University of Sussex

SONGIFAI

Exploring the use of Covariate Specific Word Embeddings

Jonathan Magbadelo

Candidate Number: 149708

Supervisor: Dr. Julie Weeds

Submitted for the degree of Bachelors of Computer Science and Artificial Intelligence

University of Sussex

April 2019

Statement of Originality

This report is submitted as part requirement for the degree of Computer Science and Artificial Intelligence at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.

Signature:

Jonathan Magbadelo

Acknowledgements

I would like to thank my project supervisor, Dr Julie Weeds for her constant support throughout the development of this project. I would also like to thank my colleagues at Brandwatch who provided expert advice whenever I required it.

UNIVERSITY OF SUSSEX

Jonathan Magbadelo

SONGIFAI

Exploring the use of Covariate Specific Word Embeddings

Summary

Word embedding algorithms such as GloVe are vector space models capable of providing a distributed representation of words. By utilising vast amounts of text corpora, these representations are able to encapsulate semantic and syntactic regularities along with relationships between words. Traditional word embedding algorithms tend to operate on corpus documents in solidarity, often neglecting additional covariate metadata attached to corpus documents.

CoVeR, an extension of the GloVe algorithm, jointly learns word embeddings together with a set of diagonal weight matrices, representing the affect of a particular covariate on the base embeddings.

This project explores the use of covariate specific word embeddings for both neural language modelling and text classification. Specifically, both models are applied to a possible use case: a songwriting assistant application. The main areas covered in this dissertation are:

- An overview of issues songwriters face whilst writing songs.
- An overview of related research
- An introduction to word embeddings, neural language modelling and text classification.
- Requirements analysis for SONGIFAI, the prototype application.
- A detailed account of the implementation process.
- An evaluation of CoVeR and its usage in each model.
- An evaluation of SONGIFAI
- Limitations and future work

Contents

List of Tables

List of Figures

The skill of writing is to
create a context in which
other people can think

EDWIN SCHLOSSBERG

Chapter 1

Introduction

1.1 Overview

Natural language is a constantly evolving system capable of producing infinite numbers of sentences each with their own distinct meaning. The task of inferring meaning from a sentence is one that is seemingly trivial for humans, even from a young age. However, getting machines to derive meaning from textual data has long been a challenge within the field of Artificial Intelligence, specifically the subfield of Natural Language Processing (NLP). Computing numeric representations of words is the first step in bridging the gap between natural language and machines. Previous methods to create such representations include the one-hot encoding method which involves encoding words as word vectors which have as many dimensions as the number of unique words within a corpus. Each dimension in the vector is assigned a zero value except for the dimension representing the word which is assigned a value of 1. For example given a corpus with the following vocabulary

{"Billie", "Jean", "is", "not", "my", "lover"}

and assuming that each word's index represents the dimension of the word, the word vector for "Jean" would be

[0, 1, 0, 0, 0, 0]

The discrete word representations obtained from these types of methods are sparse and suffer from the curse of dimensionality due to the dimensionality of the word vector growing linearly with the size of the vocabulary. Moreover, these representations fail to capture any syntactic and semantic relationships contained within textual data.

Recent word representation methods rely on the intuition that word meaning can be inferred from context. Specifically, these methods are based on the distributional hypothesis (Harris (1954)) that states that words which appear in a similar context have similar meanings. Algorithms such as GloVe (Pennington et al. (2014)) and word2vec (Mikolov et al. (2013a)) exploit this concept to produce dense continuous word representations which are able to encode syntactic and semantic regularities. These types of distributed representations, first proposed by Rumelhart et al. (1986), are also known as ***word embeddings*** due to the *"embedding"* of words into often low dimensional space.

Often accompanying text corpora are associated covariates such as authorship and publication date, which provide additional metadata about a corpus. Traditional word embedding methods such as GloVe, fail to capture this information and instead rely on individual

word embeddings for each subset of data relating to a particular covariate. This approach to creating covariate specific word embeddings has two drawbacks: **data efficiency**, as performing these methods on subsets is repetitive, and can also yield suboptimal word representations if data is scarce for a particular subset, and decreased **interpretability**, due to the random initialisation of word embeddings, which even when created with the same random seed, produce representations relative to the the structure of the text.

CoVeR (Tian et al. (2018)), a novel tensor decomposition method for learning covariate specific word embeddings, is an extension of the GloVe algorithm which aims to efficiently encode covariate information within learned embeddings whilst increasing interpretability. The goal of this project, is to explore the effectiveness of CoVeR generated word embeddings as well as its potential application within a prototype software aimed at songwriters.

The motivation behind using CoVeR is two-fold. Firstly, its efficient use of data is beneficial when data isn't readily available, which, in the case of this project and song lyrics, is true (discussed more in chapter 2) and its increased interpretability compared to traditional methods which allows for greater analysis. The next section briefly contextualises the project, highlighting common issues songwriters face during the writing process.

1.2 The Songwriters Dilemma

Songwriting is an integral part of the song making process which often draws upon past events and experiences. Structure and content both contribute heavily towards the success of a song; with the latter being a key factor on the extent to which a song resonates with a listener. A problem commonly faced by songwriters is that of word choice, through which they can express their ideas clearly and concisely.

Skilled writers are commonly attributed with having vast vocabulary ranges. For adults, the average vocabulary size ranges between 15,000 - 23,000 words (McCrum (2011)). Examining his works alone, Shakespeare is said to have had an approximate vocabulary range of 30,000 - 65,000 words[1] [2]. Nonetheless, a skilled songwriters ability to write impactful lyrics is not down to vocabulary size alone, but effective word choice.

As shown in a study examining vocabulary range within Hip Hop[3], which recently surpassed Rock as the most popular genre in the United States[4], more is not always better. The study examines the unique word count of 150 famous Hip Hop artists across their first 35,000 words used in lyrics. Aesop Rock, ranked 1st on the list, recorded a count of 7,392 unique words across his first 35,000 words. In contrast, rappers Drake and Future, who ranked 130th and 131st respectively, had an average unique word count of 3,334 words used across their first 35,000 words; a 55% decrease from Aesop Rocks count.

To validate the earlier claim that vocabulary range is not indicative of a songwriters ability to write impactful lyrics, the unique word count per artist was compared against the average number of Billboard 100[5] [6] songs an artist had obtained across their career (seen in Figure 1.1). Pearson's Correlation Coefficient, which is used to measure the

[1] https://kottke.org/10/04/how-many-words-did-shakespeare-know
[2] This range is most likely skewed as variant forms of words are included as singular counts
[3] https://pudding.cool/projects/vocabulary/index.html
[4] https://www.nielsen.com/us/en/insights/reports/2018/2017-music-us-year-end-report.html
[5] https://www.billboard.com/charts/hot-100
[6] The Billboard Hot 100 is a ranking chart of the 100 most played songs in the United States based on sales, radio play and online streaming.

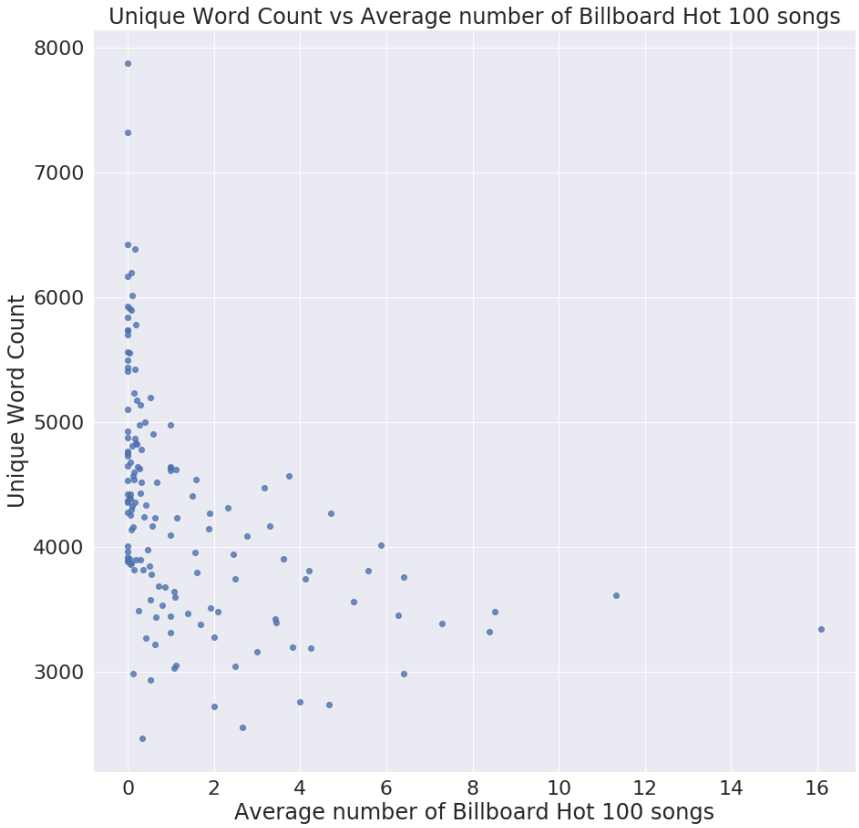

Figure 1.1: Average number of Billboard Hot 100 songs during artist activity, compared to unique word count across an artists first 35,000 words.

linear relationship between two variables, was applied to both unique word count and average number of Billboard 100 songs. This resulted in a correlation coefficient of - 0.42, indicating a weak inverse correlation between the pairs of data. This value supports the earlier claim that vocabulary range is not indicative of a songwriters ability.

Common methods used to improve songwriting competency include group writing and vocabulary expansion. More recently, software solutions such as MasterWriter[7] have been used to consolidate previous methods. An inherent drawback with applications such as MasterWriter is the static nature of features such as fixed word dictionaries and thesauruses. Consequently, these applications fail to capture the ever changing nature of language.

After the completion of song lyrics, another task in the song making process, if not already completed, is song production. For experienced songwriters, production choice might be a routine decision due to them having an established production style. On the other hand, for newer songwriters or songwriter's wishing to experiment with a new sound, this choice may not be a trivial one, especially with the constant increase in the popularity and existence of genres and subgenres, which come with their respective production styles.

[7] https://masterwriter.com/5

As shown in Figure 1.2, the distribution of music genres listened to has spread over time, even more so, the increase of distinctive sub genres within these genres.

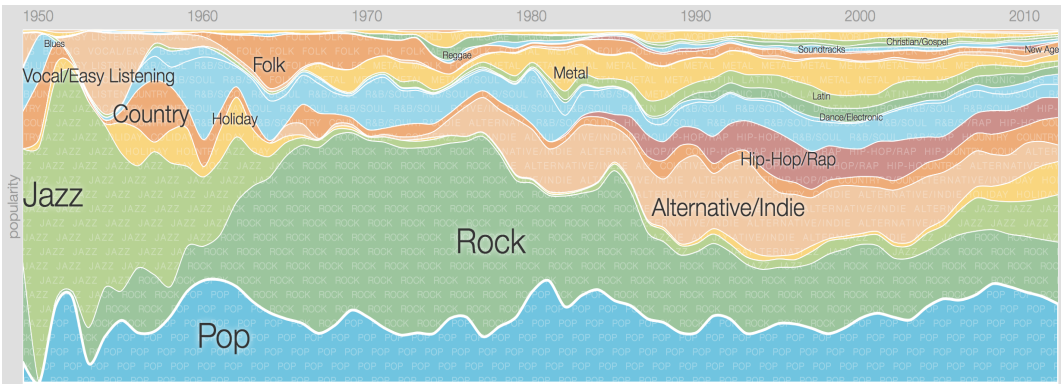

Figure 1.2: Distribution of music genres listened from 1950, based on the music library of Google Play Users.

[8]

1.3 Aims and Objectives

As previously stated, the goal of this project is to explore effectiveness of CoVeR as well as its potential usage in a software application. Specifically this takes the form of a prototype software solution to help reduce common problems faced by songwriters, namely:

1. Word Choice
2. Lyric classification

The tasks of predicting the next word in a song and lyric classification can viewed as abstracted forms of two common NLP tasks: *Language Modelling* and *Text Classification*, which are two active research areas in NLP that have benefited greatly from word embeddings. The usage of these word embeddings has helped improve accuracy in tasks such as Machine Translation (Qi et al. (2018)) and Document Classification (Manning et al. (2008)). State of the art results in both areas have also utilised recurrent neural networks (RNN), (discussed in chapter 4) which employ word embeddings within the input layer to the network.

With this in mind, a prototype solution, SONGIFAI is proposed. SONGIFAI provides two main features namely lyric assistance through predictive text and a word suggestion feature, as well as genre classification for a given song lyrics. The covariates explored in this project are the following music genres: *Pop*, *Rock* and *Hip Hop*.

As stated before, RNN's have been used successfully in both language modelling and text classification tasks. The specific variant of the recurrent neural network which has been successfully utilised in these tasks is the Long Short-Term Memory (LSTM) network (discussed in chapter 4). This network architecture is used for both the language model and text classifier which are applied to the word prediction and lyric classification features respectively. The word suggestion feature utilises the covariate specific embeddings themselves. A high level system architecture of the prototype can be seen in Figure 1.35

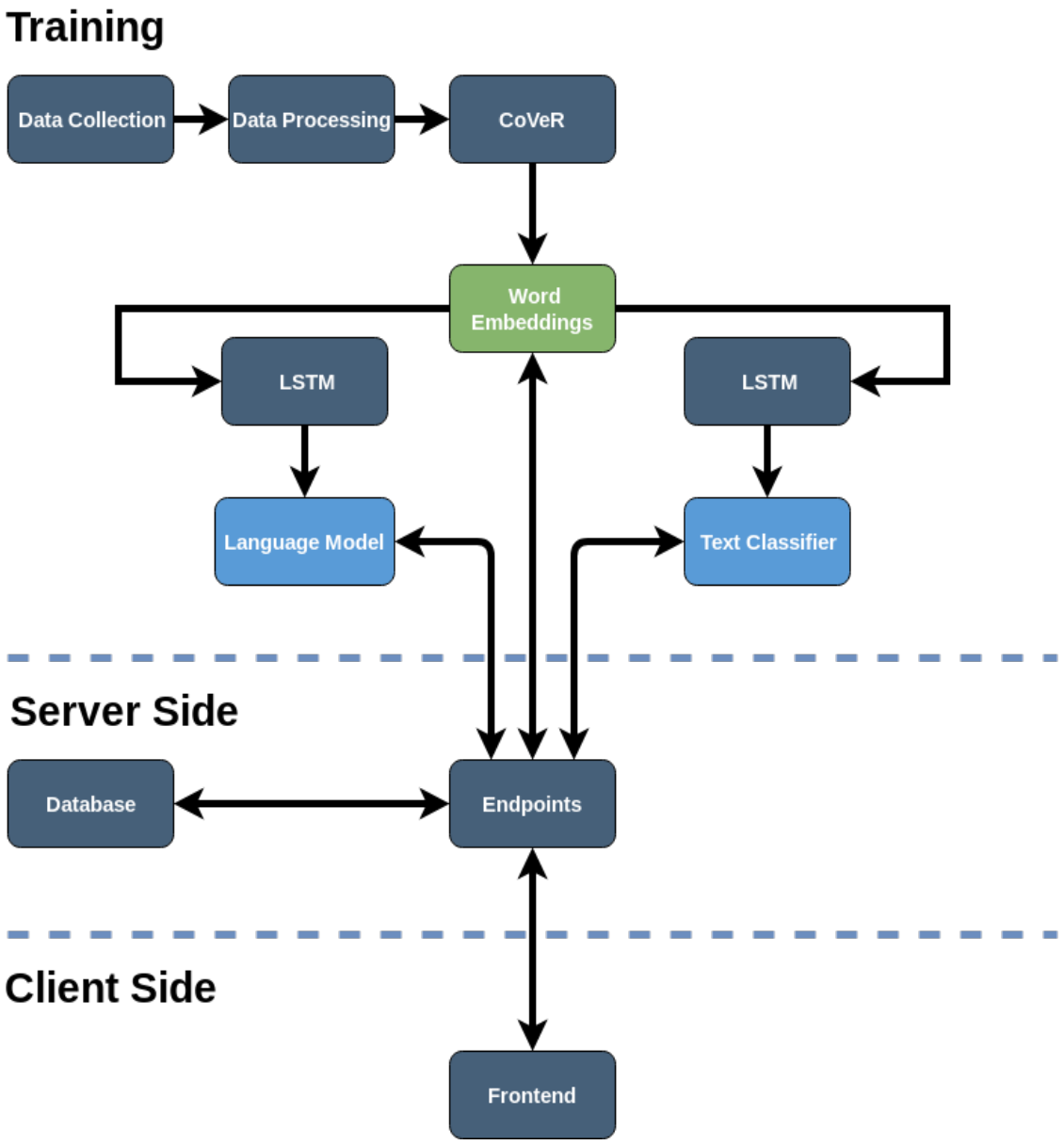

Figure 1.3: From a high level, the system can be split into three sections: **Training**, where models are created. **Server Side**, a Python based API used to interface with the trained models.**Client Side** a user interface built using JavaScript

.

Chapter 2

Professional Considerations

Throughout the development of this project both professional and ethical considerations were taken into account, including those highlighted in the British Computing Societies (BCS) Code of Conduct [1] and Code of Good Practice. This chapter outlines the relevant areas in the specified documents which have been adhered to during the project.

2.1 Code of Conduct

Professional Competence and Integrity

The completion of this project was a large undertaking due to the implementation and integration of a novel machine learning method within a prototype software application. Though the project is beyond the scope of a typical final year project, all work carried out have roots to modules taken in the University of Sussex Computer Science and Artificial Intelligence course, specifically the Neural Networks, Advance Natural Language Engineering and Software Engineering modules. In accordance with section 2.C of the BCS Code of Conduct, background research continually occurred throughout development to maintain a competent standard of professional knowledge.

Duty of Relevant Authority

In agreement with section 3.A and 3.B of the BCS Code of Conduct, all scenarios which may cause a conflict of interest between the project and the University of Sussex have been avoided.

Duty to Professionalism

In accordance with section 4.A and 4.C of the BCS Code of Conduct, the manner in which this project was conducted was one which maintained the reputation of the BCS. During meetings with other BCS members and professionals such as my project supervisor and work colleagues, appropriate levels of respect and integrity were upheld in accordance with section 4.B of the BCS Code of Conduct.

2.2 Good Practices

The motivation behind this project is one rooted in exploratory research rather than being client driven. Nevertheless, it is important that code produced is well structured and testable to ensure quality assurance. Where possible and in accordance with section

[1] https://www.bcs.org/category/6030

5.2 of the BCS Code of Good Practice, the code produced is well structured and organised to help facilitate further testing and maintainability.

The same section of the BCS Code of Good Practice refers to the following of programming language guidelines. Both Python and JavaScript were used extensively during the development of the project and where appropriate best practices and coding style/conventions have been adhered to.

2.3 Ethical Considerations

The success of a machine learning project relies heavily on data availability and quality. Currently, there exists no central repository for downloadable song lyrics due to potential copyright infringement. Under the Research and Private Study section of the Copyright, Designs and Patents Act (1988) [2], a public song lyric dataset is used solely for training machine learning models.

Finally, the project utilises textual data which may have explicit or offensive content within it. After the training of models, which will not be filtered to allow permit artistic freedom, a filtering option will be implemented in order to prevent potential users from seeing unwanted content.

[2] https://www.gov.uk/government/publications/copyright-acts-and-related-laws

Chapter 3

Related Work

This chapter briefly outlines previous work relating to the tasks of language modelling and text classification in the area of music lyrics.

3.1 Lyric Language Modelling

Generally, hobbyist approaches to modelling the language used in lyrics have utilized Markov processes, which have the special property that the future is independent of the past given the present. Naturally, as only the present or recent past (in the case of higher order Markov chains) is considered, these types of models are unable to capture longer dependencies and complex structures that appear in lyrics, such as alliteration and complex rhyme schemes; examples of which are given below:

”Tired of **injustice**
Tired of the **schemes**
Your lies are **disgusting**
What does it **mean**
Kicking me **down**
I gotta get **up**
As jacked as it **sounds**
The whole system **sucks**”

Figure 3.1: Example of complex rhyme in Michael Jackson - Scream. In this example alternate slant rhymes are used following the *ababcdcd* rhyme scheme.

”Hear Her Voice
Shake My Window
Sweet **S**educing **S**ighs”

Figure 3.2: Example of alliteration in Michael Jackson - Human Nature. In this example alliteration of the letter S is used in the last line.

Recent attempts to model creative writing have utilised neural methods, such as Zhang and Lapata (2014), who uses RNN's for Chinese poetry generation and Potash et al. (2015), who uses an LSTM, to *'ghostwrite'*[1] rap lyrics. Similarly in this project, an LSTM is used to train multiple covariate specific language models.

[1] write in the style of X without giving credit

3.2 Lyric Genre Classification

Previous attempts at genre classification predominately use audio features as opposed to lyrics, (in part due to lyric availability). Apart from audio features, the genre of a song is also connected to its lyrics as songs of varying genres use language differently. This can be seen in Hip Hop, where the use of slang, which are twisted existing words or self-made words (Edwards and Rap (2009)) is prevalent. Examples of such words can be seen in Table 3.1

Slang Word	**Meaning**
grill	teeth
lit	fun
bread	money

Table 3.1: Example slang words found in the dataset used in the project

Similar to the previous section, genre classification has also been tackled using neural methods. Using audio based features, (Irvin et al. (2016)) and (Chun Pui Tang (2018)), utilise LSTM's for classification. In relation to song lyrics, Tsaptsinos (2017) utilises lyrics to train a Hierarchical Attention Network (Yang et al. (2016)), which provide state of the art results for document classification due to their ability to encode its knowledge about the composites of a document in a hierarchical way. In the same paper, a baseline LSTM is implemented for comparative reasons. In relation to this project, the aforementioned baseline LSTM forms the basis architecturally for both the language and classification models. Rational and detail for this decision is discussed more in chapter 7.

Chapter 4

Background

This chapter provides an introduction to the theory and previous work within the areas of word embeddings, neural language models and text classification.

4.1 Word Embeddings

Word embeddings are vectors of predefined size which aim to encode a distributional numerical representation of word features. Previous techniques for creating such representations can be categorised into two categories: matrix factorization methods and shallow window-based methods.

4.1.1 Previous Methods

Global Matrix Factorization Methods

Global matrix factorization methods such as Latent Semantic Analysis (LSA) use low rank approximations to decompose large matrices containing corpus statistics. Typically, these matrices take the form of a term-document matrix, which captures the frequencies of terms across a collection of documents, or a term-term matrix, which store co-occurrence counts of terms. Matrix factorisation methods such as LSA allow for fast training and perform well on word similarity tasks by leveraging word occurrence statistics however they suffer from the disproportionate importance given to large word counts.

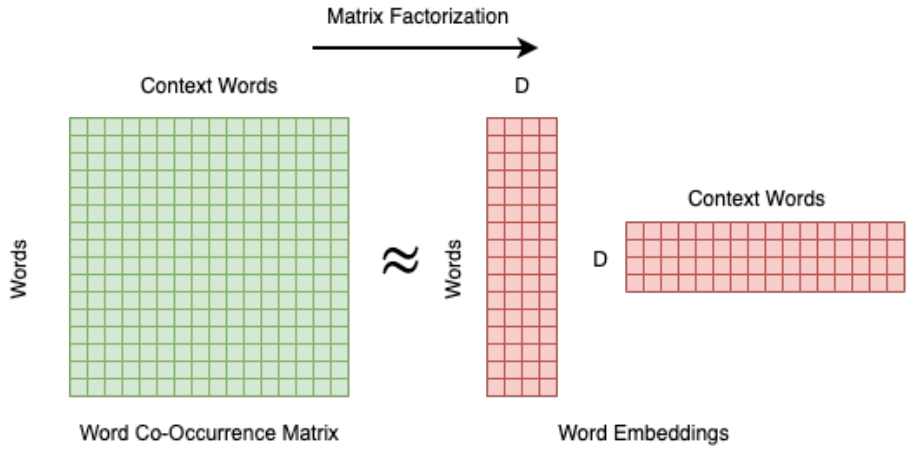

Figure 4.1: Matrix factorization of a word-word co-occurrence matrix which approximates the co-occurrence matrix as the matrix multiplication of two word vectors with dimension D.

Shallow Window-Based Methods

Shallow window-based methods provide an alternative approach to learning word representations by sliding a fixed window over the contents of a corpus and learning to predict either the surroundings of a given word (skip-gram model) or predict a word given its surroundings (continuous bag of words (CBOW)). In the case of shallow window-based methods, they are good at capturing more complex patterns and do well in the word analogy task, however they fail to leverage global statistical information such as those used in global matrix factorization methods.

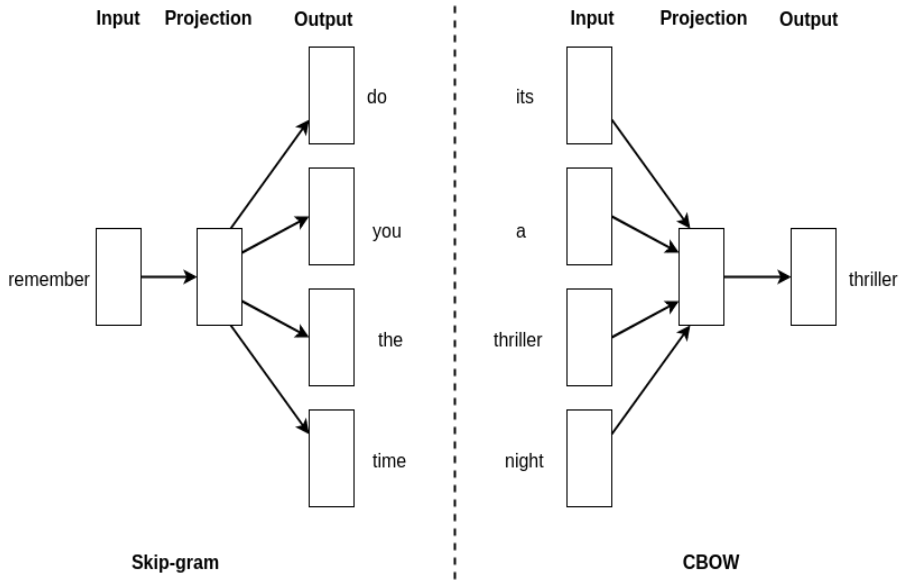

Figure 4.2: Skip-gram models attempt to predict the proceeding and successive words of a given target word, whilst CBOW models attempt to predict a target word given its surrounding words

4.1.2 GloVe

Global Vectors for Word Representation (GloVe), is an unsupervised word embedding algorithm, introduced by Pennington et al. (2014), which marries the benefits of both global matrix factorisation and shallow window-based methods. Presented as a log-bilinear regression model, GloVe makes use of global word co-occurrence statistics from a corpus. Specifically, given a corpus, which has vocabulary size N, a word-word co-occurrence matrix X, with NxN dimensions, is computed with X_{ij} being a measure of the number of times words i and j co-occur within a given context window, which in GloVe is a inverse distance function, giving closer words higher counts. For example applying a right context window of size 3 over the sentence,

I'm looking at the man in the mirror

and assuming the focal word was "man", the counts for its context words in the window would be computed as follows,

$$\text{in: } 1, \quad \text{the: } \frac{1}{2}, \quad \text{mirror: } \frac{1}{3}$$

As detailed in the paper, GloVe outperformed previous methods such as word2vec in word analogy, word similarity and named entity recognition tasks, and as seen in Figure 4.3 is able to capture linguistic substructures.

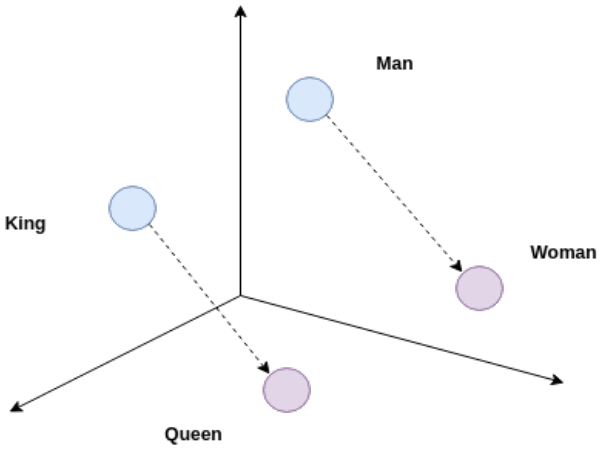

Figure 4.3: Example of the male-female relationship encoded in the word embedding space.

Conceptually, GloVe is based on the idea that ratios of probabilities of words co-occurring with respect to another word, can encode more meaning than using the raw probabilities of the words co-occurring itself.

This concept is formalised in the following equation, where the dot product of focal and context word vectors, w and $\tilde{w}$ and their associated biases, b_i and $\tilde{b}_j$, is approximately equal to the logarithm of the number of times the words co-occurred, $\log X_{ij}$.

$$w_i^T \tilde{w}_j + b_i + \tilde{b}_j \approx \log {X_{ij}}^2 \tag{4.1}$$

GloVe also utilises a weighting function f to decrease noise caused by rare word co-occurrences. The following weighting function is used in the GloVe model.

$$f(x) = \begin{cases} (x/x_{max})^\alpha, & \text{if } x < x_{max} \\ 1, & \text{otherwise} \end{cases} \tag{4.2}$$

Typically values of 0.75 and 100 are used for both α and x_{max} respectively. Combining equations 4.1 and 4.2, the GloVe model is defined as a weighted least squares regression problem.

$$J = \sum_{i,j=1}^{N} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2 \tag{4.3}$$

Training of GloVe utilises the AdaGrad optimiser (Duchi et al. (2011)) to perform stochastic gradient descent, which is an iterative method of optimising an objective function using small random samples of training data, in order to minimise the objective function above.

4.1.3 CoVeR

Covariates such as author demographics, time and location often accompany documents within a corpus. A trivial approach to obtaining covariate specific word embeddings involves applying GloVe to each subset of corpus documents relating to a particular covariate. Unfortunately, utilising GloVe this way has certain drawbacks. Firstly, GloVe must be applied and optimised to each subset of corpus data, which, depending on the number of covariates, is computationally expensive. Also as a direct result of dividing the original corpus, global co-occurrence counts are now split across covariates. Consequently, global statistics are now not shared between embeddings, which may cause sub-optimal word representations, particularly when subsets of corpus data contain minimal amounts of co-occurrences for GloVe to train on.

Another issue with applying GloVe in this manner is interpretability. Apart from visual interpretation through dimensionality reduction and clustering as shown in Figure..., traditionally, interpreting the meaning of word vector dimensions has been challenging because of their random initialisation and due to the fact that dimensions are constructed relative to the order in which co-occurrence statistics are processed during training. As a result of this, applying GloVe this way makes it more challenging to relate the separate covariate word embeddings back together.

Learning Covariate-Specific Vector Representations with Tensor Decompositions (CoVeR), proposed by Tian et al. (2018), provides an alternative to the conditional GloVe method which offers a framework to making learned embeddings more interpretable and the learning process more data efficient. Being an extension of GloVe, CoVer extends Glove's matrix decomposition of a co-occurrence matrix X_{ij} to a tensor decomposition of a co-occurrence tensor X_{ijk} , involving the joint learning of word embeddings w and $\tilde{w}$, a set of k covariate specific diagonal transformation matrices c, which represent the effect of a particular covariate on the base embeddings learned and associated biases b_{ik} and $\tilde{b}_{jk}$ for words w_i and $\tilde{w}_j$ in covariate k . Using the same weighting function as the GloVe method, the CoVeR model is presented below

$$J = \sum_{i,j=1}^{N} \sum_{k=1}^{M} f(X_{ijk})((c_k \odot w_i)^T (c_k \odot \tilde{w}) + b_{ik} + \tilde{b}_{jk} - \log X_{ijk})^2 \tag{4.4}$$

The introduction of covariate specific weight matrices into the objective function allows the authors to create covariate specific word embeddings by multiplying the base embeddings w and $\tilde{w}$ by the covariate diagonal transformation matrix c_k. This approach overcomes the limitations of data efficiency and interpretability in the conditional GloVe method by utilising global corpus statistics of all documents within a corpus, and providing a way to transform the base embeddings learnt into covariate specific embeddings through a covariate specific transformation matrix.

The following sections introduce the two research areas which relate to the project goals, namely language modelling and text classification.

4.2 Language Models

Formal languages such as programming languages are fully specified with precise syntax and semantics which dictate the usage of all words reserved within the language. Contrarily, natural languages, because of their emerging nature, are unable to be formally specified even with the existence of grammatical rules and structures. Unfortunately, rule

based systems for modelling language suffer from the endless possibilities of language usage outside of grammatical rules which are easily interpretable by humans. Moreover, the task of consistently updating rule based systems to accommodate such usage is unfavourable.

Language modelling (LM) is the task of estimating the probability distribution of various linguistic units such as characters, words and sentences. In recent years, the application of LM has been essential to many NLP tasks such as Speech to Text and Text Summarization (Rush et al. (2015)). Language models can be classified into two categories, count-based and neural language models.

4.2.1 Count Based Models

Count based models attempt to learn a probability distribution $P(w_1, ..., w_i)$ over of a sequence of words $w_1, ..., w_i$. An example of a count based model is the n-gram model.

N-gram models

An n-gram is a sequence of n words. Examples of two word sequences or bigrams include, *"Annie are"* and *"you okay"*, whilst examples of three word sequences or trigrams, include sequences of words such as *"been hit by"* and *"a smooth criminal"*. The n-gram model considers the past $n-1$ words and uses Markov assumptions to approximate the probability of word sequences $P(w_1, ..., w_i)$. This is formalised in the following equation

$$P(w_i|w_1, ..., w_{i-1}) \approx P(w_i|w_{i-n+1}, ..., w_{i-1}) \tag{4.5}$$

An inherent problem with the n-gram model is sparsity as some word sequences occur rarely or not at all, even in large text corpora. Using the standard n-gram model would yield too many zero probabilities. To mitigate this, techniques such as back-off (Katz (1987)) and smoothing exist. Another disadvantage of the n-gram model is that they rely on exact matches, meaning they fail to recognise syntactically and semantically similar sentences such as *"what about animals"* and *"what about elephants"*. N-gram models also suffer from the curse of dimensionality due to increased vocabulary sizes. As a result, limited window sizes are used, causing longer dependencies between words not be captured.

4.2.2 Neural Language Models

To overcome issues faced by count based models, deep learning methods have been used to create neural language models by simultaneously learning word embeddings and the parameters needed to create a joint probability distribution between the word embeddings. Bengio et al. (2003) proposed a feed forward neural language model to help tackle the problem of data sparsity. Recent state of the art approaches such as Mikolov et al. (2010), abstract language modelling as a form of sequential data prediction and have implored RNN's to help encode longer dependencies between sequences of words. The strength of these models comes from their ability to consider several preceding words and thus generalise well.

An overview of generic neural network architectures as well as RNN's and LSTM's are given in the following sections.

Artificial Neural Networks

In any neural network architecture, the elementary unit of computation is the artificial neuron which takes inspiration from biological neurons. The artificial neuron receives N inputs which are each weighted by N weights and summed together with a bias b. The output y of a neuron is calculated by passing the weighted sum of the inputs into an activation function f.

$$y = f\left(\sum_{i=1}^{N} x_i w_i + b\right) \tag{4.6}$$

Typical activation functions include the *Sigmoid*, *Tanh* and *ReLu* functions. The equation above serves as the formula for the simplest king of network: a single layer network, also known as a perception. The process of stacking layers on top of each other leads to multi-layer neural networks. These types of networks are also known as feed-forward networks. The learnable parameters of these networks are the set of weights and biases for each layer. A feed-forward neural network is trained using gradient descent and its parameters are updated using the *backpropagation* algorithm (Rumelhart et al. (1988)).

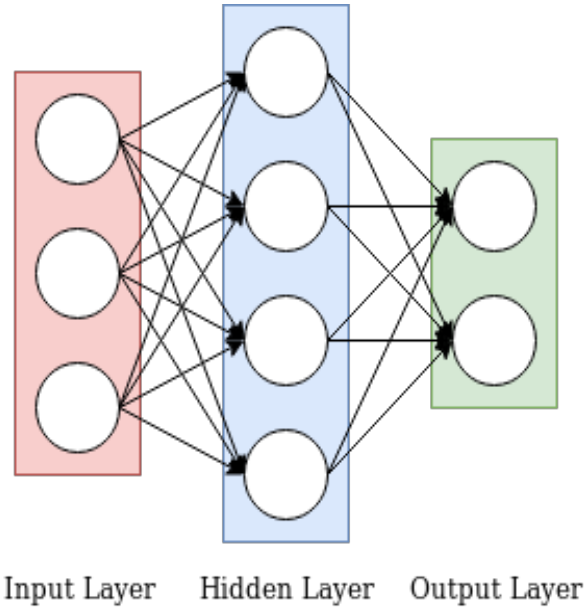

Figure 4.4: Example of a fully connected feed-forward neural network with three input neurons, four neurons in the hidden layer and two output neurons.

Recurrent Neural Network

In a feed-forward neural network, data flow is unidirectional between layers; with data passing through a given neuron at most once. These types of networks perform well on both classification and regression tasks with the assumption that inputs are independent of each other. In tasks dealing with sequential data, feed-forward networks perform poorly. To model sequential data well, a neural network must be able to model the dependencies that exist between successive inputs. The recurrent neural network (RNN) is an attempt to satisfy this requirement by utilising past inputs to help predict future outputs.

In an RNN information is cycled within the network at least once. An RNN receives a sequence of inputs x and updates its hidden state h_t by

$$h_t = \begin{cases} 0, & \text{t} = 0 \\ \phi(h_{t-1}, x_t), & \text{otherwise} \end{cases} \tag{4.7}$$

where ϕ is a nonlinear function such as *tanh* or *ReLu*. The update for the hidden state is usually implemented as

$$h_t = \phi(Ux_t + Wh_{t-1}) \tag{4.8}$$

where W and U are weight matrices.

RNN's are trained using gradient descent and backpropagation through time (BBTT), which is identical to performing backpropagation on an *"unrolled"* RNN (seen in figure Figure 4.5)

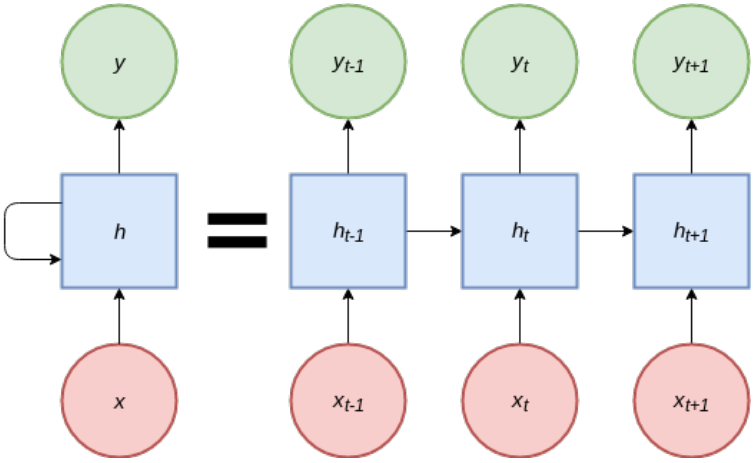

Figure 4.5: An *unrolled* recurrent neural network can be seen as a feed-forward neural network with many hidden layers.

During BBTT, back propagation is performed on an unrolled recurrent architecture, causing gradients to back-propagate through numerous network layers. Unfortunately, this has a few major problems. Firstly, a single forward/backward pass through the network is computationally expensive due to the number of hidden layers of the unrolled network being linear to the number of time steps in a sequence. Secondly, this method suffers from the issue of vanishing or exploding gradients (Bengio et al. (1994)) where gradients can exponentially decay or grow as they propagate over time, which can prevent the network from learning entirely.

Long Short-Term Memory

Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber (1997)) is a variant of the RNN which is capable of capturing longer dependencies between sequences of data without suffering from vanishing or exploding gradients. This is achieved through a feature known as gating; a mechanism which acts as a permissive or restrictive barrier to information flow.

The core component of the LSTM is the cell state which is able to propagate **relevant** information throughout the network. This is achieved within the memory cell through the forget, input and output gates. The forget gate controls how much existing memory should be forgotten, the input gate controls how much of the new cell state to keep, and

the output gate controls how much of the cell state should be allowed into the next layer of the network. An example of an LSTM memory cell is seen in Figure 4.6.

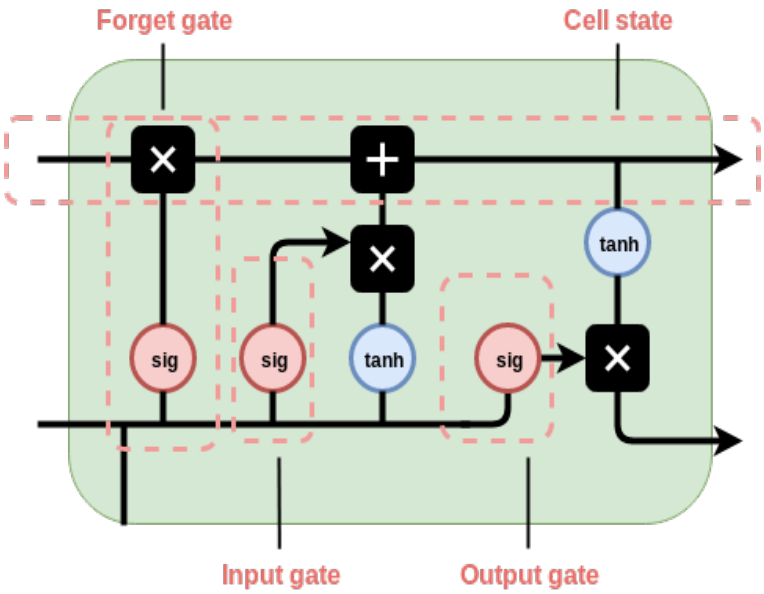

Figure 4.6: LSTM memory cell, with forget, input and output gates

4.2.3 Text Classification

Text classification is the task of assigning pre-defined labels to text according to it's contents. Machine learning methods for text classification are preferred to rule based systems due to their ability to model regardless of domain. Text classification can be approached using supervised learning, where a model is trained with labelled training data, or unsupervised learning, where training data is clustered together to see if any natural grouping exists.

Before a classifier can be trained, inputs must be transformed into numerical representations. A common method for creating such representations is the bag of words (BOW) approach which, given a vocabulary V, with n words, creates a fixed size input vector of length n for document D: representing word counts for each word in the vocabulary. For example, if we define the vocabulary V as being the set of words,

$$\{\text{"we"}, \text{"are"}, \text{"world"}, \text{"children"}, \text{"the"}\}$$

and an input document D as being

$$\textit{"we are the world"}$$

the resulting representation for document D would be

$$[1, 1, 1, 0, 1]$$

Two popular machine learning methods for text classification include the naive Bayes classifier and the Support Vector Machine. They are briefly outlined in the following section.

Naive Bayes

Naive Bayes classifiers are a class of supervised probabilistic classifiers which have been used in text classification for tasks such as Spam Filtering (Sahami et al. (1998)). At their core, these types of classifiers rely on Bayes theorem which is shown below.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{4.9}$$

The *naivety* of these classifiers comes from the assumption of conditional independence between features. Specifically, given an input x, with n features, and a target class y, naive Bayes classifiers use the following approximation to assign the probability of input x belonging to class y.

$$P(x_1, ..., x_n|y) \approx \prod_{i=1}^{n} P(x_i) \tag{4.10}$$

Given multiple classes, the classification of an input is given to the class with the maximum conditional probability as shown in the equation above. This is generalised in the formula below.

$$y = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y) \tag{4.11}$$

Support Vector Machines

Support Vector Machines (SVM) (Corinna Cortes (1995)) have also been applied successfully to text classification problems (Joachims (1998)). Unlike naive Bayes classifiers, SVM's are discriminative classifiers which aim to find a hyperplane in N-dimensional space which maximises the margin between data points of different classes. This is achieved through the use of random data points, namely *"support vectors"*, which are data points closest to the hyperplane which help maximises the margin distance between classes. Maximising marginal distance between classes provides confidence to the classification of future data points.

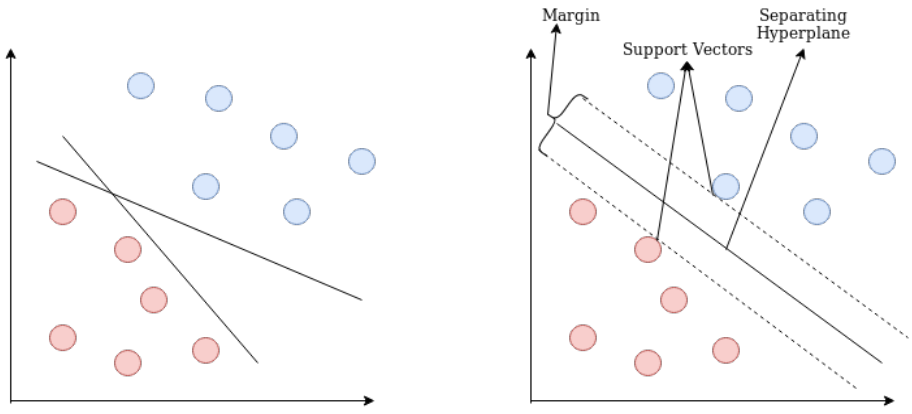

Figure 4.7: Support Vector Machine - The left set of data points can be linearly separated by many lines, but the goal of the SVM on the right is to use the support vectors to maximise the margin between the two classes. The optimal separating hyperplane is in the middle of this maximised margin

Deep Learning Methods

Until recently, state of the art methods for text classification had long used linear predictors such as the SVM (with a linear kernel) and multinomial naive Bayes classifiers using feature representation techniques such as bag-of-words (Joachims (1998) Lewis et al. (2004)). However, non-linear methods that are able to leverage dependencies between words ,such as recurrent neural networks, have provided better predictions (Dai and Le (2015)) than previous bag-of-word approaches where features are assumed to be conditionally independent and unordered.

Chapter 5

Requirements Analysis

As the project involves the development of a prototype software solution, it is important to consider the project from a software engineering perspective. Moreover, the project involves the integration of a novel machine learning model, with the success of the project relying heavily on factors such as data availability, data quality and processing power. With this and other considerations such as training time and implementation complexity in mind, it is necessary to define software requirements to limit the project scope to one that is achievable. Software requirements should also be inferred from the needs of the end-user and as such it is necessary to understand user needs through existing solutions. This chapter briefly evaluates two existing solutions and outlines the functional and non-functional requirements by which the prototype will be evaluated.

5.1 Existing Solutions

5.1.1 MasterWriter

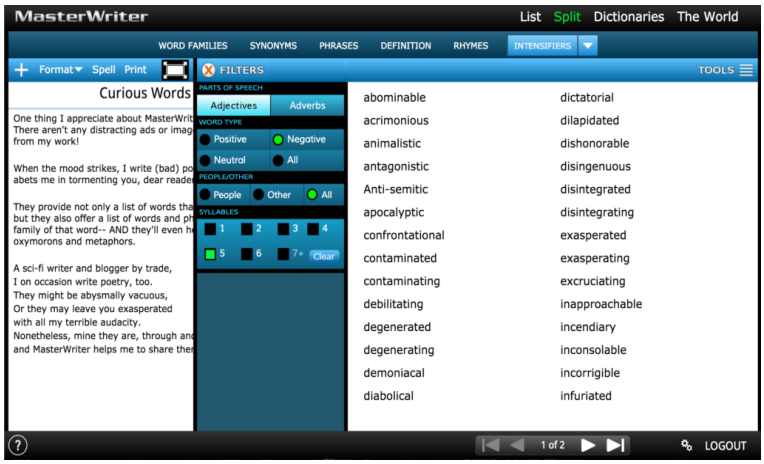

Figure 5.1: MasterWriter user interface

Self-described as ***"The most powerful suite of writing tools ever assembled in one program."***, MasterWriter is a software application aimed towards songwriters, poets and creative writers. Available as a multi-platform product, it consolidates a number of writing aids into one application. Some of the core features of MasterWriter are highlighted in Figure 5.1.1 below.

Feature	Description
Word Dictionary	Provides a word dictionary with word definitions
Rhyming Dictionary	Provides a rhyming dictionary with word/phrase rhymes
Phrase Dictionary	Provides a phrase dictionary of predefined phrases
Word Families	An extension of a typical thesaurus which provides a reference dictionary which can filter on subtle differences in word meanings
Speech Types	Provides a predefined list of different figures of speech
Synonyms	Provides a thesaurus containing synonyms

Table 5.1: MasterWriter Features

5.1.2 Rhymer's Block

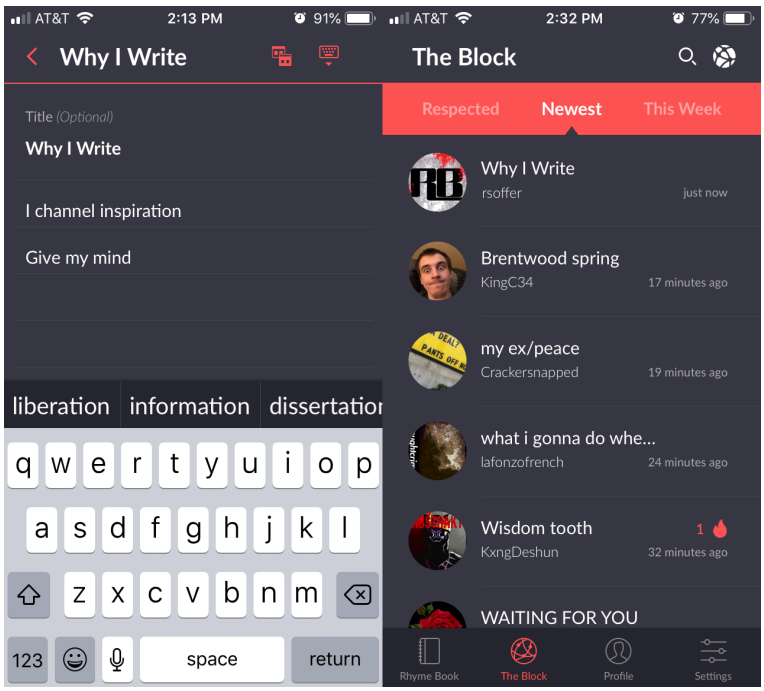

Figure 5.2: Rhymer's Block

Rhymer's block is a mobile application intended to help songwriters specifically with rhymes. Providing real time rhyme suggestions, the application allows users to quickly write song lyrics and provides a social platform through which users can share their own song lyrics, as well as review the song lyrics of other users. An overview of Rhymer's Block core features can be seen in Figure 5.1.2.

5.1.3 Requirements analysis

Both MasterWriter and Rhymer's Block provide functionality for users to write, edit and save their lyrics. Similarly, both applications offer word search functionality to help writers with word choice, a primary issue highlighted before, through features such as rhyme suggestions, word dictionaries and thesauruses. Taking inspiration from both applications and as a basis for SONGIFAI, a lyric editor and word suggestion feature are mandatory for implementation. As previously stated, software applications like MasterWriter are static in nature, and fail to capture the dynamic nature of language. Examining Rhymer's Block,

Feature	**Description**
Real-Time Rhyme Suggestion	Provides real time suggestions of rhyming words
Cloud Storage	Provides cloud storage for a user's song lyrics
Social Platform	Provides a online platform for users to share and comment on song lyrics
Word Dictionary	Provides a word dictionary with word definitions
Thesaurus	Provides a thesaurus containing synonyms
Word Analysis	Provides rhyme suggestions for words most commonly used by other users

Table 5.2: Rhymer's Block Features

a particular feature deviates away from this, namely its real-time rhyme suggestions which leverages word usage within the application's online community. The implementation of a social platform is not in the scope of this project, however Rhmyer's Block usage of its users language is comparable to a goal of this project which is to help songwriters with their own lyrics, leveraging genre specific word embeddings.

Neither MasterWriter or Rhymer's block address the secondary issue raised in chapter 1; namely helping songwriters with classifying their lyrics to help infer instrumental style. Whilst this project only considers three music genres, the potential application of such a feature becomes more apparent when considering sub-genres within music, which as explained in chapter 1 is ever increasing. Though sub genres are out of scope for this project, a basic genre classification feature will be implemented for SONGIFAI which will classify a user's submitted song lyric as either Pop, Rock or Hip Hop.

Comparing MasterWriter and Rhymer's block, software availability and accessibility are dealt with differently. Whilst MasterWriter offers its application on various platforms, Rhymer's Block utilises cloud storage for better accessibility. To address both software availability and accessibility, whilst keeping the project in scope, SONGIFAI will take the form of web application.

As a whole, the project has roots in both research and practice. Often times the interests of both these areas are incompatible. For example, research tends to focus more on advancing the current state of the art without paying too much attention to aspects such as model scalability, overall training time, ease of implementation and quality. On the other hand in practice applications are for the most part client driven meaning aspects such as delivery time become important, and the performance of a model can be suboptimal if it is robust to train and easy to deploy to production. With regards to the project, necessary compromises, from both a research and application stand point have been made in order to achieve the goals of the project. For example a typical software solution would involve an iterative design process from a low fidelity prototype to a high fidelity prototype. As stated before the goals of this project are not client driven and due to further time constraints an iterative design process will not be adhered to. In the same vein, from a research perspective, techniques such as hyperparameter tuning through methods such as grid search will also not be followed; again due to time constraints and the type of models, (RNN's) that take time to train.

Taking into consideration the project goals and what has been discussed above, the project requirements are outlined below.

5.2 Requirements

In this section the requirements for the project are set out. The functional requirements specify what the software will do whilst the non-functional requirements will detail how these will be done. Possible extensions for the project, if time permits are also outlined.

5.2.1 Functional

ID	Description	Dependency
FR1	The system should allow users to input lyrics	N/A
FR2	The system should allow users to edit, save and delete lyrics	N/A
FR3	The system should be able to classify user submitted lyrics as either Pop/Rock/Hip Hop	N/A
FR4	The system should be be able to suggest words from a given word. These words should be the most similar words in the chosen covariate word embedding space	N/A
FR5	The system should be able to provide real time text prediction whilst a user is in edit mode	N/A
FR6	The system should allow for the filtering of explicit content in both the word suggestion and word prediction feature	N/A
FR7	The user should be able to change the underlying covariate specific word embeddings used or the base embeddings if desired	N/A

Table 5.3: SONGIFAI Functional Requirements

5.2.2 Non-Functional

ID	Description	Dependency
NFR1	The system should take the form of a web application, compatible on multiple device types	N/A
NFR2	The word prediction feature should return a list of candidate words with minimal latency	N/A
NFR3	The word suggestion feature should return a list of suggested words with minimal latency	N/A

Table 5.4: SONGIFAI Non-Functional Requirements

5.2.3 Extensions

ID	Description	Dependency
E1	The system may allow rhyme words to be suggested in the word suggestion feature if time permits	N/A
E2	The system may allow sub genres to be selected for embeddings, if time and data permits	N/A

Table 5.5: SONGIFAI Possible Extensions

Chapter 6

Methodology

This chapter describes the methodology used throughout the project as well as rational behind certain methodological decisions.

6.1 Collecting Data

As previously mentioned, a central repository for song lyrics, where downloading is permissive, is non-existent. Collection of song lyrics from sites such as Genius[1] is only achievable through web scraping: a process which involves the exhaustive downloading and processing of web pages from either a predefined list of URLs or the recursive process of link extraction and following (commonly known as *web crawling*) from a given seed URL. When scraping at scale, restrictions such as the robots exclusion protocol, which specifies areas of a website that are allowed to be processed, and crawl rate, which specifies the minimum delay between requests, make data collection via this method a troublesome task. Taking into account the project aims and constraints such as time, web scraping was deemed unfavourable and hence avoided for data collection.

To fulfil the data requirements of the project, a public dataset [2] containing over 250,000 song lyrics was used. The dataset, which was scraped from Brazilian music website Vagalume.br, comprised of two *.csv* files: artists-data and lyrics-data. The artist-data file included metrics such as popularity, genre and number of songs per artist, whilst the lyrics-data file contained song lyrics for individual songs by artists.

6.2 Data Analysis and Restructuring

The CoVeR algorithm requires each document within a corpus to be associated with a covariate in order to jointly learn word embeddings and the relevant transformation matrices. To satisfy this requirement, Pandas, a Python data analysis library, was used to perform a join operation on the artists columns in both the artists-data and lyrics-data files. The result of this operation followed by the additional dropping of redundant columns resulted in the dataset used throughout the rest of the project.

A trivial approach to divide the dataset on genre would involve an equal split for equal representation, however, this method has the assumption that for each genre, word count per song is equivalent. Examining the dataset however, proved this not to be the case.

[1] https://genius.com/
[2] https://www.kaggle.com/neisse/scrapped-lyrics-from-6-genres

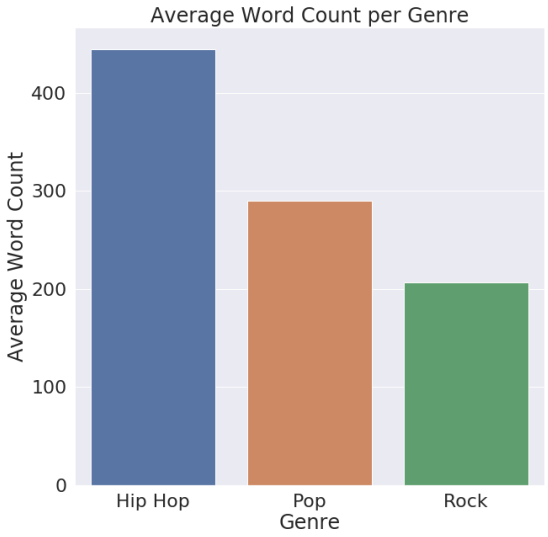

Figure 6.1: Average song word count by genre for all lyrics within the dataset.

Analysing the dataset, the mean number of words contained in a Hip Hop song was 444; being more than double the mean number of words found in Rock songs, which had a count of 207. With regards to the Pop, the mean word count per song was 289. To reflect these statistics and also due to constraints in the distribution of data by genre, 100,000 song lyrics were randomly selected to create the dataset to be used to train CoVeR. Moreover, a data split of 48:30:22 for Rock, Pop and Hip-Hop, was to maintain an even distribution of words by song genre within the dataset.

6.3 Data Pre-processing

Essential to any machine learning task is the pre-processing of input data in such a way that important features are accessible during training. In NLP, this can include techniques such as tokenisation, string cleaning, stemming and lemmatisation.

Following reconstructing of the original dataset, each song lyric in the dataset was cleansed and tokenised. The following string cleaning techniques were applied to each song lyric.

1. All characters, except for letters, were substituted with a space.
2. All letters were lowercased.
3. All text between curly and square brackets were removed. (This was to ensure text like **[Verse 1]** was not included during training).
4. All trailing white space was removed.

Tokenisation is the process of separating textual inputs into meaningful chunks called *tokens*. Naturally to create word embeddings, text is tokenised at the word level and each token is assigned a unique integer key to map the token to its future word embedding.

6.4 Hyperparameters

In machine learning hyperparameters are parameters that govern a given model. The selection of these parameters directly impacts the performance of a given training algorithm and as such, hyperparameter choice is important for producing optimal performance for a given model. Commonly, techniques such as grid search, random search and manual search are used to find optimal optimal parameters for a given training algorithm. However, due to project time constraints, hyperparameter search was limited.

Both the CoVeR algorithm and LSTM networks have important hyperparameters which are reviewed in the following sections

6.4.1 CoVeR Hyperparamters

Removing Rare Words

Word embedding algorithms such as word2vec typically remove rare occurring words within a corpus before training takes place. In the case of word2vec, rare words are removed *before* the computation of co-occurrence statistics. This method of removal has the effect of generating context windows differing to what would have been created if the corpus had remained intact. In turn, this produces different word co-occurrences and hence a different co-occurrence matrix. For example, applying a symmetric context window of size 1 over the following sentence

"We're gonna rock the night away"

and assuming the focal word was *"the"*, the original context words in this window would include *"rock"* and *"night"*. However, suppose the words *"rock"* and *"night"* were deemed as rarely occurring words and removed, the sentence above would then be processed as

"We're gonna the away"

and examining the same focal word as before, the new context words in this window become *"gonna"* and *"away"*.

Neither the CoVeR or GloVe paper, discuss the explicit removal of rare words, however many implementations of GloVe, for the sake of computation, restrict the vocabulary of a corpus to the top N most words. For this project this number was 20,000.

Subsampling

Typically found within text corpora are high frequency stop words which provide less information than infrequent or domain specific words (Mikolov et al. (2013b)). For example, from a music related corpus, co-occurring examples of the words "king" and "pop" (referring to Michael Jackson), are more beneficial to a model then examples of co-occurring words "the" and "king". This is because "the" would co-occur with the majority of words within the corpus. This concept of oversaturated co-occurrences can also be applied to word embeddings; where the word embeddings of frequent words does not change significantly after training on several examples. Existing methods for removing high frequency words involve the complete removal of stop-words during preprocessing, however, for the sake of this project, where the appearance of such words is necessary to create a language model, subsampling is used instead. Subsampling is the process of sampling infrequent words more often than frequently occurring words in order to create new training data. In the case of word embedding algorithms, subsampling increases training speed by randomly

removing very frequent word co-occurrence which provide little to no extra benefit to the model. Taking inspiration from word2vec, subsampling was used at the covariate level using the following adapted formula from the original word2vec implementation.

$$P(w_{ik}) = \sqrt{\frac{z(w_{ik})}{t}} + 1 \cdot \frac{t}{z(w_{ik})} \tag{6.1}$$

where $z(w_{ik})$ is the percentage of word w_{ik} in covariate k and t is a chosen threshold. For this project the threshold that worked well was 0.05.

Context Windows

CoVeR uses the same weighted context windows strategy as GloVe during the process of calculating co-occurrence statistics. During experiential training, a context window size of 8 is used in the paper however, the paper does not detail the type of context window used. Typically, context windows can either be asymmetric, where only preceding words are considered or symmetric, where words are considered on either side of a given focal word. An experimental finding in the original GloVe paper states that small and asymmetric context windows work well for syntactic tasks, whilst long and asymmetric tasks are good for semantic tasks. Relating back to one of the functional requirements, specifically requirement FR4, the resulting word embeddings will be used for a word suggestion feature, where suggested words correspond to the closest words in the embedding space. This requirement can be considered more of a semantic task than a syntactic task due the feature being synonymous with a thesauruses for the sake of providing words with similar meanings. For this reason, a symmetric context window is used for this project.

Embedding Size

Common embedding sizes for models such as GloVe and word2vec, range in dimensionality with typical values being between 50-300 dimensions. Currently there is no empirical method in selecting the dimension size for word embeddings. For this project, word similarity, discussed in chapter 8 was an important measure to evaluate the semantic quality of the word embeddings. Dimensionality values 50, 100, 200 yielded similar results word similarity results as shown in section 6.4.1.

Focal Word	Number of Dimensions		
	50	100	200
king	princess, queen, prince	queen, princess, hearts	queen, prince, hearts
drink	pass, beer, bottle	drink, somebody, pass	bottle, pour, lets
lord	praying, pray, child	knows, praying, please	knows, please, praying
fire	burning, fireball, desire	fireball, burning, desire	fireball, burning, light

Table 6.1: Comparison of the three most similar words to multiple focal words by dimension. The differences between dimensions was minimal and as such 50 was the chosen dimension size to increase training speed.

For the purpose of this project and to help speed up model training for later tasks, an embedding dimension of size 50 was chosen for this project.

Optimiser

Similar to the GloVe method, the CoVeR algorithm utilises stochastic gradient descent to minimise its objective function. Rather than using Adagrad, CoVeR utilises the Adam

optimiser (Kingma and Ba (2014)). For this project the same optimiser is used to train the CoVeR embeddings.

6.4.2 LSTM Hyperparameters

Both the language model and text classifier used in this project take advantage of an LSTM. Structurally, both models are the same: making use of an embedding layer, a recurrent layer, and a dense layer with a softmax activation function to help it predict the probaiilty distributioon. The general architetcure of both models can be seen below in ...

Dropout

Dropout is a regularization technique proposed by Srivastava et al. (2014), which involves the random dropping of neurons and their connections within a network, omitting them during a particular forward or backward pass. Selected neurons are picked at random using a probability known as the dropout rate. Dropout helps to decrease over-fitting as it prevents neurons from co-depending on one another as they must learn to work with a random sample of other neurons rather than the same fixed ones. This in turn forces the neuron to learn useful connections rather than relying on other neurons for correction. As utilised in Srivastava et al. (2014), dropout is applied at both the input and hidden layers, with a rate of 0.2 and 0.5 respectively.

Embedding Layer

For both the language model and the text classifier, the input layer to the network is the embedding layer which transforms words, represented as indices, into word embeddings. These embeddings can either be initialised randomly as one-hot vectors and trained with the network or they can be initialised using pre trained word embeddings, which can either be left as is or trained further. For this project the word embeddings derived from CoVeR were used as the weight initialiser for the embedding layer, and were not allowed to be trained. The motivation behind this was to see how well the genre specific word embeddings contributed towards creating genre specific language models, as well as how well they contributed towards the task of binary text classification, discussed in chapter 8.

Optimiser

Chapter 7

Implementation

Implementation of the project predominately utilised the Python[1] programming language. The motivation for using Python originated from prior familiarity with the language, and it's collection of machine learning and web development libraries which form two key components of the project. Though Python allows for relatively quick development, its known drawback of speed was a cause of concern.

As stated in chapter 5, in research, model training time is often neglected in favour of high performing models. As this project involves the creation of a prototype software application, it was important that the impact of model training time on overall development was minimal.

This chapter describes the development process for implementing the CoVeR algorithm, the language and text classification models as well as the SONGIFAI web application.

7.1 Hardware Specification

All implementation was completed on a personal machine. The hardware specifications for the machine are highlighted below

Hardware Component	Specification
CPU	Intel Core i7-8750H CPU @ 2.20GHz x 12
GPU	NVIDIA GeForce GTX 1050 Ti 4GB
RAM	16GB

Table 7.1: Hardware specification for machine used throughout development

7.2 Calculating Co-occurrence Statistics

Many unsupervised NLP methods compute co-occurrence statistics before learning takes place. Typically, co-occurrence statistics, such as GloVe's word-word co-occurrence matrix contain sparse data, and computing them can often be a computationally more expensive task than the learning itself. The original GloVe paper describes this process as a *'one-time upfront cost'*, with the assumption that selected corpora are static. Unfortunately, for many NLP pipelines such corpora are more dynamic in nature. For example, social

[1] https://www.python.org/

data from online platforms such as Twitter[2] are in constant flux and relying on pre-computed co-occurrence statistics to represent Twitter based word embeddings is sub-optimal. Compared to GloVe, computing the co-occurrence statistics for CoVeR has added complexity due to the transition from a co-occurrence matrix to a co-occurrence tensor.

Methods for efficient computation of co-occurrence statistics include the usage of distributed computing techniques such as *MapReduce*. MapReduce is a model for distributed computing which at its crux involves two functional processes: namely the mapper and the reducer. During the *map* process, data is taken in as key/value pairs and transformed to intermediary key/value pairs as output. These are then passed to the *reduce* process which aggregates data which share the same key. An example MapReduce process is walked through in Figure 7.1 below

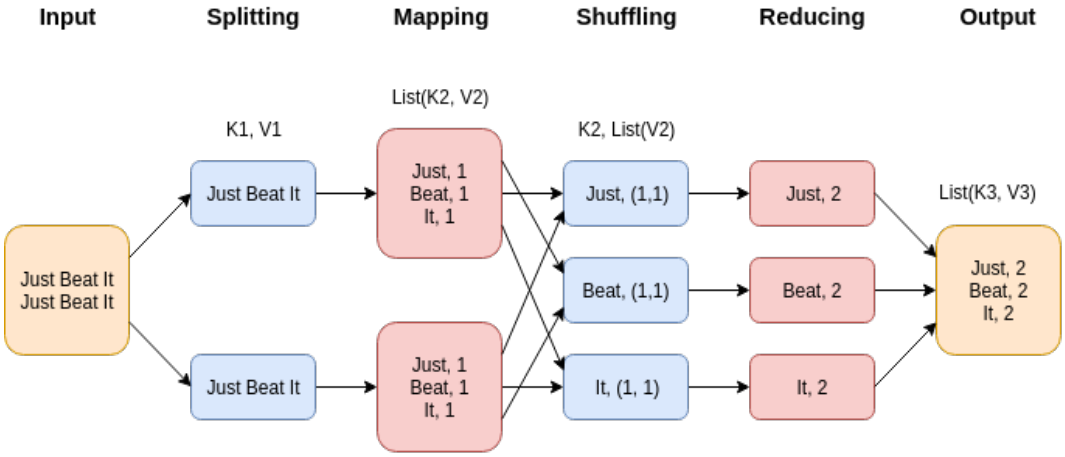

Figure 7.1: MapReduce word count example:

Apache Spark is an open-source framework, written in Scala, for distributed computing and has recently emerged as the de-facto choice for big data processing over Apache Hadoop. Like Hadoop, Spark also supports the MapReduce programming paradigm but boasts features such as enhanced speed, a distributed data structure, as well as API's written in multiple programming languages. Spark uses a master/slave architecture to achieve distributed computing. The *driver* acts as the master node and distributes tasks to many different worker nodes, also known as *executors*, which each run their own JVM processes to execute tasks.

PySpark, a Python API for the Spark framework was initially used to both pre-process and calculate co-occurrence statistics for the dataset. Unfortunately, the overhead of collecting completed executor tasks to the driver as well as the cross language communication between Python and Scala made PySpark an unfavourable option for collecting co-occurrence statistics. A similar parallelised approach which avoided cross language communication involved the use of Python's multiprocessing module. This approach suffered from the restrictions of Pythons Global Interpreter Lock (GIL) which prevents shared access of Python objects across multiple threads.

Cython is a superset of the Python programming language which aims to provide C like performance whilst maintaining the ability to write Python like code. Native Python

[2]https://twitter.com/

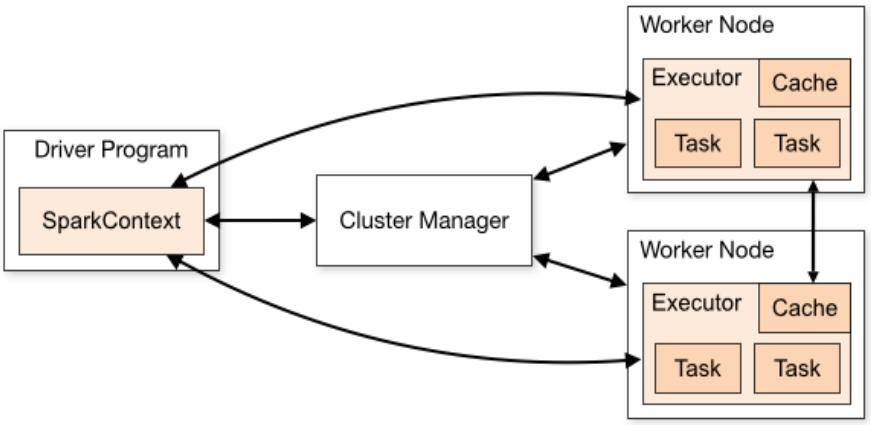

Figure 7.2: High level view of the Spark Architecture. The spark context is where the main program is defined, which is then split into tasks to be completed via numerous executors.

programs can experience major speed improvements using Cython because of its ability to compile Python to C code. Ultimately, calculating the co-occurrence statistics for CoVeR was achieved using Cython which provided major speed ups compared to the parallelised approaches outlined before. A comparison of the speed gain for computing co-occurrences can be seen in the figure below.

7.3 CoVeR Implementation

At time of writing, no public implementation of CoVeR is available. As a result and to meet the needs of this project, CoVeR was implemented from scratch using the PyTorch library. PyTorch is a Python library based on Torch, which supports Numpy like operations which can be accelerated through the GPU. All supporting code for the implementation can be found here: (LINK TO CODE)

7.4 LSTM Implementations

Implementation of both the neural language model and text classifier was done using Keras. Keras is a high level machine learning library written in Python, which runs on top of either Tensorflow or Theano. The motivation behind using Keras comes from its ease of use to quickly develop deep learning networks. In this project Keras is deployed using Tensorflow as a backend, specifically for its GPU capabilities.

7.4.1 Language Model

The structure for the language model can be seen in Figure X.X. The model consisted of an input layer, an embedding layer, a bidirectional LSTM, a dropout layer and finally a dense layer.

7.4.2 Text Classifier

The structure for the text classifier can be seen in Figure X.X. The model consisted of an input layer, an embedding layer, a bidirectional LSTM, a dropout layer and finally a

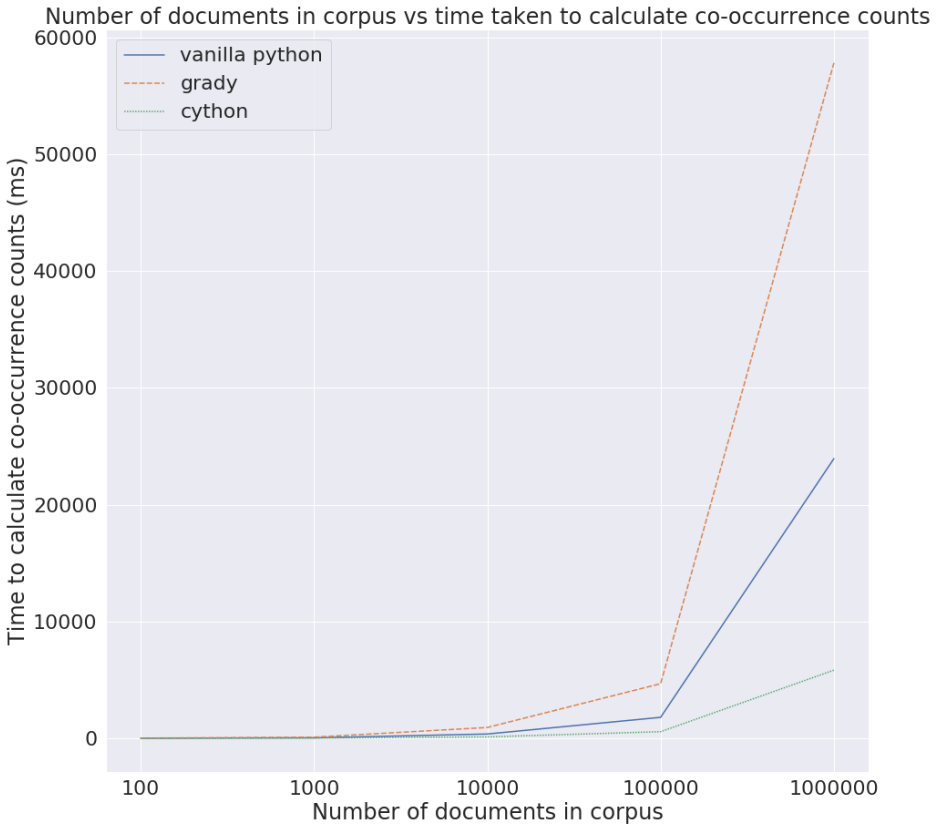

Figure 7.3: Benchmarks

dense layer.

7.5 SONGIFAI

7.5.1 Architecture

Client Side

For the client-side development of SONGIFGAI, a main requirement refers to the systems availability for web and mobile access. Being a prototype solution, it was important that the development was swift and well structured so that the research goals of the project were not hindered. To help achieve this, ReactJS was chosen as the front-end development framework.

React is a Javascript framework for building user interfaces originally developed and maintained by Facebook. The main advantages of using React

Server Side

Requirements ... refer to a user of the system being able to save, load and edit their lyrics. Moreover for easy compatibility with the Keras generated models, another Python based library was preferred as the for the server side. To meet these conditions, Django

was chosen as the development framework for the back-end of the system. Django is a python based web framework which follows the model-view-template (MVT) architectural pattern.

7.5.2 Class Overview

Chapter 8

Evaluation

8.1 CoVeR Evaluation

8.1.1 Validating Implementation

Base Embeddings

As previously stated, there currently does not exist any public implementation of the CoVeR algorithm. As a basis for validating the quality of the CoVeR model created in this project, the base embeddings generated were compared against embeddings generated using the Glove algorithm. In order to achieve this, a simple framework was set up to compare the the most similar words for each word in both the CoVeR and GloVe embeddings. For this experiment, each model was generated by using a symmetric context window of size 8 and each had a dimensionality of 50. Staying true to both papers, the Adam optimiser was used to train the CoVeR algorithm, whilst the AdaGrad optimiser was used to train GloVe. Both models were trained for 10 epochs.

To find the most similar words for a given word in both the CoVeR and GloVe model, the cosine similarity measure was used.

$$sim(A, B) = \frac{A \cdot B}{\|A\|\|B\|} \tag{8.1}$$

Covariate Specific Embeddings

The original CoVeR paper validates the quality of the learned covariate weight matrices by clustering. This was done

8.2 Model Evaluations

8.2.1 Language Model

Typically, language models can be evaluated using either extrinsic or intrinsic methods. Whilst extrinsic evaluation methods measure the performance of a model when applied to an application, intrinsic methods allow for the evaluation of a model independent of any application. A drawback of using extrinsic evaluation is the that it is computationally expensive, as it involves the the full deployment of an application to a task, which then has to be evaluated using

Intrinsic Evaluation

A typical intrinsic evaluation metric used to evaluate a language model is perplexity(REF 19, 30, 43). Perplexity is a measure of how well a given language model will predict test data. The general formula for perplexity is given below

From this

8.2.2 Text Classification

In order to further evaluate the covariate specific word embeddings produced by CoVeR, each covariate embedding was used to train a binary classifier that would discriminate lyrics as either belonging to that particular covariate or not. These were compared with GloVe trained word embeddings to see if the the covaraiets specific word embeddings did indeed capture the semantics of the genre better than generic glove embeddings. The results of the test can be seen in ... below

Embedding	**Training Loss**	**Training Accuracy**	**Validation Loss**	**Validation Accuracy**
GloVe	0.5807	0.6906	0.6194	0.6698
Pop Covariate	0.5791	0.6925	0.6122	0.6730

Table 8.1: GlovePopClass

Embedding	**Training Loss**	**Training Accuracy**	**Validation Loss**	**Validation Accuracy**
GloVe	0.5251	0.7238	0.5691	0.6985
Rock Covariate	0.5200	0.7260	0.5646	0.7009

Table 8.2: GloveRockClass

Embedding	**Training Loss**	**Training Accuracy**	**Validation Loss**	**Validation Accuracy**
GloVe	0.4511	0.7926	0.5115	0.7760
Hip Hop Covariate	0.4593	0.7878	0.4985	0.7823

Table 8.3: GloveHipHopClass

8.3 SONGIFAI

8.3.1 Requirements Evaluation

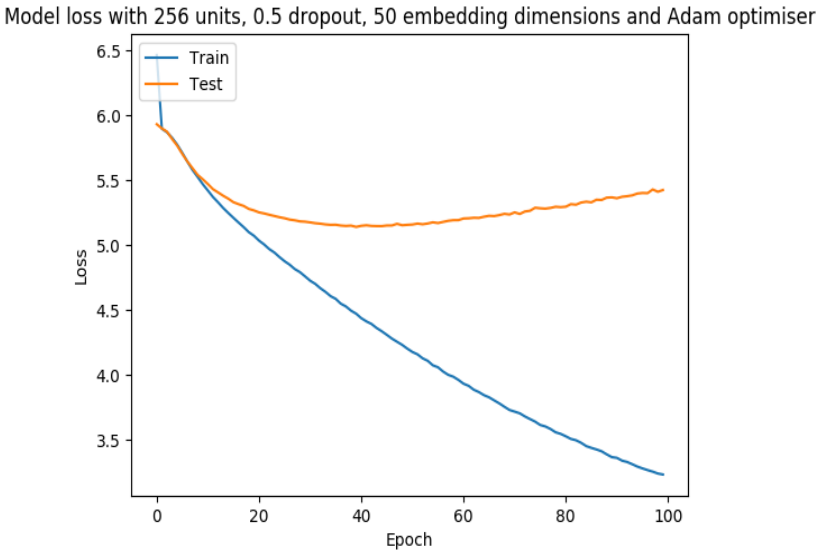

Figure 8.1: poploss

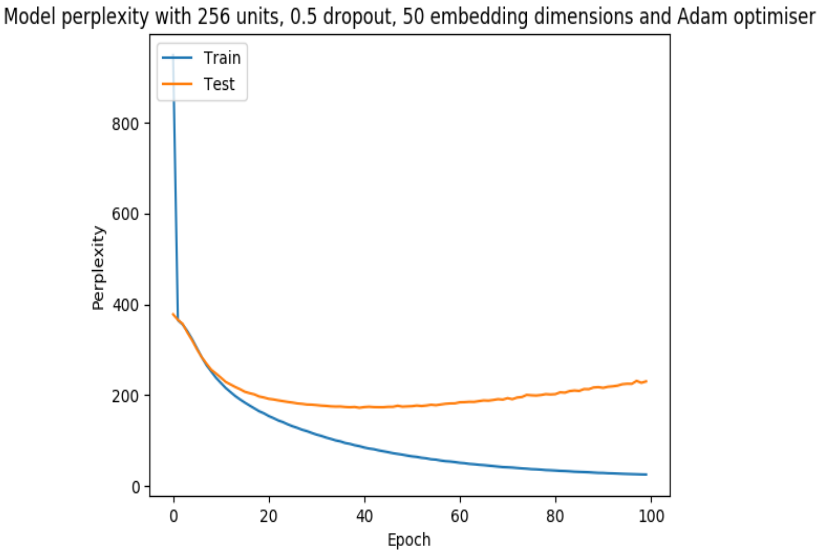

Figure 8.2: popper

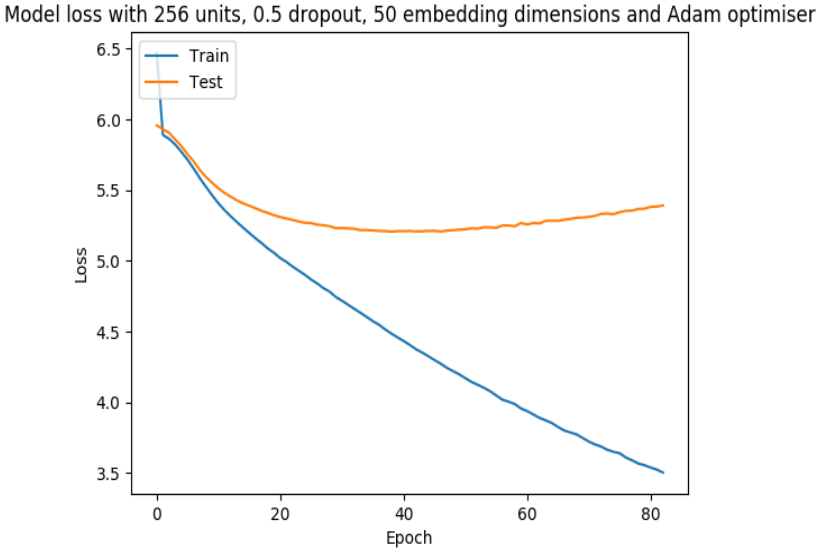

Figure 8.3: rockloss.

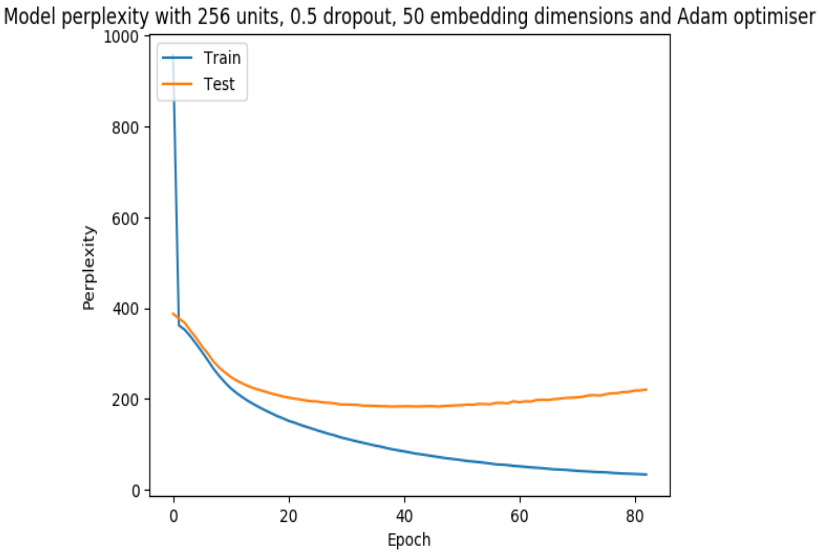

Figure 8.4: rockper.

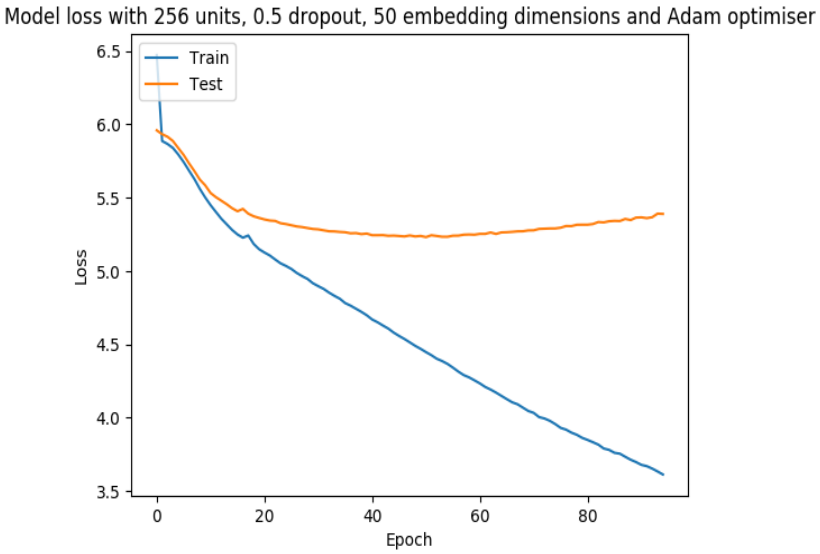

Figure 8.5: hiphoploss

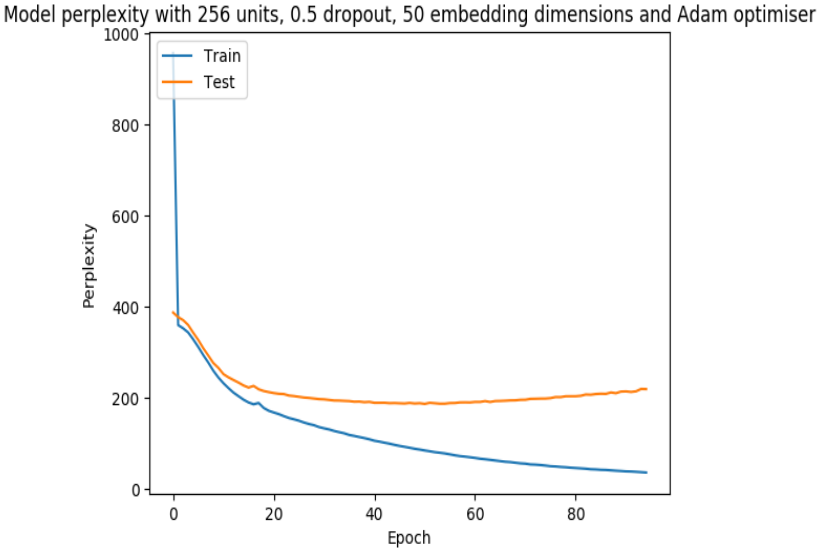

Figure 8.6: hiphopper

Chapter 9

Conclusion

I was right all along.

9.1 Limitations

I was right about the following things.

9.2 Future Work

Bibliography

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155. 14

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166. 16

Chun Pui Tang, Ka Long Chui, Y. K. Y. Z. Z. K. H. W. (2018). Music genre classification using a hierarchical long short term memory (lstm) model. 9

Corinna Cortes, V. V. (1995). Support-vector networks. 18

Dai, A. M. and Le, Q. V. (2015). Semi-supervised sequence learning. 19

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159. 12

Edwards, P. and Rap, K. (2009). *How to Rap*. Chicago Review Press, Incorporated. 9

Harris, Z. S. (1954). Distributional structure. *Word*, 10(2-3):146–162. 1

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780. 16

Irvin, J., Chartock, E., and Hollander, N. (2016). Recurrent neural networks with attention for genre classification. 9

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer. 18, 19

Katz, S. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400–401. 14

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980. 28

Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. (2004). Rcv1: A new benchmark collection for text categorization research. 19

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA. 4

McCrum, R. (2011). *The Story of English*. BBC books. Faber & Faber. 2

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*. 1

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*. 14

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119. 26

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543. 1, 11

Potash, P., Romanov, A., and Rumshisky, A. (2015). Ghostwriter: Using an lstm for automatic rap lyric generation. pages 1919–1924. 8

Qi, Y., Singh Sachan, D., Felix, M., Janani Padmanabhan, S., and Neubig, G. (2018). When and why are pre-trainedword embeddings useful for neural machine translation? 4

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA. 15

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1986). Learning representations by back-propagating errors. *Cognitive modeling*. 1

Rush, A. M., Chopra, S., and Weston, J. (2015). A neural attention model for abstractive sentence summarization. In *EMNLP*. 14

Sahami, M., Dumais, S., Heckerman, D., and Horvitz, E. (1998). A bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 workshop*, volume 62, pages 98–105. Madison, Wisconsin. 18

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958. 28

Tian, K., Zhang, T., and Zou, J. (2018). Cover: Learning covariate-specific vector representations with tensor decompositions. *arXiv preprint arXiv:1802.07839*. 2, 13

Tsaptsinos, A. (2017). Music genre classification by lyrics using a hierarchical attention network. 9

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., and Hovy, E. (2016). Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489. 9

Zhang, X. and Lapata, M. (2014). Chinese poetry generation with recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 670–680. 8

Appendix A

Code

```
10 PRINT "HELLO WORLD"
```