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 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

Both language modelling and text classification are active research areas within natural language processing. The primary goal of language modelling is to provide a probability distribution for sequences of words. Text classification, which is the task of classifying text into one or more predefined categories has applications in areas such as sentiment analysis, topic labelling and spam detection.

Recurrent neural networks have been deployed successfully in both language modelling and text classification tasks. Training these types of networks on textual data involves the conversion of text to vector representations which can result in either sparse or dense word vectors. Sparse representations of words, such as a one-hot encoding suffer from the curse of dimensionality due to the dimensionality of the word vector growing linearly with the size of the vocabulary. Dense representations of words, also known as word embeddings, offer smaller continuous word representations which, unlike their sparse counterparts, are able to encode semantic and syntactic meanings within texts.

Often accompanying text corpora are associated covariates, e.g. author demographic or publication genre, which provide additional metadata about a corpus. CoVeR (REF), a novel tensor decomposition method for learning covariate specific word embeddings, is an extension of the GloVe algorithm which aims to encode covariate information with learned embeddings.

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.

In general, skilled writers are attributed with having vast vocabulary ranges. For adults, the average vocabulary ranges between 15,000-23,000 words(REF). Examining his works alone, Shakespeare is said to have had an approximate vocabulary size of 30,000 words (REF) (FOOTNOTE HERE SKEWED). 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, which recently surpassed

Rock as the most popular genre in America (REF HERE), more is not always better. The study examines the unique word count of 150 famous Hip-Hop artists across their first 35,000 lyrics. Aesop Rock, ranked 1st on the list, recorded a count of 7,392 unique words across his first 35,000 lyrics. In contrast, rappers Drake and Future, ranked 130th and 131st respectively, had an average unique word count of 3,334 words used across their first 35,000 lyrics; a 55% decrease from Aesop Rocks count.

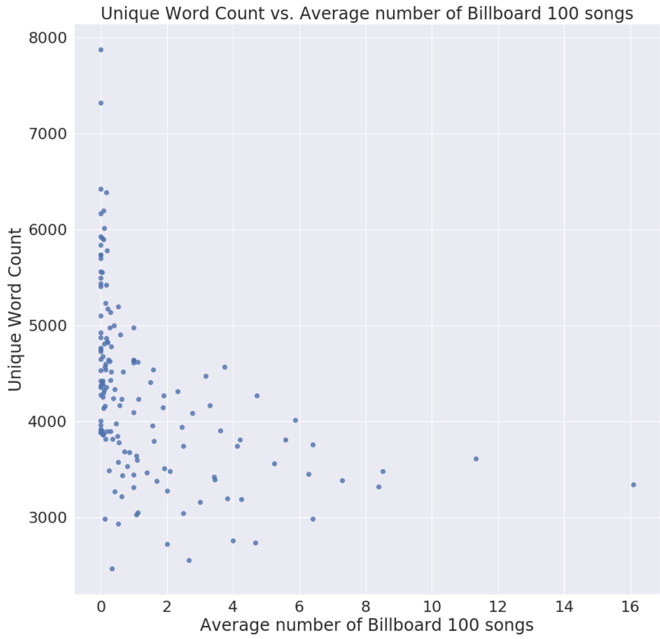

Figure 1.1: Average number of Billboard 100 songs during artist activity, compared to unique word count across an artists first 35,000 words.

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 songs across an artist had across their career. Pearson's Correlation Coefficient , which is used to measure the 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(REF) have been used to consolidate previous methods. An inherent problem within software solutions like MasterWriter is the static nature of features such as fixed word and rhyming dictionaries. Consequently, these applications fail to address the ambiguous usage of words resulting from the emerging nature of natural language.

After the completion of song lyrics another secondary problem often faced by less experienced songwriters is choice of instrumental style.

1.3 Goals and Objectives

1.3.1 SONGIFAI - A proposed solution

The goal of this project is to explore the use of CoVeR derived word embeddings to help with both language modelling and text classification tasks. To contextualise the project aims, both models are applied to a possible use case: a protoype software solution to help reduce common problems faced by songwriters. With this in mind, a prototype solution, SONGIFAI is proposed. SONGIFAI provides two main features namely lyric assistance through predictive text and word suggestions, as well as lyric genre classification. The covariates explored in this project are the following music genres: *Pop*, *Rock* and *Hip-Hop*.

1.3.2 High Level Architecture

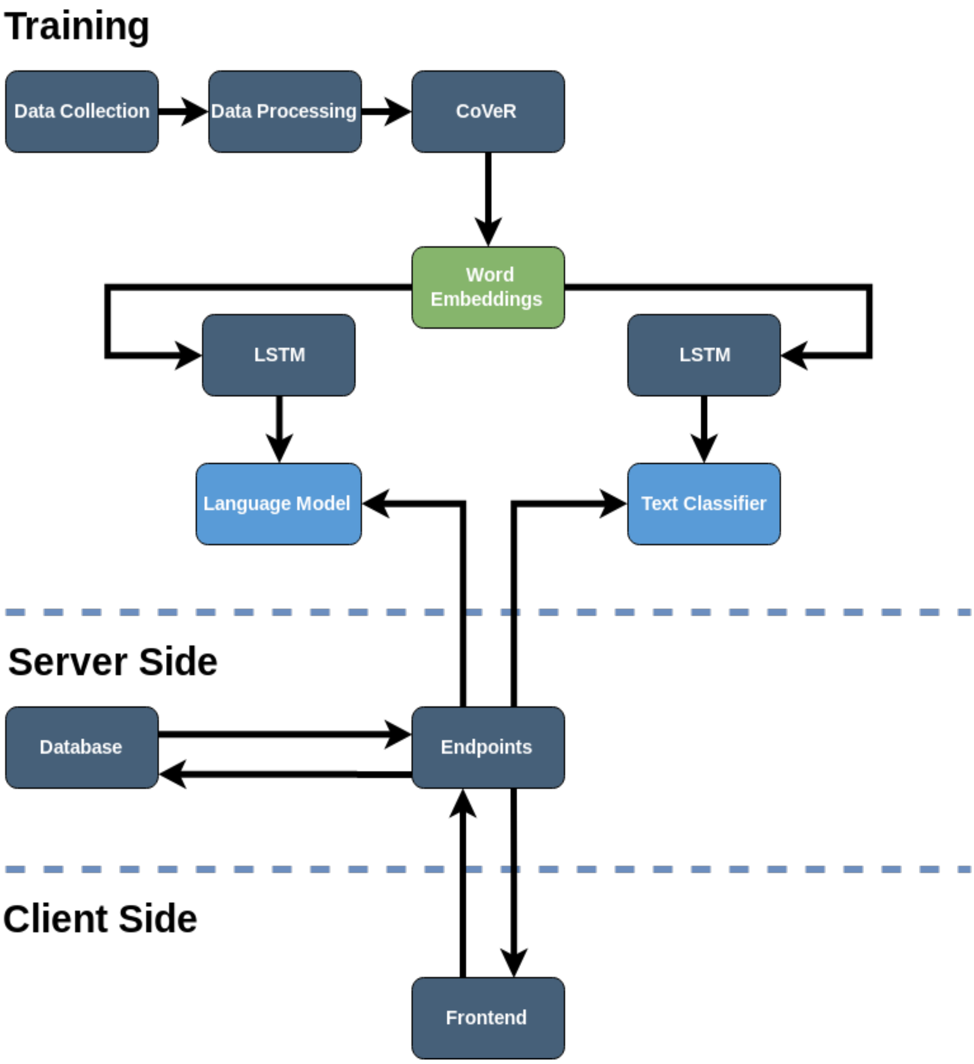

Figure 1.2: High level architecture for the project

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 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 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 possible best practices and coding style/conventions have been adhered to where appropriate.

2.3 Ethical Considerations

The success of a machine learning project relies heavily on data availability and quality. Regarding song lyrics, there exists no central repository where lyrics, along with the required metadata for the project, are stored. Consequently, a publicly available dataset was used throughout this project.

Additionally, the project utilises textual data which may have explicit 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 explicit content.

Chapter 3

Background

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

3.1 Word Embeddings

Word embeddings are vectors of predefined size which aim to encode a distributional numerical representation of word features. They have found usage in a variety of applications such as document classification (REF) and named entity recognition (REF). Conceptually they are based on the distributional hypothesis that states that words which appear in a similar context have similar meanings. Recent aforementioned methods of learning these representations include both the GloVe and Word2Vec algorithms. The utilization of word embeddings has been highly successful in many natural language processing tasks such as sentiment analysis (soccer, perelygin, wu, cghuang, manning, ng. Potts 2013) and syntactic parsing(soccer, Bauer, manning ng 2013). Previous techniques for creating such representations can be categorised into two categories: matrix factorization methods and shallow window-based methods.

3.1.1 Historic Methods

Global Matrix Factorization methods

Global matrix factorization is the process of using matrix factorisation in order to perform rank reduction on a large term-frequency matrix. Within natural language processing, these matrices usually take one of two forms, term-document frequencies, where each entry represents the count of a particular word within a document, and term-term frequencies, which measures the co-occurrence of words within a given context. Matrix factorisation techniques such as Latent Sentiment Analysis (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.

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). 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.

3.1.2 GloVe - Global Vectors for Word Representation

GloVe (Global vectors for word representation) (REF) 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 a global co-occurrence statistics from a corpus. As detailed in the paper, Glove outperformed previous methods such as Word2vec in word analogy, word similarity and named entity recognition tasks. Conceptually, GloVe is based on the idea that ratios of probabilities of words co-occurring have the potential to encode meaning which is encoded as vector differences. This concept is formalised in the following equation, where the dot product of focal and context word vectors, w and $\tilde{w}$, is equal to the logarithm of the probability of the words co-occurring, $\log X_{ij}$.

$$w_i^T \tilde{w}_j + b_i + \tilde{b}_j = \log {X_{ij}}^2 \tag{3.1}$$

A weighting function $f(X_{ij})$ is used to decrease noise caused by very frequent word co-occurrences. The following weighting function is used in the GloVe model.

$$f(x) = \begin{cases} (x/x_{max}), & \text{if } x < x_{max} \\ 1, & \text{otherwise} \end{cases} \tag{3.2}$$

Combining equations 3.1 and 3.2, the GloVe model is defined as a weighted least squares regression problem.

$$J = \sum_{i,j=1}^{N} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2 \tag{3.3}$$

3.1.3 CoVeR - Covariate-Specific Word Embeddings

Covariates such as author demographics, time and location often accompany documents within a corpus. A trivial approach to learning covariate specific word embeddings would involve applying GloVe to each subset of a corpus relating to a particular covariate. A weakness in this approach is that information from each of the specific covariate co-occurrence matrices is not shared.

CoVer, proposed by Tian et al Stanford, provides an alternative to the conditional GloVe method. Being an extension of GloVe, CoVer extends Glove's matrix decomposition of co-occurrence matrices to tensor decomposition of co-occurrence tensors, involving the joint learning of word embeddings and covariate specific transformation matrices which represent the effect of a particular covariate on the base embeddings learned. The CoVeR model is presented below.

$$J = \sum_{i,j=1}^{N} \sum_{k=1}^{M} f(X_{ijk})((c_k \odot w_i)^T (c_k \odot \tilde{w}) + b_{ik} + \tilde{b}_{jk} - \log X_{ijk})^2 \tag{3.4}$$

3.2 Language Models

Formal languages such as programming languages are fully specified with precise syntax and semantics which dictate the usage of all reserved words within a 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 suffer from the endless possibilities of language usage outside of grammatical rules which are still easily interpretable by humans. Moreover the task of consistently updating rule based systems to accommodate such usage is suboptimal.

Language modelling 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 natural language process tasks such as speech to text and text summarization. Language models can be classified into two categories, count-based and continuous-space language models.

3.2.1 Count Based Models

Count based methods such as statistical language models attempt to create the joint probability distribution of a sequence of words. An example of a count based method is the n-gram model.

An n-gram is a sequence of N words. Examples of a two word sequences or bigrams include, ”My name” and ”is Aubrey”, whilst examples of three word sequences or trigrams, include sequences such as ”Hello my name” and ”is Aubrey Graham”. The n-gram model which considers the past n-1 words can be formailsed as

The n-gram model relies on Markov assumptions to model the probability of word sequences P(w1....wm) as being equal to a limited number of previous words. 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 to many zero probabilities. To circumvent this, techniques such as back-off and smoothing exist. Another disadvantage of n-gram models is that they rely on exact patterns, meaning n-gram models fail to recognise syntactically and semantically similar sentences such as ”the cat sat on the mat” and ”the dog sat on the mat”. 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 to not be captured.

3.2.2 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, abstract language modelling as a form of sequential data prediction and have implored recurrent neural networks 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 recurrent neural networks and Long-Short Memory networks 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.

$$y = \sum_{i=1}^{N} x_i w_i + b \tag{3.5}$$

Typical activation functions include *Sigmoid*, *Tanh* and *ReLu*. A single layer neural network is defined by k neurons sharing the same input in the same layer. Single layer neural networks have been proven to be *'universal approximators'* meaning any continuous function can be approximated using this type of network. 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 (REF).

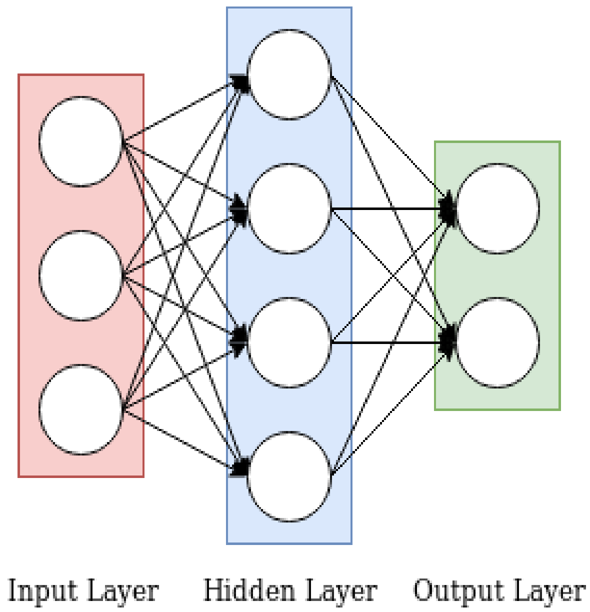

Figure 3.1: Example neural network, three input nodes, four hidden and two outputs

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

$$h_t = \begin{cases} 0, & \text{t} = 0 \\ \phi(h_{t-1}, x_t), & \text{otherwise} \end{cases} \tag{3.6}$$

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

$$h_t = \phi(Ux_t + Wh_{t-1}) \tag{3.7}$$

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 X.X)

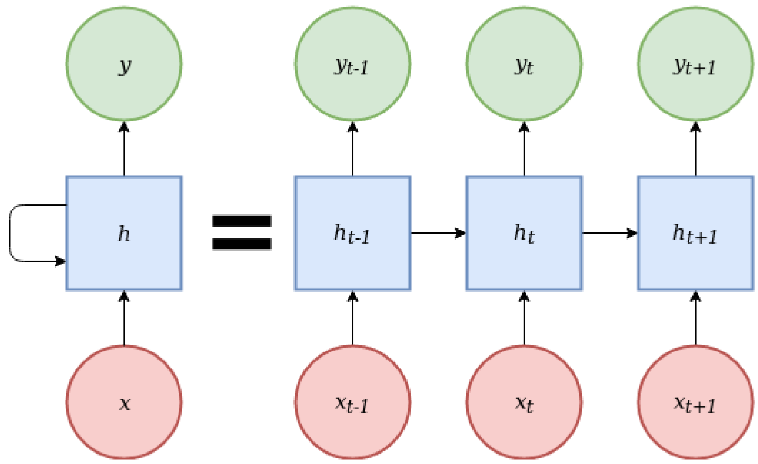

Figure 3.2: An *unrolled* recurrent neural network can be seen as a feed-forward neural network with many hidden layers

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(REF) where gradients can decay or grow exponentially as they propagate over time

In BBTT, the gradient is back-propagated through network layers at each time step to adjust weights accordingly. During this process, weights in each layer are adjusted using previous gradients from output layers causing gradients to become increasing smaller. Ever decreasing gradients, or "vanishing gradients", can prevent the network from learning entirely due to the minimal updates applied to weights in earlier layers.

Long Short-Term Memory

Long Short-Term Memory (LSTM) (Hochreiter, 1997) is a variant of the recurrent neural network which is capable of capturing longer dependencies between sequences of data without suffering from vanishing 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 gate. The forget gate regulates how much of the existing memory should be forgotten, the input gate regulates how much of the new cell state to keep, and the output gate regulates how much of the cell state should be allowed into the next layer of the network.

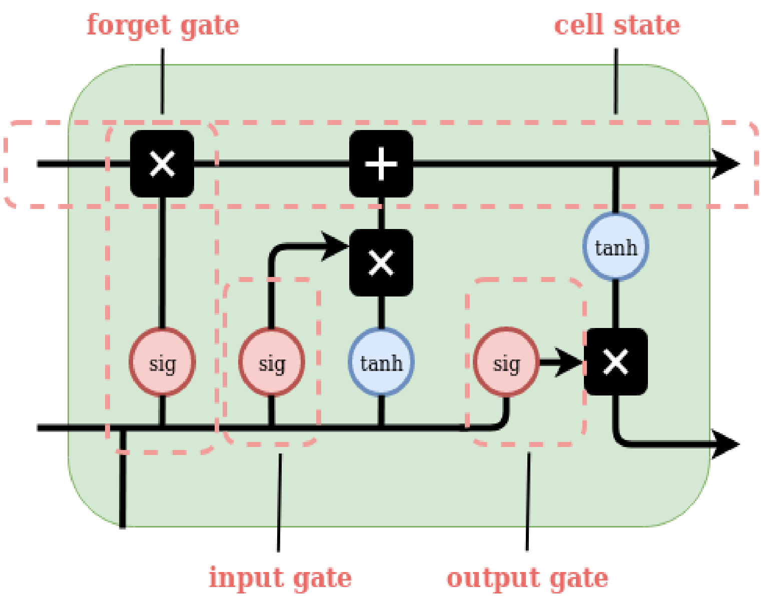

Figure 3.3: LSTM memory cell, with forget, input and output gates

3.2.3 Text Classification

Text classification is a supervised NLP task which involves assigning pre-defined labels to text according to its content. Automated classification of text can be achieved through rule based and machine learning based systems. Rule based methods tackle classification through the use of handcrafted linguistic rules, which assign patterns in text to predefined categories. For example, given two word lists which Rule based systems don't come without drawbacks, firstly to create such a system requires deep domain knowledge. Moreover, unlike the previous example, creating, maintaining and scaling such rules is challenging and time consuming.

Traditional Methods

Before a classifier can be trained, textual inputs must be transformed into numerical representations in a process known as feature extraction. A common method for feature extraction is the bag of words approach which given a vocabulary set V, creates an input vector which represents word counts for each word of the vocabulary. For example, if we define the vocabulary V ..

Two common classifiers, namely Naive Bayes classifiers and support vector machines are outlined below

Naive Bayes

Naive Bayes Classification is a generative approach which uses bayes theorom to learn a join probability distribution .... Naive bayes classification is a robust method for training a text classifier which can achieve accurate results without large amounts of training data.

Naive bayes is a classification techinque based on bayes theroem with an assumpoion of inde[endence among preictors. A naives bayes c;assifier assumes that the precence of a particular fgeature in a class is unrelsted tp the [recence of any other fetaure.

Support Vector Machines

A Support Vector Machines (SVM) (Vapnik et al 1995) is a discriminative classifier that aims to find a linear classification boundary or *hyperplane* to discriminate between classes in high dimensional space. SVM's achieve this through support vectors, random training data points, which are used to maximise the margin between ... Similar to naive bayes classifiers, SVM's do not require large amounts of training data to achieve accurate results. SVM's were introduced to the problem of text classification by Joachims(REF)

Deep Learning Methods

Chapter 4

Requirements Analysis

As the project involves the development of a prototype software system, 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 in order to constrain the project goal 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.

4.1 Existing Solutions

4.1.1 MasterWriter

Self-described as ”The most powerful suite of writing tools ever assembled in one program.”, MasterWriter 5 is a software application which aims to help songwriters, poets and creative writers with their works. Available as a desktop, mobile or tablet application, it consolidates a number of writing tools into one application. These tools are outlined in the table below:

4.1.2 Rhymer's Block

Rhymer's block is a mobile application intended to help writers specifically with rhymes. Providing real time rhyme suggestions, the application allows users to quickly write lyrics and provides a social feature in order to share lyrics and review lyrics from other users.

4.1.3 Evaluation of existing solutions

A common feature to both software solutions is that of word suggestion, specifically suggestion of rhyming words. Furthermore both solutions provide functionality for users to write, edit and save lyrics within the application.

4.2 Requirements

In this section the requirements for the project will be set out. The functional requirements will specify what the software will do whilst the non-functional requirements will detail how these will be done.

4.2.1 Functional

Table 4.1: SONGIFAI Functional Requirements

ID	Description	Dependency
FR1	The system should allow users to input lyrics	N/A
FR2	The system should allow users to load/save 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 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 required	N/A
FR8	10C	N/A

4.2.2 Non-Functional

Table 4.2: SONGIFAI Non-Functional Requirements

ID	Description	Dependency
NFR1	The system should take the form of a web application and be able to be rendered on different device types	N/A
NFR2	The word prediction process should return a list of candidate words in real time	N/A
NFR3	10C	N/A

Chapter 5

Methodology

This chapter details the methodology used to collect, analyse and process the dataset used to derive the CoVeR word embeddings.

5.1 Collecting Data

As previously mentioned, there exists no central repository from which song lyrics can be obtained. Though lyric hosting websites such as Genius(FOOTNOTE) exist, selective collection of song lyrics is only achievable through the process of web scraping. Web scraping is the process of exhaustively downloading web pages, from either a predefined list of URLs or through link extraction. Large scale scraping is usually achieved through parallelised methods due to restrictions such as Robots.txt and download latency. Taking into account the project objectives and goals, web scraping was avoided.

In view of this, a publicly available dataset containing over 250,000 lyrics was used. The dataset comprised of two .csv files: artists and lyrics. The artists .csv file mapped individual artists to their respective genres/sub genres, whilst the lyrics .csv file contained data on individual songs, mapping lyrics to artists.

5.2 Data Analysis and Restructuring

The CoVeR algorithm requires sub corpora to be labelled in order to jointly learn word embeddings and the relevant transformation matrices. With regards to this project a mapping between lyrics and genre was required. To fulfil this requirement Pandas, a Python data manipulation/analysis library was used perform a SQL like join on both sets of data, specifically on the artist name.

100,000 song lyrics were decided on to be used as training data. A trivial approach to split the data on the genre covariate would involve an equal split for equal representation, however, this method has the assumption that for each particular genre, word usage is the same on average.

Examining the dataset proved this not to be the case, with Hip-Hop songs containing 444 words on average compared to the 207 found in Rock songs and 289 found in Pop songs. To reflect these statistics in the training data, a training split of 48:30:22 for Rock, Pop and Hip-Hop was used.

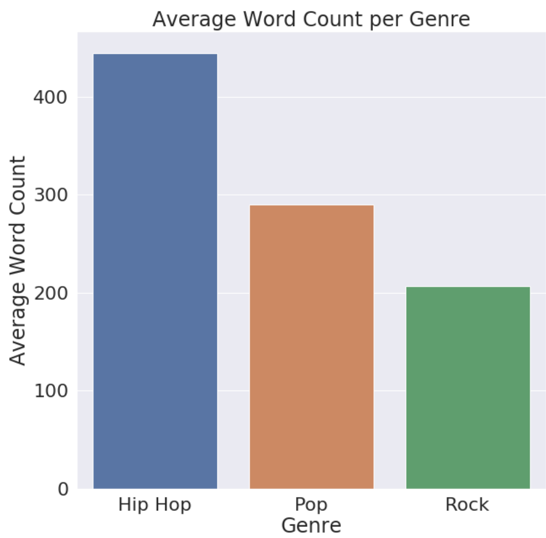

Figure 5.1: Average word count per lyric per genre in the dataset

5.3 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 natural language processing, this can include techniques such as tokenisation, string cleaning, stemming and lemmatisation.

Following the lyric data reconstruction, each lyric in the corpus was cleansed and tokenised. The following string cleaning techniques were applied to each lyric in the dataset.

1. All letters were lowercased.
2. All characters, except for letters, were substituted with a space " ".
3. All text between 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 has a one-one mapping with a unique numerical key, which is used to transform each lyric in the corpus into a list of numerical keys. For training, only tokens which appeared a minimum number of times were kept.

Typically found within text corpora are high frequency stop words such as 'the', 'a', and 'in' which provide less information than rarely occurring words(REF-DISTRIBUTED REP OF WORDS). For example... This concept can also be applied to word embeddings; where the word embeddings of frequent words does not change significantly after training on several examples. Similar to (REF), subsampling was used at the covariate level using the following adapted formula from the original word2vec implementation.

$$P(w_{ik}) = \sqrt{\frac{z(w_{ik})}{t}} + 1 \cdot \frac{t}{z(w_{ik})} \tag{5.1}$$

where $z(w_{ik})$ is the percentage of word w_{ik} in covariate k and t is a chosen threshold.

5.4 Hyperparameters

Hyperparameters within machine learning are parameters that govern a given model. The choice of these parameters directly impacts the performance of a given training algorithm and as such, optimal hyperparameter selection is vital for yielding optimal performance for a given model. All three models implemented, namely CoVeR, the BiLSTM language model and the BiLSTM text classifier, have hyperparameters which are discussed in the following sections.

5.4.1 Context Windows

Like GloVe, CoVeR also uses weighted context windows during the process of calculating co-occurrence statistics. The CoVeR paper uses a context window size of 8, however the paper does not specify whether they used symmetric or asymmetric windows during their experiments.

5.4.2 Embedding Size

5.4.3 Optimisation Methods

5.4.4 title

5.4.5 Dropout

Dropout is a regularization technique proposed by Srivastava et el 2014), which involves the random dropping of nodes and their connections within a network. Selected nodes are picked at random using a probability known as the dropout rate. Dropout helps to decrease over-fitting as

5.4.6 Embedding Layer

Chapter 6

Implementation

6.1 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
4	545

Table 6.1: Hardware specification for machine used throughout development

6.2 Calculating Co-occurrence Statistics

Many unsupervised natural language processing methods compute co-occurrence statistics before learning takes place. Typical co-occurrence statistics, such as GloVe's word-word co-occurrence matrix are very sparse in nature, and computing them can often be a computationally more expensive task than the learning itself. Examining GloVe, where a corpus has vocabulary size N, a word-word co-occurrence matrix X is computed with X_{ij} being a measure of the number of times words i and j co-occur within a given context window.

The original GloVe paper describes this process as a *'one-time upfront cost'*, with the assumption that selected corpora are static. Unfortunately, for many natural language processing pipelines such corpora are more dynamic in nature. For example, social data from online platforms such as Twitter are in constant flux and relying on pre-computed co-occurrence statistics 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 involves two functional processes namely map and reduce. 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.

Apache Spark is an open-source framework, written in Scala, for distributed computing and has recently emerged as the preferred option 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.

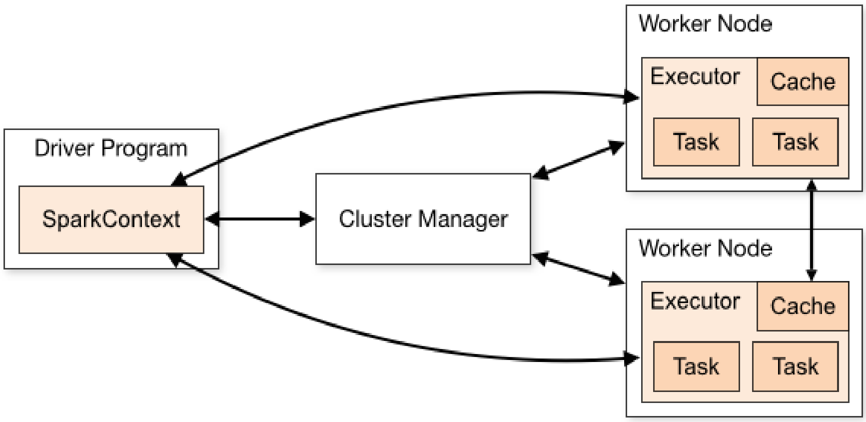

Figure 6.1: High level view of the Spark Architecture. The spark context is where the main program is defined, which is then split into tasks to be completed via numerous executors.

PySpark, a Python API for the Spark framework was initially used to both pre-process data and calculate co-occurrence statistics for the corpora. 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 programs can experience major speed improvements using Cython because of its ability to compile Python to C code.

6.3 CoVeR Implementation

At time of writing, no publicly available 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)

6.3.1 Initialisation of Learnable Parameters

6.3.2 Hyperparameters

6.4 Model Implementation

Implementation of both the language model and the 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.

6.4.1 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.

Hyperparameter Tuning

6.4.2 Text Classifier

Hyperparameter Tuning

6.5 SONGIFAI

6.5.1 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.

6.5.2 Class Overview

Chapter 7

Evaluation

7.1 CoVeR Evaluation

7.1.1 Validating Implementation

Base Embeddings

As stated earlier, no publicly available version of the CoVeR algorithm exists. Being an extension of the Glove algorithm which still learns base word embeddings for focal and context vectors, comparing the outputs of both base embeddings served as a good metric. To measure the similarity between the two base embeddings generated by Glove and Cover, the F1 scores where generated to compare the

Covariate Specific Embeddings

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

7.2 Model Evaluations

7.2.1 Language Model

Perplexity

Penn Treebank Dataset

7.2.2 Text Classification

7.3 SONGIFAI

7.3.1 Requirements Evaluation

7.3.2 Expert User Testing

Chapter 8

Conclusion

I was right all along.

8.1 What was I right about?

I was right about the following things.

8.1.1 Previous theories were wrong

People thought they understood, but they didn't.

8.1.2 My new idea is right

Of course.

Bibliography

Appendix A

Code

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