Enhancing Lexical Sentiment Analysis using LASSO Style Regularization

By
Jeremy Brett Blanchard

A THESIS

submitted to

Brock University

in partial fulfillment of
the requirements for the
degree of

Master of Science in Mathematics and Statistics

Presented 18 January, 2023
Commencement September 2020

AN ABSTRACT OF THE THESIS OF

Jeremy Brett Blanchard for the degree of Master of Science in Mathematics and Statistics presented on 18 January, 2023. Title: Enhancing Lexical Sentiment Analysis using LASSO Style Regularization

Abstract approved:

Dr. William Marshall

In the current information age where expressing one's opinions online requires but a few button presses, there is great interest in analyzing and predicting such emotional expression. Sentiment analysis is described as the study of how to quantify and predict such emotional expression by applying various analytical methods. This realm of study can broadly be separated into two domains: those which quantify sentiment using sets of features determined by humans, and approaches that utilize machine learning. An issue with the later approaches being that the features which describe sentiment within text are challenging to interpret. By combining VADER which is short for Valence Aware Dictionary for sEntiment Reasoning; a lexicon model with machine learning tools (simulated annealing) and k-fold cross validation we can improve the performance of VADER within and across context. To validate this modified VADER algorithm we contribute to the literature of sentiment analysis by sharing a dataset sourced from Steam; an online video game platform. The benefits of using Steam for training purposes is that it contains several unique properties from both social media and online web retailers such as Amazon. The results obtained from applying this modified VADER algorithm indicate that parameters need to be re-trained for each dataset/context. Furthermore that using statistical learning tools to estimate these parameters improves the performance of VADER within and across context. As an addendum we provide a general overview of the current state of sentiment analysis and apply BERT a Transformer-based neural network model to the collected Steam dataset. These results were then compared to both base VADER and modified VADER.

Key Words: Machine Learning, Sentiment Analysis, VADER, LASSO, Simulated Annealing

Corresponding e-mail address: jb13vo@brocku.ca

Contents

1 Introduction

1.1 What is sentiment analysis

Sentiment analysis is the process of collecting and analyzing the thoughts and opinions expressed by individuals. The exponential growth of sentiment analysis as a field of study witnessed in recent years can be attributed to the global adoption of social media and the development of one's digital identity online. The growth of social networking sites such as Facebook and Instagram have produced vast quantities of data [10]. This highly opinionated data has allowed researchers to test and validate novel sentiment analysis in record time. The definition of a digital identity is any demographic information an individual posts or shares online, this can include posting a comment (either through voice or by text), sharing an image of one's self or even video on social media. The goal of sentiment analysis is to apply various analytical methods to such demographic data so that we can predict the opinion communicated [6]. Sentiment analysis can broadly be categorized into grammar and machine learning based approaches: the former includes the use of lexicon and grammatical rules to estimate the polarity associated with some given body of text; the latter consists of supervised, semi-supervised and unsupervised machine learning techniques. The current state of art focuses on the development of deep learning sentiment analysis models.

1.2 Introducing Steam

Steam is the dominant platform for the purchase of video games on Windows, Linux and Macintosh operating systems [5]. In addition to being an online retailer of video games, this Steam service also has its own social network where users can interact with each other through various means, such as private and public text based communication, playing video games together and sharing modifications to video games on forums. This newly available dataset from Steam is currently the only one from a hybrid-social media and retail platform and is very unique in the body of sentiment analysis literature as opposed to more standard data sources such as IMDB, Amazon and Twitter. Steam allows its users to write reviews on games that they have purchased. In these reviews users can recommend (positive) or not-recommend (negative) a game and are required to publish some text alongside their rating. The allowed text in Steam reviews can be any alpha numerical character or emojis and emoticons. One way that Steam validates the credibility of each review is by allowing users to independently react to video game review. Currently a user can label a review as either helpful, not-helpful or funny. For example a review that discusses a video game's game-play mechanics, visuals or sound are more likely to be rated by other users as being helpful. We leverage this user rating system to filter out sub-standard reviews that could impact the quality of the results. In addition other pre-processing methods have been employed such as automated word correction.

Steam contribution to literature We contribute to the literature by introducing a novel dataset of text based reviews collected from the video game online store Steam. This dataset provides an additional source of testing and validation for proposed methods. The reviews available are curated by Steam itself using a combination of both manual and algorithmic methods. The resultant reviews are thus highly reliable and consistent even at high volumes which is important for machine learning applications that are sensitive to noise. In terms of unique properties the morphology of reviews on the Steam platform has been heavily influenced by PC gaming. In regards to this paper we have leveraged this "gamer slang" to compare how well VADER generalizes to other datasets.

1.3 Introduction of the VADER model and outline of thesis

VADER is a sentiment analysis rule/lexicon based algorithm introduced by Huto et al. In the original paper they describe a "gold" grammatical rule set that works favorably when applied to social media. We apply VADER to a dataset of Steam reviews and find that the methods used to estimate the parameters which control the grammatical rules do not generalize favorably across context. Given that VADER does not generalize can we leverage statistical learning techniques to make VADER robust across context. To this aim we have employed simulated annealing and k-fold cross validation to improve the estimation of these grammatical rules across context. We aim to submit this chapter as a registered report. In chapter 3 we provide a review of neural network methods commonly used for sentiment analysis. We discuss the current state of deep learning based sentiment analysis and apply a fundamental model to the same Steam review data as mentioned previously; the deep learning model applied was a Bidirectional Encoder Representations from Transformers (BERT). The reason we used BERT was because many other methods are derived from its architecture.

2 Enhancing Lexical Sentiment Analysis using Statistical Learning Methods

2.1 Introduction

Sentiment analysis is the application of natural language processing via algorithms. The main purpose being to analyze text, spoken voice, or other methods of communication to identify and quantify subjective data. One of the common uses for sentiment analysis is to apply it to text reviews of products from websites such as Amazon [38] to understand the degree of positivity and negativity within each review. However, language is not typically so easy to classify and does heavily depend on the context [29] itself whether it be Twitter posts [10], movie reviews [16] from IMDB or business reviews from Yelp [20]. The field of sentiment analysis has existed for several decades but major advances have occurred only relatively recently in large part due to the advent of the internet in addition to exponential improvements in computing performance. The internet gave researchers a cost and time effective method of accessing massive amounts of opinionated text data. Leveraging these new technological achievements, researchers were more easily able to test, validate and improve natural language processing models.

Related works Methods of analyzing the sentiment within text can be broadly separated into two groups; those based on grammatical rules and others that are data-driven (e.g. machine learning). Typically, machine learning methods have a higher degree of accuracy but come at the cost of reduced interpretability, which makes it akin to a "black box" of knowledge. In lexical methods of analysis, the rules are defined and it is, therefore, easier to apply towards the goal of gaining a clearer understanding of where sentiment exists within text.

Turney defined a grammatical ruleset based on the semantic orientation of words [1]. Using these findings as a basis, Mellville et al developed a model that incorporated a lexicon into the work done by Turney and applied it to the same dataset where they found that including a lexicon into a grammar based sentiment model improved the overall accuracy substantially [28]. Mckeown et al published a paper that discussed the development of a dictionary of words using the 1987 Wall Street Journal and constructed a list of positive and negative adjectives where they demonstrated that the emotional expression using adjectives have some dependency on orientation. Various other ideas for creating a dictionary of words have been proposed, these include both manual and automatic. Islam et al proposed an approach to classify online news. In their paper, sentiment analysis was done at the sentence level by using a dictionary of predefined positive and negative words that were then added together to obtain an estimate of sentiment within text. Reis and Olmo proposed a methodology to explore the relationship between sentiment polarity and news articles [31]. An example of a lexicon-based approach that involves a massive human sentiment labeling of words is described by Dodds et al [30]. This sentiment dictionary contains approximately 5 million human

sentiment assessments of 10,000 common words, each in 10 languages and labeled 50 times. Another well-known sentiment lexicon is SentiWordNet [3], constructed by applying semi-supervised machine learning to a massive bank of words. The total number of words in sentiWordNet is approximately 100,000 words, but is limited to English only. VADER which stands for Valence Aware Dictionary and sEntiment Reasoner is a lexical and grammar-based sentiment analysis algorithm that has been trained using a dictionary of experimentally derived sentiment values using Amazon Mechanical Turk (AMT). AMT is a micro labour website where workers perform minor tasks in exchange for a small amount of money. Applying a wisdom of the crowd approach [34] along with AMT. Leveraging these methods the publishers of VADER then surveyed each parameter and lexical feature commonly used in social media and micro blogs. This method of sampling from Tweets does explain why VADER performs better on social media data like Twitter than product reviews such as Amazon and IMDB.

According to the literature, support vector machines were one of the first methods to be used for sentiment analysis. The motivation being that it is easier to represent a set of grammatical rules as vectors to capture more of the underlying semantic structure of a language [28]. For a few years, support vector machines outperformed all previous sentiment and classification models. At the time of writing there now exists several other machine learning algorithms for sentiment analysis such as bag of words [35], naïve bayes [19], maximum entropy and decision trees [15]. Pang et al has compared various machine learning methods including the ones stated; they found that as the amount of text in a training sample increases, there exists a tendency for over-fitting to the context of the data [25]. Recurrent neural networks are an appropriate fit for these types of problems since language is sequential in nature and thus have consistently outperformed other machine learning and lexical approaches to sentiment analysis tasks. One benefit of neural networks when applied to language processing tasks is that little external linguistic knowledge is required to obtain optimal results. A critical step with language analysis especially when in regards to neural networks is that text must be transformed into some input (typically called encoding) either through by converting to a numerical dictionary, mapping words to a vector space or as a parsed syntactic word tree [25] and [32]. As conjectured by Li et al they suggest that sequence-based models are sufficient to for capturing the sub-dimensional semantics for sentiment classification [21]. The introduction of BERT [7] and its immediate and widely adoption as a framework has led current deep learning models to be, for the most part, iterations of BERT. Examples of two models that leverage BERT's framework and have achieved state of the art performance on natural language benchmarks such as GLUE [37] and RACE [18] are RoBERTa [22] and XL-Net [40]. Currently there are two deep learning models that achieve state of the art performance on natural language tasks DeBERTa [13] (BERT) and ERNIE [42] (GPT-3) and in regards to the GLUE benchmark have obtained identical overall scores of 91.1. All of this to say that sequential based algorithms may indeed be the future of natural language comprehension.

The VADER approach is a lexicographical method that reveals the grammatical rules

that underlie sentiment, while being able to classify Twitter data with high accuracy. However, the method is not robust, and thus the grammatical rules may not apply in other contexts. It is the hope that by combining a lexical method; in our case this being the VADER algorithm [16] with some statistical learning tools we can obtain a model that is more robust and that can hopefully be used to better estimate the grammatical rules used and to learn about sentiment across context. To evaluate our proposed method, we will compare the performance of the original VADER algorithm to our modified approach on two datasets: a novel dataset of videogame reviews from Steam, and an existing dataset from IMDB [2]. We will be testing two hypotheses: (1) that the VADER modified algorithm outperforms the original VADER algorithm at classifying video game reviews; and (2) that our modified algorithm will outperform VADER on an independent dataset for which neither were trained. The remainder of this paper is organized as follows. In section 2 we review VADER in detail and our extension of the method using statistical learning. Initial experimental results of our pilot Steam data are presented in section 3. Discussion and future work is presented in section 4 and section 5 concludes the paper.

2.2 Methods

At the time of writing there does not exist any curated Steam dataset that can be used for Steam game reviews, this is due to video gaming as a medium being a relatively "new" source of data as opposed to movie or product reviews which have been the standard since sentiment analysis' development as a field of study. For this publication, it was decided to use video game reviews since there exists a large online collection that is openly available on the Steam store. Steam was launched in September of 2003 and was initially used by Valve (the company that owns Steam) to automatically update and distribute video games they have developed such as Portal, Counter-Strike and the Half-Life series. Today, Steam is the dominant digital distribution platform for PC gaming with a market share of 75% [33]. In general, the user base of Steam is relatively young ranging between 10 to 30 and so has its own unique sub-culture with a predilection towards the use of memes, emojis, and sarcasm to convey sentiment. The use of this data for sentiment analysis is novel because it contains linguistic properties that exist in both social media datasets (emojis and colloquialisms) and other more conventional sources of review data such as IMDB (popular culture references) [39]. As with movie reviews, there is no need to independently label the data since users summarize their overall sentiment with a machine interpret-able rating indicator; for Steam reviews, the rating indicator is a binary "thumbs-up" (positive) and "thumbs-down" (negative). Since the distribution of movie reviews on a 5 point scale is not uniformly distributed; IMDB like Steam is usually positioned as a binary classification problem (positive or negative) without sacrificing granularity [24]. Another benefit is that the original VADER paper has already applied the IMDB movie reviews (F1 score of 0.61) [16]. The IMDB dataset was collected by Maas et al [24] that provides a set of 50000 movie reviews from IMDB. The dataset itself contains an even number of both positive and negative reviews. Each review within this dataset was annotated as either positive (when the review has a score ≥ 7 out of 10) and negative ≤ 4 out of 10, where neutral reviews were omitted.

As an exploration of our idea, we collected a pilot dataset of 50000 video game reviews from the Steam web store. These reviews are randomly sampled between each video game product. The distribution is equally distributed both with respect to video game products and the ratio of positive (25'000) and negative reviews (25'000). As an initial step we applied our pilot Steam dataset to the base VADER algorithm. To better compare our results to the ones presented in the original VADER publication, we assessed the performance of the model using binary class classification metrics of precision, recall and F1 score. We define the performance metrics used to evaluate each model as follows: precision is a measure of how many of the positive predictions made are true. Recall is the number of positive cases the classifier correctly predicted vs all positive cases. The F1 score being the harmonic mean of precision and recall.

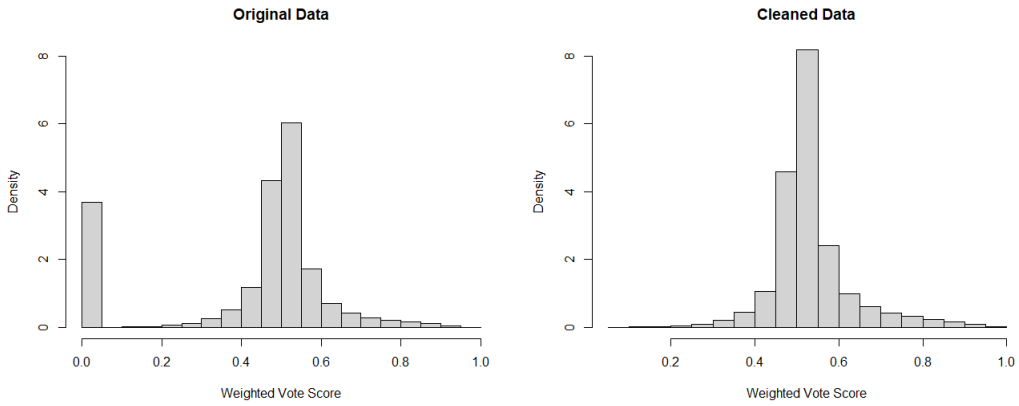

Figure 1: Weighted score before and after pre-processing

2.3 Pre-processing

Natural language pre-processing is a critical aspect of sentiment analysis. User-generated content is typically unstructured and certain steps must be followed to transform it into a format that can be understood by a computer. The main steps performed in normalizing our text are summarized in figure 2.

Spam filtering A standard but naive method of filtering spam reviews is by the removal of reviews that contain less than a certain number of words (Less than 2). A complication of this approach is that Steam shares many similarities with Twitter when it comes to an emphasis on brevity of word length and so another method will be included as well one built by Steam itself. As explained in the Steam section, each review is accompanied with a usefulness (spam) score that ranges from 0 to 1 where the closer this score approaches 0 the higher the odds a review is considered spam and vice versa. In order to better normalize the distribution of our review data with respect to spam scores we filtered out any reviews that had a review score of less than or equal to 0.05. As shown in figure 1 where we see the effect this step has on the distribution of weighted vote scores in the Steam review data.

Colloquial and incorrectly spelled Words replacement In non-formal texts it is common for Steam reviews to contain poorly spelled and exaggerated expressions to emphasize sentiment, this type of miss-classification is a generally known issue with lexical methods since the dictionary of words is static and does not adapt dynamically to non standardized contexts. For example, the colloquial word "Happyy" would be converted to the formal word "Happy". One thing to note is that only characters within a word are impacted and not their degree of capitalization since in the VADER algorithm a critical step is the measure of capitalization within a word. To accomplish this task a spelling corrector script was written that identifies incorrectly spelled word within a

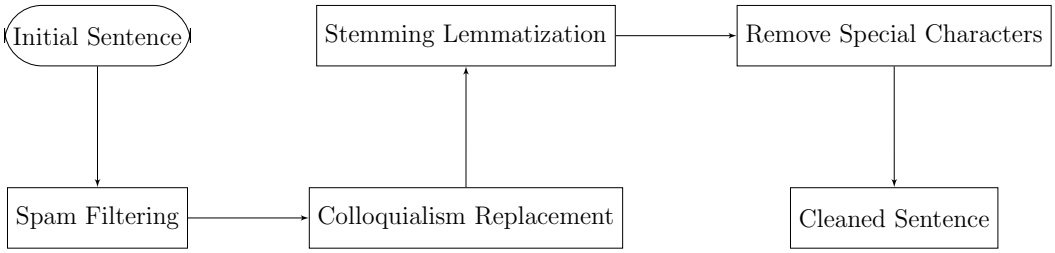

Figure 2: Pre-processing steps

Steam review by comparing them to a dictionary of correctly spelled words available in the NLTK library [23]. This list of identified misspelled words is then looped through and we calculate the Levenshtein distance. The Levenshtein distance is defined as the minimum number of single character edits to change one word into another) [11]. This distance measure is equivalent to the L_1 norm but where word characters are represented as a vector. The objective of taking the L_1 norm is to select a new word vector from the total word space where the computed distance is minimized.

Lemmatization Lemmatization is the process of simplifying language (reducing dimensionality) by grouping together the various inflected forms of a word (tenses) so that they can be analyzed as a single term. This is accomplished by using a combination of algorithms and lookup-tables. Our goal with including lemmatization in our pre-processing methodology is to reduce the complexity of our dataset by transforming related word forms (e.g. "performed" and "performing" are both inflections of "perform") to a common base. Lemmatization was performed on the Steam dataset by applying the Word-Net-Lemmatizer function found in the NLTK library [23]. Lemmatization is similar to stemming but it brings context to the words by including other subtle grammatical features such as parts of speech tagging. Lemmatization is preferred over stemming since the former method provides a more precise approach to reducing natural language dimensionality.

Remove Links and Special Characters Links and non-alphanumeric characters are removed using regular expressions. If a string within the text matches the form "http://" or "https://", it is removed. Similarly, any non-alpha-numeric characters including those found within a word were removed.

2.4 VADER

In this section, the intricacies of the VADER algorithm will be defined in the context of grammar and will then be tied to how this algorithm calculates the sentiment score of a provided text. VADER can be summarized as follows. A lexicon of known words will assign an initial sentiment score to each word given some initial input of text and then

these initialized sentiments are modified using the following sequence of grammatical rules.

Lexicon from TURK The VADER lexicon sentiment scores were obtained using Amazon Mechanical Turk (AMT), a crowd-sourcing platform where workers perform minor tasks in exchange for a small amount of money. The publishers of VADER [16] constructed a list of both words and emoticons by compiling existing sentiment word banks. (LIWC [17], ANEW [4] and HU-KDD [15]). Each of these textual features were then rated from (-4) "Extremely Negative" to (4) "Extremely Positive" by 10 independent AMT workers. Each sentiment word was calculated by taking the mean of these 10 reviews keeping only lexical features that had a non-zero mean rating, and whose standard deviation was less than 2.5.

Idioms An idiom is a phrase or group of words with a metaphorical (sentimental) meaning which is not equal to the sum of sentiment values of their individual words. Idioms can be considered as the building blocks of languages and how the combination of words can completely change the sentiment value of a sentence. Idioms themselves are found more than just the English language but in the context of VADER we will only examine a few idioms in the English language. A set of idioms was selected by comparing the difference between the sum total of words versus the actual rating from AMT keeping the later as the correct sentiment value. When an idiom is identified the calculated sentiment of based on the component words that makeup an idiom are replaced by the sentiment defined in the Idiom dictionary.

Emoji handling Provided some symbolic representation of each emoji found within a dictionary VADER maps each of them to their literal equivalent. So, if the text fed into VADER was presented with the emoji :D then it would be replaced by the text "happy face".

Grammatical rules In the original VADER publication [16] determined and validated a set of grammatical rules by selecting a sample of the most 400 positive and 400 negative social media tweets from an initial set of 10000 random public tweets. Two human experts scrutinized these 800 tweets and independently scored their sentiment intensity on the same scale as used for the VADER lexicon. Using qualitative analysis techniques such as grounded theory, they identified 4 grammatical rules (Punctuation, Capitalization, Booster and Conjunctions). Following these qualitative results, a controlled experiment was designed to evaluate the impact of grammatical rules on sentiment intensity. This was accomplished by selecting 30 baseline tweets and testing (using 30 independent AMT workers) how the insertion or deletion of grammatical features impact sentiment. A summary of these four grammatical rules and their estimated values are found in table 1.

Punctuation VADER only considers punctuation marks such as "!" and "?" as sentimentally significant, where these symbols increase or decrease the overall magnitude of a phrase without modifying the overall meaning. For example, if the following sentences are compared with respect to VADER. "VADER is good" and "VADER is good!"; the algorithm would respond by increasing the total sentiment of the phrase. This amount is estimated by taking the total number of exclamation (up to a maximum of 4) and question marks (up to a maximum of 4) found within the sentence; multiplying each of these totals by their associated constant (see figure 1) and then adding the summation of these two punctuation groupings to the sentiment of the phrase.

Negation The following list of words "least, without, doubt, never, so, without, doubt and never-so, never -this" are negation words. Similar to punctuation the VADER algorithm searches for the existence of negation words and if one is detected then the overall sentiment score of the sentence will be multiplied by the constant -1.25. In plain English, the valence score of a sentence is flipped and increased by 25%. There exists a separate method for the existence of "No" including "or and nor" where if it is found in the three prior word locations then the overall sentiment of the phrase is multiplied by the constant -0.74. However, we did find that VADER struggles with the identification of complex negative expression such as in the case of a double negative.

Boosting and dampening words Boosting and dampening words are a list of special adjectives and adverbs (e.g absolutely, uber, less and little) which impact the degree of sentiment intensity by increasing the absolute sentiment score of the following three words as VADER loops through each word in a sentence. One important detail to note is that boosting words increase the absolute value of sentiment of the following word so represented in table 1 as two separate parameters (+0.293 and -0.293) but this does not imply that these two values were estimated separately as according to the original paper this constant was calculated based on the average absolute effect a boosting word has on sentiment in text. By applying this definition of a boosting word, the developers of VADER were able to obtain a dictionary of Boosting words by applying the Pattern.en module from the NLTK library [23] to the set of 800 most positive and negative tweets.

Conjunction A conjunction is a word that joins together other words or groups of words. VADER applies these grammatical rules to the overall sentiment of a sentence if and only if the word "but" is found. A critical aspect of this grammatical rule is that it is location-dependent. As an example of how this works consider the set of modified words after VADER has applied all other sentiment-based grammatical rules. Given this set of modified words, VADER iterates through the list, and if "but" is found before the current sentiment word then that word's sentiment is multiplied by 0.5. Conversely, if "but" is found after the current sentiment word then that word's sentiment is multiplied by 1.5.

VADER algorithm sentiment classification As represented in equation 1 we define y as a binary categorical variable with two possible values; +1 for positive sentiment and -1 for negative sentiment. Following this definition we represent $f(x)\epsilon\{-1, ..., 1\}$ as the output from the VADER algorithm 3 where the input x is some block of text.

$$y = \begin{cases} +1 & \text{if } f(x) > 0, \\ -1 & \text{if } f(x) < 0. \end{cases} \tag{1}$$

2.5 Modified VADER algorithm

We propose a modified VADER algorithm that leverages simulated annealing and k-fold cross validation to estimate the grammatical results estimated in the original VADER publication [16]. We re-framed each of these grammatical rules as a parameter to be estimated via machine learning. Table 1 shows a quick summary of each parameter and their associated grammatical rule within the VADER algorithm.

Furthermore, we state that the VADER algorithm can be re-framed in the context of statistical learning. In the original implementation of VADER, several static parameters control the weighting of each heuristic grammatical rule. Since these weights are constant values they were not defined in the original VADER function. These parameters were estimated using a wisdom of the crowd approach via the Amazon Turk framework. Leveraging supervised machine learning techniques along with video game reviews from Steam we hope to find estimates of the parameters that generalize favorably across contexts. The proposed modification to the original VADER algorithm $f(x)$ is represented below where we have re-framed these constant grammatical weights as an additional set of parameters β to be estimated using simulated annealing.

$$\hat{y} = \begin{cases} +1 & \text{if } f(X;\beta) > 0, \\ -1 & \text{if } f(X;\beta) < 0. \end{cases} \tag{2}$$

Recasting VADER as a statistical learning problem we aim to estimate a set of parameters β which have been trained on cleaned user review data from the Steam webstore, rather than the approach used in the original Vader paper [16] which employed crowd-sourcing techniques facilitated by Amazon Mechanical Turk. Once we have estimated our set of parameters β and have determined the performance of our model on some Steam training data we compute its performance using some loss function $L(\beta)$. To mitigate the risk of over-fitting we subtract a regularization term λ multiplied by the sum of the estimated $|\beta|$s from the computed loss to obtain the penalized performance metric $\hat{L}(\beta)$. Regularization wise if $\lambda = 0$ the method returns the unconstrained optimal value for β and as $\lambda \to \infty$ all β's go to zero. This regularization step is applied at the end of each iteration during simulated annealing.

Input : A sentence where each word contains some initial sentiment
Output: The output is the sum of the sentiment scores for each word in the list

for Each Word_i within a Sentence **do**
  if Word_i in Sentiment Dictionary **then**
    if is word no(Word_i) AND Not the end of Sentence **then**
      sentiment = 0
    end
    if is No(Word_{i-1}) AND Word_i.sentiment > 0 **then**
      Word_i.sentiment = B
    end
    if word is capitals(Word_i) AND sentence is not capitals **then**
      if Word_i.sentiment > 0 **then**
        Word_i.sentiment + = A
      else
        Word_i.sentiment − = A
      end
    end
    if Is a Known Booster Word $\text{Word}_{i-1..i-3}$ **then**
      scalar_i = H
      if Word_i < 0 **then**
        scalar_i *= -1
      end
      if word is capitals(Word_i) AND sentence is not capitals **then**
        if Word_i.sentiment> 0 **then**
          scalar_i += A
        else
          scalar_i -= A
        end
      end
      Word_i.sentiment + = scalar_{i-1} + scalar_{i-2} * 0.95 + scalar_{i-3} * 0.90
    end
    if negated word($\text{Word}_{i-1..i-3}$) **then**
      Word_i.sentiment + = scalar * B
    end
    if idiom sentiment($Word_{i-1..i-3}$) **then**
      Word_i.sentiment = Idiom Sentiment
    end
    if word is before But(Word_i) **then**
      Word_i.sentiment ∗ = F
    end
    if word is after But(Word_i) **then**
      Word_i.sentiment ∗ = G
    else
    end
  else
    Word_i.sentiment=0
  end
end
Sentiment Score=$\sum_{n=1}^{\infty}$ Word_i.sentiment
return Sentiment Score

Figure 3: VADER algorithm

Parameter	Grammar Rule Operation
Negation	If Negation exists multiply current sentiment
Negation (Special Case)	If Negation exists and is preceded by "never (so, this)" or (without doubt) multiply current sentiment
Capitalization	If current word is capitalized then add a constant
Booster Word Increase	If the following word is positive then increase
Booster Word Decrease	If the following word is negative then decrease
Exclamation Punctuation	If word contains exclamation point then add to current sentiment
Question Punctuation Low	If word contains question mark then add to current sentiment
Question Punctuation High	If word contains more than 3 question marks then add a different constant
Conjunction Decrease	If the preceding word is a conjunction then multiply the current sentiment
Conjunction Increase	If the following word is a conjunction then multiply the current sentiment

Table 1: Names of default parameters mapped to definitions

$$\hat{L}(\beta) = L(\beta) - \lambda \sum_{j=1}^{10} |\beta_j| \tag{3}$$

Using a data-driven approach along with regularization techniques we define each of the stated parameters as a set of parameters β to be estimated. To derive this newfound set of parameters β we will perform simulated annealing (see section 2.5 for further details) fit to the output of our VADER perceptron. Furthermore, to minimize the odds of parameters being over-fit to the training data we will perform penalized regularization methods similar to what is done in LASSO. The hyper-parameter λ will be estimated using k-fold cross-validation where we split our training data into folds to determine the λ value that maximizes the validation accuracy.

Simulated annealing Simulated annealing is a search optimization algorithm inspired from the mechanical process of annealing [12]. Annealing from physics is defined as the process of heating a solid up to some temperature which is then cooled at an extremely slow rate until it achieves its most stable (lowest energy) state. In effect what annealing achieves from a physical perspective is the determination of the global optima of some object. Taking this concept of annealing we can see that it can be modeled mathematically. In this section we will discuss our understanding of simulated annealing and what specific methods were used to search for optimal parameters for VADER.

Regardless of the type of simulated annealing algorithm applied the main concept is as follows. At the start of each iteration the algorithm randomly generates a new set of parameters. The performance of the current and the newly spawned solutions are computed. If the new solution demonstrates improvement it is always accepted, while a fraction of inferior solutions are accepted in the hope of escaping local optima in search of global optima. The probability of accepting an inferior solution depends on a temperature parameter, which is typically non-increasing with each iteration of the algorithm. The main benefit of incorporating simulated annealing into some optimization problem is that it provides a simple and effective method for an algorithm to escape the local

optima without resorting to brute force methods such as in the case of grid search.

We will now define the specific features of a simulated annealing algorithm and how it relates to solving discrete optimization problems such as in the case of optimizing parameters used in the VADER sentiment analysis algorithm.

Definition of simulated annealing algorithm The basic elements of a simulated annealing algorithm are as follows.

1. A finite set S of possible parameters β
2. A real value cost function $f(x)$ which is defined from the set of parameters S.
3. A non-increasing function $T(k)$, where k is the number of iterations.
4. For every iteration k , there exists a unique collection of probabilities that describes the likelihood of accepting a particular solution within the set S.

Let β be the current set of parameters from S we randomly generate a proposed set new of parameters $\hat{\beta}$. We will demonstrate how simulated annealing updates β into $\hat{\beta}$ based on the probability of accepting a worse solution within this local neighborhood as follows. To begin we will permute our initial set of βs to obtain $\hat{\beta}$.

$$\hat{\beta} = \beta + R - 0.5 \tag{4}$$

The set of estimated $\hat{\beta}$ must be between -2.0 and 2.0 and R is a random number that follows a uniform distribution $R \sim U(0, 1)$. Using our objective function $f(x)$ we compare the performance of both sets of parameters by taking the difference between the two where if $\Delta f > 0$ we return the probability of accepting the new worse performing solution $\hat{\beta}$. In the other two cases where $\hat{\beta}$ performs either better or equivalently ($\Delta f \leq 0$) to the current set of β the probability of accepting the new solution is 100%.

$$P(\text{accept a new solution}) = \begin{cases} \exp\left(\frac{f(\hat{\beta}) - f(\beta)}{T(k)}\right) & \text{if } \Delta f > 0, \\ 1 & \text{if } \Delta f \leq 0. \end{cases} \tag{5}$$

We define $T(k)$ as the remaining thermal energy within the system at iteration k. Given that d is limited to being a positive constant we see that as the number of iterations k increases the probability that a worse solution will be accepted decreases. Thus the cooling schedule stated in equation 6 is the mechanism used to control how quickly the SA algorithm converges.

$$T(k) = \frac{d}{\ln k} \tag{6}$$

By iterating these steps a multitude of times we aim to obtain a new set of parameters $\hat{\beta}$ which minimizes the loss $\hat{L}(\hat{\beta})$ of our modified VADER model on the training and validation data from Steam reviews for a series of folds $k = 5$.

Statistical analysis We will then compare the performance of the modified algorithm with the original VADER parameters on two datasets - the Steam dataset and the IMDB dataset. The performance is primarily evaluated using accuracy. We also present precision, recall, etc. To evaluate the performance of our modified VADER algorithm we will use a 5X2 cross validation (CV) combined F test which is a procedure for comparing the performance of two classification models. The 5X2 CV combined F test will be used to evaluate our two classification models (Base Vader and our Modified VADER) is positioned by the author Alpaydin [9] as a refinement of Dietterichs (5X2 CV student's t-test) [8] approach to estimating two classifiers. The main benefit of the former approach as opposed to Dietterichs is that there is a lower risk of type one error and higher levels of statistical power. The general method so defined by both authors are explained in the following paragraph.

Consider two models A and B. Furthermore, we have a labelled dataset D and additionally some performance metric p. In the case of VADER we used classification accuracy. As with other hold-out methods such as k-fold CV and 10X10 CV we split the dataset into two parts: a training and a validation set. Specifically for the 2 fold cross validation repeated 5 times CV paired F test, we repeat the split (50% training and 50% test data) 5 times with different random seeds. In each of these 5 iterations, we fit both A and B to the training split and compare their performance ($p_A^{(1)}$ versus $p_B^{(1)}$) and ($p_A^{(2)}$versus $p_B^{(2)}$) on the test split by taking their difference. Then, the training and validation sets are swapped, and we compute their performance again, which results in us obtaining two performance measures $p^{(1)}$ and $p^{(2)}$ as defined below.

$$p^{(1)} = p_A^{(1)} - p_B^{(1)} \text{ and } p^{(2)} = p_A^{(2)} - p_B^{(2)} \tag{7}$$

Provided these two estimates we can now compute the mean $\overline{p}$ and variance s^2.

$$\overline{p} = \frac{p^{(1)} + p^{(2)}}{2} \tag{8}$$

and

$$s_i^2 = (p^{(1)} - \overline{p})^2 + (p^{(2)} - \overline{p})^2 \tag{9}$$

Using the formula in 10 we compute our F statistic which follows an approximate F distribution with 10 and 5 degrees of freedom where the mean and variance are calculated for each fold i which are our 5 random seeds and j represents the 2 fold CV.

$$F = \frac{\sum_{i=1}^{5} \sum_{j=1}^{2} (p_i^j)^2}{2 \sum_{i=1}^{5} s_i^2} \tag{10}$$

. Using this F statistic, a p value can be obtained and compared with a selected significance level $\alpha = 0.05$. Using a two sided test; if the p value is smaller than the selected significance level, we reject the null hypothesis and find that there exists a statistically significant difference between models A and B.

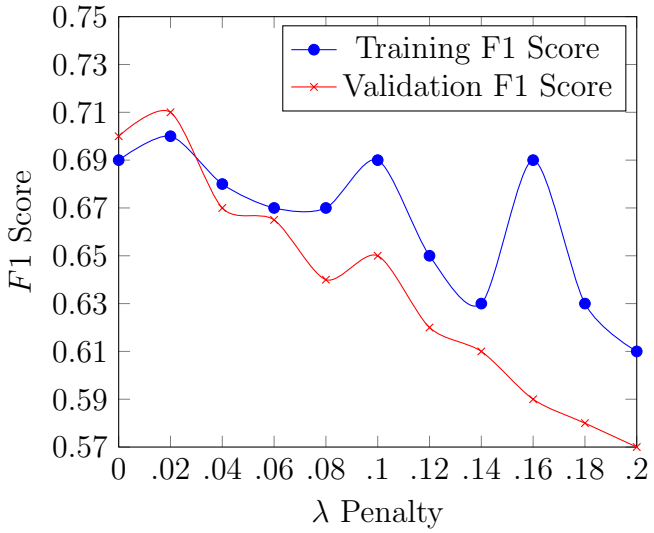

Figure 4: Plot of lambda vs F1 scores

2.6 Results

Modified VADER training The Modified VADER algorithm was fit to the pilot Steam dataset using 5 fold cross validation. Taking a range of values we selected a penalty term λ that best maximized the performance using the F1 score. It was determined that the penalty term that best maximizes the F1 score of the modified VADER algorithm is $\lambda = 0.02$. The full range of values and accompanying F1 scores is represented in 4.

Using the identified training penalty $\lambda = 0.02$ the modified and Base VADER algorithm were applied to the same validation set of Steam reviews, the calculated F1 scores being 70% for the former and 68% for the later. Comparing the results shown in table 2 these outcome it is observed that there is a 2% increase in performance. Applying the 5X2 cross validation F test the determined F statistic to compare the performance of these two models is found to be F = 4.7. From the F statistic the p = 0.049 value is obtained. Comparing this p value to the criteria $p < 0.05$ meaning that the performance improvement using the Modified VADER algorithm is statistically significant when applied to a testing Steam dataset but only barely. Given that modified VADER was trained on a validation Steam dataset it is unsurprising that the performance difference as compared to base VADER is at least comparable if not slightly improved.

Two datasets were sourced from Stanford (IMDB and Twitter) to compare the performance of modified and Base VADER across context. Following the same design setup for evaluating Steam dataset. The calculated F1 scores shown in table 2 for Modified and Base VADER when for IMDB movie reviews are 74% and 73% respectively. The F statistic being 2.7 (p = .14). Given the significance criteria $p < 0.05$ it is found that

Experiment	Precision	Recall	F1 Score
VADER Steam	0.61	0.77	0.68
Modified VADER Steam	0.66	0.76	0.70
VADER Twitter	0.66	0.63	0.65
Modified VADER Twitter	0.67	0.63	0.67
VADER Movie	0.64	0.85	0.73
Modified VADER Movie	0.65	0.85	0.74

Table 2: Comparison of base vs modified VADER's performance on some datasets

Modified VADER and Base VADER perform equally well. The same findings were also obtained after Modified and BASE VADER were applied to the Twitter dataset. Same order as before the F1 scores obtained for each algorithm were 0.65% and 0.67% with a 5X2 F statistic of 3.6 and p = .083. The findings provide evidence that the parameters estimated by Modified VADER using the Steam dataset perform equivalently to Base VADER across context. Furthermore, the obtained set of for each Algorithm are found to be distinct as represented in table 3. The implication being that Both BASE VADER and Modified VADER have estimated a unique set of parameters that have the same degree of performance when applied IMDB and Twitter but neither are a global set of parameters that generalize favourably across context.

Examining the differences in table 3 between the estimated parameters from simulated annealing versus the base VADER algorithm a few key differences are observed. Comparing both the positive and negative boosting word we see that the weighted contribution towards the estimated sentiment has increased and is no longer symmetric. Given this lack of symmetry, positive boosting words have a greater weighted effect on the calculated sentiment as opposed to negative boosting words. Similar changes have also been seen in exclamation punctuation, question high punctuation, conjunction decrease and especially in the case of negation with an estimated value of -2.0. In a similar way to L_1 regularization we see that the weights from the other parameters have been shrunk to zero which is interesting since in the original BASE VADER parameters these values were shown to contribute significantly to the estimated sentiment such as in the case of conjunction decrease. A possible explanation for how these estimated parameters are so different could be attributed to the unique properties of Steam reviews since the diction used relies heavily on positive boosting words, conjunction and excessive punctuation to express sentiment.

Parameter	Modified VADER	Base VADER
Booster Word Increase	1.922	0.293
Booster Word Decrease	-0.963	-0.293
Capitalization	0.518	0.733
Exclamation Punctuation	1.45	0.292
Question Punctuation Low	-0.022	0.18
Question Punctuation High	0.208	0.96
Negation	-2.0	-0.74
Negation (Special Case)	0.027	1.25
Conjunction Decrease	0.914	0.5
Conjunction Increase	-0.007	1.5

Table 3: Comparison of derived parameters between base and modified VADER

3 Neural Network Sentiment Analysis

In this section we will describe the current state of the art of neural networks for natural language processing. Furthermore, we will describe an implementation of BERT trained on the Steam data-set and then compare performance of BERT against our modified VADER algorithm as a binary sentiment classifier.

3.1 Neural networks

Artificial neural network A neural network is a mathematical model that emulates the decision-making and learning abilities of a mammalian brain. A neural network can generally be described as a type of non linear model that is composed of a number of activation functions (neurons) that forward, transform, filter and exchange information between each other. Based heavily in graph theory, neural networks are generally categorized into feed forward neural networks and recurrent neural networks. In figure 5 we have included a basic three layer network (Input, Hidden and Output). The lines connecting each circle represent a connection on how information flows through the network. Each connection is associated with a weight, which is a value used to control the strength of a signal between two neurons. The learning of a neural network is accomplished by adjusting these weights where the goal is to optimize for some measure of loss. Recurrent neural networks will be discussed after we introduce the basic structure of a feed forward neural network since they share many similarities.

Classification or regression using neural network Given some neural network we describe the input layer $f_k(x)$ as being composed of M number of nodes. For each node a randomly assigned weight w from a uniform distribution is assigned. These weights impact how much the output of each node in a layer affects its contribution in the following layer. Given that this network is fully connected between layers, the total number of inputs forwarded to each node within a layer would be equal to the number of nodes k of the preceding layer. The layer or layers h_j composed of N number of nodes that are in between the initial and final layer of a neural network are known as hidden layers. A hidden layer behaves similarly to a filter where the weighted sum of the inputs from the input layer is only passed along if it reaches some threshold based on the output from some function. In addition each node in a hidden layer contains some constant α_i value that is individual to each neuron. This filter mechanism is accomplished by the inclusion of some function σ that only activates if a specific value or condition is met. Common examples of such functions are sigmoid, tanh and ReLU the details of which will be discussed in the next paragraph. After the sum of the weighted inputs has been filtered in the hidden layer it is then fed and summarized into the final layer g which is composed of a single neuron and it is this neuron that computes the estimated output based on the sum of the weighted outputs from the hidden layer h_j added along with some bias b_0. The activation function used in g can be different than the one used in

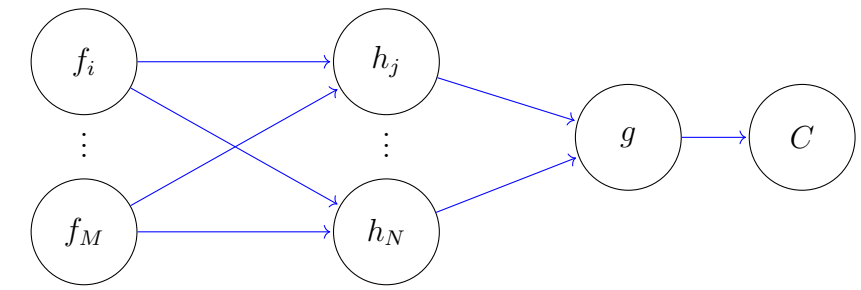

Figure 5: General feed forward network

the hidden layer.

$$\begin{aligned} f_i &= w_i * x,\ i \in \{1, 2, ..., M\} \\ h_j &= \sigma(b_j + \sum_{i=1}^{M} f_i),\ j \in \{1, 2, ..., N\} \\ g &= \lambda(b_0 + \sum_{j=1}^{N} w_j * h_j),\ j \in \{1, 2, ..., N\} \end{aligned} \tag{11}$$

Common choices for σ are the sigmoid function, hyperbolic tangent function (tanh), or rectified linear function (ReLU). Their equations are as follows.

$$\begin{aligned} \text{sigmoid}(w^t x) &= \frac{1}{1 + \exp(-w^t x)} \\ \tanh(w^t x) &= \frac{\exp(w^{tx}) - \exp(-w^{tx})}{\exp(w^{tx}) + \exp(-w^{tx})} \\ \text{ReLU}(w^t x) &= \max(0, w^t x) \end{aligned} \tag{12}$$

The sigmoid function takes as input a real value number and transforms it to be between 0 and 1. The use of the sigmoid function in neural networks has fallen out of favour due to two majors problems. Firstly that the sigmoid function losses effectiveness with large positive or negative values and secondly that the output of the sigmoid function is not zero centered which is a problem because it can introduce some unbalance in how weights are updated in a neural network. [12] The first issue also applies to the Tanh function. Compared with the sigmoid function and the tanh function, ReLU is easy to compute, fast to converge in training and yields equal or better performance in neural networks.

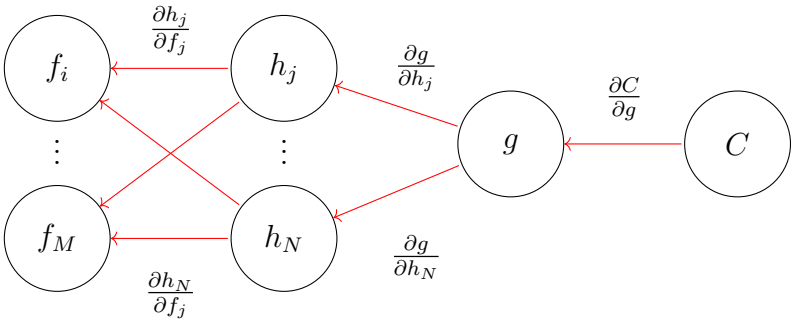

Figure 6: Back propagation in a feed forward network

Training a neural network A neural network learns by taking the output of some cost function whereby the weights and biases for each node in each layer are changed where the objective is to minimize the cost of the network. In other words, maximize its performance. The level of adjustment is determined by computing the gradient of the cost function. The gradient in a neural network is the weighted direction of how much each parameter (node) with respect to the cost function C needs to change to minimize the cost of the network.

$$\begin{aligned} w_{t+1} &= w_t - \alpha \frac{\delta C}{\delta w_t} \\ b_{t+1} &= b_t - \alpha \frac{\delta C}{\delta b_t} \end{aligned} \tag{13}$$

In equation 13 we show how by taking the partial derivative with respect to the cost function C backwards through each layer it allows for the efficient estimation of derivatives for each nodal connection within a network. Once we have obtained the partial derivatives with respect to each weighted connection in a neural network. We subtract the original weights and biases with the product of a constant hyper-parameter α and the associated partial derivative with respect to the cost function C. The hyper-parameter α controls the rate of change between each learning epoch so a higher value shortens the rate of convergence for a neural network model. Similar to simulated annealing this process is repeated multiple times until the desired minimum cost value has been obtained or if there is no change between each learning cycle.

3.2 Recurrent neural networks

A recurrent neural network is a type of neural network that allows for the exchange of information between nodes in the same layer. In graph theory this type of network would be described as a cycle. Unlike in a feed-forward network, information can pass multiple times through the same node. This introduction of a cycle into a neural network , allows for the past h_{t-1} output from nodes to affect future inputs x_t to the same nodes. The

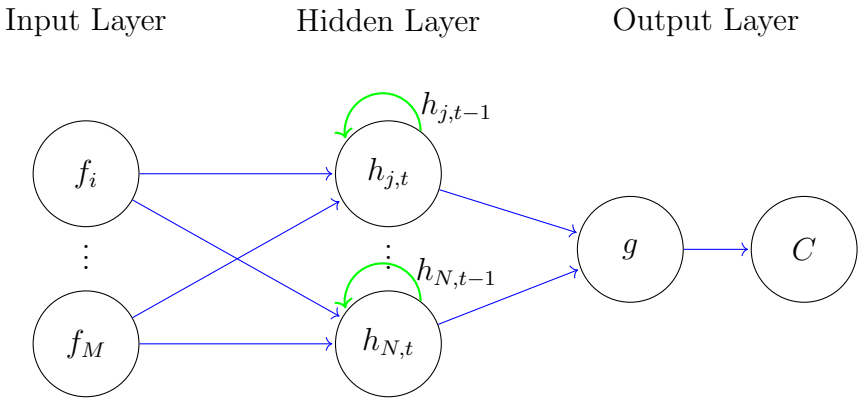

Figure 7: An example of a recurrent neural network

mathematical expression of a standard recurrent neuron is stated in equation 14.

$$\begin{aligned} h_t &= \sigma(w_h h_{t-1} + w_x x_t + b) \\ y_t &= h_t \end{aligned} \tag{14}$$

The symbols x_t, h_t and y_t denote the inputs which are fed in sequentially based on some order $t \, \epsilon \, \{0, ...\infty\}$. All three of the inputs of a RNN are fed in sequentially where t is the current time step of the fed in data. based on the information in the cycle, and the output of the cell at time t respectively; w_h and $w_x$are the weights; and b is the bias. RNNs have difficulties with long term dependencies in sequential data. Hochreiter et al [14] analyzed fundamental reasons for the vanishing/exploding gradient problem and found that during back-propagation the weighted cost gradient of Recurrent Neural networks tend to either approach infinity or zero. This gradient issue is explained by how during back-propagation the product of the estimated weights from prior and current cycles are used in estimating the gradient for the current cycle. Given that the weights per each cycle in a RNN are relatively similar these values can either decrease (if they are small) or increase (if they are large) too quickly before the RNN can converge.

LSTM Aiming to solve the vanishing/exploding gradient problem which has impeded Recurrent neural networks from being used in a broader set of sequential applications, Hochreiter and Schmidhuber [14] proposed a modification to the standard recurrent cell that improved the long term memorization capabilities of a standard recurrent neuron by introducing the concept of memory gates. This is known as a Long Short Term Memory (LSTM) neuron. A standard LSTM cell (represented in figure 8) is composed of an input (left tanh function) and output (right tanh function) gate. Those gates act on the inputs they receive, and similar to the neural network's nodes, they block or pass on information based on some minimum threshold which they filter with their own sets of weights. Those weights, like the weights that modulate input and hidden states, are adjusted via the recurrent networks learning process. The inclusion of these

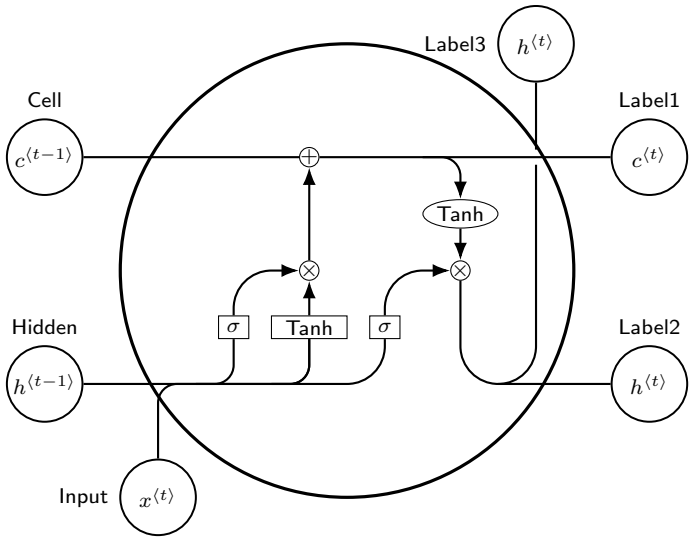

Figure 8: A LSTM neuron with a Forget, Input and Output Gate

gates changes the dynamics of how weights are updated since now there is an additive property in the estimation for each neuron. By increasing the number of elements within each neuron it decreases the odds that the weights per each cycle will be in effect the same.

Word vector representation A word vector is the representation of the meaning of a word in a vector space. Each word is represented as a unique multi-dimensional vector of real numbers where each point captures a dimension of the words meaning. For example words such as car and train would be in the same geometric region since they are both vectors of the word vehicle. The numbers in a word vector represent the word's distributed weight across dimensions where each dimension represents a meaning and the word's numerical weight on that dimension captures the closeness of its association. Thus, the semantics of a word are embedded across the dimensions of a vector space. An example of such a method is Global Vectors for word representations (GloVe) [26]. Glove is a type of unsupervised machine learning model that represents a word as a vector by examining a large of corpus of text calculating it's degree of semantic similarity. A limitation of representing words in this manner is that it does not account for contextual differences in text

Using neural networks to represent word vectors To improve the contextual representation of word vectors pre-trained neural networks have been the default. An example of such a model is ELMo [27] which is a set of two bi-directional LSTM language models stacked on top of each other. In order to capture the inner structure of a word, a character-level convolution neural network is used to represent each word as a vector of characters. Given a sequence of N tokens (this could be a whole sentence, or at least

a part of a sentence) the bidirectional language model computes the forward and also the backward probability. When fitting such a network the goal is to to maximize the log likelihood of both the forward and backward probability.

3.3 BERT applied to sentiment analysis

A transformer is a deep learning model that adopts the mechanism of self-attention, deferentially weighting the significance of each part of the input data. Transformers originally introduced in the paper Attention is all you Need [36] were used for natural language processing with the purpose of further optimizing the performance of word vectors. Similar to recurrent neural networks, transformers are designed to process sequential input data such as natural language. Dissimilar to RNNs and LSTMs, a transformer processes the entire input of a collection of text at the same time. This idea of a transformer processing unit in natural language processing is similar to how the human brain learns and interprets a language. By reading the entire sequence the attention mechanism can better calculate the relative context for any position in an inputted sequence. BERT (Bidirectional Encoder Representations from Transformers) is in essence a language model that consists of a set of bi-directional Transformers stacked on top of each other. In the default form, a Transformer includes two separate mechanisms - an encoder that reads the inputted text and a decoder that interprets the inputted text and provides some predicted output on the meaning of the word. Before feeding the sentence into BERT, 15 percent of the words in each sequence are hidden. The model then predicts the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence. The BERT loss function takes into consideration only the prediction of the masked values and ignores the prediction of the non-masked words. This masking can be thought of as a way to prevent the neural network from over-fitting to the training data. The second mechanism used in BERT during the training process is as follows: the model receives a pair of sentences as input and learns to predict if the second sentence in the pair is the following sentence in a paragraph. The paired sentences are taken from two separate pools of sentences where the second pool has been randomly sampled. Both of these methods (Masked LM and Next Sentence prediction) are trained in the same instance where the goal is to minimize the sum of the calculated loss function from both methods. A limitation of BERT is that by randomly masking some percentage of words within a text the relationship between each masked token has been assumed to be independent from one another. This independence assumption prevents the estimation of the joint probability for a given masked token and impedes how efficiently BERT can estimate long term dependencies in text.

RoBERTa This deep learning model is a variants of BERT which was developed by the Facebook AI research group. RoBERTa stands for Robustly Optimized BERT Pre-training approach. As the name implies the differences between these two models arise

not from a change in architecture but in the methods used during training. Comparing the two models BERT was trained for approximately 1 million steps with a batch size of 256 sequences. In RoBERTa, the authors increased the size of each batch to 2000 sequences and halved the number of iterations. The larger batches improved the learning rate on masked language modelling objective and as well as end-task accuracy. Going beyond batching methodologies the authors of RoBERTa noticed that the masking process used by BERT is statically defined during training. This was substituted with a dynamic masking algorithm in which the location of words to be masked changes each time data is passed into the model. This change is important during training of larger data-sets and in addition is shown to perform comparably and in some cases marginally better than static masking on Next Sentence Prediction (NSP) tasks [22].

XL-Net XL-Net is a state of the art deep learning method that solves the masking Independence assumption in BERT. The general idea behind XL-net is that by re-framing the training objective (prediction of these masked tokens) as a conditional probability distribution sampled from all possible permutations given a masked token to predict. Taking the average over all possible permutations in a sequence of text the XL-net model can then calculate the join probability for a given masked token. This removes the independence assumption found in BERT and allows XL-NET to more efficiently estimate long term dependencies in text making it the current state of the art for deep learning sentiment analysis applications.

BERT specific text pre-processing The BERT algorithm has its own specific set of rules to encode text. The developers of BERT have engineered a set of 3 features that allows BERT to perform better when learning the underlying structure of some provided body of text. Position Embedding which encodes the position of each word within the imputed text. This is included to overcome a limitation of Transformers since unlike LSTM and RNN it does not have a built-in mechanism that learns sequential patterns. Segment embedding, BERT labels each sentence pair as inputs for next sentence prediction. This aids the model in learning the relationship between these two sentences. Token Embedding: Using Word Piece a machine learning model that constructs a dictionary of words interpret-able by neural networks such as BERT.

Tuning of BERT hyper-parameters When fine tuning a pre-trained a model there exists a significant risk for information from the previously learned task to be abruptly overridden as new information is incorporated. This phenomenon is commonly known as catastrophic forgetting. Sentiment classification is conducted by adding a dense layer after the last hidden state of the BERT transformer. As in the literature this is the recommended practice when applying BERT with the goal of binary classification. This modified network is trained on the labeled sentiment Steam review dataset. The same hyper-parameter tuning strategies were used as recommended by Devlin et al [7]. we fine tune uncased BERT (BASE) using a batch size of 32 and a maximum learning rate

Experiment	Precision	Recall	F1 Score
VADER Steam Reviews	0.61	0.77	0.68
Modified VADER IMDB	0.66	0.76	0.70
BERT Steam Reviews	0.86	0.30	0.79
BERT Movie Reviews	0.80	0.45	0.80

Table 4: Comparing performance of BERT vs base and modified VADER

of $1e-4$. To minimize the loss of our BERT model the learning rate is linearly increased from 0 to 1e-4 and linearly decreased to 0 afterwards for the first 10% of iterations. Dropout is applied with probability p = 0.1 and weight decay of $\lambda = 0.01$. BERT is trained for 3 epochs with global gradient clipping active. As is recommended by Zhang et al [41] the ADamW optimizer is used with bias compensation.

The results in table 4 indicate that BERT outperforms both modified and base VADER when trained and validated on both the Steam and IMDB review datasets. However, this does come at a cost with respect to the time taken since BERT requires several hours of run time in order to attain performance metrics that defeat VADER. We also see that when BERT is then applied a contextually independent dataset (IMDB movie reviews) accuracy is decreased and recall (loss) increases.

BERT does outperform BASE and Modified VADER but unlike neural network methods the training requirements are lessened somewhat and the results are interpretative. We find evidence that confirm the findings from Huto Et [16] al that these set of grammatical rules can be applied across context. However, the set of weights used within the algorithm itself are dependent on the context itself and are not generalized. We find that training the Modified VADER algorithm for each set of sentiment analysis word data can improve the performance by at least a few percentage points.

4 Conclusions

In this thesis, we have explored statistical methods to further extend, improve and develop lexicographical sentiment analysis methods within and across contexts. To this aim we have described VADER, a lexical sentiment analysis algorithm as one that lacks robustness and have proposed that by leveraging statistical learning methods improvements can be made that are statistically significant when applied to a new context. As an additional contribution to the field of sentiment analysis. A novel dataset from Steam, a hybrid social media and video game retail platform. This Steam dataset was used to estimate the grammatical weights that control the estimation of sentiment of the modified VADER algorithm. Taking a separate validation set of Steam reviews the performance of Base and Modified VADER were compared where it was found that using statistical learning methods such as simulated annealing does improve the performance but only marginally. However, the same was not the case for Twitter and IMDB datasets where the performance differences between the two models were found to not be statistically significant.

A few outcomes are summarized. Firstly, that the estimated parameters do not perform favorably across context, we use the fact that neither Modified VADER nor BASE VADER performed any better than the other across context as evidence of this observation. Based on this information we conclude that the grammatical rules that define VADER are not robust and do not generalize across context. This does explain why the parameters determined by Modified VADER are so vastly different as opposed to those estimated by BASE VADER. It is suspected that the use of a larger more general set of grammatical rules (such as using syntax trees) in addition to contextual features may need to be used in order to properly estimate sentiment within text. Future work would be in exploring a larger set of parameters and training the modified VADER algorithm on a mixed data-set composed of Twitter comments, IMDB movie and Steam video game reviews. Using the same design methodologies as presented in this thesis the aim would be in confirming with greater certainty if there exists a global set of grammatical rules that perform well regardless of context.

References

[1] Peter D. Turney Alpher. "Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews". In: *ACL* 40.1 (2002), pp. 417–427.

[2] Raymond E Daly Dan Huang Andrew L Maas and Andrew Y Ng. *Learning Word Vectors for Sentiment Analysis*. 2011.

[3] Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. "SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining". In: *LREC*. Ed. by Nicoletta Calzolari et al. European Language Resources Association, 2010. ISBN: 2-9517408-6-7. URL: http://nmis.isti.cnr.it/sebastiani/Publications/LREC10.pdf.

[4] Margaret Bradley and Peter Lang. "Affective Norms for English Words". In: *NIMH* 49.1 (2011), pp. 142–150.

[5] Tyler Clark. "Characteristics of Online Gaming Market Structures: Evidence from Steam's Online Gaming Marketplace". In: *NCUR* (2019).

[6] Nhan Cach Dang, Maria N. Moreno Garcia, and Fernando de la Prieta. "Sentiment Analysis Based on Deep Learning: A Comparative Study". In: *CoRR* abs/2006.03541 (2020). eprint: 2006.03541.

[7] Jacob Devlin. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Google AI Language* (2018).

[8] T. G Dietterich. *Approximate statistical tests for comparing supervised classification learning algorithms*. 1998.

[9] Alpaydin E. *Combined 5 x 2 cv F test for comparing supervised classification learning algorithms*. 1999. DOI: 10.1162/089976699300016007.

[10] Theresa Wilson Efthymios Kouloumpis and Johanna Moore. "Twitter Sentiment Analysis: The Good the Bad and the OMG!" In: *AAAI* (2011).

[11] Rishin Haldar and Debajyoti Mukhopadhyay. *Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach*. 2011. DOI: 10.48550/ARXIV.1101.1232. URL: https://arxiv.org/abs/1101.1232.

[12] Trevor Hastie. *The Elements of Statistical Learning*. 2009.

[13] Pengcheng He et al. *DeBERTa: Decoding-enhanced BERT with Disentangled Attention*. 2020. DOI: 10.48550/ARXIV.2006.03654. URL: https://arxiv.org/abs/2006.03654.

[14] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[15] Minqing Hu and Bing Liu. "Mining and summarizing customer reviews". In: *ACM* 40.1 (2004), pp. 168–177.

[16] C.J. Hutto and Eric Gilbert. "VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text". In: 2014.

[17] James W. Pennebaker, Roger J. Booth, and Martha E. Francis. *LIWC 2022*. Version 22. July 19, 2022. URL: https://www.liwc.app/.

[18] Guokun Lai et al. "RACE: Large-scale ReAding Comprehension Dataset From Examinations". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 785–794. DOI: 10.18653/v1/D17-1082. URL: https://aclanthology.org/D17-1082.

[19] Kushal Dave Steve Lawrence and David M. Pennock. "Mining the peanut gallery: opinion extraction and semantic classification of product reviews". In: *World Wide Web* 12.1 (2003), pp. 519–528.

[20] Na Li. "SENTIMENT FEATURES FOR YELP NO TURES FOR YELP NOT-RECOMMENDED ONLINE -RECOMMENDED ONLINE REVIEWS STUDY". In: *Open Access Digital Commons* (2018).

[21] Jiwei Li1 and Eduard Hovy. "Reflections on Sentiment/Opinion Analysis". In: *Language Technology Institute* 40.1 (2015).

[22] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. DOI: 10.48550/ARXIV.1907.11692. URL: https://arxiv.org/abs/1907.11692.

[23] Edward Loper and Steven Bird. "NLTK: The Natural Language Toolkit". In: *CoRR* cs.CL/0205028 (2002). URL: http://dblp.uni-trier.de/db/journals/corr/corr0205.html#cs-CL-0205028.

[24] Andrew Maas Raymond Daly Peter Pham Dan Huang Andrew Ng and Christopher Potts. "Learning Word Vectors for Sentiment Analysis". In: *ACL* 49.1 (2011), pp. 142–150.

[25] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. "Thumbs up? Sentiment Classification using Machine Learning Techniques". In: *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*. Association for Computational Linguistics, July 2002, pp. 79–86. DOI: 10.3115/1118693.1118704. URL: https://aclanthology.org/W02-1011.

[26] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162.

[27] Matthew E. Peters et al. *Deep contextualized word representations*. 2018. DOI: 10.48550/ARXIV.1802.05365. URL: https://arxiv.org/abs/1802.05365.

[28] Wojciech Gryc Prem Melville and Richard D. Lawrence. "Sentiment Analysis of Blogs by Combining Lexical Knowledge with Text Classification". In: 2009, pp. 234–778.

[29] Fernando Rada and Carlos A.Iglesias. "Social context in sentiment analysis: Formal definition, overview of current trends and framework for comparison". In: *Information Fusion* 52.1 (2019), pp. 344–356.

[30] Andrew J. Reagan et al. *Benchmarking sentiment analysis methods for large-scale texts: A case for using continuum-scored words and word shift graphs*. 2015. DOI: 10.48550/ARXIV.1512.00531. URL: https://arxiv.org/abs/1512.00531.

[31] Julio Reis et al. *Breaking the News: First Impressions Matter on Online News*. 2015. DOI: 10.48550/ARXIV.1503.07921. URL: https://arxiv.org/abs/1503.07921.

[32] Christopher D. Manning Andrew Y. Ng Richard Socher Alex Perelygin Jean Y. Wu Jason Chuang and Christopher Potts. "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank". In: *stanford* (2013).

[33] *Steam Web Store*. May 2022. URL: https://store.steampowered.com/games/.

[34] J Surowiecki. "The wisdom of crowds: Why the many are smarter than the few and how collective wisdom shapes business, economies, societies, and nations". In: *AAAI* (2004).

[35] Janyce Wie Theresa Wilson and Paul Hoffmann. "Recognizing Contextual Polarity in Phrase-Level Sentiment Analysis". In: *HLT* 5.1 (2005), pp. 347–354.

[36] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[37] Alex Wang et al. *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*. 2018. DOI: 10.48550/ARXIV.1804.07461. URL: https://arxiv.org/abs/1804.07461.

[38] Qizhe Xie et al. *Unsupervised Data Augmentation*. cite arxiv:1904.12848. 2019. URL: http://arxiv.org/abs/1904.12848.

[39] Zhilin Yang and Zihang Dai. "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: *NeurIPS* 33 (2019).

[40] Zhilin Yang et al. *XLNet: Generalized Autoregressive Pretraining for Language Understanding*. 2019. DOI: 10.48550/ARXIV.1906.08237. URL: https://arxiv.org/abs/1906.08237.

[41] Chiyuan Zhang et al. *Understanding deep learning requires rethinking generalization*. 2016. DOI: 10.48550/ARXIV.1611.03530. URL: https://arxiv.org/abs/1611.03530.

[42] Zhengyan Zhang et al. *ERNIE: Enhanced Language Representation with Informative Entities*. 2019. DOI: 10.48550/ARXIV.1905.07129. URL: https://arxiv.org/abs/1905.07129.