality and the complexity grows. Since usually there are not enough training samples to cover up for the dimensionality growth, statistical noise is induced. Which in terms of resulted WE could provide less meaningful word representations.

### 2.3.3 Dense Reduced Continuous Vector Space

It is obvious that with discrete data modeling the resulted language models will be inefficient in terms of representations and computationally slow. In this work [8] a more grounded state of the art Neural Network Language Model(NNLM) implementation was introduced.

Based on the Bayes concept of conditional independence, one could statistically express a language model as a conditional probability of the next word given the previous content. Since the previous content is represented as a combination of n-grams, it is mathematically expressed as

$$P\left(w_t \mid w_{ngram}^{t-1}\right) \equiv P\left(w_t \mid w_{t-n+1}^{t-1}\right) \tag{2.2}$$

Where $w_t$ is the t-th word and $w_{ngram}^{t-1}$ is the n-gram combination of previous words. However there are quite a few problems in that implementation. Since the model is not taking into account content further than 1 or 2 words given in the n-gram window. Secondly the model don’t capture the similarity between words. For instance if the model observed the sentence “The cat was purring in the bedroom” in the training phase. It should generalize to make the sentence “The dog was barking in the hall” as much alike. Because words like “dog” and “cat” have similar semantic and grammatical roles.

To overcome the COD, different approach rather than modeling discrete data was proposed. To associate each word in the vocabulary with a distributed word feature vector. A feature vector of each word will represent different aspects of that word. The overall number of features (e.g. m = 30,50,100 e.t.c.) will be much smaller than the overall vocabulary size(tens of thousands, maybe more). Recall that in discrete data modeling each word dimension was of the size of the overall dictionary, something that consequently led to high dimensionality problem.

The revised NN approach was based on traditional Bayes concept and NLM architecture where joint probability of random variables $Z_1 ... Z_n$ is expressed as a product of conditional probabilities

$$P(Z_1 \dots Z_n) = \prod_i P(Z_i \mid g_i(Z_{i-1},\ Z_{i-2} \dots Z_1)) \tag{2.3}$$

Where g() is a calculation function represented in the NN with a left-to-right architecture, **Figure 2.2**. The $g_i$ output block computes the parameters of conditional distribution of $Z_i$ given the previous content of n-grams.

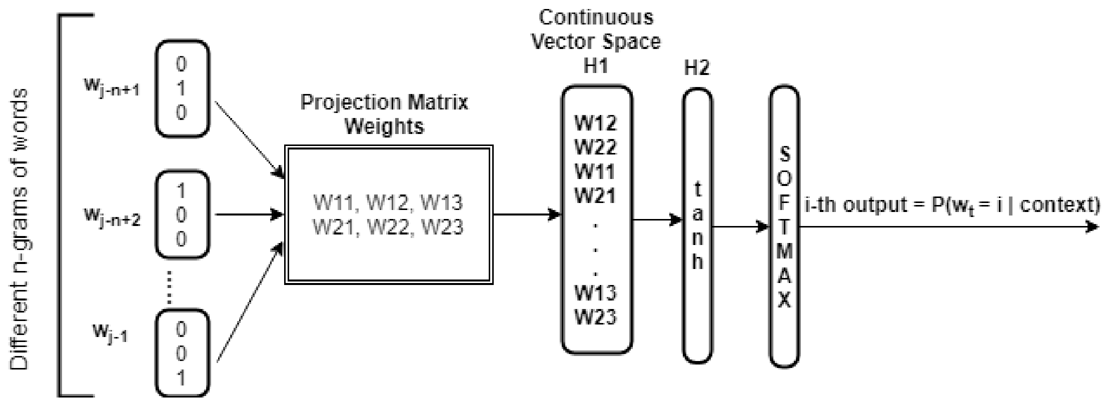

Different n-grams of words
$w_{j-n+1}$
0 1 0
$w_{j-n+2}$
1 0 0
$w_{j-1}$
0 0 1
Projection Matrix Weights
W11, W12, W13
W21, W22, W23
Continuous Vector Space
H1
W12 W22 W11 W21 . . . W13 W23
H2
tanh
SOFTMAX
i-th output = P($w_t$ = i | context)

Figure 2.3: Neural Network Language Model(NNLM) pipeline

From **Figure 2.3** you can observe the projection matrix which maps the discrete word indices of different n-grams to continuous vector space. The projection layer is shared across different n-grams and has intrinsic topology. The topology of the projection matrix efficiently increases the amount of available training data, since each word from each of the contexts contributes individually to the weight matrix update.

The way that projection into reduced continuous vector space works, is simply by assigning zero-to-one input representation of the context, to the according position in the projection matrix. In such a way a similar weight set is applied to contexts that contain the same word multiple times. As a result, the models capture syntactic and semantic information between different contexts. The softmax function normalizes the computations done in the non-linear hidden layer into a probability distribution. The final output is the probability distribution of i-th word over the previous context. After a certain number of training epochs, the hidden layer("weights") of the network is extracted, forming the actual dense word vectors(embeddings).

#### 2.3.3.1 Linquistic Regularities

Dense Continuous Vector Space(CVS) models have demonstrated quite satisfying results in numerous NLP downstream tasks. Mainly because the distributed representations achieve a level of generalization that typical discrete n-gram VSM can not. Since the discrete data modeling with n-grams works only in a way that the same n-grams have no inherent relation to one another. Contrary to CVS modeling, where similar words or words that occur in the same context tend to have similar representation in vector space, due to their distributional property.

Furthermore, besides the fact that CVS models effectively capture syntactic and semantic similarities, each of these relations can be interpreted as vector offset calculations. Something that allows a vector oriented interpretation of different relations between words, based on their vector offsets. For instance the gender(male/female) relation is automatically extracted from model's training content. By utilizing vector representations "King" - "Man" + "Woman" results in a vector quite close to "Queen". Another example of regularities between pairs of words in CVS, is the plural/single relation. Where, if denote $x_i$ as vector(x) of word(i), you can observe that $x_{apple} - x_{apples} \equiv x_{car} - x_{cars}, x_{family} - x_{families}$ [35].

Different relations between words is something that is called a multiple degrees of similarities. Since each of the word vectors, encapsulate different relations in high dimensional vector space, **Figure 2.4**.

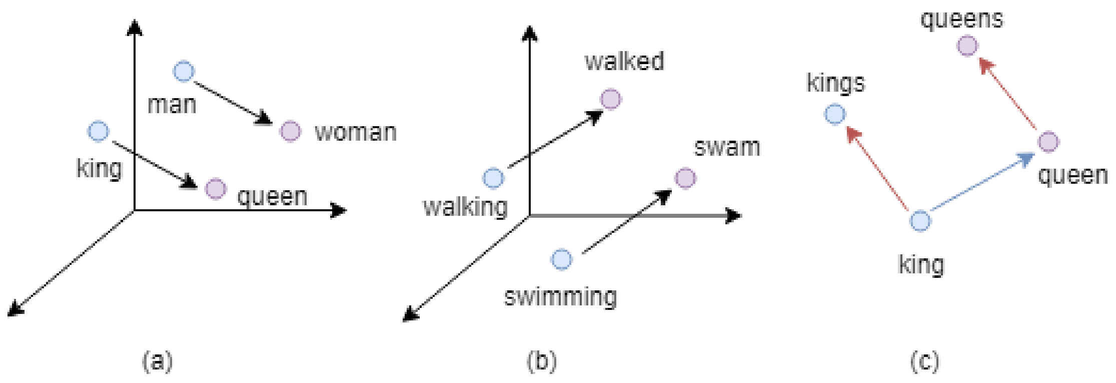


Figure 2.4: (a) Gender regularities between pair of words. (b) Verb tenses regularities. (c) Projection of plural/single with extra relation of gender, in high-dimensional space multiple relations can be embedded in a single word.

We have seen that syntactic and semantic similarities between words, have been formulated in high dimensional CVS in a form of analogies. In order to provide answer to these analogies, one should compute the cosine distance between the word vectors offsets. Thus in order to answer the analogy question $a : b \xrightarrow{analogy} c : d$, where d is the unknown. We first compute the respected word vectors $x_a, x_b, x_c$ and after we find the relation $y = x_b - x_a + x_c$. Where y is the resulted CVS representation, which is considered to be the "best" answer to our question.

$$w^* = argmax_w \frac{x_w y}{||x_w|| \; ||y||} \tag{2.4}$$

Of course no specific word might exist at that exact position within the CVS. Thus what should be done is to find a closest word($w^*$) whose embedding in CVS has the biggest cosine similarity to y [35].

# Chapter 3

# Related Work

There has been significant research regarding NLP downstream tasks such as sentiment and semantic relation extraction based on sparse and dense text representations.

In [9] linear models such as Support Vector Machine(SVM) and Multinomial Naive Bayes(MNB) were introduced, to classify the sentiment polarity in short text data(microblog). Different feature extraction techniques were used, from the regular sparse BOW approach to POS extraction and n-grams.

In [2] different customization features of cross-domain classification models were examined. To provide better adaptation to new target domains, in the absence of sufficient training data. Sparse frequency models were used in cross-domain classifications combined with n-grams combination, indicating that n-grams usually contribute to better generalization across models.

In [36] feature extraction methodology was presented based on hand-made syntactic(Turney) and rule-based features(Semantic Orientation(SO), WordNet)). These features were used, to obtain favorability in movie reviews. SO was used to express a real value representation of positive or negative sentiment expressed in word or phrase, based on PMI metrics. While WordNet was used to extract relationship metrics between sentiment properties(potency, activity, evaluative) of words and adjectives.

In [14] an empirical survey was provided regarding different word embedding models(SENA, Turian, Huang, HLBL) and the quality of the semantics that they captured. Furthermore intrinsic characteristics of the models were tested such as dimensionality property, reflecting the quality of the information encapsulated in multi-dimensional VSMs. Different evaluation tasks were compiled, which involved sentiment and semantic quality assessment of the produced word embeddings. Based on linear(LR) and non-linear(SVM with RBF kernel) ML models.

In [58] a refining technique was used on pre-trained WE models, to capture better sentiment representation in multi-dimensional vector space. The refinement process was built around pre-trained models(Word2Vec, Glove). By augmenting the produced word vectors with an extra feature of sentiment ranking score, which derived from sentiment lexicon. The augmented embeddings were rearranged in terms of their inter-domain relations, based on their valence(positive, negative) score. The proposed method slightly reduced the number of sentimental dissimilar relations in the embedding space, compared to the pre-trained models.

In [53] the proposed method was focused on encoding sentiment information in CVS using harvested twitter data as a form of weak supervision in the training process. The so-called Sentiment Specific Word Embeddings(SSWE) were trained on a shared neural network that incorporated the sentiment polarity of n-grams in its loss function. Specifically, the shared neural network was a composition of three different neural network architectures. Each serving as a weaker model that combined learn both syntactic and sentiment information of n-grams. The resulted model yielded better results compared to raw pre-trained embedding models(C&W, Word2Vec, e.t.c).

In [12] the empirical study provided insights about cross-domain relations of WE models combined with factors such as domain characteristics(thematic elements, corpus size). The source domains comprised of five different corporas(songs, reviews, twitter, news, common text). Different metrics took places such as domain richness, the dimensionality of the embedding models and the learning algorithm(Glove, Word2Vec). The evaluation process proceeded with the semantic and sentiment assessment of the quality of the produced embeddings.

In [10] aspect-based sentiment analysis approach was introduced, by encoding the aspect terms of a sentence into a feature vector. The feature vector was zero-one padded accordingly, to produce normalized vectors of fixed size across all the sentences. Another feature extraction technique was used such as location encoding of terms and their respected context, along with sparse VSM TFIDF. After the preprocessing and feature extraction, certain machine learning models and deep learning models were evaluated. Where deep learning models, especially DNN served the best F1 scores on sentiment classification tasks.

In [45] WordNet embedding model was introduced, which is based on a lexical ontology graph that comprised of different types of word semantic relations. These relations were captured and transitioned through various information extraction techniques (normalization, PMI) to high-dimensional VSM. Furthermore, the dimensionality reduction technique(PCA) was used to reduce the overall dimensionality of the embeddings. The produced WordNet embeddings showed better efficiency against the Word2Vec model in word similarity tasks.

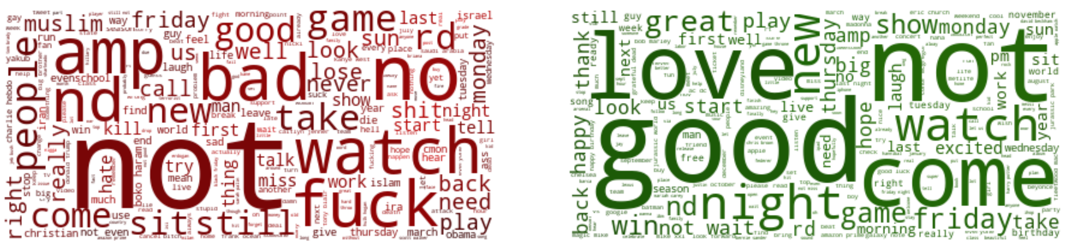
# Chapter 4

# Technical Overview

## 4.1 Datasets

### 4.1.1 Twitter: Semeval2016

This dataset [37] comprises of opinionated short-sentenced twitter sentences with negative and positive sentiment polarity. The original size of the dataset was about 42k sentences. The overall size of the sampled dataset is roughly 15k sentences divided equally(class balance) in the positive and negative class. Since we are performing cross-domain correlations with the TFIDF algorithm. It is quite obvious that we need more data than usual(twitter domain), to provide sustainable internal representations of the content inside the TFIDF model. Hence each time we obtain the model's internal dictionary representations out of source domain training data to transform the test data of the targeted domain.



(a) Negative Frequent Words

(b) Positive Frequent Words

Figure 4.1: Twitter: Semeval2016 Dataset, Wordcloud Frequency Overview

In **Figure 4.1**, we can observe the Wordcloud representation of the twitter dataset. Words with larger fonts are more frequent compared to words with smaller fonts. On the left, we have all the sampled words from negative class distribution and on the right side the positive one. We can observe that the negative word "not" is quite frequent in both positive and negative word domains. Positive sentiment words such as "love" and "good" are more related to the positive domain than the negative ones. This is quite natural for the reason that these specific sentiment words are hardly expressed in combination with the usual negations. The overall data morphology was quite noisy due to the short text sentences, domain-specific slang, and verbal expressions.

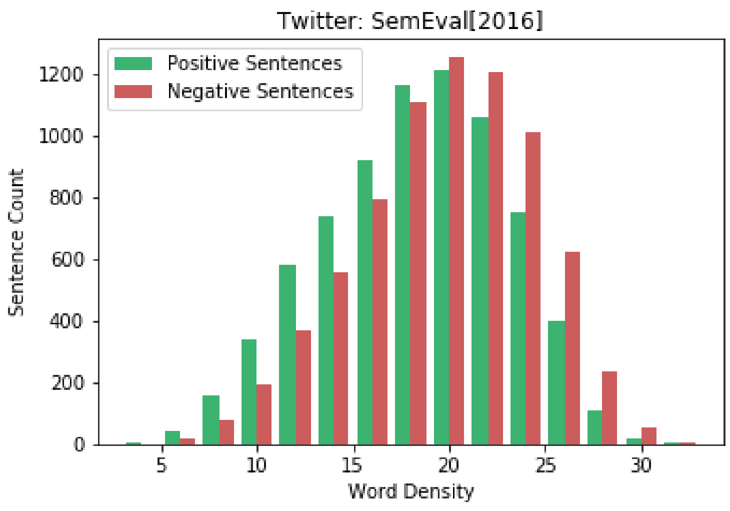
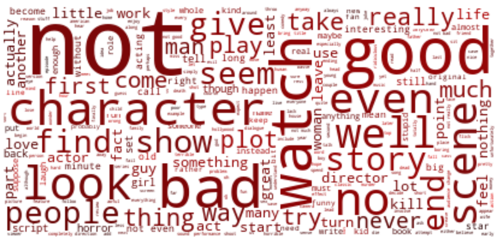


Figure 4.2: Word density histogram in Semeval2016 dataset

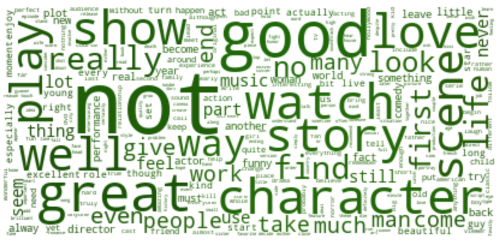
In **Figure 4.2** we can observe the word frequency distribution over the twitter dataset sentences. Roughly 80% of the data is gathered around sentences with more than 5, up to 25 words. We also can observe that as word frequency rise above 20 words per sentence the negative sentences tend to be more frequent than the positive ones. One assumption that can be made is that positive opinions are generally more straightforward regarding the context size of the information than the negative ones.

### 4.1.2 IMDB: Movie Reviews

This dataset [31], comprises of opinionated long-sentenced movie reviews with negative and positive sentiment polarity. Original size of the dataset is about 50k sentences. Since the length of the sentences is quite big compared with other twitter datasets. We sampled 4k sentences, equally divided(class balance) in two classes(positive, negative). The size was empirically set, to satisfy enough training and testing data in both sparse and dense VSM experiments.



(a) Negative Frequent Words



(b) Positive Frequent Words

Figure 4.3: IMDB: Movie Dataset, Wordcloud frequency overview

In **Figure 4.3**, we can observe the Wordcloud representation of the IMDB dataset. On the left, we have all the sampled words from negative class distribution and on the right the positive ones. We can observe that the negative term "not" and positive term "good" are both roughly equally present in both positive and negative sentences. This can be explained since negative term "not" and positive term "good" are frequently combined in word formations to express negation. As we described in the previous sections such linguistic phenomena tend to create ambiguity in sentiment classification models. Since the model itself cannot rely on sentiment words alone, to determine the overall sentiment outcome. On the other hand sentiment words such as "great" and "love" contextually are more related to positive sentiment statements than the negative ones.

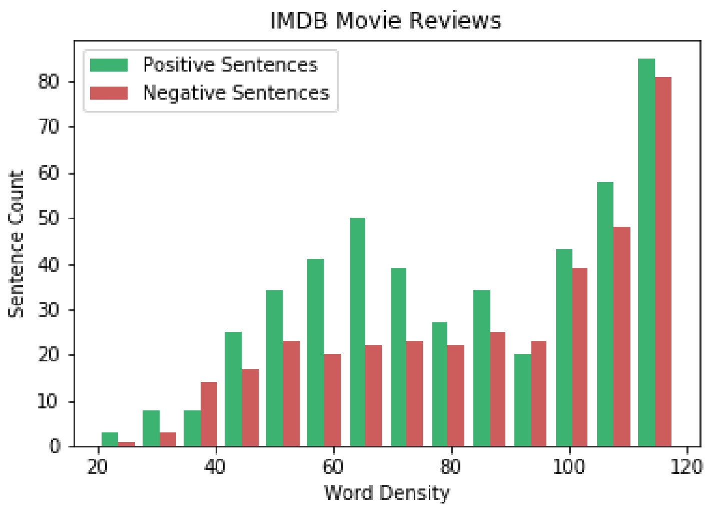
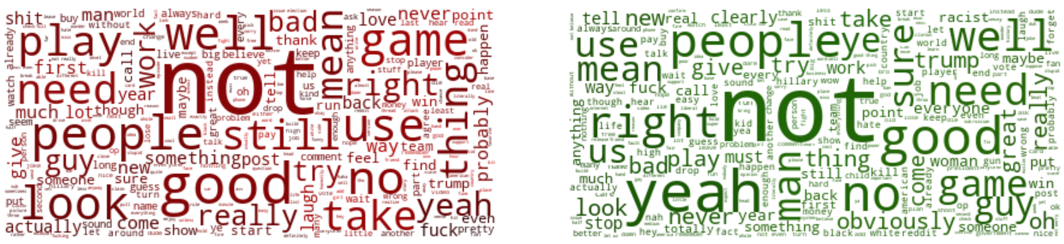


Figure 4.4: Word density histogram in IMDB dataset

In **Figure 4.4** we can observe the word frequency distribution over the IMDB movie dataset. Here we treat each collection of sentences as a single document, with positive or negative polarity annotation. Generally, the information spread across the sentences tends to be low at 20 to 80 words per document. As the word frequency rises above 80 words per sentence the word density is also getting bigger. A significant amount of sentences have high word density. Another interesting fact is that positive documents have more word density than the negative one, in contrast to twitter domain word density distribution.

### 4.1.3 Twitter: Sarcasm

This corpora [26] comprises of self-annotated sarcastic and neutral statements of more than a million sentences. The dataset size that we chose is roughly 15k sentences divided equally(class balance) in both positive and negative classes. The major difference with this dataset is the level of ambiguity and bias that introduces the sarcastic statements. Usually one should provide large volumes of training data to smooth out the bias factor in the training phase of the classifier. Although for our research we included roughly equal content with the other two datasets.



(a) Non-Sarcasm Words

(b) Sarcasm Words

Figure 4.5: Twitter: Sarcasm dataset, Wordcloud frequency overview

In **Figure 4.5**, we can observe the Wordcloud representation of the sarcasm dataset. On the left, we have all the sampled words from neutral class sentence distribution and on the right the sarcastic ones. One of the observations of the word frequency is that the noun “yeah” is far more frequent in the sarcastic collection. The same goes for the adverb “sure” and adjective “well” which are probably used in a sarcastic manner of a false positive statement. Something that adds a quite significant level of ambiguity in terms of classification.

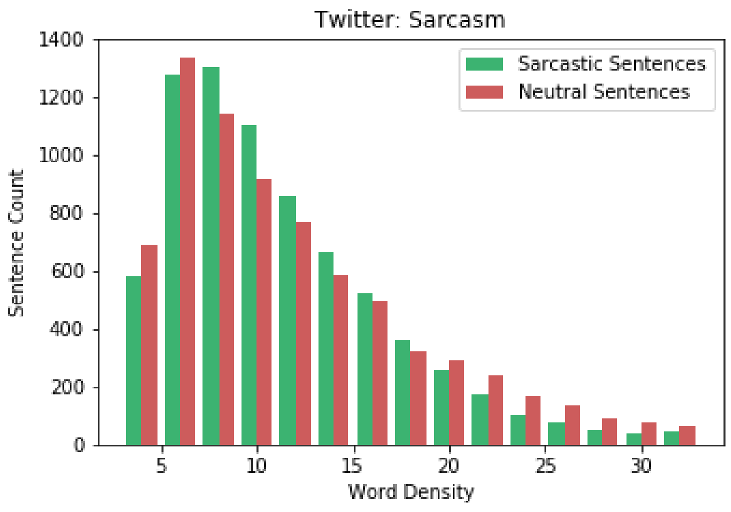
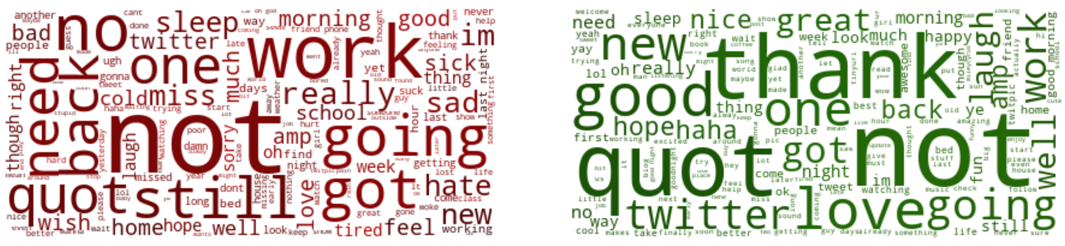


Figure 4.6: Word density histogram in Sarcasm dataset

In **Figure 4.6** we can observe the word density distribution over the annotated sarcasm dataset. Roughly 80% of the data is gathered around sentences with 3 to 20 words. With the majority of the sentences to fall below 15 words per sentence. In general sarcastic statements are not that long in terms of words. Of course, the source domain is also a reason behind the word density, there is a huge word difference between a statement and a review. The above corpus is drawn from different domains, thus it's difficult to make precise observations over the nature of sarcasm in text. In general, both classes(neutral, sarcasm) have a roughly equal context in terms of word density per sentence.

### 4.1.4 Twitter: Sentiment140

This Twitter corpora [21], comprises of self-annotated sentiment polarity(negative-positive) sentences. The size of the corpora is around 1.5 millions of sentences. The self-annotation procedure was totally automated by extracting emojis and assigning sentiment polarity based on those emojis. Thus the sentiment data is noisy, first due to the annotation and secondly due to the twitter content. We sampled a dataset of 15k sentences, equally divided(class balance) in two classes(positive, negative).



(a) Negative Frequent Words (b) Positive Frequent Words

Figure 4.7: Twitter: Sentiment140 dataset, Wordcloud frequency overview

In **Figure 4.7**, we can observe the Wordcloud representation of the Sentiment140 dataset. On the left, we have all the sampled words from negative class sentence distribution and on the right the positive ones. We can observe some sentiment words such as "good" and "love" expressed straightforwardly and positively. Also a word "work" is more expressed in negative sentiment class rather than the positive one.

In **Figure 4.8**, we can observe the word density distribution over the annotated Sentiment140 dataset.The majority of the data is gathered around 3 to 20 words per sentence. Pretty much natural since it is twitter domain, the data has a declining trend in volume from 8 words per sentence onward. With only a few sentences contain more than 20 words.

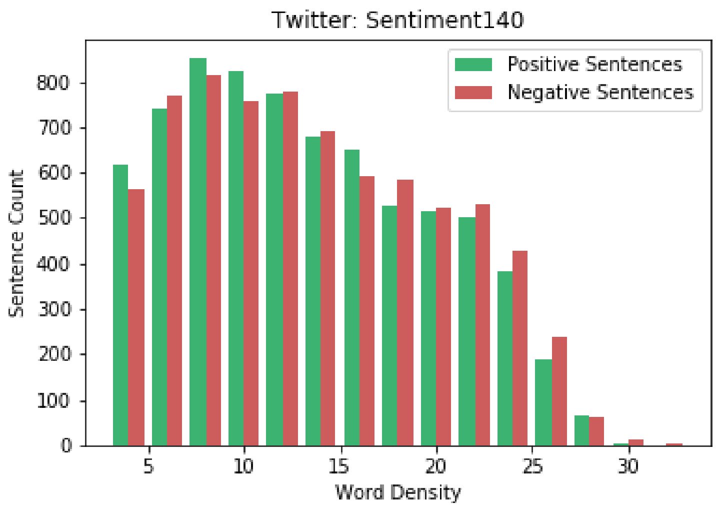


Figure 4.8: Word density histogram in Sentiment140 dataset

## 4.2 Text Preprocessing

In our experiments we perform sentence-level sentiment analysis, thus we treat each sentence of the dataset as a single data point in multidimensional space. Each input sentence is tokenized, preprocessed with certain NLP methods and reassembled as a postprocessed sentence. Thereafter, the cleaned sentences are mapped into multidimensional VSM through transformation algorithms(TFIDF) or fixed word-to-word mapping in the case of pre-trained and self-trained word embedding models.

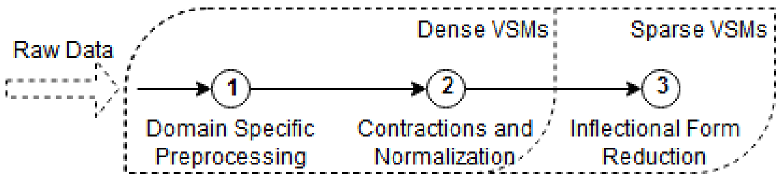

Raw Data
Dense VSMs
Sparse VSMs
1
2
3
Domain Specific Preprocessing
Contractions and Normalization
Inflectional Form Reduction

Figure 4.9: Text preprocessing pipeline steps for sparse and dense VSMs

As you can observe from **Figure 4.9**, where the preprocessing pipeline is depicted for both sparse and dense VSMs. The pipeline steps one and two are common for both of the experiments(dense and sparse representations) except the lemmatization in phase 3, which is used solely for TFIDF. We used spaCy, an advanced NLP library to perform the lemmatization procedure along with some other parts of the preprocessing.

### 4.2.1 Domain specific preprocessing

As the first phase of the preprocessing, we tokenize the input sentence, extracting single word tokens per sentence. Tokens that are related to domain-specific elements such as hashtags(#), references(@), links(HTTP) or user mentions are removed along with any numerical data present. We strictly focus on raw text data, since the purpose of this research work is to expose different aspects of VSM models towards sentiment analysis tasks. Thus we do not count domain-specific elements or any other numerical type of data as features. Although quite a lot of research work has been focused on domain-specific feature extraction, providing additional feature information from different domains [17] [6].

### 4.2.2 Word Form Expansion and Normalization

Non-standard words and word formations are present in many text domains. From the advertisement, reviews to short sentenced Twitter data. They are often related to specific domain terminology and expressions which users mostly use for their convenience of speech and communication. Although these lexical expressions often degrade the precision and the resulted quality of language models. Since they contain terms that are only present in these domains and considered as an out of vocabulary(OOV) for the general vocabulary [22]. Thus at the second phase of the preprocessing, we performed contraction expanding and domain-specific term preprocessing.

By meaning contractions, we refer to shortened versions of words and word groups, formed by omissions of internal letters. For instance, given the contraction formation "they aren't like us" becomes "they are not like us". So basically the shortened version is expanded to the full context of the word group. Moreover expanding contractions contributes to emphasizing the negation phrases in the sentence. Adverbs as "not" are restored to their canonical lexical forms. Regarding the text normalization, we compiled a list of highly frequent twitter abbreviated terms. We use this rule-based approach to replace frequent domain slang with its respective canonical form. For instance, in our mapping dictionary, "fyi" is expanded in its canonical form as 'for your information'. Text preprocessing and normalization heuristics tend to improve the performance of sentiment analysis models [1].

### 4.2.3 Inflectional Form Reduction

A high level of entropy in classifiers' decisions indicates a high level of uncertainty upon a decision-making process. Thus high entropic data tend to be less prototypical [15], by adding more diversity and often noise to the dataset. When it comes to words, different word forms such as adjectives endings "ing" and "ed" tend to raise the overall entropy of the dataset. Adding diversity to the lexical terms and expressions, something that often leads to increased noise inside the dataset. Of course, the whole procedure of entropy reduction is hugely dependant upon the task at hand.

In our work, we use lemmatization as a method of reducing inflectional word forms but only in the first part of the experiments with sparse VSMs. Since entropy reduction is a quite common technique for sparse frequency models, such as TFIDF [4]. Using lemmatization with dense VSMs is not such a good practice since most of the word embedding models are pretrained on huge corpora with a priory high level of the word diversity. Lemmatization is a technique reduces words to their basic forms. For example, verb forms "play-ing,ed,s" become "play" which is the common root of these forms.

## 4.3 Feature Extraction Techniques

### 4.3.1 Sparse Vector Space Models

Count based models are information retrieval techniques that utilize the word occurrence factor inside a collection of documents, or simply text. As we described before, the overall vocabulary of resulted occurrence matrix per document is quite huge and a big majority of words do not occur in every document. Something that leads to a sparse representation of a document(for instance, sentence) in multidimensional vector space.

Since the majority of the vector features regarding a single document will have numerous zero elements resulting in sparse document representation. The opposite side of sparse representations is the dense ones. Where information is distributed inside the multidimensional vector space with a certain degree of density, hence the lack of zero elements or else blank information.

#### 4.3.1.1 N-Grams

N-grams are groups of contiguous words. These groups range from a single word to any number of word combinations of preferred window size. The sampling window size impacts the retrieved descriptive information. For instance, the sentence "global warming is real" can be decomposed to Unigrams ("global,warming,is,real"), to Bigrams("global warming, warming is, is real") or to any number n-grams combinations.

The occurrence count of those n-grams combinations can be provided as input to any machine learning algorithm. It is worth mentioning that having a small number of n-grams will result in poor context capturing(for instance, negations). Contrary to having a large number of n-grams will result in rare representations of words, which will only add noise to the overall weighting scheme. Thus the number of n-grams should be empirically picked, according to the task at hand.

#### 4.3.1.2 Term Frequency Model (TFIDF)

Term frequency times inverse document frequency or else TFIDF is a statistical representation algorithm that consists of weighted frequency terms drawn from a collection of documents [51]. The terms frequency is weighted by its inverse document frequency(IDF) by sampling the percentage of documents that the specific terms appear. This weighting option allows some terms to have a stronger influence over other terms. Specifically, terms that are more related to few classes tend to have a stronger influence, rather than terms that appear more often in a different context.

This logic favors terms that carry knowledge over the text, rather than frequent words that appear statistically nearly everywhere in a text. Thus frequent terms such as stop-words, will have less impact on the weighting scheme of the algorithm, and generally will matter less. The **Equation 4.1** shows how the weighting term($t$) is calculated. Where $df(d,t)$ is the document frequency of the specified term and $n_d$ is the number of documents where that term appears.

$$idf(t) \; = \; log\left(\frac{1+n_d}{1+df(d,\; t)}\right) \; + \; 1 \tag{4.1}$$

## 4.3.2 Continuous Dense Vector Space Models

### 4.3.2.1 Word2Vec

In **Figure 2.3** the first complete implementation of feed-forward NNLM is depicted, where the projection layer is shared among different n-gram context and the hidden layer is a non-linear function. Thus, much of the model's computational complexity was caused exactly because of the non-linear hidden layer computations. Furthermore, there were certain restrictions in training data, since feeding too much data resulted in the model's inefficient performance. In [34] simpler architecture was proposed to train efficiently NNLM on much bigger training data, called Word2Vec. Word2Vec mainly has two training objectives techniques.

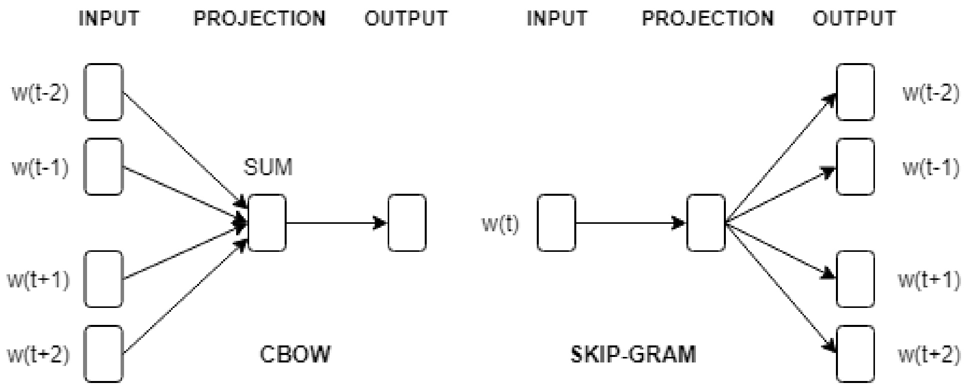


Figure 4.10: Word2Vec architectures overview

#### 4.3.2.1.1 Continuous Bag-of-Words Model

This architecture is closely related to the feed-forward NNLM, where the nonlinear hidden layer which was responsible for the computational overhead is removed. The projection layer, instead of being shared only by the contextual n-grams(just the projection matrix) is shared among all the words. Thus all the words are projected into the same position and their vectors are averaged. Hence the bag-of-words name, since the word order in history does not impact the projection. The best performance of the CBOW architecture was obtained when both future($before$) and history($after$) words were present in the sampled input. With the training criterion of predicting the middle word.

$$p(w_j \mid w_I) \;=\; \frac{exp\left(v_{w_j}^{'}\,{}^{T}\, v_{wI}\right)}{\sum_{j'=1}^{V} exp\left(v_{w_{j'}}^{'}\,{}^{T}\, v_{wI}\right)} \tag{4.2}$$

This architecture in its simplest form can be expressed mathematically as in the **Equation 4.2**. In this case we assume that the model will predict one target word($w_j$), given one context word($w_I$). Where $v_w$ and $v_w^{'}$ are different vector representations for the word $w$. Specifically the $v_w$ is the "input-vector" which is in fact is the input-hidden weight matrix of CBOW model, **Figure 4.10**. While $v_w^{'}$ is the so called "output-vector", which is actually the hidden-output weight matrix of the model [43].

#### 4.3.2.1.2 Continuous Skip-gram Model

It is similar architecture with the CBOW, but instead predicting a word based on contexts(future and history), Skip-gram tries to maximize the classification of a word based on another word in the same sentence. Specifically, each word is served as an input to a log-linear classifier with a continuous projection layer. To predict words within a certain window range before and after the input word. Empirically increasing the range of the prediction window(before words, after words) results in a better quality of the vectors, though adds more computational complexity.

$$p(w_{c,j} \; = \; w_{O,c}|w_I) \; = \; y_{c,j} \; = \; \frac{exp(u_{c,j})}{\sum_{j'=1}^{V} exp(u_{j'})} \tag{4.3}$$

In the output layer of Skip-Gram model, in **Figure 4.10**. Instead of outputting one multinomial distribution the Skip-Gram model outputs in total $C$ distributions. Where $w_{c,j}$ is the $j - th$ word on the $c - th$ panel of the output layer; $w_{O,c}$ is the actual $c - th$ word in the output context words; $w_I$ is the only input word; $y_{c,j}$ is the output of the $j - th$ unit on the $c - th$ panel of the output layer [43].

#### 4.3.2.2 Glove

Instead of utilizing shallow neural networks in a probabilistic fashion based on word contexts. Glove utilizes statistical information of word occurrences inside a given corpus. Specifically, Glove is an unsupervised learning algorithm in which the learning process is based on a global word to word co-occurrence statistical matrix created from the corpus.

The basic concept of this algorithm is to examine the relationship between words, by comparing their co-occurrence probabilities ratio, with various probe words $k$. Where $P_{ik}/P_{jk}$ is the probabilistic ratio of co-occurrence between words $i$ and $j$. The algorithm learns only the non-zero elements in the global co-occurrence matrix, rather than the whole sparse matrix. Since the number of non-zero elements is significantly smaller than the overall corpus word size, the training iterations of the algorithm progress significantly fast. A weighted least-squares regression of the Glove algorithm goes as follows.

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \left(w_i{}^T \, w_{\tilde{j}} \; + \; b_i \; + \; b_{\tilde{j}} \; - \; log \, X_{ij}\right)^2 \tag{4.4}$$

Where $X$ is the co-occurrence matrix, $X_{ij}$ is the frequency of word $i$ co-occurring with the word $j$, $w$ is the word embedding and $w^{\sim}$ is the separate context word embedding. $f(X_{ij})$ is a weight function which regulates and constrains the co-occurrence overweighting [41].

#### 4.3.2.3 FastText

Unlike with previous embedding models(Glove, Word2Vec) which entirely ignored the internal word structure. Since the n-gram input was based on word-level combinations. In cases of morphologically rich languages where different parts of speech have numerous inflected forms, the word representations may result in lower quality. FastText embeddings capture the internal structure of words by adding additional character level features. Where each word is represented as a bag of characters n-gram. For instance, the word "where" with character n-gram window of three, would be decomposed to its respective n-gram combinations as ("wh, whe, her, ere, re").

The word itself is also used with its n-grams so the model can learn words and characters representations. Thus having a dictionary of n-grams size of $G$ and a word $w$, $G_w \in (1, ...G)$ as the set of n-grams appearing in $w$. Let $Z_g$ be the vector representation of each of the n-gram $g$. The scoring function of the model will be a representation of a word as a sum of vector representations of its n-grams.

$$s(w, c) = \sum_{g \in G_w} Z_g{}^T V_c \tag{4.5}$$

#### 4.3.2.4 Doc2vec

Doc2vec or else Paragraph Vector is an unsupervised learning algorithm that learns fixed-length feature representations. From text pieces of variable length such as sentences, paragraphs or even completed documents. Doc2vec implies the basic concept of Word2Vec implementation, where prior context knowledge is used to predict the next word. Doc2Vec consists of two different training models.

##### 4.3.2.4.1 Distributed Memory Model(DMM)

Each sentence, paragraph or document is mapped to unique paragraph ID which is represented by a column in matrix $D$ [29]. Every word is also uniquely mapped to a column in word matrix $W$. The resulted paragraph vector and the word vectors are averaged into a single vector and used to predict the next word in the context. The paragraph ID token could be seen as a memory module since it remembers what is missing from the current context, **Figure 4.11**.

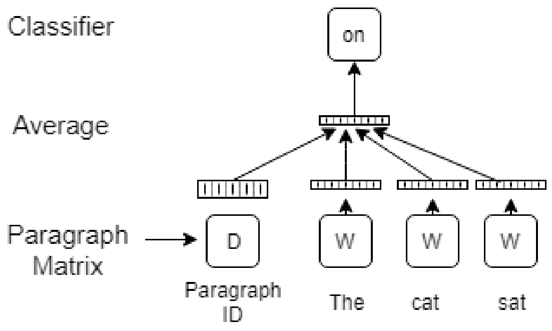


Figure 4.11: Doc2Vec, DMM Architecture Layout

#### 4.3.2.4.2 Distributed Bag Of Words(DBOW)

In this approach, while the concept of predicting the new word based on previous contexts exists. The context words are ignored and there is only Paragraph ID present in the input [29]. Thus in each optimization cycle(gradient descent) small text window is sampled, together with a random word within that text window. Lastly given the Paragraph Vector classification task is performed on the sampled data,**Figure 4.12**.

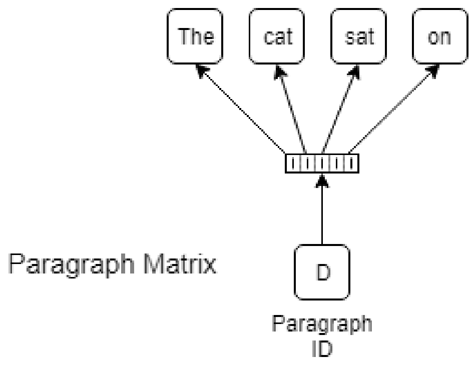


Figure 4.12: Doc2Vec, DBOW Architecture Layout

### 4.3.2.5 Universal Sentence Embeddings

Two main models are used to produce general-purpose sentence embeddings Universal Sentence Embeddings(USE) [13]. Both of these models are trained in a way to be as general-purpose as possible. This was achieved by using the same encoder to feed multiple NLP downstream tasks.

#### 4.3.2.5.1 Transformer

This sentence encoder model produces sentence embedding by using the encoding sub-graph of the transformer architecture [57]. This sub-graph utilizes the attention to produce context-aware representations in which both words position and orders are taken into account. These representations are transformed into fixed-size sentence embeddings by summing the representations at each of the word positions in the sentence. The encoder takes as input a lowercase PTB formatted token sequence and outputs a 512-dimensional sentence embedding.

#### 4.3.2.5.2 Deep Averaging Network

The second model utilizes Deep Averaging Network(DAN) [24]. In which input word embeddings and their bigrams are firstly averaged and later on passed through a deep neural network(DNN), to produce sentence embeddings. The number of feed-forward non-linear layers in DNN varies from task to task and the last non-linear layer is used to perform linear classification on its representations. Similarly to the Transformer model, the input PTB format and the output format dimensions are the same.

### 4.3.2.6 Deep Contextualized embeddings - Elmo

Elmo contextual embeddings consist of word vectors that are extracted from internal layers of stacked bidirectional language models(biLMs), that were pretrained on large corporas. For the bidirectional model, bidirectional LSTMs were used in synergy with different Language Model objectives to produce word representations.

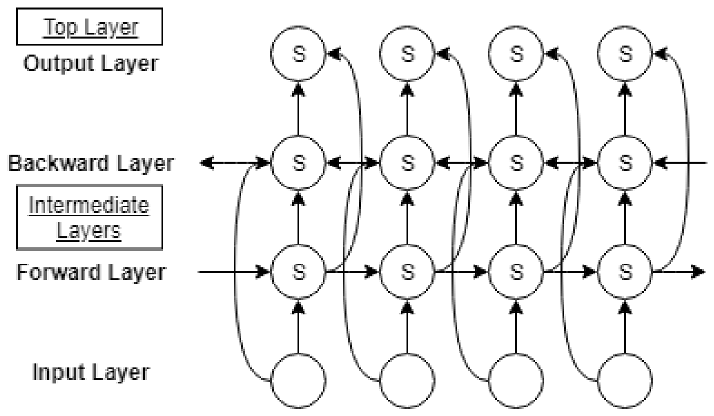


Figure 4.13: Bidirectional Language Model Overview

Elmo embeddings are computed on top of two-layer biLMs, with character convolutions. Since the higher-level states of LSTM can capture context-dependent aspects of words. The lower-level states can capture syntactic aspects of words. The top layer of single biLM is a function of a linear combination of its intermediate layer states, **Figure 4.13**. In the case of Elmo embeddings, the two LSTMs are stacked on top of each other, while the final representations calculated as the average of all three layers weights(two top layers of LSTMs, plus the character convolutions input layer) [42].

Given a sequence of $N$ tokens, $(t_1, t_2..t_N)$ a forward language models models the probability of token $t_k$ given the history tokens $(t_1...t_{k-1})$. While the backward language model models the probability of token $t_k$, given the future tokens$(t_{k+1}...t_N)$. The combined model(biLM) jointly maximizes the log probability of backward and forward models as it is expressed in **Eqution 4.6**.

$$\sum_{k=1}^{N} (log\, p\, (t_k \mid t_1 \,...t_{k-1};\; \theta_x, \theta_{LSTM}^{\rightarrow}, \theta_s) + log\, p\, (t_k \mid t_{k+1} \,...\, t_N;\; \theta_x,\; \theta_{LSTM}^{\leftarrow},\; \theta_s)) \tag{4.6}$$

The token representations $\theta_x$ and Softmax layer $\theta_s$ parameters are bound in both forward and backward directions. While LSTM parameters are kept separately.

## 4.4 Machine Learning Models

### 4.4.1 Multinomial Naive Bayes

Naïve Bayes(NB) is a probabilistic classifier based on the Bayes Theorem. The basic Bayes hypothesis states that the probabilities between interchanging events are totally unconnected/independent. Something that hardly happens when it comes to statistically modeling real-life events. But despite that fact, NB is quite efficient due to this simplicity of independent probabilities. The major concept of the NB is oriented around the observation of an event backed by prior knowledge.

Specifically Multinomial Naive Bayes(MNB), computes class probabilities for a given document as fallows.Let $C$ be the set of classes and $N$ the overall size of the vocabulary. Then MNB assigns a test document $t_i$ to the class that has the highest probability outcome $Pr(c|t_i)$ which utilizing the Bayes principle is formulated as:

$$Pr(c \mid t_i) \;=\; \frac{Pr(c)\; Pr(t_i \mid c)}{Pr(t_i)}, \qquad c\; \in C \tag{4.7}$$

The class prior knowledge or else Pr(c) is estimated by dividing the number of documents belonging to class c by the total number of documents. While $Pr(t_i|c)$ is the probability of obtaining a document like $t_i$ in class $c$ [27].

### 4.4.2 Logistic Regression

Logistic Regression(LR) or else Maximum Entropy classifier is a statistical way of describing and modeling relationships between categorical outcome variables and one or more continuous or categorical predictors. Since LR introduces non-linearity by applying logit($ln$) transformation of the dependent variables in the prediction.

In practice the LR algorithm models the logit(natural logarithm $ln()$ of odds, or else ratios of probabilities) of event $Y$ happening. General formula of LR can be seen at **Figure**

**4.8**, where the probability of the occurrence of event of interest is modeled. Or else $\pi = Probability(Y = outcome\ of\ interest \mid X = x,\ a\ specific\ value\ of\ x)$ is equal to logit transformation:

$$\pi = \frac{e^{a+bx}}{1 + e^{a+bx}} \tag{4.8}$$

Where $\pi$ is the outcome probability of the event, $\alpha$ is the $Y$ intercept and $\beta$ is the regression coefficient. As you can observe, The relationship between the probability categorical variable $Y$ and the predictor $X$ is non-linear [40].

### 4.4.3 Support Vector Machine

Support Vector Machine(SVM) belongs to a family of classifiers that deterministically find the best dividing point between two or more classes. Specifically, the SVM algorithm is regulating the boundaries(support vectors) between classes that are separated by a maximum equal distance. The SVM algorithm is described thoroughly in [11].

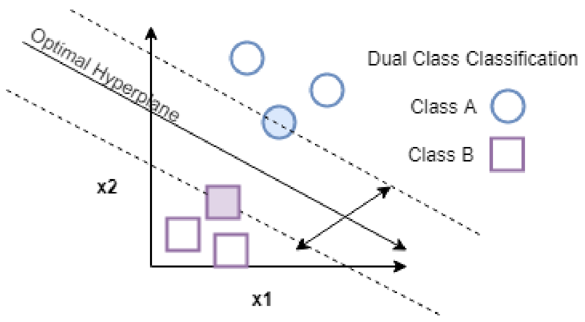


Figure 4.14: Dual-Class Problem, SVM model

The so-called boundaries or else support vectors are the class instances that are used to draw the optimal hyperplane between classes. You can see them in **Figure 4.14**, where the marked class instances are used to calculate the optimal hyperplane. The rest of the figures are class instances that lie behind the support vector planes. Through the training process, the optimal hyperplane is established, if the maximal-margin property between classes is satisfied.

### 4.4.4 Random Forests

Random Forests(RF) are the family of ensemble classifiers that form different groups of predictors. That are made of decision trees, trained in isolation. Unlike the boosting base models, which combine different group regularizers and weighting schemes.

RF algorithm generates(top-bottom) random trees that are trained separately. Where each tree branch represents the outcome of the test, while the leaf nodes represent the class labels. Finally, each of the predictions of the trees is averaged into a single prediction. Hyper-parameters of the RF model is the overall depth, feature randomness, and type of the predictors in leaves. RF is quite robust to overfitting since it normalizes the variance observed in the sampled data. Due to chunking data to different test samples [19].

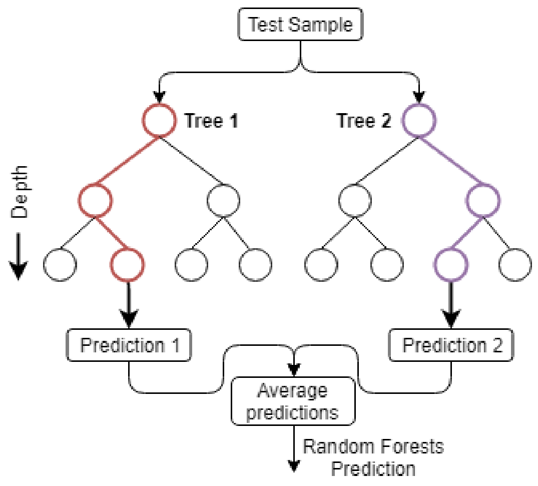


Figure 4.15: Random Forests Classifier[Tree No: 2, Depth: 2]

### 4.4.5 Adaboost

Adaboost is a boosting algorithm proposed by [50]. The purpose of this algorithm is to combine $t$ weak classifiers into strong one. By updating and adjusting their weights accordingly at each iteration. The general principle of Adaboost could be summarized as :

$$F(x) = sign\left(\sum_{t=1}^{T} a_t h_t(x)\right) \tag{4.9}$$

Where $h_t$ is the weak hypothesis $t$ and $a_t$ is the appropriate weight that has been assigned to this hypothesis. Practically the weak hypothesis represents the base/weak estimators that Adaboost uses in boosting capacity. In this work, we used as a base estimator for Adaboost, RF algorithm. Specifically, after the grid-search procedure with RF algorithm, we obtained the best hyperparameters for the weak estimator(RF) and used the same model in boosting capacity(Adaboost).

## 4.5 Machine Learning Data Preparation

### 4.5.1 Datasets Split

Since we mostly used Twitter corporas in our experiments, which are characterized as noisy datasets due to the high level of informality of speech. For that reason, we made certain post-processing steps, which involved sanitizing corpus(keep sentences from three words onward). To provide more expressive sentences as training data to our supervised classification tasks.

Table 4.1: Training-Testing split size per dataset

| Datasets | Size | Train-Set | Test-Set |
|---|---|---|---|
| Semeval2016 | 15k | 12250 | 3750 |
| Sentiment140 | 15k | 12250 | 3750 |
| Sarcasm | 15k | 12250 | 3750 |
| IMDB | 4k | 3000 | 1000 |

Furthermore, the sample size of the datasets(chunk size of k sentences as training and testing data) was preserved throughout all of the experiments. The train/test split ratio between the training and testing data was 70%/30%, **Table 4.1**.

### 4.5.2 Hyperparameter Tuning

Table 4.2: Hyper-Parameter Tuning, Grid-search

| ML Models | HyperParameter[1] | HyperParameter[2] |
|---|---|---|
| LinearSVC | C: [0.01, 0.1, 1.0, 10] | Iterations: 10k |
| LR | C: logspace(-3,3,7) | Iterations: 10k |
| RF | Depth: [12, 24] | Estimators: [400, 800] |
| MNB | ————————- | ————————- |

After preprocessing the textual data of the datasets, feature extraction techniques were performed to map text data to Vector Space. We performed Hyperparameter tuning on our ML models in order to obtain better results, **Table 4.2**. Precisely we tuned the C regularizer parameter for both linear models. We also raised the maximum number of iterations to

10k. Dense high-dimensional representations such as WEs make linear models such as LR, LinearSVC hard to converge. Lastly, we tuned the overall number of leaves in RF model, and the maximum number of trees(estimators).

### 4.5.3 Cross-Validation

To train less-biased models with better generalization. We performed cross-validation in $K$-Folds fashion, with overall 10 folds. Cross-Validation splits the training data into $K$ folds, and $K$ different models are trained using one fold for testing and the remaining $K - 1$ for training. Partitioning training data results in less variance across produced models, thus less overall bias.

## 4.6 Evaluation

One of the most common evaluation metrics in ML is Accuracy, which is defined by the number of correct instance classifications over the overall number of classified instances. While accuracy is one of the indications of the model's performance, it is not as reliable as it seems. For instance, in the case of class unbalance(more samples from class A rather than class B in training phase), the Accuracy metric could by high. Since the classifier could be assigning the majority of the instances to the dominant class. Something that is not what was intended in the first place. Clearly, the model is biased towards a certain class, due to the class imbalance in the training phase.

$$\begin{gathered} Precision = \frac{tp}{tp + fp} \\ Recall = \frac{tp}{tp + fn} \end{gathered} \tag{4.10}$$

- **True positive**($tp$): correctly attributed instances to the relevant class
- **True negative**($tn$): correctly attributed instances to the other class
- **False positive**($fp$): : mistakenly attributed instances from the other class to the relevant class class;
- **False negative**($fn$): instances from the relevant class mistakenly attributed to the other class.

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{4.11}$$

The metric that we are using in this work is a balanced combination of Accuracy but also the Recoil of a model. The F1-score is a useful metric of testing models performance when it comes to binary classification problems. F1 represents harmonic-mean of the Precision and Recall of the model, **Equation 4.11**. Since our dual sentiment classification problems involve two classes, the F1-score is computed individually for each of them and then is averaged.

# Chapter 5

# Empyrical Findings

## 5.1 Sparse Vector Space Models Experiments

### 5.1.1 TFIDF Cross-Domain Experiments

#### 5.1.1.1 Data Preparation

For this part of the experiment, we evaluated the cross-domain relations of the TFIDF sparse model. For that purpose, we chose three different datasets(IMDB, Semeval2016, Sarcasm), with different word density, context, and domain(sentiment polarity - sarcasm). While Semeval and ImDb datasets are both domain-related(sentiment polarized datasets) their main differences are the word density and a figure of speech. In contrary Sarcasm dataset consists of polarized sarcastic and non-sarcastic sentences, instead of sentiment annotated ones. Thus we treat the sarcasm dataset as the outer domain with respect to the other two sentiment domains.

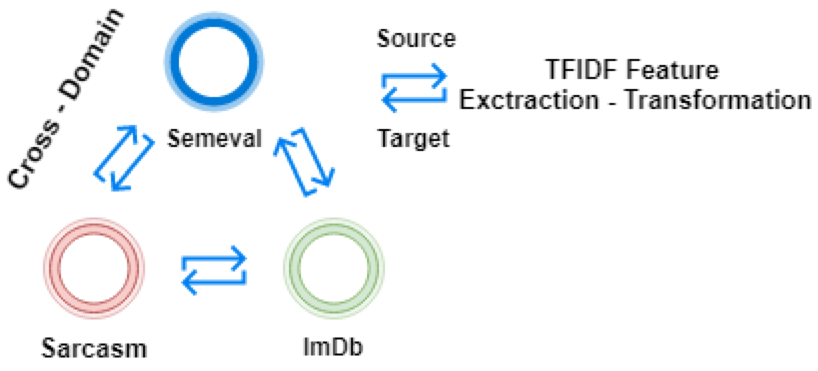


Figure 5.1: TFIDF Cross-Domain pipeline

Therefore each of the domains acts as a Source. The training data is fitted to create the internal dictionary representation of TFIDF algorithm of the Source domain, which contains mapped relations of terms and their respective weights. After we use the same representations to transform the training and the testing set of Target domains. In this way, we cross-utilize the TFIDF representations between different domains. In **Figure 5.1** the cross-domain pipeline is depicted.

Table 5.1: TF-IDF Hyperparameters

| analyzer | min_frequency | n-grams | tokken_pattern | norm |
|---|---|---|---|---|
| $word$ | $count\_of[3]$ | $[1, 3]$ | $r'\backslash w\{2,\}$ | $l1$ |

In **Table 5.1**, Hyperparameters that we used in TFIDF cross-experiments can be observed. The analyzer of TFIDF is focused on words(more than 2 characters), with a minimum frequency of 3 words per corpus. This threshold discards the very rare domain-specific words, that usually add noise to internal TFIDF representations. We also combined the usage of n-grams, ranging from a monogram to trigram as word formations.

#### 5.1.1.2 Sparsity

Since sparsity is of the main weaknesses of the TFIDF algorithm, due to large internal dictionary representation and consequently plenty of zero elements in document feature vectors. Besides, there is no information sharing in sparse VSMs since data points are represented in the sparse non-distributed context. Scaling up the training data of the algorithm scales up the internal dictionary representation dimensions, which consequently leads to underfitting and the lack of generalization [28].

Another reason which can harm the sparsity factor is the domain-specific context. Special terminology or slang elements that are bound to the specific way of speech inside different domains(for instance, Twitter). Will most likely add nothing but noise in the general weighting scheme of the TFIDF algorithm. To deal with this problem, we use certain NLP techniques to unfold non-canonical word forms. In **Figure 4.9**, you can observe the preprocessing steps we apply to raw data of each of the domains(IMDB, Semeval2016, Sarcasm). Since we are dealing with sparse VSM we also apply inflectional forms reduction(lemmatization) to our data, to improve the recall factor of TFIDF.

#### 5.1.1.3 Results

After data is preprocessed we apply n-grams as input to TFIDF algorithm. The reason behind utilizing n-grams is to capture as much content as possible and create TFIDF internal representations including these word combinations. Since in the sentiment domain it is also crucial to deal with different forms of negations, that are accompanied by specific negation tokens inside the text context. In general, the usage of n-grams with TFIDF instead of simple unigrams tends to have a positive impact on sentiment analysis tasks with linear classifiers [54].

Finally, we use three different linear classifiers(SVM, MNB, LR) to perform cross-domain sentiment analysis tasks. The overall performance of the linear models was quite close since all of them are handling sparse data pretty well. Thus we aggregated the results per model to present the average F1 as a trusted evaluation factor of the experiments.

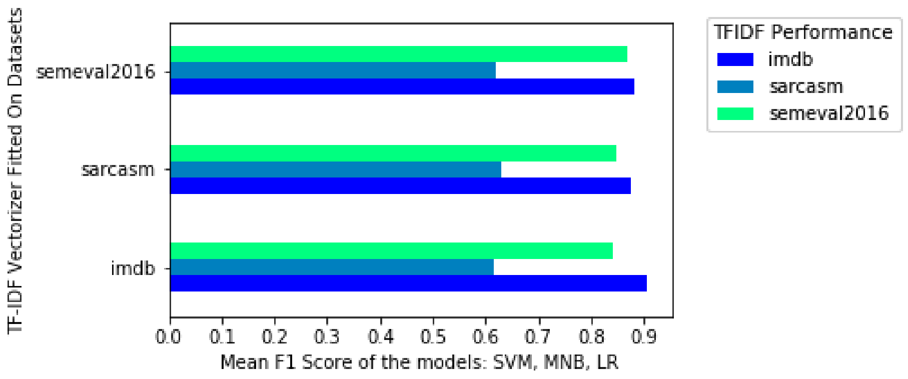


Figure 5.2: TF-IDF cross-domain performance

Table 5.2: TF-IDF Cross-Domain evaluation of mean F1[SVM, MNB, LR]

| Source Domain | IMDB[F1] | Sarcasm[F1] | Semeval2016[F1] |
|---|---|---|---|
| IMDB | **0.91** | 0.61 | 0.84 |
| Sarcasm | 0.87 | **0.63** | 0.85 |
| Semeval2016 | 0.88 | 0.62 | **0.87** |

As it is seen from the **Table 5.2**, the general rule of thumb states that when TFIDF dictionary is drawn from the same source, the F1 performance metric is always higher. Something that makes sense, since the test set will have a quite similar distribution with the train set, which is used to create the internal dictionary of TFIDF. Although it is quite evident that, because of the count-based nature of the TFIDF algorithm which disregards the syntactic information. TFIDF adapts quite well to different domains and lexical structures.

It is also worth mentioning that the preprocessing steps of unfolding domain-specific content and minimizing the word entropy bear positive results over cross-domain representations. The divergence of the F1 metric was quite small in the correlation results, with values ranging from [1% to 3%] of a total performance difference.

In the case of sarcasm, the F1 score(62%) was quite low in contrast with the other two sentiment datasets. Sarcasm is still one of the challenging fields in NLP and classification since there are many factors that one should account for to perform an efficient sarcasm classification task. By adding more feature options, interpreting specific domain elements, linguistic elements related to sarcasm, or even categorizing sarcasm into one of several types [16].

### 5.1.2 Sparse Linear Model Analysis

In this part of the experiment, we will use one of the best linear models of the previous experiment.

$$SVM(c = 0.01), TFIDF(ngrams = [1, 3]), Dataset(Semeval2016)$$

Furthermore we used eli5[source: **eli5**] framework to analyze each individual lexical feature contribution to the final outcome of the classifiers prediction. In **Figure 5.3** we can see the internal organization of eli5 prediction analysis. To make the prediction analysis, one should supply the eli5 framework with a pre-fitted classifier and vectorizer that has already internally mapped the feature names.

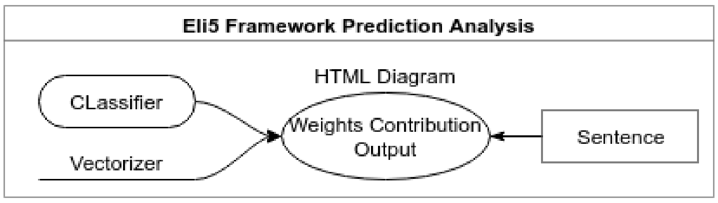

Eli5 Framework Prediction Analysis
HTML Diagram
CLassifier
Weights Contribution Output
Sentence
Vectorizer

Figure 5.3: Eli5 prediction explainer

Next, we performed prediction analysis on misclassified(FP) testing data drawn from Semeval2016 dataset. In **Table 5.3** we can observe the top three positive and negative text features and their contribution to the overall weights of our classifier. Features with slight positive or negative weights are also depicted in between. Since we included both(unigrams,bigrams,trigra in our TFIDF vectorizer, as extra features. Positive/Negative weights signify the relationship between the given n-gram with Positive/Negative sentiment class.

Different combinations of word n-grams can be observed, reflecting their respective weight contribution to the classifier's prediction. Different n-gram combinations of the word "dysneyland" results in positive sentiment class weights. While, unigram features such as sentiment word "sad" and negation "not", have huge contribution towards negative sentiment class. Since as unigrams, there are more frequently related to negative sentiment context. Although a bigram "sad to" has slightly positive weight(+0.028) simply because this bigram is quite common in both positive and negative sentences.

Table 5.3: Eli5 framework false-positive(fp) prediction analysis
Sentence : "sad to not be go to disneyland tomorrow", the word "go" is a stemmed version of its gerund form "going".

| **n-gram combination [1, 3]** | **SVM weights** |
|---|---|
| disneyland tomorrow | +0.357 |
| to disneyland | +0.328 |
| disneyland | +0.316 |
| go to disneyland | +0.257 |
| to disneyland tomorrow | +0.182 |
| to not | +0.124 |
| tomorrow | +0.093 |
| <**BIAS** > | +0.049 |
| be go | +0.044 |
| go to | +0.039 |
| go | +0.029 |
| sad to | +0.028 |
| to | +0.022 |
| be go to | +0.008 |
| not be | -0.022 |
| be | -0.117 |
| not be go | -0.230 |
| not | -0.417 |
| sad | -0.866 |
| **Weight Contribution** | **y=1 (score +0.224)** |

The BIAS term is there because of the intercept of the linear model(SVM), which can be seen directly through the eli5 model. The overall weight contribution sums up to the class prediction, with negative weight sum resulting in negative prediction and visa-Versa. The weighted sum(+0.224) of this prediction resulted in a positive score, hence the false positive(fp) class prediction. Even without the presence of BIAS(+0.049) the resulted prediction score would be positive(sentiment), even though it is not.

This observation provides significant evidence about n-gram domain dependencies. The proper combination of n-grams should be empirically set, based on domain observations. Including all combinations of unigrams, bigrams and trigrams as features might reflect poorly in terms of performance.

## 5.2 Continuous Dense Vector Space Models Experiments

In this section, we perform sentiment analysis based on continuous dense text representations such as word embedding models.

The first part of the experiment is oriented around the performance of self-trained word embedding models, that are trained on sentiment corpora Sentiment140.

In the second part of the experiment, we utilize already pretrained word embedding and contextual embedding models to perform sentiment analysis.

The ML models that we used in dense VSMs experiments are:

- SVM
- Adaboost(RF)
- LR

Linear models indeed have some difficulties when it comes to converging, because of the dense multi-dimensional input of word embeddings. Although one can counter this problem by raising the number of convergence iterations of the model, **Section 4.5.2**. The only challenge that we faced was using MNB with dense text representations, which had negative values. MNB genuinely is quite sensitive to positive arithmetic data. Countering this issue(min-max scale) introduced other performance issues thus, we picked another model.

Instead of MNB that we used for sentiment analysis with Sparse VSMs. To evaluate dense text representations, we used a generative approach of RF. But instead of using directly RF algorithm, we used a boosting variance Adaboost(RF). The performance of Adaboost(RF) was quite close, even some times outperforming the other two linear models(SVM, LR). The purpose of the 3rd model was to enforce a more robust performance estimation.

### 5.2.1 Self-Trained Word Embeddings

For this part of the experiment, we evaluated self-trained embedding models such as Doc2Vec(PV-DM, PV-DBOW), Word2Vec(DBOW, DM) and Glove on sentiment annotated corpus Sentiment140. The overall training corpus consisted of 1.6 millions of annotated sentiment sentences. The preprocessing pipeline consisted of the usual steps that we described in Text Preprocessing, [**Section 4.2**].

We performed the same preprocessing set up for both training and evaluating the word representations. We used Gensim library implementations of Doc2Vec and Word2Vec algorithms and python Tensorflow implementation of Glove model. Specifically, the models that were used:

- Doc2Vec
  - Doc2Vec Gensim Model [source: **d2v**]
- Word2Vec
  - Word2Vec Gensim Model [source: **w2v**]
- Glove
  - Glove Tensorflow Implementation [source: **glove**]

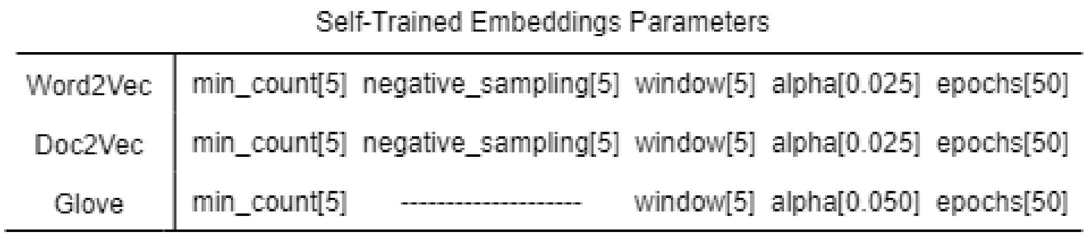
#### 5.2.1.1 Text Preprocessing Options

We performed different forms of preprocessing such a stop words removal, we also used compiled stop words list specifically tailored for sentiment datasets. Where any stop word related to sentiment inference(negations, sentiment stop words) were excluded from the compiled source. Both of the approaches resulted in less significant performance on the metric score F1, as they introduced more bias.

Unlike TFIDF approach, word embeddings training process is highly related to syntactic and structural principles(Word2Vec, Doc2Vec). Since each time a word is predicted based on the specific context, or vice versa. By removing the stop words, these structural dependencies are no longer effective and the prediction capacity of the model is handicapped.

We also excluded word entropy reduction techniques and inflated form reduction(Lemmatizing, Stemming). Since word embeddings value structural and syntactic dependencies, while the training process is conducted on a large variety of word forms(for instance adjective endings). By reducing inflated word forms, much of the embedding information is lost, resulting in lower performance.

#### 5.2.1.2 Training Parameters


Self-Trained Embeddings Parameters

| | | | | | |
|---|---|---|---|---|---|
| Word2Vec | min_count[5] | negative_sampling[5] | window[5] | alpha[0.025] | epochs[50] |
| Doc2Vec | min_count[5] | negative_sampling[5] | window[5] | alpha[0.025] | epochs[50] |
| Glove | min_count[5] | ------------------- | window[5] | alpha[0.050] | epochs[50] |

Figure 5.4: Training Hyper-Parameters settings

##### 5.2.1.2.1 Minimum Number Of Occurrences

Is the actual frequency number of a token, that is required to include the specific token in the overall internal dictionary of the embeddings. If the number is low, more tokens will be added in the overall distribution of words, something that might add statistical noise to the embeddings representations. If the number is too high, fewer words will be added to the overall distribution of embeddings with a consequence of less generalization on unseen content. For our experiments, since we are using Twitter data we set the number of occurrences to five. Twitter data is usually noisy and contain terms that adds nothing but noise to the distribution of the overall representation, despite all the preprocessing efforts.

##### 5.2.1.2.2 Negative Sampling

In the case of word embedding models such as Word2Vec, Doc2Vec that are based on a shallow neural network approach. In the training phase, when a training sample is present. The whole purpose of the training procedure is to adjust the neuron weights slightly so that the training sample could be predicted more accurately. Meaning that a single training sample impacts all the weights in the neural network in distributional fashion. To tackle this problem, NS was introduced. Which instead of modifying all the weights at once, updates a small percentage of them. Specifically, it updates weights for only a random selection of words that are related to negative or positive class outcomes. In our experiments we considered the default value of Negative Sampling(NS)=5 in Gensim implementation of Word2Vec and Doc2Vec models.

##### 5.2.1.2.3 Context Window

Context Window(CW), is a sampling window of the previous or future context, depending on the training algorithm implementation. Maximizing CW, usually results in better accuracy of the model as more content is available for word prediction. Although it is more computationally expensive and must be used wisely with an analogy of the training size of the corpus. In our experiments since the average word count in a Twitter sentence is below ten words, the context window was set to overall five words. Increasing the context window to more than five words did not produce any meaningful results.

#### 5.2.1.2.4 Learning Rate - alpha

Regarding the learning rate, we experimented with different custom techniques of manually decaying the learning rate at each of the learning iterations. While providing a low-threshold cap to learning rate each time. The resulted trained representations of both Word2Vec and Doc2Vec models were underperforming. Thus we set the learning rate for those models to its default values alpha=0.025. For Glove model training, the referred learning rate was quite slow to capture meaningful representations. Thus the learning rate for Glove was set to alpha=0.050, since it yielded better inference results.

#### 5.2.1.2.5 Learning Epochs

Usually, to sufficiently capture word representations with self-trained models, one should provide an excessive amount of training data(billions of words). The way that pre-trained embeddings were trained, for instance, Google team used large text corpora with billions of word tokens. In this case just a few epochs are enough to capture text representations. Because the training corpus size(1.6 million sentences) that we used contained a small amount of text data. We increased the number of epochs to 50, to minimize the bias gap, introduced as a result of inefficient training data. Lowering the learning epochs to something less than 40, resulted in performance drops.

### 5.2.1.3 Results

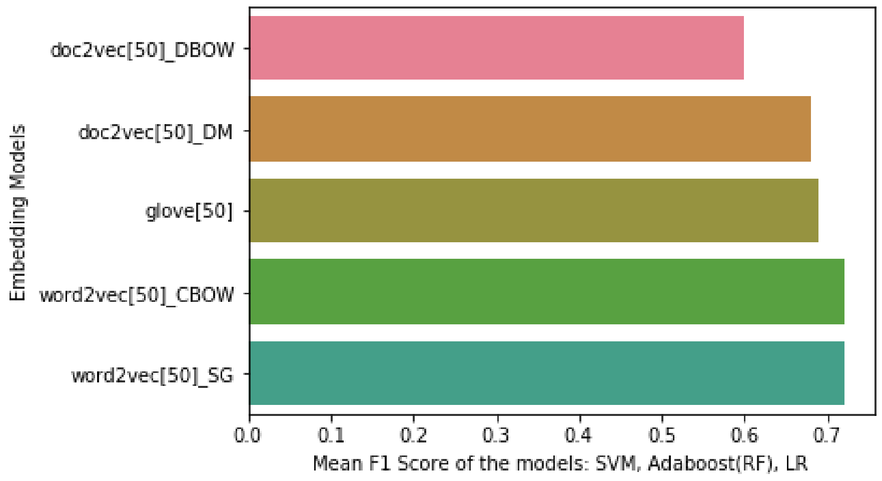


Figure 5.5: Mean F1 Performance evaluation of pre-trained word vectors on sentiment corpora Sentiment140

We performed evaluation(F1 metric) of pretrained embedding models that were trained on sentiment corpora Sentiment140. The models that we used to determine the performance of the embeddings were SVM, Adaboost(RF) and LR. The mean F1 metric was calculated by aggregating the individual model results into a mean score.

As it can be seen from **Figure 5.5**, the worst-performing learning algorithm was DBOW from Doc2Vec implementation. It is pretty much clear that the specific training option required more training data to obtain meaningful representations. While DM training mode performance was quite close to Glove's F1(69%).

Word2Vec implementation outperformed both Doc2Vec and Glove in terms of performance, with CBOW and SG training techniques, both scoring F1(72%).

Since CBOW implementation is ignoring the word order but predicting the centered word based on future and past content. It seems that in sentiment analysis tasks it is quite efficient after all, considering elements such as negations that might be present in different areas of proximity with sentiment words. The same goes for SG, which is the quite opposite of CBOW. Where the prediction of future content is based on the input word.

If we compare the results of self-trained word embedding models with the pre-trained ones **Table 5.5**, on sentiment dataset Sentiment140 . We can see that both word2vec models, performed almost the same($\approx \%1$).

Although our pretrained model was trained only on 1.6 million of annotated sentiment sentences, less than 50 million tokens. The pretrained Word2Vec model was trained on 100 billions of tokens from random corporas, **Table 5.4**.

### 5.2.2 T-SNE Analysis

In this section, we performed a 2-D overview of the internal relations(topology) of sentiment specific word embeddings based on T-SNE algorithm [32]. Which is unsupervised non-linear dimensionality reduction technique primarily used for data exploration and visualization in low dimensional spaces(usually in 2D or 3D).

The purpose of this algorithm is to provide insights into the relations between the data in high-dimensional Euclidean spaces. The main logic behind the T-SNE is to reduce dimensionality while at the same time preserving pairwise relations between data points. Thus data points that are far away from each other in multidimensional space, will be proportionally distance separated in the resulting 2D or 3D dimensional output. Generally, T-SNE algorithm maps data from multidimensional domains to 2D or 3D object representations.

#### 5.2.2.1 Sentiment Orientation Analysis

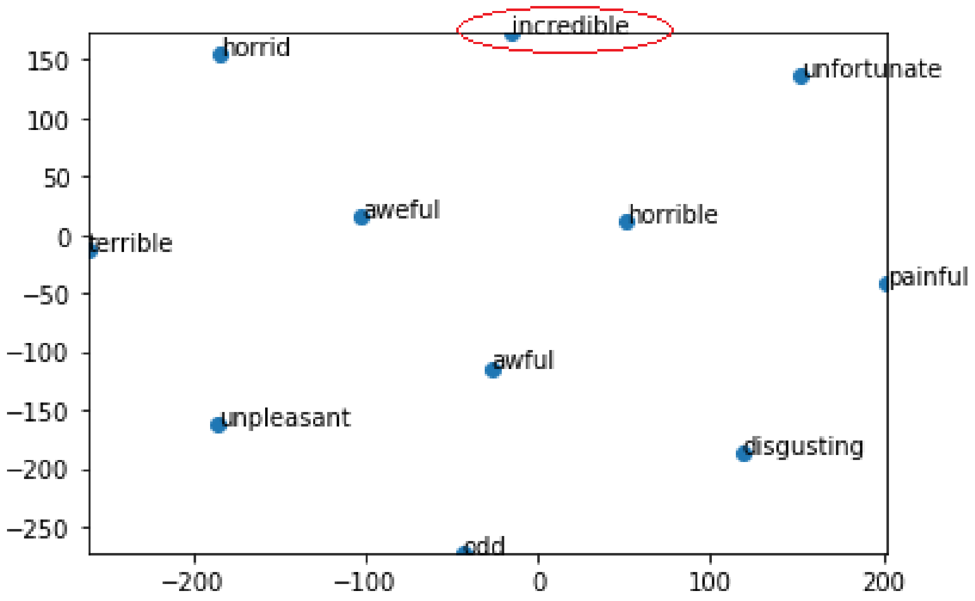
We picked one of our best model(word2vec[50]_CBOW) of the previous experiment, **Figure 5.5**. To analyze some of the relations and analogies that it captured among sentiment words. After training the model on sentiment corpora, Sentiment140. Specifically, we sampled the closest $n(10)$ words in a word cluster of sentiment words such as “awful” and “great”. When it comes to multidimensional word representations, in practice the neighboring(syntactical, meaning) words are forming word clusters. If the word embeddings distances are close(same cluster), the words will be semantically related.

Since $n = 10$ neighboring word vectors is a significant sample of closest words in terms of linguistic relation and meaning. If we add more samples to T-SNE observations, of course, you will see more noisy word embedding presence. Words that are barely meaningfully related or where more syntactically frequent with targeted sentiment word.

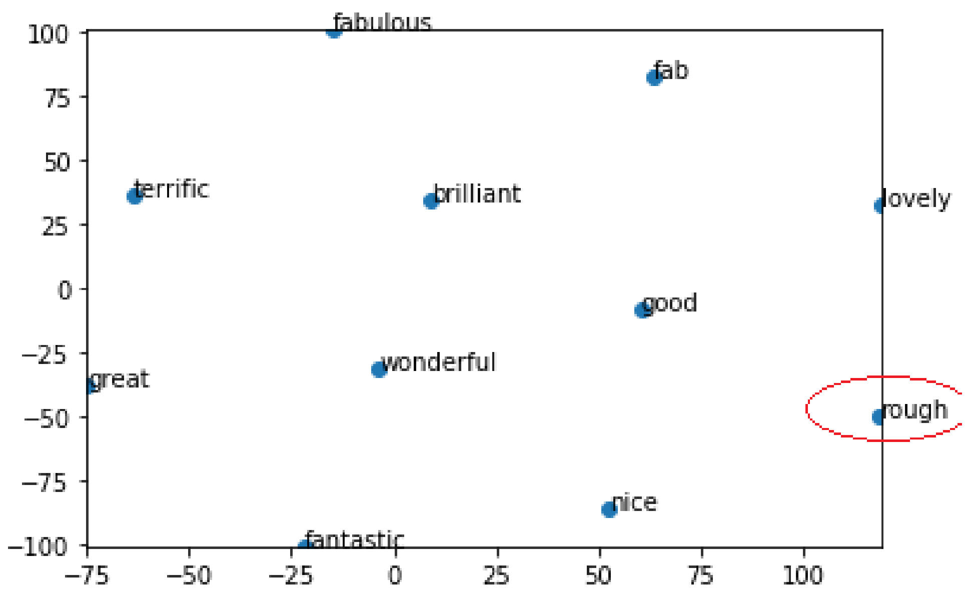
As can be seen from **Figure 5.6**. Closest 10-word embeddings were captured, that were related with sentiment words “great” and “awful”. It is worth mentioning that both 10 words are sentimentally related to the targeted sentiment word. This is because the sentiment corpus that we used for training, contained sentiment sentences with sentiment phrases and words. Thus sentiment word relations were captured more “focused”, than in generic embedding model which was trained on random corporas.

In the case of sentiment word “awful”, we can see that 90% of clustered words express the same negative sentiment. While the word “incredible” definitely reflects opposed emotion of joy, thus positive sentiment class. The same behavior can be seen with targeted positive sentiment word “great”, where the presence of negative sentiment word “rough” is in the same word cluster.

The main observation on both of these sentiment word samples is that shallow NN model architectures such as Word2Vec. Cluster anti-diametrical sentiment words, probably because these same words were obtained through a similar context. Thus words with similar vector representations tend to have an opposite sentiment polarity(positive, negative). Something that is a degrading factor when it comes to sentiment analysis [58].



(a) Top 10 closest word embeddings of sentiment word, “awful”



(b) Top 10 closest word embeddings of sentiment word, “great”

Figure 5.6: T-SNE 2D analysis of sentiment words, “awful” and “great”. Sentimentally opposed words are depicted in red circles. Something that could add significant noise in sentiment analysis tasks. Since anti-diametrical emotions will be close to each other in the embedding space, thus treated as related.

### 5.2.3 Pre-Trained Word Embeddings

For this part of the experiment, we evaluated pretrained word embedding and contextual models. In **5.4**, inner parameters and training capacity of the models can be seen.

Table 5.4: Pre-Trained Word Embedding Models Overview

| Pretrained Models | Vocabulary | Dimensions | Training | Source |
|---|---|---|---|---|
| Word2Vec | 3m words | 300 | 100b tokens | **w2v** |
| Glove | 1.9m words | 300 | 42b tokens | **glove** |
| FastText | 2m words | 300 | 600b tokens | **ftext** |
| U.S. Encoder(DAN) | N.K. | 512 | N.K. | **u.s.e.** |
| Elmo | N.K. | 1024 | 1b tokens | **elmo** |

The training capacity of shallow NN models such as Word2Vec and FastText, but also statistical-based models such as Glove. It can be extended up to several hundreds of billions of tokens. Due to the limited training complexity of the model. In contrast with more complex language models such as Elmo, which used only 1 billion of tokens for the training process.

What also can be seen is that shallow NN models usually cap their dimensionality up to 300 dimensions per embedding. It is still debatable whether the full amount of dimensions(300) encapsulates significantly more information than lesser dimensional models[50,100,200]. A recent study suggests that the dimensionality of the word embedding model should be tuned according to the task at hand. Empirical studies [33], show that 300 dimensions per word embedding is enough to encapsulate most of the information for downstream NLP tasks. Including sentiment analysis.

Although usually when the model complexity is high(u.s.e. , elmo), the number of dimensions tend to increase. Since more information is being extracted on a multiple levels (char level analysis, sub-word n-grams, attention, e.t.c.) from the training data.

#### 5.2.3.1 Text Preprocessing Options

We followed exactly the same text preprocessing procedure as with the self-trained word embedding models, **Section 5.2.1.1**

#### 5.2.3.2 Results

From **Table 5.5**, you can observe that typical NN and count-based models as W2V and Glove, have almost the same performance($\approx 1\%$) across all of the three datasets. While more advanced LMs performed slightly better results, around [2 - 3]% percent performance gap, except the Sentiment140 dataset where the performance gap was [2 - 5]% in the overall performance.

Table 5.5: Pretrained Language Models evaluation of mean F1[SVM, Adaboost(RF), LR]

| WE Models | IMDB[F1] | Semeval2016[F1] | Sentiment140[F1] |
|---|---|---|---|
| w2v[300] | 0.84 | 0.83 | 0.73 |
| glove[300] | 0.85 | 0.84 | 0.73 |
| ftext[300] | **0.86** | 0.85 | 0.76 |
| use[512] | **0.86** | 0.85 | **0.78** |
| elmo[1024] | 0.84 | **0.86** | 0.75 |
| TFIDF | **0.91** | **0.88** | **0.79** |

It is also worth mentioning that Elmo LM had the best performance on Semeval2016 dataset with F1(86%) score evaluation. While at the same model, was trained only on 1 billion tokens as input data, **Table 5.4**.

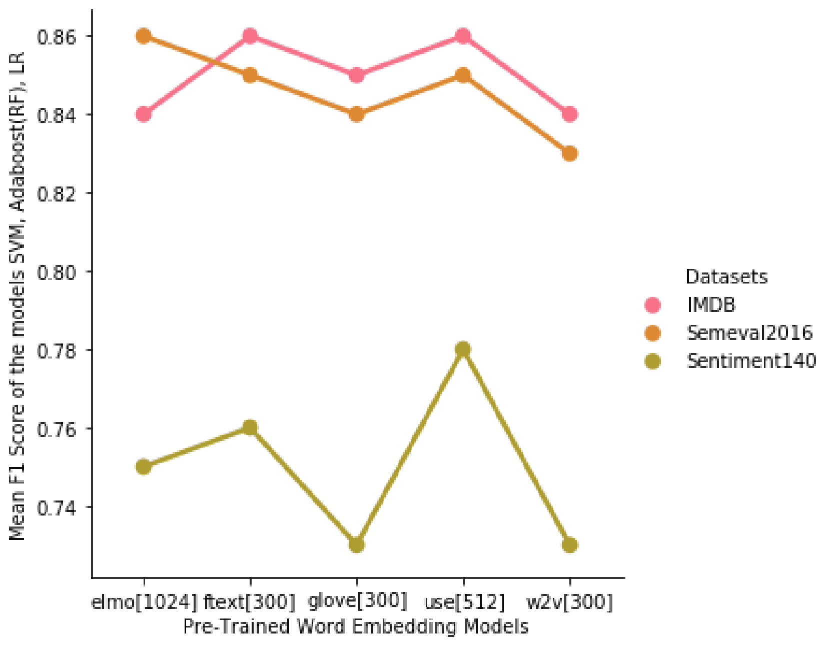


Figure 5.7: Mean F1(SVM, Adaboost(RF), LR) performance evaluation of pre-trained word vectors on sentiment datasets(IMDB, Semeval2016, Sentiment140)

The USE model achieved high-performance F1 score on IMDB(86%) and Sentiment140(78%). The USE embeddings were trained on deep NN setup, known as DAN. Where the training data, words and n-grams were first averaged together and after passed to deep FFNN architecture.

Comparing with the results of sentiment analysis on sparse VSMs in datasets(ImDb, Semeval2016 and Sentiment140), **Table 5.5**. It is obvious that TFIDF has better performance on both datasets. Where F1 scores of IMDB(91%) and Semeval2016(87%) datasets with TFIDF transformation algorithm were higher than the best performing pretrained LMs, F1 score IMDB(86%) and Semeval2016(86%).

It is quite interesting, that simple sparse count-based models such as TFIDF outperform complex neural LMs, when it comes to sentiment analysis. Although, dense VSMs such as word embeddings or contextual embeddings usually have been trained on general data(corporas, crawls). Without any specific domain orientation or NLP task focus, regarding its training data. Thus the fact that pretrained general word embedding models don't have a significant impact on sentiment analysis tasks, is not surprising.

On the other hand, word embeddings also introduce the sentiment bias in word representations. As it was discussed in **Section 5.2.2.1**, shallow NN word embeddings that have been trained on sentiment corporas. Capture semantically related words, with antidiametric sentiment polarity. Which is also a problem regarding sentiment analysis tasks. Since the sentiment information would be biased in vector representations, of sentiment words that appear in the same context.

# Chapter 6

# Conclusions and Future Work

## 6.1 Conclusions

In this work we provided detailed observations on sparse and dense text representations, regarding their performance on sentiment domains. In both of the experiments we used custom rule-based text preprocessing techniques in order to enrich the information representation and to counter OOV instances.

Regarding sparse VSMs, we performed cross-domain experiments in order to measure the information transfer from one sentiment domain to another, through TFIDF transformations. TFIDF is quite good at adapting and wrapping around different domains, since it is term-frequency model. Despite of being one of the oldest information retrieval techniques. We also showed how weights are interpreted inside linear SVM model in conjunction with TFIDF transformation algorithm. Technically exposing weaknesses of the TFIDF model, regarding proper weight assignment and text(n-gram) transformation.

In response to the research questions on sparse VSMs:

**RQ1** *What is the capacity of TFIDF algorithm in terms of cross-domain information transfer?*

**A1** *TFIDF retains the information quite well across different domains, when provided with enough data to formulate a balanced internal dictionary of representations*

**RQ1.1** *Does utilizing richer n-grams combinations always improve the TFIDF performance on sentiment domains?*

**A1.1** *FP analysis prediction with eli5 model showed that an arbitrary number of n-grams combination does not always reflect better on the performance. N-Grams representations are quite domain-dependent and should be used according to the domain's preferences*

Regarding dense continuous representations we performed two individual experiments with self-trained and pretrained word embeddings.

The purpose behind the self-trained embedding models, was to capture relations through shallow NN(Word2Vec, Doc2Vec) and statistical(GloVe) LMs, in sentiment domain Sentiment140. While at the same time, exploring different training algorithm concepts, whether they impact sentiment performance.

As for the pretrained versions of WE models, we performed classification tasks on sentiment domains, using word(Word2Vec, Glove) and contextual(FastText, U.S.E, Elmo) LMs.

In response to the research questions on continuous-dense VSMs:

**RQ2** *Does domain specificity reflects on better word embeddings representations, regarding sentiment classification tasks?*

**A2** *While sentiment-domain specificality plays important role in capturing more expressive sentiment representations. In contrast with generically trained WE models that were trained on random corporas. They also introduce anti-diametrical SO in word embedding representations, something that degrades performance in sentiment analysis tasks.*

**RQ3** *What is the performance difference between TFIDF, pretrained contextual and word LMs on sentiment domains?*

**A3** As the results showed, there are slight performance differences of the models, but not as significant as one would expect. Especially for deep NN architectures, like USE and stacked biLMs(Elmo). Low-performance gap didn't justify the models' perplexity, although they performed better than their predecessors (Glove, Word2Vec) in most of the cases. In terms of performance with sparse representations such as TFIDF. Surprisingly TFIDF outperformed any pretrained WE model. Regardless perplexity level or training principles. Although TFIDF limitations are quite noticeable, with difficult lexical expressions such as negations and sarcastic content).

Strong factor that degrades the performance of pretrained models on sentiment data, is the general training perspective of the pretrained models. Since models where trained on gigantic corporas and crawls, without any specific training or domain preferences. In order to be as general as possible for different downstream NLP tasks. Thus the resulted representations between words are quite general and not suited directly for sentiment analysis tasks.

## 6.2 Future Work

In the course of this work, we have presented the major aspects, but also weaknesses of sparse and continuous dense text representation. When it comes to sentiment analysis tasks.

While sparse models can capture term-frequency patents and wrap around any text domain. In more complex sentiment analysis tasks, such as sarcasm. Sparse frequency models quickly loose their higher ground, since they encapsulate far less information than needed in order to properly classify sarcasm. Beside sarcasm, same poor performance can be obtained with sentiment sentences that include complex linguistic expressions and negations.

While in dense representations, problems such as opposed sentiment polarity are frequent, at least with shallow NN models. Count-based statistical model such as Glove, encounter the same performance threshold when it comes to sentiment analysis.

More complex models, such as attention models or customized biLMs are really promising in the field of deep learning. Since these models have the capacity of extracting more sophisticated features on multiple level. Such as, different levels of attention(character /word/sentence) when it comes to attention models or more complex learning of different lexical features(biLms). Something that is quite beneficial for sentiment analysis.

This study showed that pre-trained deep learning models do not have that much impact on performance of sentiment analysis. Thus they should not be treated directly, but instead used in the capacity of transfer learning. Meaning that the pretrained models, have been trained on billions of tokens of data with a generic training routine. Hence, what should be done is to use the already generic pretrained models and fine-tune them based on task at hand. In this way, the pretrained "knowledge" of the models could be expanded with domain specific knowledge(for instance, sentiment) through a fine-tuning procedure.

There are many ways in order to perform fine-tune on certain deep learning models. One could freeze the pretrained model's early layers weights, or replace the last layer(softmax) according to specific downstream task. In both of the deep learning models(elmo, use), fine-tune is presented as only alternative of further improving the performance on selected tasks [42][13].

# Bibliography

[1] *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, Copenhagen, Denmark, Sept. 2017. Association for Computational Linguistics.

[2] A. Aue and M. Gamon. Customizing sentiment classifiers to new domains: A case study. In *Proceedings of recent advances in natural language processing (RANLP)*, volume 1, pages 2–1. Citeseer, 2005.

[3] D. BAHR and E. PASSERINI. Statistical mechanics of opinion formation and collective behavior: Micro-sociology. *The Journal of mathematical sociology*, 23(1):1–27, 1998.

[4] V. Balakrishnan and E. Lloyd-Yemoh. Stemming and lemmatization: a comparison of retrieval performances. 2014.

[5] R. Bar-Haim, E. Dinur, R. Feldman, M. Fresko, and G. Goldstein. Identifying and following expert investors in stock microblogs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1310–1319. Association for Computational Linguistics, 2011.

[6] L. Becker, G. Erhart, D. Skiba, and V. Matula. Avaya: Sentiment analysis on twitter with self-training and polarity lexicon expansion. In *Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, volume 2, pages 333–340, 2013.

[7] Y. Bengio and S. Bengio. Modeling high-dimensional discrete data with multi-layer neural networks. In *Advances in Neural Information Processing Systems*, pages 400–406, 2000.

[8] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[9] A. Bermingham and A. F. Smeaton. Classifying sentiment in microblogs: is brevity an advantage? In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1833–1836. ACM, 2010.

[10] A. Bhoi and S. Joshi. Various approaches to aspect-based sentiment analysis. *arXiv preprint arXiv:1805.01984*, 2018.

[11] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[12] E. Çano and M. Morisio. Word embeddings for sentiment analysis: A comprehensive empirical survey. *arXiv preprint arXiv:1902.00753*, 2019.

[13] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, et al. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018.

[14] Y. Chen, B. Perozzi, R. Al-Rfou, and S. Skiena. The expressive power of word embeddings. *arXiv preprint arXiv:1301.3226*, 2013.

[15] W. Daelemans, H. Groenewald, and G. Van Huyssteen. Prototype-based active learning for lemmatization. *International Conference Recent Advances in Natural Language Processing, RANLP*, 01 2009.

[16] A. D. Dave and N. P. Desai. A comprehensive study of classification techniques for sarcasm detection on textual data. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 1985–1991. IEEE, 2016.

[17] D. Davidov, O. Tsur, and A. Rappoport. Enhanced sentiment learning using twitter hashtags and smileys. In *Proceedings of the 23rd international conference on computational linguistics: posters*, pages 241–249. Association for Computational Linguistics, 2010.

[18] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[19] M. Denil, D. Matheson, and N. De Freitas. Narrowing the gap: Random forests in theory and in practice. In *International conference on machine learning*, pages 665–673, 2014.

[20] N. Farra, E. Challita, R. A. Assi, and H. Hajj. Sentence-level and document-level sentiment mining for arabic texts. In *2010 IEEE international conference on data mining workshops*, pages 1114–1119. IEEE, 2010.

[21] A. Go, R. Bhayani, and L. Huang. Sentiment140. *Site Functionality, 2013c. URL http://help. sentiment140. com/site-functionality. Abruf am*, 20, 2016.

[22] B. Han and T. Baldwin. Lexical normalisation of short text messages: Makn sens a #twitter. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 368–378, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[23] G. E. Hinton et al. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, volume 1, page 12. Amherst, MA, 1986.

[24] M. Iyyer, V. Manjunatha, J. Boyd-Graber, and H. Daumé III. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1681–1691, 2015.

[25] F. H. Khan, U. Qamar, and S. Bashir. A semi-supervised approach to sentiment analysis using revised sentiment strength based on sentiwordnet. *Knowledge and information Systems*, 51(3):851–872, 2017.

[26] M. Khodak, N. Saunshi, and K. Vodrahalli. A large self-annotated corpus for sarcasm. *arXiv preprint arXiv:1704.05579*, 2017.

[27] A. M. Kibriya, E. Frank, B. Pfahringer, and G. Holmes. Multinomial naive bayes for text categorization revisited. In *Australasian Joint Conference on Artificial Intelligence*, pages 488–499. Springer, 2004.

[28] R. G. Krishnan, D. Liang, and M. Hoffman. On the challenges of learning with inference networks on sparse, high-dimensional data. *arXiv preprint arXiv:1710.06085*, 2017.

[29] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196, 2014.

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

[31] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pages 142–150. Association for Computational Linguistics, 2011.

[32] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[33] O. Melamud, D. McClosky, S. Patwardhan, and M. Bansal. The role of context types and dimensionality in learning word embeddings. *arXiv preprint arXiv:1601.00893*, 2016.

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

[35] T. Mikolov, W.-t. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, 2013.

[36] T. Mullen and N. Collier. Sentiment analysis using support vector machines with diverse information sources. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, 2004.

[37] P. Nakov, A. Ritter, S. Rosenthal, F. Sebastiani, and V. Stoyanov. Semeval-2016 task 4: Sentiment analysis in twitter. In *Proceedings of the 10th international workshop on semantic evaluation (semeval-2016)*, pages 1–18, 2016.

[38] T. Nasukawa and J. Yi. Sentiment analysis: Capturing favorability using natural language processing. In *Proceedings of the 2nd international conference on Knowledge capture*, pages 70–77. ACM, 2003.

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

[40] C.-Y. J. Peng, K. L. Lee, and G. M. Ingersoll. An introduction to logistic regression analysis and reporting. *The journal of educational research*, 96(1):3–14, 2002.

[41] J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[42] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

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

[44] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[45] C. Saedi, A. Branco, J. A. Rodrigues, and J. Silva. Wordnet embeddings. In *Proceedings of The Third Workshop on Representation Learning for NLP*, pages 122–131, 2018.

[46] M. Sahlgren. *The Word-Space Model: Using distributional analysis to represent syntagmatic and paradigmatic relations between words in high-dimensional vector spaces.* PhD thesis, 2006.

[47] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. Technical report, Cornell University, 1987.

[48] G. Salton et al. The smart retrieval system, 1971.

[49] G. Salton, A. Wong, and C.-S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[50] R. E. Schapire, Y. Freund, et al. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780, 1999.

[51] K. Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[52] B. Tang, M. Shepherd, E. Milios, and M. I. Heywood. Comparing and combining dimension reduction techniques for efficient text clustering. In *Proceeding of SIAM International Workshop on Feature Selection for Data Mining*, pages 17–26. Citeseer, 2005.

[53] D. Tang, F. Wei, N. Yang, M. Zhou, T. Liu, and B. Qin. Learning sentiment-specific word embedding for twitter sentiment classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1555–1565, 2014.

[54] A. Tripathy, A. Agrawal, and S. K. Rath. Classification of sentiment reviews using n-gram machine learning approach. *Expert Systems with Applications*, 57:117–126, 2016.

[55] A. Tumasjan, T. O. Sprenger, P. G. Sandner, and I. M. Welpe. Predicting elections with twitter: What 140 characters reveal about political sentiment. In *Fourth international AAAI conference on weblogs and social media*, 2010.

[56] P. D. Turney and P. Pantel. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37:141–188, 2010.

[57] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[58] L.-C. Yu, J. Wang, K. R. Lai, and X. Zhang. Refining word embeddings for sentiment analysis. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 534–539, 2017.