Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

Sentiment Analysis From the Reader's Perspective

Author:
Chien Hong Tan

Supervisor:
Dr. Anandha Gopalan

Second Marker:
Professor Julie McCann

June 25, 2020

Abstract

Most sentiment analysis research is not explicit with respect to the point of view. Those that do, tend to focus on the Writer's perspective only. In this project, we first focus on developing a model for the Reader's perspective. We succeed in doing so, and our final model is an ensemble combining different but related affective features to perform its final Reader sentiment predictions. We also use available datasets and our constructed models to perform an investigation of various emotional relationships.

Furthermore, we seek to demonstrate applications of a Reader-centric sentiment analysis model. We do this in two ways. First, we develop a news application serving positive content to the user, using our sentiment analysis model as its prediction engine. Second, we use our model in the extraction of emotional trajectories from various fictional works, and show that we can identify common emotional arcs via clustering.

Acknowledgements

I would like to thank Dr. Anandha Gopalan for offering his continuous guidance throughout the project. I would also like to thank my friends, who have accompanied me throughout university. Finally, I would like to thank my parents, for supporting me throughout my entire life.

Contents

Chapter 1

Introduction

The point of view is a factor often ignored in sentiment analysis research and applications, but one which has meaningful implications [59]. There are two main points of view with respect to a given text – the *Writer's* perspective and the *Reader's* perspective. To highlight the distinction between the two, take the following news headline as an example:

"Oh no! Dogs United has beaten Cats F.C 6-0."

In this example, the emotional attitude as expressed by the Writer, likely a Cats F.C fan, can be deemed to be very negative. However, a Dogs United fan will likely be experiencing feelings of joy and happiness.

For this project, we explicitly focus on perspectives of sentiment analysis, specifically, the Reader's perspective. This is in contrast with most sentiment analysis research, which primarily deals with the Writer's perspective, whether explicitly or implicitly [27]. This means that we are focused on the task of predicting the reader's emotional reaction to a piece of text, as opposed to the emotional state as expressed in the piece of text by the writer.

In addition to that, we also focus on a mixed view of sentiment. This means that the predicted sentiment is not restricted to a single category. The Writer's perspective tends to be a singular one, meaning that a single classification label of writer emotion can generally apply (although there may be varying interpretations). However, meaningful information can be lost when reducing predicting Reader emotion to a classification problem. Classifying reader emotion as positive or negative, or a specific emotion such as "Happy", means that information regarding how various people will react differently to such a statement is lost.

Reusing the news headline example, classifying the statement strictly as positive or negative is not entirely accurate, as fans of the different teams will react very differently. A mixed view, indicating the varying nature of Reader emotional reactions, is therefore preferable.

1.1 Objectives

Our objectives are as follows:

Main Objective

The development of a machine learning model which performs sentiment analysis from the point of view of the Reader, in a fashion which indicates the varying reactions of Readers.

Secondary Objective

To explore the applicability of Reader sentiment analysis by using the developed model for concrete tasks and applications.

1.2 Challenges

The main challenges of this project are as follows:

- Finding previous work - Previous work in sentiment analysis which emphasises the differing perspectives is sparse [59], so finding related research was a challenge.
- Obtaining data - Finding datasets explicitly attuned to different perspectives was difficult, with few existing datasets available which suit our needs.

1.3 Contributions

The overall contributions of this project are as follows:

- **Reader Sentiment Analysis Model** - We achieve our main objective of developing a sentiment analysis model which predicts Reader emotions in a mixed fashion. Our final model consists of an ensemble of two base Reader models, which each predict different but related affective (emotional) information. See Chapter 3 and Chapter 4.
- **Application: Positive Stories News Application** - We implement a practical application of our model, a news application which shows users articles likely to evoke positive emotions. See Chapter 5.
- **Application: Analysis of Emotional Trajectories in Stories** - We successfully demonstrate a research application of our model, using it to analyse the emotional trajectories of a corpus of fictional works. See Chapter 6.
- **Analysis of Emotional Relationships** - Using our collected datasets and constructed models, we demonstrate results of our analysis on different representations of emotions and perspectives. See Chapter 7.

Chapter 2

Background

In this chapter, we explore the background and related works associated with this project.

2.1 Representing Emotions

This project heavily involves the representation of emotions. Within psychology, there exists two broad ways to represent emotions, dimensional and categorical [28].

In the categorical model, all humans have an innate set of universal basic emotions that form discrete categories. An example of a categorical model of emotion was defined by Ekman, consisting of six basic cross-cultural emotions - anger, fear, happiness, disgust, sadness and surprise [36].

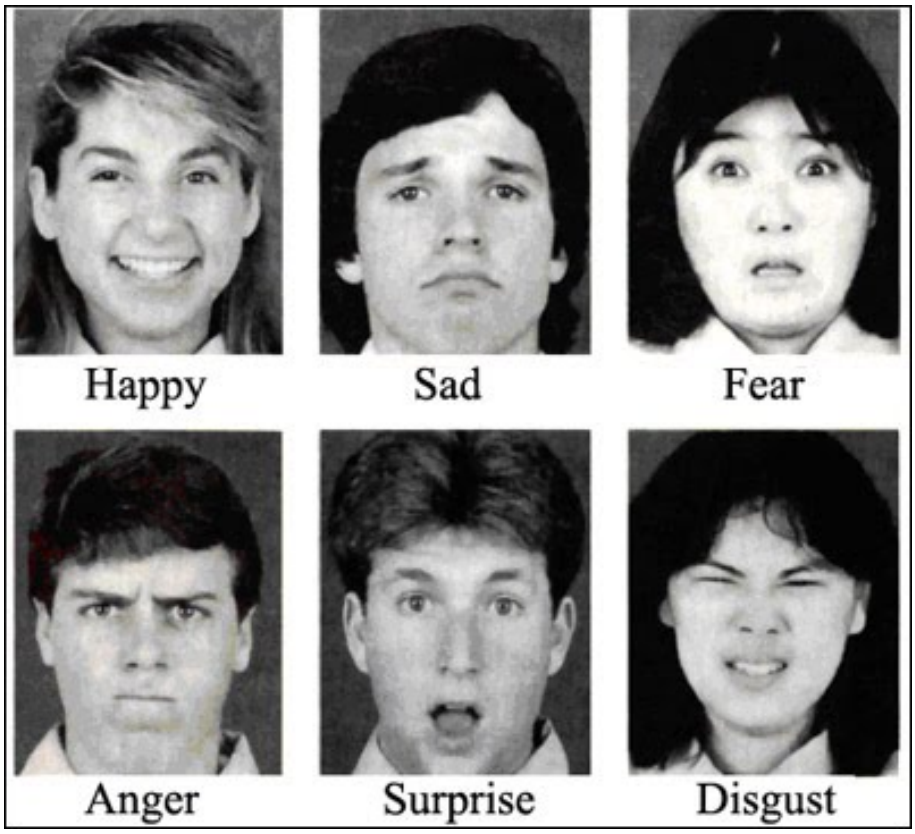


Figure 2.1: Paul Ekman's Six Basic Emotions [36]

This is in contrast with the dimensional model, where emotional states are instead defined by points in a multi-dimensional space [28]. There are various examples of such dimensional models. A common model is the Valence-Arousal-Dominance (VAD) model. In this model, the multi-dimensional space consists of three dimensions [64]. Valence concerns the level of experienced pleasure or displeasure, Arousal concerns the level of calmness or excitement and Dominance concerns the level of perceived control of the cause of the emotional experience. Another common model, Russell's circumplex model of emotion [80], leaves out the Dominance dimension, and instead defines a two-dimensional space consisting solely of Valence and Arousal. Their argument is that the Dominance dimension is of a cognitive as opposed to emotional nature. Converting between the categorical and dimensional models is possible by mapping regions of the multi-dimensional to the discrete categories.

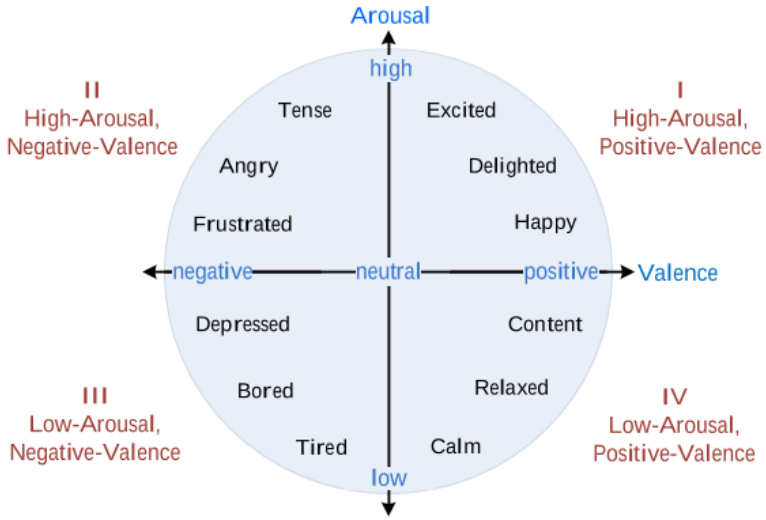


Figure 2.2: Circumplex/Valence-Arousal Model of Emotion [28]

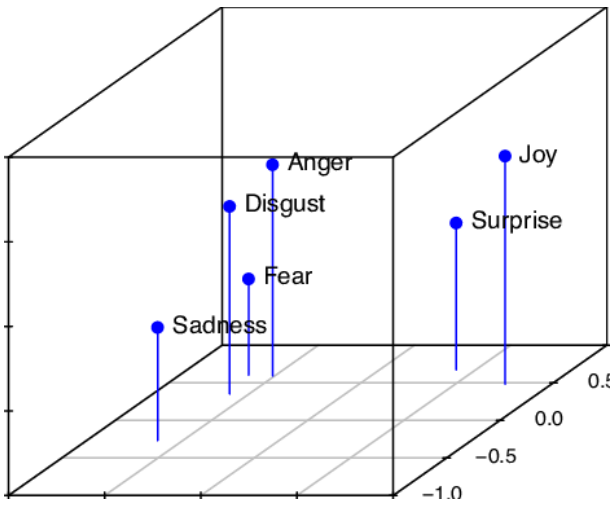


Figure 2.3: Valence-Arousal-Dominance Model of Emotion [64]

2.2 Facebook Reactions

Starting February 2016, Facebook introduced Reaction buttons to exist alongside the standard Like button, allowing users to choose from a range of emotional responses as opposed to simply pressing "Like" when responding to content [85]. The five Reactions added were "Love", "Haha", "Sad", "Wow" and "Angry". Since its introduction, work has been done explicitly focused on emotional reactions of the Reader, using these Facebook Reactions as "ground-truth" labels.

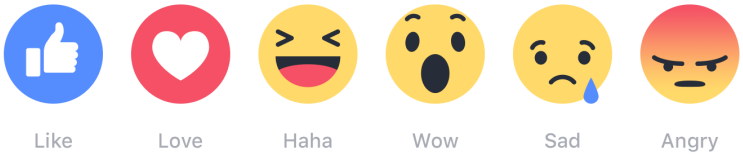


Figure 2.4: Facebook Like and Reactions [85]

A common concern with using reaction data is that users may be indicating their emotional response to the content that is being linked to in the post, as opposed to the text of the post itself. However, studies have shown that people rarely read beyond the headline in news articles shared on social media, so this should not be an issue [31]. It is also worth noting that some websites besides Facebook have provided reaction functionality, for example Buzzfeed (OMG, LOL, LOVE, CUTE etc.).

2.3 Sentiment Analysis

Sentiment analysis is a field of study within Natural Language Processing (NLP) [59]. It is focused on the analysis of information contained within text pertaining to individuals' affective states, attitudes and opinions, often towards a specific entity. There are many alternate names for the field of sentiment analysis, such as opinion mining, sentiment mining, emotion analysis etc. Furthermore, although the definition of sentiment analysis can be broad, with a variety of differing sub-tasks, it is most often used to refer to the specific task of determining the polarity (positivity/negativity) of a given text [27]. An extension to this task is concerned with determining specific emotional states ("angry", "happy", "sad" etc.) as opposed to simply binary polarity [71].

Sentiment analysis has many applications, from companies using it to conduct reputation management and market research [20] to identifying which programming languages are associated with more unhappy GitHub commits [42]. There are various approaches to the problem of sentiment analysis [29]. They can broadly be divided into two categories:

- Knowledge/rule-based techniques – These determine the sentiment of a body of text by using manually defined rules [29]. An example of such an implementation would involve first defining two separate lists of positive and negative words, and comparing the count of positive and negative words in a body of text. If the number of positive words exceeds the number of negative words, the text is classified as being positive. These approaches may require substantial human effort to maintain and develop depending on the complexity of the task. However, they tend to be less computationally expensive.

- Machine learning techniques – Most state-of-the-art results in sentiment analysis have been obtained via machine learning [34]. They tend to require more data and are more computationally expensive, but less manual feature engineering and maintenance is required. Today, machine learning tends to be the preferred approach in sentiment analysis, and it is what we focus on for this project.

2.4 Machine Learning

2.4.1 Multilayer Perceptrons (MLP)

Multilayer Perceptron refers to a class of artificial neural networks [44]. An MLP has at least three fully-connected layers of neurons, an input layer, one or more hidden layers, and an output layer. The hidden and output layers both use nonlinear activation functions. MLPs can approximate every continuous function that maps intervals of real numbers to some output interval of real numbers.

To learn, MLPs often use a technique known as back-propagation [44]. Output values are compared with correct values, the error computed using an error function. The error is fed back through the network, and the network adjusts the weights of each connection with the purpose of reducing the value of the error function.

Within the context of NLP, MLPs allows the ability for non-linear modelling. For example, MLPs provide the ability to distinguish data in an embedding space that is not linearly separable. This led to improvements in various tasks, however unlike sequential models like CNNs and RNNs, they are limited by the fact they do not model word ordering, an important factor in complex linguistic tasks [12].

2.4.2 Convolutional Neural Networks (CNNs)

CNNs led to state-of-the-art performances in computer vision before being applied to NLP [35]. The defining aspect of a CNN is the application of a sliding window function (also known as a kernel, filter of feature detector) to a matrix.

In NLP, the matrix represents a sequence of tokens [30]. Each row of the matrix corresponds to a token's word embedding (or one-hot encoding). The width of the filter corresponds to the embedding dimension, and the height (window size) can vary. A window size of n means that the filter is applied to n words (rows) at a time, that is, an n-gram at a time. Sliding the filter across the entire sequence n-gram at a time, feature extraction occurs, and repeated application of filters results in hierarchical feature extraction from lower-level features to higher-level features.

As opposed to RNNs, CNNs recognize patterns across space, working well for detecting local and position-invariant patterns [35]. For example, key phrases that express sentiment, like “I hate”. They are hence popular for sentiment analysis and text classification in general. Further study by Jacovi and Golberg et al. indicate that CNNs model local ordering by enabling n-gram feature extraction of embeddings [48]. Filters learn to capture linguistic meanings of n-grams, and, accompanied with global max-pooling, less important n-grams are then filtered out.

2.4.3 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks have seen success in a wide variety of areas where longer-term modelling is helpful, including NLP [60]. They work by retaining a "memory" of information from past sequential inputs. At each step, a vanilla RNN unit is fed both the input and its "hidden" state, which is its output from the previous timestep [84].

However, vanilla RNNs have difficulty in learning long-distance dependencies, suffering from vanishing and exploding gradients. RNNs use backpropagation through time (BPTT) to update [94], which involves unrolling a RNN temporally before applying backpropagation. Vanishing and exploding gradients are a result of the multiplication of many partial derivatives that comes with BPTT . Exploding gradients can be dealt with by clipping gradients when reaching a certain threshold. To deal with vanishing gradients, Gated RNNs were created.

The idea behind Gated RNNs was to create paths through time that meant gradients would neither explode nor vanish [84]. This is done by allowing gradients to flow without being suppressed or amplified. Intuitively, Gated RNNs use internal gates to allow the retention and removal of information based on input and history, while vanilla RNNs overwrite history unconditionally. There are many types of Gated RNNs, one of the most popular being Long Short-Term Memory (LSTM) [45].

Using RNNs within the context of NLP is straightforward, usually tokens are just fed in sequentially to the network as input [60]. RNNs are trained for temporal patten recognition, and hence perform well for NLP tasks where understanding longer-range semantics is important for performance, such as Part-Of-Speech tagging [60].

2.5 Word Embeddings

We can represent words in a vector space, with semantically similar words located closer to each other. These vector representations are called word embeddings [66]. There are multiple approaches to developing such word embeddings. A common approach is word2vec [66]. Training word-embeddings from scratch requires a very large dataset, and words are often not unique to a dataset. It is hence popular to use pre-trained embeddings, which are then transferred into an embedding layer in a neural network.

2.5.1 Using Word Embeddings

There are two main ways to then use these word embeddings — freezing pre-trained embeddings and fine-tuning pre-trained embeddings [46]. These can be further trained (fine-tuned), along with the rest of the network, on a dataset for the specific task. This is more computationally expensive, but can lead to better performance. This contextual tuning enables embeddings trained on one dataset to become better aligned with new language patterns present in another dataset. Not training embeddings further is called freezing.

2.5.2 Embeddings With Affective Information

Most existing approaches for learning word representations rely on the distributional hypothesis of linguistics, which states that words which appear in similar contexts have a tendency to also be semantically similar [66]. However, this means that emotionally different words such as "sad" and "happy" occurring in similar contexts would result in embeddings which are closer than their emotionally similar counterparts, such as "sad" with "disappointed" and "happy" with "joyful". This is not ideal for tasks to do with affective state, such as sentiment analysis.

In their paper "Learning Emotion-enriched Word Representations", Agrawal et al. propose a method of obtaining word representations imbued with affective information, meaning that emotionally similar words will occur closer in vector space than emotionally dissimilar words [17]. They use distant supervision to obtain a large dataset for training, and Recurrent Neural Networks (RNNs) for learning the embeddings, which they refer to as Emotion-aware Embeddings (EWE).

Retrofitting is another approach to enriching embeddings with new information. In their paper "Retrofitting Word Vectors to Semantic Lexicons", Faruqui et al. use semantic lexicons to determine what words are more or less strongly associated with each other, according to the nature of the lexicon (for example, a synonym lexicon would mean synonyms would have a stronger association than non-synonyms) [37]. They then use these associations to move the embeddings of associated words closer to each other. This technique can also be applied using emotion lexicons, resulting in an overall effect similar to that of Emotion-aware Embeddings. However, in contrast with emotion-aware embeddings, affective information is integrated into the embeddings after training has already occurred.

2.6 BERT

BERT was a model developed by researchers at Google, and was presented in the paper "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", by Devlin et al [34]. In Computer Vision, Transfer Learning has helped achieve state-of-the-art results through the fine-tuning of pre-trained models to specific tasks. Similarly, BERT as a pre-trained language model has been used to achieve state-of-the-art performance across a variety of NLP tasks, such as question answering and natural language inference.

Its key innovation involves the application of Transformers, a deep learning architecture, to language modelling. Transformers usually consist of two components, an encoder which reads input, and a decoder with produces a prediction. As the goal of

BERT is to generate a language model, the decoder is excluded. The encoder reads an entire sequence of input at once, and hence can be said to be bi-directional. Due to this, it gains a more sophisticated sense of language context than conventional single-direction models. Pretraining language models requires prediction objectives. For BERT, these are Masked Language Modelling (MLM) and Next Sentence Prediction (NLP). In MLM, 15% of the words in each input sequence is replaced with a `[MASK]` token. The model then attempts to predict the masked words by the context. In NSP, a pair of sentences are received as input, and the objective is to determine whether the second sentence follows the first.

There are two main ways BERT is used:

- Fine-tuning BERT - This refers to fine-tuning BERT models on downstream tasks using our task-specific data. When fine-tuning BERT, we take a pre-trained BERT model and use its outputs as features to an additional untrained neural network, defined as per our downstream task. The downstream neural network is often a simple MLP or just a single layer. The pre-trained model and the additional model are then trained using our task-specific inputs and outputs, with parameters being updated end-to-end. Like word embeddings, parameters can also be frozen, with only parameters of the downstream neural network being updated. However, fine-tuning for BERT usually implies end-to-end updates.

- BERT embeddings - This refers to extracting high-quality features from BERT models and using them as inputs to other downstream models. BERT embeddings are context-informed, meaning that unlike embedding models like word2vec where an embedding for a specific word is static, the embedding of that same word will differ depending on the context it appears in. The BERT model is sometimes fine-tuned on a task before embeddings are extracted to obtain task-adapted embeddings.

It is also possible to train your own BERT model from scratch, however a lot of data and computational power is required to obtain comparable results.

Fine-tuning possesses numerous advantages over training a model from scratch:

- Less training required - Pre-trained models already contain a lot of linguistic information which would otherwise need to be learned from scratch when training a new model. Less time is therefore required to "adapt" to a specific NLP task, and achieving excellent results is possible with only 2-4 epochs of training.

- Less data required - Related to the previous point, pre-trained weights also allow us to obtain excellent results with less data than would otherwise be required, as we do not need to encode linguistic information as per our downstream task from scratch. This saves time and energy which would otherwise be put into collecting data and model training.

- State of the art results - This procedure has been shown to achieve state of the art results across multiple tasks (although the BERT embedding approach is competitive).

However, extracting fixed embeddings from BERT is advantageous in certain scenarios:

- Task-specific model architectures - Not all tasks can be easily represented by the BERT architecture. Extracted BERT embeddings can be used to feed models better designed for such tasks.

- Computational benefits - With this approach, we can pre-compute a representation of the dataset just once, and then run multiple experiments with comparatively cheaper models using this representation.

Various modifications of standard BERT have since emerged. We now introduce the variants we use in our project.

2.6.1 RoBERTa

RoBERTa was proposed in "RoBERTa: A Robustly Optimized BERT Pretraining Approach" by Liu et al [61]. RoBERTa is in fact a pretraining approach as opposed to a new type of implementation, but BERT models trained in such a way are often referred to as RoBERTa models. In their paper, they acknowledge the significant performance gains that come with using pretrained language models, but also highlight the difficulty in making comprehensive comparisons between different approaches. This is necessary to determine what aspects (such as hyperparameters and dataset size) actually correspond to improved performance, and whether new forms of modeling are actually more effective or previous models are simply underutilised. This difficulty stems from a number of factors:

- Training is computationally intensive, making extensive fine-tuning of hyperparameters difficult. It is therefore difficult to comprehensively evaluate performance of a model across many combinations of parameters, and with respect to other approaches.

- Datasets are kept private and are of varying sizes, again making it difficult to determine whether improved performance from newer models are a result of increased training size or improvements in methodology.

With these factors in mind, they carefully replicate BERT pretraining and adjust key factors such as training dataset size and hyperparameter choice. They show that BERT was in fact undertrained, and significant performance gains were possible with the original BERT architecture. They therefore propose an improved training methodology for BERT models, RoBERTa. The modifications to training include the following:

- Increase training time, increase batch size, and increase amount of data trained on.

- Remove the Next-Sentence Prediction objective.

- Train on longer sequences.

- Dynamically change the masking pattern applied to training data.

When used to train models with the BERT architecture, performance exceeds those of all post-BERT models at the time of publishing of the paper.

2.6.2 ALBERT

The ALBERT model was proposed in "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations" by Lan et al [57]. They were motivated by the fact that an increase in model size when pretraining language representations is often associated with an increase in performance for downstream tasks. However, an increase in model size also means longer training times and larger memory consumption. Hardware limitations hence become a real obstacle to furthering state-of-the-art performance, as state-of-the-art models often involve hundreds of millions to billions of parameters. A Lite BERT (ALBERT) was therefore designed for quicker training than standard BERT, and also for lower memory usage. It does this via two parameter reduction techniques:

- Repeating layers split among groups
- Decomposing the embedding matrix into two smaller matrices

In their paper, Lan et al. demonstrate that these proposed methods lead to better scalability compared to standard BERT, and also lead to better generalisation by acting as a form of regularisation. They also address issues with Next-Sentence Prediction in standard BERT, leading to improvements in multi-sentence input tasks. ALBERT achieves state of the art performance across a variety of benchmarks while having fewer parameters than BERT.

2.6.3 DistilBERT

The DistilBERT model was proposed in the paper "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter" by Sanh et al [83]. They were motivated by the fact that large pretrained models were becoming increasingly common in NLP, achieving state-of-the-art results, but often usage of these models was constrained by limited computational resources for training and inference. They therefore propose a method for pretraining a smaller language representation model, called DistilBERT, which is distilled from standard BERT. To achieve this, they combine language modeling, distillation and cosine-distance losses to train their model. Similar to larger pretrained models, DistilBERT can be fine-tuned on downstream tasks. The model is 40% the size of BERT while being 60% faster to train and retains 95% of performance compared to BERT on the GLUE benchmark.

2.7 Hierarchical Agglomerative Clustering

Clustering is the task of grouping items together in such a way that items within a group (cluster) are more similar, according to some defined measure, than items in other groups [62].

In this project, we use hierarchical clustering, a specific clustering method that aims to build a hierarchy of clusters [70]. More specifically, we use hierarchical agglomerative clustering. This refers to building the hierarchy of clusters in a bottom-up fashion. It starts with each item belonging to its own separate cluster, with pairs of clusters merged as one moves up the hierarchy. To decide what clusters to merge, the dissimilarity between sets of items must be determined. The distance between sets of items is usually how this dissimilarity is measured.

2.7.1 Metric

To do so, an appropriate distance metric must first be selected. Different metrics will impact the resulting shape of clusters in different ways. This is because the distance between two items will be different depending on what metric is used. In hierarchical clustering, there is no specific distance metric which must be used. In practice, the distance metric is not even necessary, as a matrix of distances will suffice. Popular metrics used for hierarchical clustering include:

Distance Metric	Formula
Euclidean	$\|a-b\|_2 = \sqrt{\sum_i (a_i - b_i)^2}$
Manhattan	$\|a-b\|_1 = \sum_i \|a_i - b_i\|$
Maximum	$\|a-b\|_\infty = \max_i \|a_i - b_i\|$

2.7.2 Linkage Criterion

After selecting a distance metric, a linkage criterion must be selected, which measures the distance between sets of items, and hence their dissimilarity. It is a function of the pairwise distances between items, as measured by the selected distance metric. Some popular linkage criteria between sets of items A and B include:

Linkage Criterion	Formula
Complete	$\max \{\, d(a,b) : a \in A,\, b \in B \,\}$.
Single	$\min \{\, d(a,b) : a \in A,\, b \in B \,\}$.
Average	$\frac{1}{\|A\| \cdot \|B\|} \sum_{a \in A} \sum_{b \in B} d(a,b)$.

Ward's method is another popular linkage criterion [69]. Each step of the method involves finding the pair of clusters that, after merging, leads to the smallest increase in total within-cluster variance. The pair is then merged.

2.7.3 Obtaining clusters

Let's take the following as an example of raw data to be clustered:

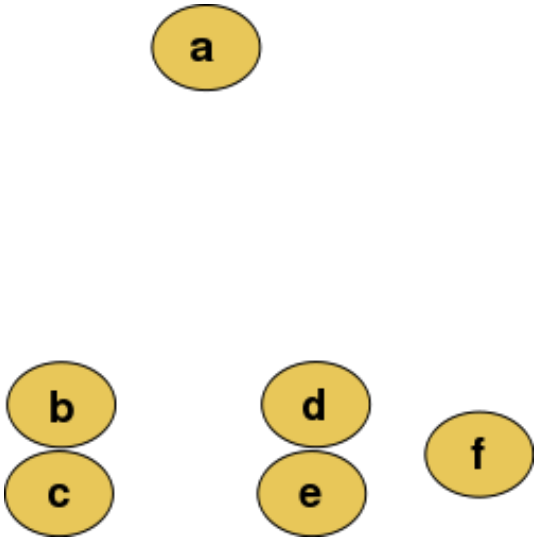


Figure 2.5: Raw Data to Cluster [6]

After selecting a distance metric and linkage criterion, applying hierarchical agglomerative clustering results in a hierarchy of clusters. Constructed hierarchies are usually illustrated via a dendrogram - a diagrammatic representation of a tree. The resulting dendrogram would hence look like the following:

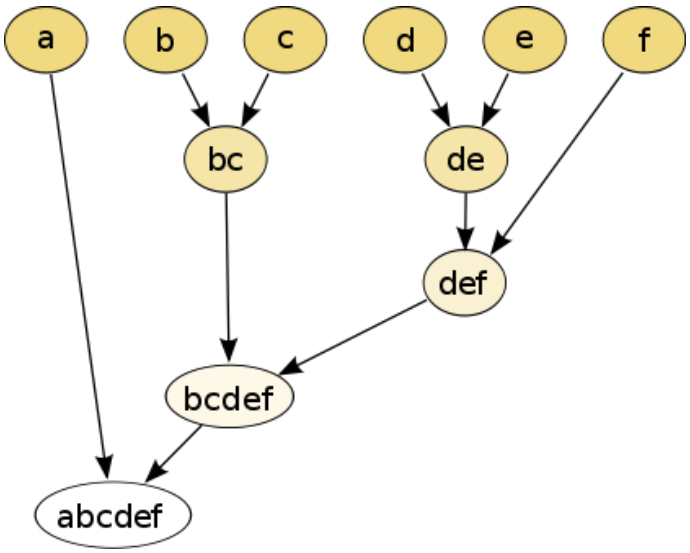


Figure 2.6: Constructed Dendrogram [7]

We can specify the precision of our clustering by "cutting" the dendrogram at specific heights. For example, cutting above the row containing "bc" and "de" yields all items as individual clusters. Cutting below the row containing "bc" and "de" will yield larger but fewer clusters "bc", "de", "a", and "f". At what level to cut the dendrogram is usually specified by either 1) determining the number of desired clusters 2) setting a maximum threshold value for the linkage criterion, so that clusters which are too dissimilar are not considered.

2.8 Previous Work

We now outline previous research related to the work we do in this project.

2.8.1 Sentiment Analysis - Reader vs. Writer

The point of view is sometimes ignored or assumed to be the writer in Sentiment Analysis research and applications [59]. However, there exists some awareness of its importance. Godbole et al. were one of the first to highlight the existence of different perspectives [40]. Buechel et al. [27] argue for a third perspective, that of the text. Lin et al. studied the classification of news articles explicitly in terms of the readers' perspective [58]. Some work has also been done on the relationship between the affective states of the Reader and Writer. Chen and Tang [89] examined the relationship between the sentiment of micro-blog posts (Writer) and the comments (Reader).

2.8.2 Sentiment Analysis – Dimensional vs Categorical

Sentiment Analysis within the context of NLP usually concerns sentiment polarity, with respect to the point of view of the Writer. Polarity can be said to be equivalent to the aforementioned Valence dimension, also known as Pleasure. Sentiment Analysis has often been viewed as a classification problem, with the determined polarity/valence being positive, negative or even neutral. When viewed as a regression/ordinal problem, also known as fine-grained Sentiment Analysis [97], a score is given to indicate the intensity. This can involve a single pole for both positivity and negativity, or independent positive and negative poles (or additionally, neutral) [54]. Recently, more work including the emotional dimensions besides Valence has been done. In such instances, the problem is considered a regression problem with predicted scores for each individual dimension. For example, Wang et al. performed dimensional sentiment analysis using a regional CNN-LSTM model [92]. .

Work involving the categorical model of emotion has also been done [96], where text is classified as being one of the categorical emotional states. Similarly, a regression/ordinal perspective is sometimes used, where each emotion is scored depending on intensity or probability.

2.8.3 Facebook Reactions

There are two broad approaches to the utilisation of Facebook Reactions as labels. The first is taking the Reaction with the highest count and using it as the assigned emotional label for a Facebook post. This is a classification problem and is the approach used by Pool et al. [74] and Tareaf et al [75]. The other approach is to predict the distribution of Reactions, which is a regression problem. To simplify the problem, the distributions are converted to sum to 1, so that the task is to predict the ratio of each reaction as opposed to the actual count. It is important to note that the ratios do not represent intensity of emotion. Only Krebs et al. [55] have previously used this approach for Facebook Reactions.

Goodman et al. use BuzzFeed article reactions in a similar way, predicting reaction distributions, but instead use the the full content of an article to perform

predictions [41]. This is in contrast to the previously mentioned Facebook-centred approaches, which use just the post text, which is often just a summary or headline which may link externally to the full content.

2.8.4 Positive News

A growing number of very popular websites, Facebook pages, YouTube channels and podcasts have sprung up over the past few years, with the goal of serving consumers feel-good news [79]. To the best of our knowledge, these mainly involve human-curation of news content as opposed to fully automated processes.

In terms of previous work, applying sentiment analysis to news is a very popular area of research. Godbole et al. present a system which assigns negative or positive sentiment to each identified entity in a large-scale text corpus of news articles and blogs, identifying what entities have more positive or negative associations [40]. They use a lexicon-based approach. Balahur et al. perform a series of experiments using a lexicon-based approach to identify key issues with sentiment analysis as applied to news [23]. Among their insights are 1) the difference in vocabulary used between news and more subjective content like movie reviews, 2) the existence of different perspectives, such as the reader versus the writer, 3) the importance of identifying the source and target of sentiment.

In specific relation to research on the development of news applications which provide positive news, we identify "EmoNews: An Emotional News Recommender System" by Kazemifard et al [72]. Their research involved the development of a mobile application which provides positive news articles, but uses a recommender system approach as opposed to a sentiment analysis-based approach. Users provide ratings indicating their emotional reaction as opposed to their content preference, which in turns improves the recommendations of their hybrid collaborative and content-based recommender system approach.

2.8.5 Emotional Arcs

Reagan et al. claim that their analysis of works from Project Gutenberg reveal that the emotional arcs of stories are dominated by six basic shapes [78]. The shapes are as follows:

- Rags to riches – Ongoing emotional rise.
- Riches to rags – Ongoing emotional fall.
- Man in a hole – Emotional fall followed by a rise.
- Icarus – Emotional rise followed by a fall.
- Cinderella – Emotional rise-fall-rise.
- Oedipus – Emotional fall-rise-fall.

They use a combination of techniques – Singular Value Decomposition, Hierarchical Clustering, and Self-Organizing Maps (SOM) to back up their claims. They use a lexicon-based approach (labMT lexicon [77]), constructing emotional arcs by analysing the sentiment of 10,000 word windows per text. In addition to that,

they demonstrate that certain core arcs are associated with higher or lower levels of popularity, measured by number of downloads.

Jockers also uses a lexicon-based approach (NRC lexicon [68]), but performs sentence-level sentiment analysis as opposed to sliding word windows [49]. Discrete Cosine Transforms (DCT) are then used to smooth the resulting trajectories. Jockers uses hierarchical clustering on a corpora of 40,000 novels to reveal between six and seven core arcs.

Vecchio et al. shift domains from literature to movies, exploring the relationship between emotional arcs and box office performance [32]. They follow the approach used by Jockers to construct emotional arcs from scraped movie subtitles, and use k-means clustering to identify universal arcs. They succeed in obtaining the same six emotional arcs as Reagan et al. They reveal that certain emotional arcs are, namely *Man in a Hole*, are associated with better box office performance.

The previously mentioned works are only with respect to the Valence dimension of emotion. To the best of our knowledge, research concerning the Arousal and Dominance dimensions of emotion has not been done with respect to emotional arcs. However, some work has been done with respect to categorical representations of emotion.

Working with the Fairytale Corpus and Corpus of English Novels (CEN), Mohammad et al. [67] use the NRC lexicon to construct emotional arcs for various categories of emotion (joy, surprise, sadness etc.) by splitting each text into segments.

Samothrakis et al. take a lexicon-based (Wordnet-Affect [87]), per-sentence approach, with smoothing conducted via a Hamming smoother [82]. They conduct an analysis of texts from Project Gutenberg to show that emotional content is a good predictor of genre.

Kim et al. use a lexicon-based approach (NRC lexicon) to construct emotional arcs by splitting each text into five segments [52]. They then average the arcs per-emotion and per-genre to show that each genre has characteristic emotional arcs.

Chapter 3

Reader Model Development – Part 1

3.1 Introduction

Our aim is to develop a machine learning model which performs sentiment analysis from the Reader's perspective, in a way which indicates the mixed nature of Reader reactions. We call this model our Reader model, and it forms the foundation for the rest of the work in this project. This chapter outlines the process we take in obtaining such a model.

3.2 Emotional Reaction Data

To train our machine learning model, we first need to obtain a suitable dataset. The two main sources of mixed emotional reaction data we considered were Facebook and BuzzFeed, as they are two popular platforms with "reaction" features as previously described. Such data is suitable for our needs, as they are explicitly from the Reader's perspective, and the distribution of reactions signifies the mixed nature of emotional response.

3.2.1 Data Collection Challenges

The introduction of Facebook Reactions in early 2016 [85] has made Facebook a wealthy source of data for studying the emotional responses of the reader. However, Facebook has recently made changes to its Graph API [5], making large-scale data collection of any kind on its platform challenging. Previous work utilising Facebook reaction data, or Facebook data in general, mostly occurred before this change[74][55]. Web scraping is an alternate approach to data collection, with various platform-specific tools available, but there exists uncertainty with respect to its legality [15], especially when considering that web scraping for this project may occur in different countries. Some previously cited works have chosen this approach [55], but we choose not to engage with any data collection ourselves, and opt to only use publicly available datasets from previous academic work. BuzzFeed does not have an API, and the same concerns with respect to web scraping apply.

3.2.2 Choosing a Dataset - Martinchek Dataset

We evaluate various publically available datasets. The dataset we opt for is the Martinchek dataset [1], which was previously used for an analysis of political coverage in the media [16]. It contains Facebook post data from the public Pages of various news outlets, namely, Fox and Friends, Fox News, The Washington Post, The Huffington Post, NBC News, CBS News, The New York Times, ABC News, CNN, The Los Angeles Times, USA Today, The Wall Street Journal and NPR. We can then extract from the data news headlines, which we deem suitable as they are short and often designed to provoke an emotional reaction in the reader to capture their attention. In addition to that, the size of the dataset is much larger than the others we encountered [55].

For this project, we therefore predict an emotion distribution consisting of five categories corresponding to the Facebook Reactions – "Love", "Haha", "Sad", "Wow" and "Angry". These can be mapped to defined categories of emotion "Love", "Amusement", "Sadness", "Surprise" and "Anger" [73]. However, we maintain the original naming scheme for the sake of simplicity.

The Martinchek dataset consists of posts from January 1st 2012 to November 8th 2016. It consists of csv files organised per news source, which we then combine. Table 3.1 demonstrates the columns contained within each csv file as well as the values for an example Facebook post. The original dataset consists of a total of 534,391

Column Name	Example Value
"id"	5550296508_10155412956406509
"page_id"	5550296508
"name"	"At least 264 people in Haiti have been killed by Hurr..."
"message"	"Three days later, scale of Hurricane Matthew's..."
"description"	-
"caption"	www.cnn.com
"post_type"	link
"status_type"	shared_story
"likes_count"	826
"comments_count"	143
"shares_count"	416
"love_count"	7
"wow_count"	84
"haha_count"	3
"sad_count"	779
"thankful_count"	0
"angry_count"	3
"link"	http://cnn.it/2dWLFRC
"picture"	https://external.xx.fbcdn.net/safe_image.php?d=...
"posted_at"	2016-10-07 07:01:02

Table 3.1: Sample from Martinchek Dataset

sample posts and their associated data, with a total of 129,193,794 Reactions and

1,726,119,512 Likes. We can also find many concrete examples of mixed sentiment, for example in Figure 3.1.

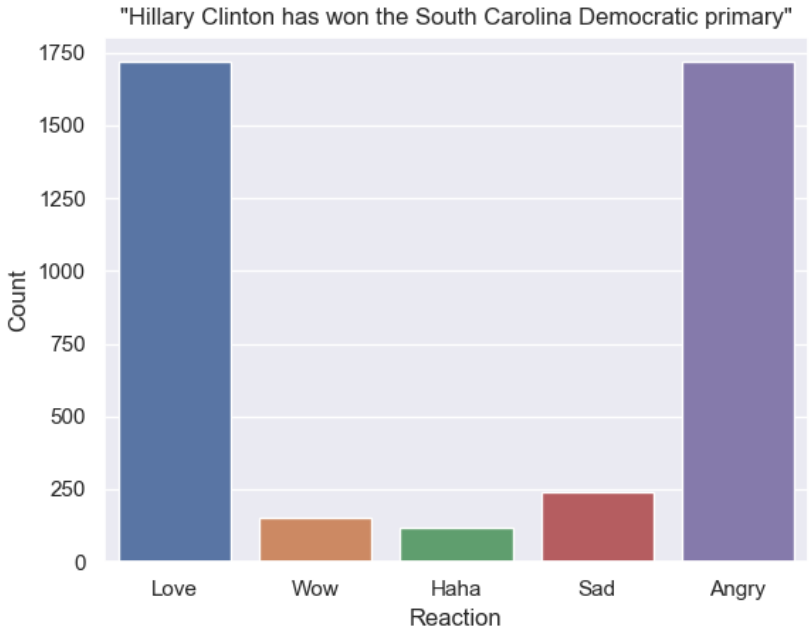


Figure 3.1: Headline With Mixed Reactions (Excluding Likes)

This provides further evidence of the utility of a mixed view of sentiment.

3.2.3 Processing the Dataset

We first apply some filtering to our dataset. We filter out posts which do not have a `post_type` of "link", as other post types, for example "image" or "video" tend to contain no text or noisy text such as "WATCH THIS VIDEO NOW". As we desire a full distribution of emotional responses, but the dataset contains both posts before the introduction of Reactions and those with no Reaction responses, we also filter out posts with 0 Reactions. It is important to note we consider Likes separate from Reactions. Figure 3.2 shows the change in dataset size after applying these conditions.

We then have to decide which field will constitute our headlines. We have the fields `name`, `message`, and `description` as viable options. We select `name` as from manual inspection, it is the most consistently used field while the others occasionally do not contain text. In addition to that, we We also make the decision to normalise each post's Reaction and Likes distribution to sum to 1, so the counts for each category are now *ratios*. Predicting the Reaction distribution as a series of counts is a harder problem than just predicting the ratios. Predicting the counts would also mean changing the focus of the project to include predicting the popularity and reach of a post, as opposed to just focusing on the emotional information which can be gleaned from the text.

An important decision we make is the exclusion of Likes from our emotion distribution, leaving only Reactions. We experiment with both including and excluding Likes in the distribution. As illustrated in Figure 3.3 it can be seen that performance when including Likes (mean squared error of 0.0063) is substantially better

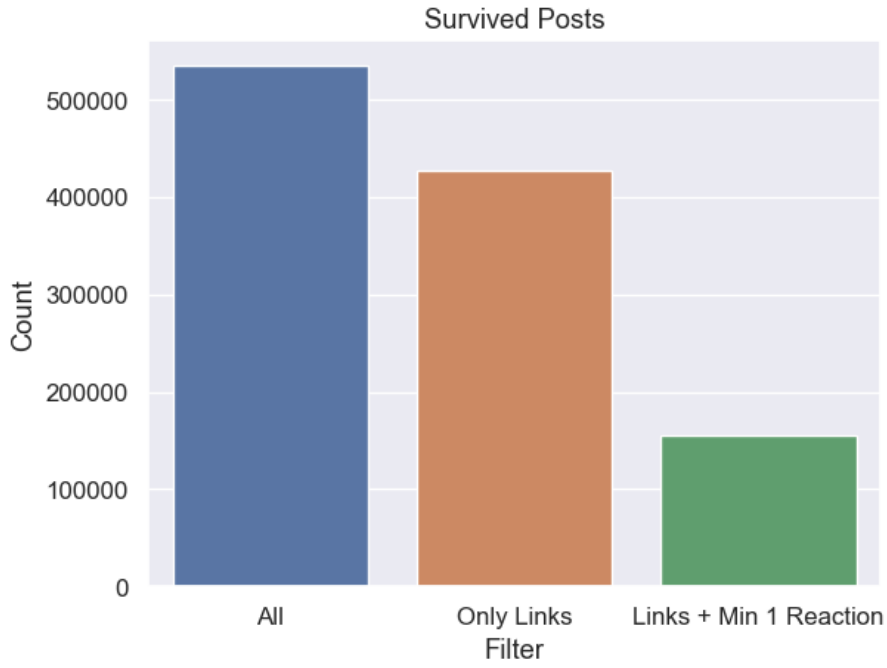


Figure 3.2: Number of Survived Posts After Filtering

than when only using Reactions (mean squared error of 0.0518). Further details are in Appendix A.1.

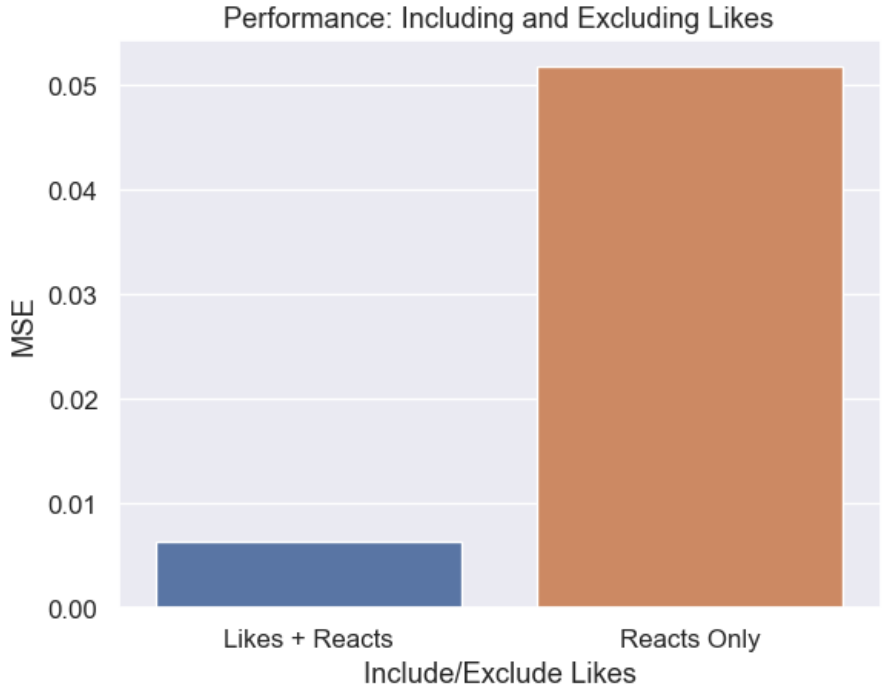


Figure 3.3: Comparing Performance When Including and Excluding Likes

However, inspecting model predictions shows that the model predicts a value close to 1 for Like every time. This can be attributed to the fact that the Like response is much more common than the Reaction responses, as illustrated in Figure 3.4. We suspect that this is due to the fact that, although "liking" something can

be seen to be a positive emotional response, "Like-ing" something on Facebook is commonly used as a “catch-all” response, with people selecting it as their response to a post no matter the actual emotional response. However, as our focus is on the actual emotions experienced by the users, we decide to exclude Likes from the predicted distribution, leaving Love, Wow, Haha, Sad and Angry. Figure 3.5 and Table 3.2 demonstrate the new distribution.

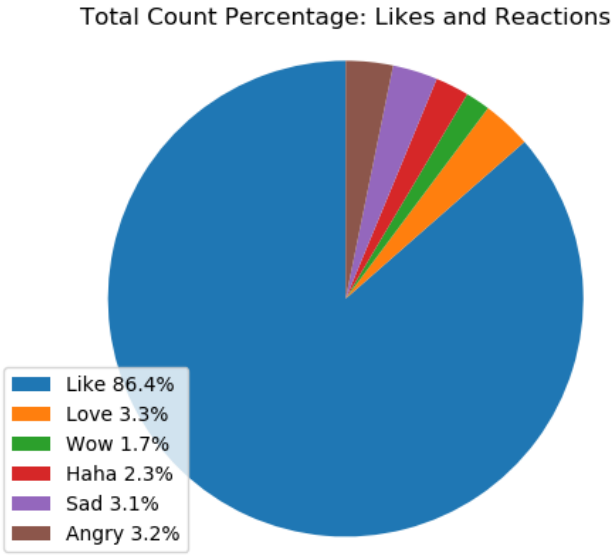


Figure 3.4: Percentage of Likes and Reactions (Filter: Link + Min. 1 Reaction)

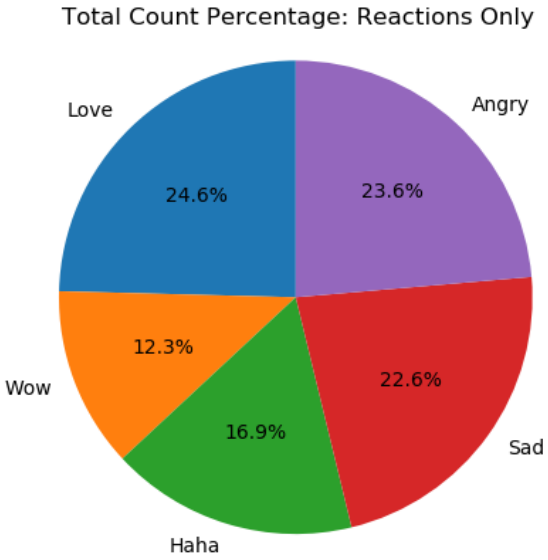


Figure 3.5: Percentage of Reactions Only (Filter: Link + Min. 1 Reaction)

Reaction	Count
Love	18,147,501
Wow	9,107,811
Haha	12,447,344
Sad	16,662,370
Angry	17,435,906
Total	73,800,932

Table 3.2: Total Reaction Counts (Filter: Link + Min. 1 Reaction)

After excluding `likes_count`, we also remove `thankful_count`, as the associated Thankful Reaction was a temporary Reaction introduced only for Mother's Day 2016 [14]. We then remove other unrelated fields like the number of comments. Table 3.3 indicates the final columns of our dataset.

Column Name	Example Value
headline	"At least 264 people in Haiti have been killed by Hurr..."
love_ratio	0.007
wow_ratio	0.095
haha_ratio	0.003
sad_ratio	0.889
angry_ratio	0.03

Table 3.3: New Dataset Format

We then perform some additional pre-processing on values of the `headline` column:

- Remove non-ASCII characters
- Lowercase all the text
- Separate punctuation and words

We perform the above additional pre-processing using `nltk`, a popular Python library for natural language processing.

3.3 Developing the Reader Model

Overall, we experiment with three different approaches to developing our Reader Model:

- Bag-of-words approach
- Sequence vectors approach
- BERT approach

Machine learning algorithms do not directly accept text as input. Hence for each of these approaches, we need to perform tokenization and vectorization so we can represent our text as numerical vectors. Tokenisation refers to dividing text into tokens, with the purpose of enabling the relationship between texts and labels to be generalised. Tokens are themselves usually words or smaller sub-texts like characters. Tokens are what make up the vocabulary of the dataset, the vocabulary being the set of unique tokens present. Vectorization refers to the manner in which we represent text as vectors.

3.3.1 Bag-Of-Words Approach

We first use the bag-of-words approach. With this approach, text is represented quite literally as a bag of words, disregarding word order and grammar [13]. Unlike a set, a bag allows for multiple instances of each element to exist.

Tokenization and Vectorization

In the bag-of-words approach, we first tokenize text into n-grams. n-grams refer to sequences of adjacent tokens of length n, the value of which we must decide. We then assign indexes to the vocabulary (set of unique tokens) of the dataset.

We then need to perform vectorization. We use and evaluate three commonly used approaches:

- One-hot encoding: In this approach, each text sample is represented as a vector. A zero at the index of a token represents its presence. A one represents its absence.

- Frequency encoding: Similar to the above, but each value at the index of a token refers to its frequency in the sample.

- TF-IDF encoding: Short for term frequency-inverse document frequency, TF-IDF was developed to avoid a problem with frequency encoding. Namely, the most frequent tokens in a document are often the most frequent tokens in the entire corpus. It is a more meaningful statistic indicating the importance of a word to a document in a corpus (collection of documents) [76].

In practice, this means a matrix with column size equal to vocabulary size, and row size equal to total dataset size (number of text samples), is produced. Values then depend on the type of encoding used.

We use `sklearn`, a popular machine learning library for Python which includes tools for natural language processing, to perform the above tokenization and vectorization operations. We also experiment with varying the number of features used, as the result after tokenization can be a very large number of unique tokens (features). Not all will contribute to better performance, such as those which occur extremely rarely in the text, and hence can be disregarded. We select features by frequency, only considering the top x features (tokens) across the corpus, ordered by count. We also experiment with using bigrams in addition to unigrams.

Machine Learning Model

Now that our input is in a form which can be fed into a machine learning model, we must now decide what model to use. As the bag-of-words approach is orderless, machine learning models which do not recognise sequential order are often used. Examples of such models are Naïve Bayes, Gradient Boosting Machines, Logistic Regression, Support Vector Machines, and Multilayer Perceptrons (MLP). Our model of choice for this approach are multilayer perceptrons, as a comprehensive study by Google showed that, with respect to performance and computational cost, they are a preferred option when used in conjunction with bag-of-words [12].

We use `keras`, a neural network API written in Python, for training and evaluation of our models. The evaluation results are as follows:

Reader Model Evaluation: Bag-Of-Words		
Vec. Encoding	Model	MSE
One-hot	MLP	0.0518
Frequency	MLP	0.0522
TF-IDF	**MLP**	**0.0486**

We see that the best performing model is MLP with TF-IDF encoding, which is not unexpected. Further details, such as architectures, hyperparameters and training and evaluation procedures used, can be found in Appendix A.2.

3.3.2 Sequence Vectors Approach

Word order is often an important consideration in text, and taking sequential structure into account can help improve performance. For example, the meaning of the sentence "I used to love my job. My new boss changed that." can only be accurately surmised when read sequentially. This approach hence preserves order by representing text as a sequence of tokens, and by using machine learning models which can derive information from such sequential ordering [13].

Tokenization and Vectorization

In terms of tokenization, text is tokenized into sequences of words or characters. The latter is usually only used when dealing with text of a differing language but similar characters, or text with frequent spelling mistakes [13]. We hence follow convention and tokenize our dataset into words.

Unique tokens (the whole vocabulary) are then assigned indexes, and the text samples, which are sequences of tokens, are then converted to sequences of these indexes. It should also be noted that sequences should be of fixed length. We set the length to be the maximum number of tokens in a sentence in our corpus, and truncate or pad accordingly. To then vectorize these sequences of indexes, we consider two main options:

- One-hot encoding - Each token in a sequence is represented by a vector of length equal to the vocabulary (unique tokens in dataset) size. This can lead to very large and sparse vectors when the vocabulary of words is large. This approach is more suitable for when tokenizing as characters, as character sets are usually much smaller.

- Word embeddings - Represent each unique token as point in a dense vector space. A word embedding captures the meaning of a word by placing semantically similar words closer together in the vector space.

We opt to use word embeddings instead of one-hot encoding as they provide a semantically richer representation of words. We consider whether to use pre-trained embeddings or train them from scratch, as well as whether to freeze the embeddings or fine-tune them when training. We ultimately decide to both use pre-trained embeddings and train our own from scratch, as we can obtain our own corpus from our Facebook data. We then decide to freeze the embeddings during training, because of the high computational cost associated with fine-tuning.

We implement and evaluate a variety of word2vec word embedding models:

- Google word2vec - This refers to pre-trained embeddings produced by Google [65]. They were trained on part of a Google News dataset using word2vec, with the model containing 300-dimensional vectors for over 3 million words and phrases.

- Facebook word2vec - We train our own word2vec word embeddings from scratch, using the original Martinchek dataset which consists of over 500,000 text samples.

- Emotion-aware embeddings - This refers to the emotion-enriched embeddings provided by Agrawal et al [17]. Although not explicitly in word2vec embedding format, conversion was straightforward as doing so simply involved prepending vocabulary size to the front of the provided embeddings file.

- Retrofitted FB word2vec - We "retrofit" the above Facebook word2vec embeddings with affective information, achieving a similar effect to the above emotion-aware embeddings. To accomplish this, we use the retrofitting tool provided by Faruqui et al [37]. and the NRC emotion lexicon [68], which is a list of English words and their associations with eight basic emotions, manually annotated.

We use `gensim`, a Python library for natural language processing with word embedding functionality, to train and load the above embedding models as necessary. We use a multilayer perceptron in conjunction with each embedding model (embeddings loaded into an embedding layer) to decide which to perform further experimentation with. `keras` was again used for this.

Embedding Model Selection		
Embedding Type	Model	MSE
Google word2vec	**MLP**	**0.0532**
Facebook word2vec	MLP	0.0572
Emotion-aware embeddings	MLP	0.0541
Retrofitted FB word2vec	MLP	0.0547

We can see that Google word2vec is the best performing embedding, likely due to it being produced from the significantly largest corpus. The rest of the embedding models perform worse, likely due to them being produced from smaller corpora. Retrofitted FB word2vec and emotion-aware embeddings perform similarly, with

both performing better than their vanilla counterpart, indicating the effectiveness of the retrofitting process. As results from Google word2vec embeddings were satisfactory, we did not also apply the retrofitting process to them, due to the very high computational cost.

Machine Learning Models

Some model types, such as CNNs, RNNs and their variants, can learn from the sequential nature of text [12]. We evaluate the performance of various sequential models to use in conjunction with our best performing embedding mode, Google word2vec. We again use `keras`.

Reader Model Evaluation: Sequence Vectors		
Embedding Type	Model	MSE
Google word2vec	MLP	0.0532
Google word2vec	CNN	0.0531
Google word2vec	**RNN (LSTM)**	**0.0517**

We can see that the best performing model is Google word2vec used in conjunction with the LSTM. This suggests that modelling longer-range dependencies, which LSTMs do, is more immediately helpful for this problem despite text lengths being relatively short. CNNs' focus on local patterns is less helpful. Further details, such as architectures, hyperparameters and training and evaluation procedures used, can be found in Appendix A.3.

3.3.3 BERT Approach

Bidirectional Encoder Representations from Transformers (BERT) has led to state of the art performance across multiple natural language processing tasks [34]. However, BERT and its variants have yet to be applied to our specific sentiment analysis task of predicting reaction distributions.

We consider both approaches to using BERT. Namely, fine-tuning and extracting embeddings for external use. After considering both approaches, we decide to use the fine-tuning approach over the embedding approach, as it was the procedure used in obtaining state of the art results in the original BERT paper as well as further research. Furthermore, the scenarios where the embedding approach may prove better are not as applicable to us.

We also consider what variants of BERT to evaluate, settling on the following:

- Standard BERT - As we are evaluating BERT-like models, it makes sense to evaluate the original [34].
- RoBERTa - The original BERT architecture, maximised for performance [61].
- DistilBERT - Computationally inexpensive relative to other options [83].
- ALBERT - Same reasoning as for DistilBERT [57].

We also consider evaluating other BERT and transformer-like approaches, but decide to start with the above.

Tokenization and Vectorization

We need to prepare our text for input into a BERT model. Although some differences exist in terms of tokenization and vectorization procedures for each BERT variant, the following steps all apply:

1. Tokenize text sample into tokens

2. Add special token `[SEP]` – Append `[SEP]` to the start of each text sample. This token is required to distinguish between sentences in multi-sentence input tasks, but is still necessary for single sentence input tasks like ours.

3. Add special token `[CLS]` – Prepend `[CLS]` to each text sample. This was intended for classification tasks, but is still applicable to us. Each layer in a transformer architecture takes in an equal number of token embeddings as it outputs. However, only the `[CLS]` token embedding is used for classification tasks, with it being fed into the downstream-task specific layer on top of the transformer architecture. This means that the model is trained to imbue all required information for classification into this single token. This means that pooling over all output embeddings into a single embedding is not required, as `[CLS]` effectively already acts as a sequence embedding. See Figure 3.6.

4. Pad or truncate text samples – Inputs have to be of a fixed length (maximum 512 tokens). Padding or truncation is therefore required. For our project, we take the maximum length in our corpus and pad everything to that length. Padding is done via another special token, `[PAD]`, with vocabulary index 0.

5. Convert tokens to their corresponding vocabulary indexes

6. Create attention mask – We need to differentiate the "pad" tokens from the "real" tokens with an array of ones and zeros (the attention mask). This is so "self-attention" [91], an important mechanism in BERT, is not applied to these tokens. The masks are fed in as input alongside the input sequences (of vocabulary indexes) to the model. See Figure 3.7.

We use `transformers`, a `pytorch` library developed by HuggingFace, to perform the appropriate steps. We use `transformers` due to its relative ease-of-use and excellent documentation, and it also has support for many other transformer architectures. `pytorch` is a machine learning library developed by Facebook [51].

Machine Learning Models

We use `transformers` to load the models and pretrained weights of the above BERT variants. It also provides models which come attached with downstream task-specific layers which sit on top of the main transformer architecture. However, we had to implement our own as they did not have support for downstream multi-dimensional regression tasks. As `transformers` is a `pytorch` library, we train and evaluate using `pytorch` as well.

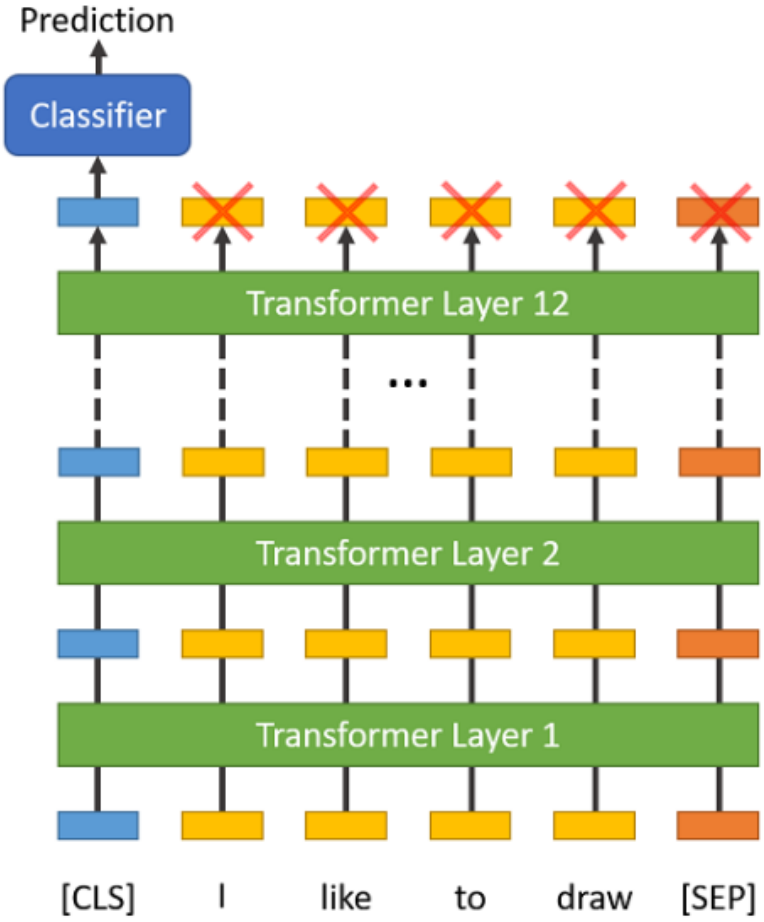


Figure 3.6: Addition of Special Tokens [2]

Reader Model Evaluation: BERT	
Model	MSE
DistilBERT	0.0441
BERT	0.0439
RoBERTa	**0.0411**
ALBERT	0.0473

The results are as expected. Larger pretrained models tend to have better performance [61], hence why the smaller models DistilBERT and ALBERT have worse performance (although DistilBERT performance is not far from BERT). We can see that RoBERTa is the best performing model, which also makes sense as it is the same architecture as BERT but pretrained more effectively. We are overall satisfied with performance, and therefore do not pursue evaluation of other BERT and transformer-like approaches.

Further details, such as architectures, hyperparameters and training and evaluation procedures used, can be found in Appendix A.4.

3.3.4 Comparing All Approaches

We compare the best model of each of the different approaches.

Figure 3.7: Addition of Attention Mask [2]

Reader Model Evaluation: Overall		
Approach Type	Approach	MSE
Bag-Of-Words	TFIDF + MLP	0.0486
Sequence Vectors	Google w2v + LSTM	0.0517
BERT	**RoBERTa**	**0.0411**

We can see that RoBERTa is the best performing model, outperforming the other approach types by a significant margin. Given the state-of-the-art performances from BERT and BERT-like architectures across a variety of tasks, this is somewhat expected.

3.3.5 Improving Performance Further: Minimum Reaction Threshold

We previously made the decision to filter out posts with a Reaction count of 0 from our dataset (minimum Reaction threshold of 1). Using the RoBERTa model, we experiment with increasing this threshold to higher values to observe changes in performance. The reasoning behind this is that an increased minimum threshold would mean that only posts with a larger sample size to constitute a more accurate and less "noisy" emotion reaction distribution would be included.

We apply minimum thresholds of 1, 5, 10, 15, 20, 25 to both the training and test data.

Performance Across Min. Thresholds		
Data Applied	Min. Threshold	MSE
Train + Test	1	0.0411
Train + Test	5	0.0395
Train + Test	10	0.0377
Train + Test	15	0.0361
Train + Test	20	0.0334
Train + Test	25	0.0329

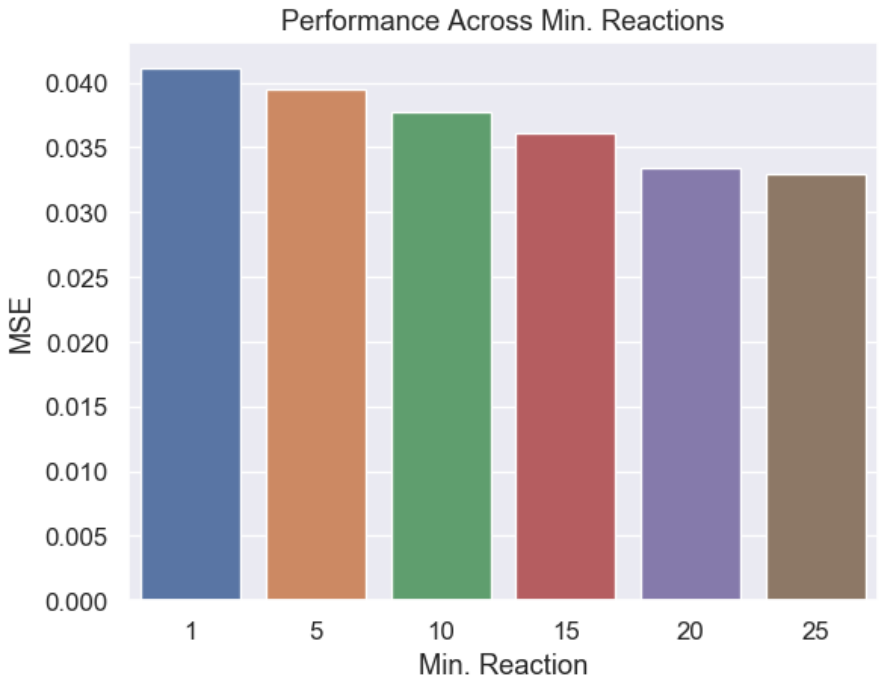


Figure 3.8: Performance Across Min. Reaction Thresholds

We observe from Figure 3.8 that performance increases. Note that filtering the test data in such a manner does not mean we are increasing performance in predicting "real-world" Facebook Reaction distribution samples. Indeed, when applying the minimum thresholds only to training data and not test data, performance decreases the higher the minimum threshold.

Performance Across Min. Thresholds		
Data Applied	Min. Threshold	MSE
Train Only	1	0.0416
Train Only	5	0.0420
Train Only	10	0.0419
Train Only	15	0.0423
Train Only	20	0.0428
Train Only	25	0.0430

We suspect this is because the test data remains noisy due to posts with low sample sizes of Reactions, and training the model with cleaner data does not improve performance on the noisy data. In addition to that, the training dataset size has been reduced as can be seen from Figure 3.9.

Whether to apply a higher minimum threshold of Reactions therefore depends on the objective:

- Objective 1: To be more accurate in predicting "true" emotion reaction distributions (the distribution of emotions of as given by an infinitely large sample size).
- Objective 2: To be more accurate in predicting "raw" Facebook Reaction distributions (predicting concretely the distribution of Reactions on any given Facebook post, with no filtering condition such as a minimum threshold of Reactions).

For this project, we are interested in Objective 1. The performance gains around a minimum Reaction threshold of 25 are beginning to be marginal. This is perhaps because a similar increase in threshold does not lead to as significant a reduction in noisy data, which can be again observed from Figure 3.9.

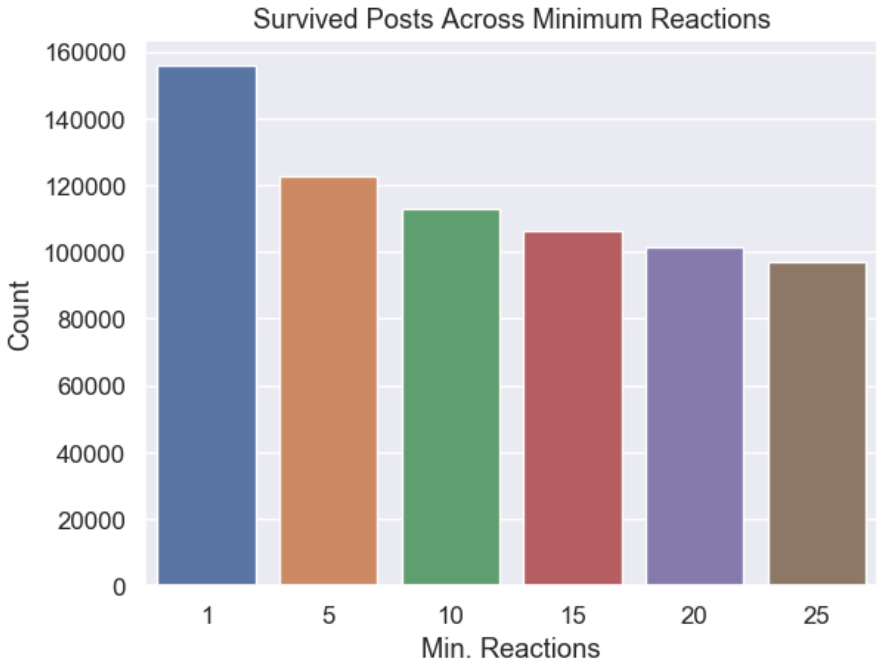


Figure 3.9: Survived Posts After Applying Minimum Reaction Thresholds

We use the model trained with the minimum threshold of 25 as our base Reader model, with a mean squared error of **0.0329**.

Chapter 4

Reader Model Development – Part 2

4.1 Introduction

We now have a Reader Model which fulfills our main objective – predicting Reader emotions in a mixed fashion. In this chapter, we investigate the feasibility of integrating predictions from other types of sentiment analysis to further improve performance.

4.2 Related Affective Information

We have so far taken a narrow view of affective information within the context of sentiment analysis. We have focused on categorical representations of emotion, the Reader's perspective, and stemming from the latter, the mixed nature of Reader emotional response. However, other types of sentiment analysis exist, for example sentiment analysis with respect to dimensional representations of emotion from the Writer's perspective.

Following that, a thus far unexplored avenue for improving performance in sentiment analysis is the integration of different, but related, affective features which can be obtained from these other types of sentiment analysis. Our task is therefore to explore this approach.

4.3 Modes of Sentiment Analysis

We refer to the different types of sentiment analysis we consider as "modes". To clarify our options, we propose the following classification for modes, across three main dimensions:

- Perspective (Writer vs. Reader)
- Emotional Representation (Categorical vs. Dimensional)
- Measurement Type (Classification, Mixed, Fine-grained etc.)

For example, a common form of sentiment analysis is simply determining whether expressed emotion in a text is positive or negative. This falls under the Writer perspective, and since positivity or negativity of text refers to the Valence dimension

of emotion, it falls under the Dimensional category. As we are simply predicting "positivity" or "negativity", this is a classification problem.

Our own Reader model can be classified as the following:

- Perspective: Reader
- Emotional Representation: Categorical
- Measurement Type: Mixed

We hence investigate the usage of modes different to our own.

4.4 Challenges

We first investigate various tools and previously constructed models to look for ones with modes different to our own. This is to see if we can conduct our investigation without having to build our own separate model. However, we face some challenges. It is clear that determining the Emotional Representation used by a tool is straightforward. The Emotional Representation can be deemed to be Categorical or Dimensional simply by reading the documentation or inspecting outputs. Similarly for Measurement Type. However, determining the Perspective is challenging as it is often not explicitly stated, can be inconsistent, or not obvious when reading documentation or assessing outputs by hand.

To be clear on what alternative mode of sentiment analysis we are using, we instead decide to train our own model. However, dataset availability is now a major consideration. We run into similar problems in determining the specific mode of a dataset. However, we do manage to find a single dataset, EmoBank, which suits our needs.

4.5 EmoBank

EmoBank [26] is a manually annotated dataset with two separate parts (each in a csv file). It contains the following two modes:

EmoBank Writer Dataset

- Perspective: Writer
- Emotional Representation: Dimensional
- Measurement Type: Regression

EmoBank Reader Dataset

- Perspective: Reader
- Emotional Representation: Dimensional
- Measurement Type: Regression

EmoBank uses the Valence-Arousal-Dominance model of emotion, with scores for each emotional dimension rated between 1-5 (3 denotes a neutral affect). To the best of our knowledge, it is the only existing dataset which is explicit with respect to the Reader and Writer perspective.

It consists of 10,0062 samples across various categories, as can be seen in Table 4.1. The Reader and Writer dataset have the same set of text samples.

Category	Count
fiction	2893
letters	1479
newspaper	1381
blog	1378
headlines	1250
essays	1196
travel-guides	971

Table 4.1: EmoBank Categories and Counts

We remove unnecessary fields from the datasets:

Text	V	A	D
"Fuck you"	1.2	4.2	3.8
"I was feeling calm and private that night."	3.1	1.8	3.1

Table 4.2: Processed EmoBank Writer Format

We then apply the same additional pre-processing as for the Martinchek Facebook dataset (remove non-ASCII characters, lowercasing, separating punctuation and words) using `nltk`.

4.6 Choosing a Mode and Developing a Model

Given the EmoBank dataset, we now have the option of developing a model for a mode of sentiment analysis different to our own. In actuality, we could build two models, one for each EmoBank dataset. However, we decide to build just one due to time and computational resource limitations. Given our overall project focus on the Reader's perspective, we decide to build a model with the EmoBank Reader dataset, with a mode of **Reader-Dimensional-Regression**. The predictions of this model are the aforementioned related, but different affective features which we will later utilise.

We take the same approach and process as when developing our original Reader model. For more detail, refer to Chapter 3.

4.6.1 Bag-Of-Words Approach

Writer Model Evaluation: Bag-Of-Words		
Vec. Encoding	Model	MSE
One-hot	MLP	0.1549
Frequency	MLP	0.1680
TFIDF	**MLP**	**0.1227**

Further details, such as architectures, hyperparameters and training and evaluation procedures used, can be found in Appendix B.1.

4.6.2 Sequence Vectors Approach

Writer Model Evaluation: Sequence Vectors		
Embedding Type	Model	MSE
Google word2vec	**MLP**	**0.0844**
Facebook word2vec	MLP	0.1180
Emotion-aware embeddings	MLP	0.1012
Retrofitted FB word2vec	MLP	0.0948

We can see that Google word2vec is the best performing embedding. We then evaluate the performance of various sequence models to use in conjunction.

Writer Model Evaluation: Sequence Vectors		
Embedding Type	Model	MSE
Google word2vec	**MLP**	**0.0844**
Google word2vec	CNN	0.0949
Google word2vec	RNN (LSTM)	0.0927

Interestingly, the MLP outperforms the sequence models CNN and RNN (LSTM), which was not the case when developing our original Reader-Categorical-Mixed model. Further investigation is necessary to understand why. Further details, such as architectures, hyperparameters and training and evaluation procedures used, can be found in Appendix B.2.

4.6.3 BERT Approach

Writer Model Evaluation: BERT	
Model	MSE
DistilBERT	0.0725
BERT	0.0733
RoBERTa	**0.0698**
ALBERT	0.0769

Further details, such as architectures, hyperparameters and training and evaluation procedures used, can be found in Appendix B.3.

4.6.4 Selecting the Best Model

We can see that the RoBERTa is the best performing model, similar to previous results.

Writer Model Evaluation: Overall		
Approach Type	Approach	MSE
Bag-Of-Words	TFIDF + MLP	0.1227
Sequence Vectors	Google w2v + MLP	0.0844
BERT	**RoBERTa**	**0.0698**

Further details, such as architectures, hyperparameters and training and evaluation procedures used, can be found in Appendix B.3.

4.7 Final Ensemble Reader Model

Our final system is an ensemble, combining different but related affective features to perform Reader-Categorical-Mixed predictions (Ensemble R-C-M).

It consists of two base models:

- Our original Reader-Categorical-Mixed model as developed in Chapter 3 (Base R-C-M).
- Our Reader-Dimensional-Regression model as just developed (Base R-D-R).

On top of the base models is the meta-learner, which is a multilayer perceptron. Further details, such as architectures, hyperparameters and training and evaluation procedures used, can be found in Appendix C.

The whole system works as follows. For each text sample, the prediction from the Reader-Categorical-Mixed model (shape 1×5) and the prediction from the Reader-Dimensional-Regression model (shape 1×3) is stacked into a single vector (shape 1×8). The meta-learner then takes this combined vector as input, and recomputes the Reader-Categorical-Mixed prediction. This is illustrated in Figure 4.1.

We compare performance for both the ensemble Reader-Categorical-Mixed model and the base Reader-Categorical-Mixed model :

Ensemble Model Evaluation: Ensemble vs. Base	
Approach Type	MSE
Ensemble R-C-M Model	**0.0304**
Base R-C-M Model	0.0329

The ensemble performs better. This means that utilising Reader-Dimensional-Regression features can help improve performance for Reader-Categorical-Mixed sentiment analysis models. Considering the entire system as our final Reader model, the final Reader model performance is a mean squared error of **0.0304**.

This suggests that in general, related affective information from a different mode of sentiment analysis can indeed help improve predictions for other modes of sentiment analysis. However, more extensive work must be done across various modes.

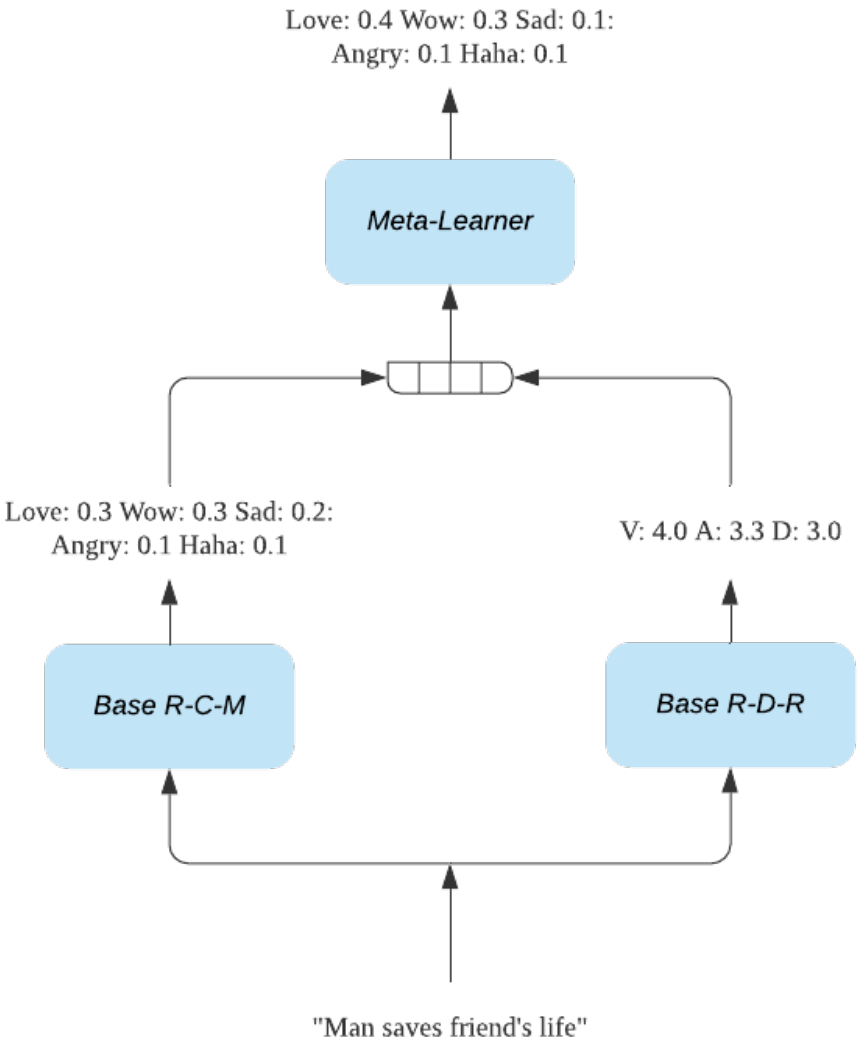


Figure 4.1: Ensemble Pipeline

4.8 Evaluating Classification Performance

An alternative way of using Facebook Reaction data is through the classification perspective, with the Reaction with the highest count being the "ground-truth" Reaction label for a post. Most work dealing with Facebook Reactions have used data in this way, and hence train machine learning models for classification. Tasks where we classify text with categorical emotions in such a manner are also known as "emotion recognition" tasks. Even though not explicitly trained to do so, we can also use our model for classification/emotion-recognition, by selecting the Reaction with the highest ratio in the predicted distribution as the predicted Reaction label for a post. In other words we can evaluate our ensemble Reader-Categorical-Mixed model on a Reader-Categorical-Classification task. In this section, we evaluate performance of our model for emotion recognition by using popular emotion recognition datasets. We compare our results to other works which have also used these datasets for

evaluation.

4.8.1 Datasets Used

We briefly introduce the datasets we use for evaluation. Note that the Perspective (Reader vs. Writer) of these datasets is not explicitly mentioned.

- Affective Text - This dataset was constructed by Strappavara et al. for Task 14 of SemEval-2007, an NLP workshop focused on semantic tasks. Similar to the Martinchek dataset, it consists of headlines from a variety of news sources, labelled with one of six emotions – "Anger", "Disgust", "Fear", "Joy", "Sadness" and "Surprise". There are 1,250 headlines in total. [86].

- Fairy Tales - This dataset was constructed by Alm et al. and contains 176 fairy tales [19]. Each sentence in a fairy tale is annotated with one of seven emotions – "Angry", "Disgusted", "Fearful", "Happy", "Neutral", "Sad" and "Surprised".

4.8.2 Previous Work

We briefly introduce the previous works we compare to.

- Pool et al. [74] - The work of Pool et al. uses Facebook Posts and Reactions as their dataset, similar to our work. Their approach uses GloVe embeddings, a variety of standard textual features and Support Vector Machines (SVM). They present several models, each trained on a variety of combinations of Facebook pages. We refer to the model they call B-M as POOL-B, the FT-M model as POOL-FT and the ISE-M model as POOL-ISE. They evaluate their models on several datasets, including Fairy Tales and Affective Text.

- Strapparava et al. [86] - The best performing model of Strapparava et al. is based on Latent Semantic Analysis (LSA) and the use of the WordNet and WordNet Affect lexicons. They evaluate on the Affective Text dataset only. We refer to their model as STRAP.

- Kim et al. [53]- The work of Kim et al. consists of using unsupervised techniques and information from lexicons. They evaluate on Affective Text and Fairy Tales. We refer to their best performing model as KIM.

- Tareaf et al. [75] - The work of Tareaf et al. also uses Facebook Posts and Reactions as their dataset, and makes use of word2vec embeddings, standard textual features and Naive Bayes classifiers. They evaluate on a variety of datasets, including Affective Text and Fairy Tales. We refer to their model as TAREAF.

4.8.3 Standardizing Emotions

The datasets used for evaluation and Facebook Reactions use differing categories of emotion. We therefore map the emotions of Affective Text, Fairy Tales and Facebook to a consistent set of emotions:

Affective Text	Fairy Tales	Facebook	Mapped
Anger	Angry-Disgusted	Angry	Anger
Disgust	Angry-Disgusted	-	Anger
Fear	Fearful	-	-
Joy	Happy	Haha-Love	Joy
Sadness	Sad	Sad	Sadness
Surprise	Surprise	Wow	Surprise

This is consistent with the mapping used by Pool et al. and Tareaf et al.

4.8.4 Results

Note that results of other works are reported as stated in their respective papers – no models were re-implemented for evaluation purposes.

Affective Text Evaluation				
Approach	Joy (p, r, f)	Surprise (p, r, f)	Sadness (p, r, f)	Anger (p, r, f)
Ours	0.60, 0.87,**0.71**	**0.34**,0.16,**0.22**	**0.81**, 0.50, **0.61**	0.46, 0.55, **0.50**
TAREAF	0.56, 0.73, 0.63	0.25, 0.16, 0.20	0.53, 0.32, 0.40	0.33, 0.46, 0.38
POOL-B	0.39, 0.85, 0.54	0.20, 0.05, 0.08	0.51, 0.21, 0.30	0.50, 0.35, 0.41
POOL-FT	0.41, 0.77, 0.54	0.25, 0.17, 0.20	0.53, 0.28, 0.37	**0.51**, 0.30, 0.38
POOL-ISE	0.39, 0.82, 0.53	0.27, 0.08, 0.12	0.49, 0.21, 0.29	0.48, 0.35, 0.40
STRAP	0.19, **0.90**, 0.31	0.08, **0.95**, 0.14	0.12, **0.87**, 0.22	0.06, **0.88**, 0.12
KIM	**0.77**, 0.58, 0.65	-	0.50, 0.45, 0.48	0.29, 0.26, 0.28

We can observe that our model performs the best across a range of metrics. For those where other models outperform our approach, our model is the second best or remains competitive.

Fairy Tales Evaluation				
Approach	Joy (p, r, f)	Surprise (p, r, f)	Sadness (p, r, f)	Anger (p, r, f)
Ours	0.48, 0.87,0.62	**0.37**, 0.15, **0.22**	0.44, 0.56, 0.49	0.65, 0.03, 0.06
TAREAF	0.40, **0.90**, 0.55	0.36, 0.05, 0.08	0.34, 0.23, 0.28	0.62, 0.02, 0.05
POOL-B	0.49, 0.77, 0.60	0.12, 0.04, 0.06	0.43, 0.39, 0.41	0.33, 0.04, 0.07
POOL-FT	0.49, 0.69, 0.58	0.14, **0.33**, 0.19	0.50, 0.24, 0.33	0.27, 0.02, 0.04
POOL-ISE	0.48, 0.81, 0.60	0.17, 0.04, 0.07	0.43, 0.34, 0.38	0.36, 0.05, 0.08
KIM	**0.80**, 0.76, **0.78**	-	**0.71**, **0.82**, **0.77**	**0.77**, **0.56**, **0.65**

We can observe that overall, KIM is the best performing model. However, our approach still remains competitive with the other approaches. We also compare the micro F-scores between our approach and that of Tareaf et al. and Pool et al.

Avg. Micro F-Score		
Approach	Affective Text	Fairy Tales
Ours	**0.57**	**0.47**
TAREAF	0.47	0.39
POOL-B	0.41	0.46
POOL-FT	0.41	0.40
POOL-ISE	0.41	0.46

Our approach obtains the best results compared to the other two approaches. Note that Strapparava et al. and Kim et al. are ommitted as they do not provide this information. Overall, performance of our model is still competitive when applied to a task it is not explicitly trained for.

4.9 Emotion Distribution Benchmark

We would also like to compare performance of our approach to those of previous work, with respect to the original emotion distribution (mixed) perspective. However, to the best of our knowledge there are no standard datasets which could be used for such purposes, and datasets in other works have not been released publicly [55]. This would be a useful area of further work, in order to assess the state-of-the-art.

Chapter 5

Application: Positive Stories News App

5.1 Introduction

We seek to investigate real-world applicability of our Reader model. In this chapter, we use our model in a web application dedicated to providing news of a positive nature.

5.2 Reading the News

Reading the news is not without downside. The psychological impact of consuming negative news is well documented [93], with exposure to negative news contributing to increased levels of anxiety, depression and stress, which in turn also leads to negative physical health outcomes.

Studies have shown that in contrast to negative news, exposure to positive news is associated with decreased levels of stress and anxiety. Stories that induce positive emotions and are framed in a solutions-oriented manner have even been shown to cause people to take a more optimistic and solutions-focused view of the world, increasing their commitment to taking positive action [22].

We create a web application that provides positive news to the user, using our Reader model to perform article selection. The application allows users to view news articles and provide feedback in the form of emotional reactions.

5.3 System Overview

The overall system design is illustrated in Figure 5.1.

5.4 Deployment

We use Heroku for deployment. We chose Heroku as it abstracts away management of hardware or virtual machines via containerisation. We use Heroku's Free tier to deploy our front-end web application, as it is suitable for our requirements [9]. We use Heroku's paid Production tier to deploy our back-end service, as the

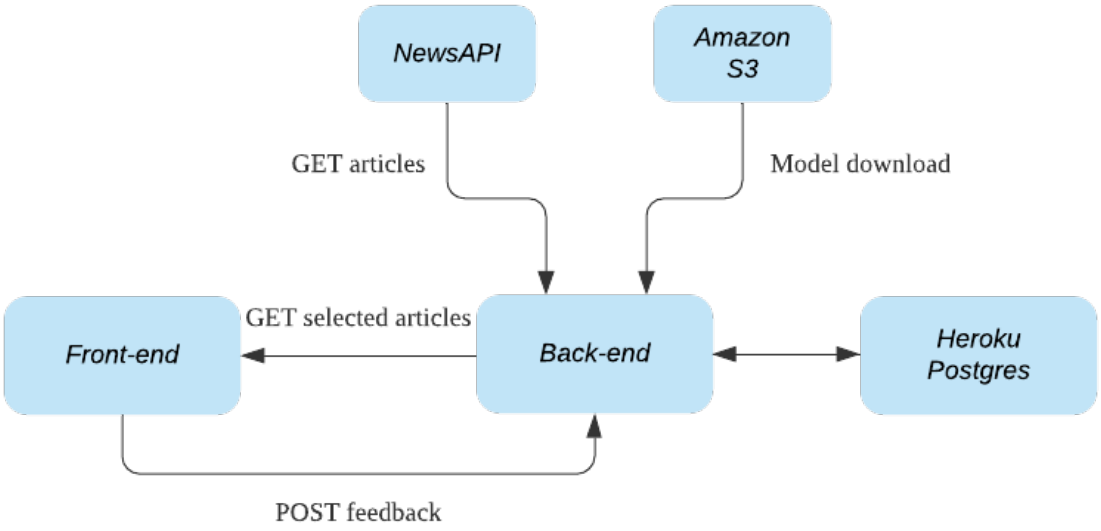


Figure 5.1: System Components

Production tier gives us access to more powerful compute options which are necessary for performing model prediction. Application deployments to Heroku are done via the version control system Git. Application code is committed to a Git repository, and new versions of an application are deployed by pushing the latest version of the corresponding repository to a Heroku-hosted remote that's associated with the application. When applications are deployed to Heroku, the application code is then packaged into containers which Heroku refers to as "dynos" [4]. These containers contain necessities such as compute, memory, an operating system, and an ephemeral filesystem. We use GitHub for repository hosting and management.

5.5 Front-end Application

We create a simple front-end design for our web application. This is because our objective was to mostly evaluate how our model was performing in a real-world setting. We hence chose the tools we were most familiar with, and just made sure the application was intuitive enough to enable smooth user feedback. Our front-end framework of choice was Angular 9, as it was the framework we had the most prior experience with.

The main interface is the news feed. It enables the user to view a daily selection of news articles as chosen by our machine learning model. Users can click on a news item to read the full article at its original webpage and provide feedback in the form of a reaction akin to Facebook Reactions.

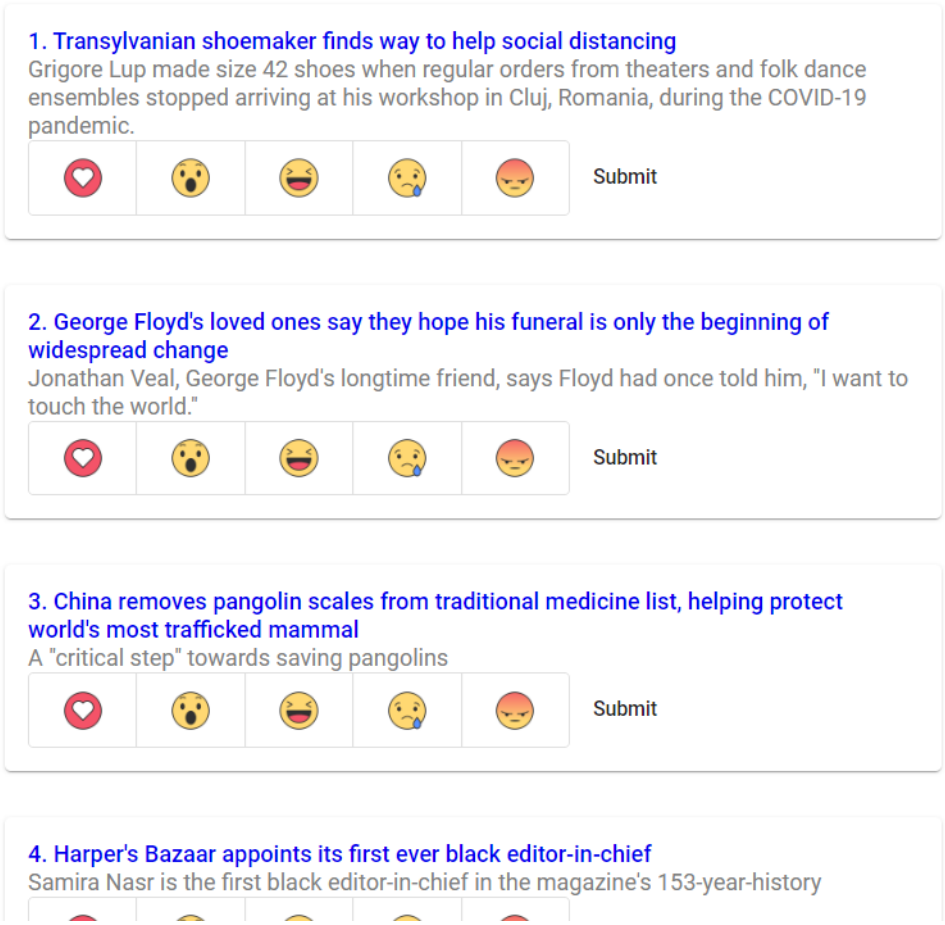


Figure 5.2: Screenshot of NewsFeed

5.6 Back-end Service

Our back-end service handles the interactions between our front-end application, database, and model prediction service. We use Flask as our back-end framework. This was because we wanted our back-end to be written in Python to enable easier interactions with our Python machine learning model. Flask is also lightweight and easy to get started with, which was important as we did not have much previous experience with Python web frameworks.

It is important to note that we perform article selection once a day, first downloading the model, performing predictions on obtained articles, then storing the selected articles in the database. This is for the following reasons:

- NewsAPI limits - NewsAPI's Developer (Free) tier has a limit of 500 requests per day [8]. 500 requests per day can be easily reached with many users or frequent access of the application. Requesting for articles once a day, and storing the articles in the database for future access, means that the number of API requests we need to perform are limited.

- Data sparsity - Similar to our work with the Reader model, where we filter out Facebook posts with below the minimum number of Reactions, more Reactions

per post will make it easier to get an accurate distribution of emotion. We take a classification perspective, using the highest Reaction ratio as our "ground-truth" Reaction label. Constantly changing the articles shown to users, or showing a very large number of articles, could make it difficult to determine an accurate "ground-truth" Reaction. This is why we select a fixed number of articles per day, so that the same set is collectively shown to all users.

- Ephemeral filesystem - GitHub has a strict upper limit of 100 MB for file sizes in a repository [3]. In addition to that, the maximum size of our application after being compressed by Heroku is 500 MB [10]. As our saved machine learning models are very large, including them in our application repository to be deployed means we frequently exceed both these limitations. However, "dynos", the containers to which applications are deployed, have ephemeral filesystems. This means that any changes to the filesystem do not persist beyond dyno shutdowns or restarts. In addition, dynos routinely perform one restart a day to "maintain the health of applications". Due to this, manually uploading a model to a dyno separately from an application is not feasible as the associated files will not persist. To deal with this challenge, we instead upload our model files to Amazon S3, which provides object storage through a web service interface. The latest model is then downloaded at the daily prediction time, as part of a scheduled job. This ensures that the model is available for prediction when necessary.

The downside of doing once-a-day prediction is reduced article freshness. However as our objective for this application is more to demonstrate the usefulness of our model in a real-world setting, rather than a production-grade application, we can accept the trade-off.

This once-a-day scheduled job is implemented using Heroku Scheduler. Heroku Scheduler is an add-on for running jobs on a Heroku application at scheduled time intervals, much like `cron` in traditional server environments. Everyday at a fixed time, Heroku Scheduler executes our scheduled job, which is defined in a Python script in our back-end application. The following steps are executed as part of the job:

1. Perform GET requests to NewsAPI to retrieve articles from a variety of news sources.

2. Download the latest version of the models from Amazon S3.

3. Predict an emotion distribution using each article's title using our ensemble model.

4. We seek to provide content which will evoke positive emotions in the user. We decide to use "Love" as our positive target emotion of choice. Therefore, ten articles with the highest "Love" ratio are selected as our "batch" of articles for the day.

5. Store the batch of articles in our database, to be later retrieved and displayed by the front-end application as necessary.

5.7 Database

We originally chose SQLite as our relational database management system (RDBMS) due to prior familiarity. In contrast to other systems, SQLite does not use a client–server database approach, and instead stores data on a local file in the application [11]. However, we quickly ran into problems of changes not persisting, which we then realised was due to Heroku's aforementioned ephemeral filesystem.

We then decided to use Heroku Postgres, a Database-as-a-Service provided directly by Heroku, for our database needs. Heroku Postgres abstracts away any management we would otherwise have to do when self-hosting, for example installing and maintaining the required hardware and software. In addition to that, it is well-integrated within the Heroku ecosystem and is especially convenient when used in conjunction with Heroku-deployed applications such as ours.

We use SQLAlchemy, an SQL toolkit and Object-Relational Mapper for Python, to interface with our database. This is because of the good support available for its usage in Flask applications.

5.8 Authentication

For authentication, we choose to look for Authentication-as-a-Service options as they allow for much quicker integration into our application. In addition to that, it means we can offload some level of responsibility to an external party. This allows us to focus on the other core aspects of our application.

We opt to use Firebase Authentication, which is part of Google's Firebase Backend-as-a-Service platform. This is because we had prior experience with using Firebase and could therefore integrate it quickly into our application.

5.9 NewsAPI

External APIs are convenient options for sourcing new content. We use NewsAPI, due to its ease-of-use, excellent documentation, and availability of a free tier. To obtain a large enough corpus of news items to select from, we use NewsAPI's available endpoints to obtain the 100 most "breaking" articles from each of ten different news sources, totalling 1,000 articles. Sources include CNN, BBC and Fox News.

5.10 Results

We evaluate our system for a period of 10 days, with 3 users providing feedback for 10 articles each day. This yields feedback for 100 articles, with 3 Reactions for each. Taking the most common Reaction as the "ground-truth" label, only 28% of the articles our model predicted as "Love" were actually labelled with a Love Reaction by our users. Although a deeper evaluation of our collected data is required, we suspect our poor results can be explained by the following reasons.

5.10.1 Negative News

News is mostly negative [63]. Manually inspecting the articles returned by NewsAPI for our various sources confirms this. Poor performance may hence be attributable to the model not receiving enough positive articles, even with a selection pool of 1,000 articles per day. To counteract this, we could do better source selection, and perhaps take greater care in selecting topics besides the more reactionary and negative "breaking" topic provided by NewsAPI.

5.10.2 Concept Drift

Another reason why we achieve poor performance may be the fact that as our model was primarily trained on a Facebook news dataset containing data from 2012 to 2016, it does not perform as well on more current news. This effect, where a changing target variable results in a degradation in model performance, is also known as concept drift [38]. More analysis is required to confirm this.

To counteract this effect, we considered using the feedback from users to assess how to improve and adapt the model over time. The data would allow us to experiment with different concept drift adaptation approaches, such as periodic model retraining [38]. However, we suspected the scale of data we were collecting, as well as the time span in which we collected data, would be insufficient for analysis of how to best design the model adaptation process. We then considered artificially replicating a longer time span of data collection with a larger amount of user feedback, by scraping Facebook posts across a long time span and using the Reactions as feedback. This way, we could get a large number of Reactions, and experiment with various techniques to see how the model adapts over time. However, we ultimately decided against it due to the unclear legalities surrounding web scraping.

Chapter 6

Application: Emotional Arcs of Stories

6.1 Introduction

We seek to investigate the applicability of our Reader model in a more research-focused setting. In this chapter, we use our model to analyse the emotional trajectories of a corpus of fictional works.

6.2 Emotional Arcs

Storytelling is an important part of the human experience. The advent of large digital linguistic resources such as Project Gutenberg and Google Books have made large scale analysis of stories feasible, allowing those in the digital humanities to better understand the stories we tell by identifying patterns and trends.

A vital aspect of storytelling are the emotional ebbs and flows throughout the narrative. In recognition of this, the emotional arcs of stories have gathered much interest in recent years in the digital humanities [78] [32] [49]. The identification of universal emotional arcs is a common area of research, with recent studies claiming that there are core emotional arcs from which all stories derive. The idea of universal *plot* structures in stories is not a new one. Various theories identifying universal narratives have been proposed over the years, from Vonnegut's master's thesis [33] to Booker's seven basic plots [25]. However, distinction must be drawn between the plot, which concerns the structure of the events in a story, and the emotional arcs, which instead concerns the emotional structure of a story. The work on universal emotional arcs goes beyond just identification, with the understanding of the relationship between emotional arcs and genre, and emotional arcs and popularity, being common subjects of research [32].

6.2.1 Datasets

Our corpus consists of texts from both books and movie scripts, totalling 2,861 documents.

6.2.2 Project Gutenberg

Project Gutenberg is an online digital library consisting of over 60,000 works, with most being in the public domain [88]. Project Gutenberg is a common linguistic resource used in the digital humanities for statistical analysis of the evolution of language and culture. We collect 1,737 works in total from Project Gutenberg. For selecting which works to use, we choose to use the same set of books as that in the paper "The emotional arcs of stories are dominated by six basic shapes" by Reagan et al [78]. We use the third-party `gutenberg` Python library [95] to perform the data collection. Project Gutenberg permits such forms of large-scale data collection from its website.

6.2.3 Internet Movie Script Database (IMSDb)

The Internet Movie Script Database (IMSDb) is a large online resource of movie scripts. We use a Python library, `imsdb_download_all_scripts` [56], to download all available movie scripts, totalling 1,124.

6.3 Constructing an Emotional Arc

We now go through the various steps of our approach to constructing an emotional arc for a text – 1) Pre-processing, 2) Predicting emotional values, 3) Smoothing.

6.3.1 Pre-processing

We first use the `gutenberg` Python library to strip unnecessary header and footer text from each text. This includes unneeded information such as the release date, language, character encoding and Project Gutenberg license. Similarly, we use `imsdb_download_all_scripts` to strip unnecessary header and footer text from each movie script.

We then use `nltk`, a popular natural language processing library for Python, to split the text into sentences and perform additional pre-processing (remove non-ASCII characters, lowercasing, separating punctuation and words).

6.3.2 Predicting emotional values

We then use our Reader model (the Reader-Categorical-Mixed prediction ensemble) to predict the mixed distribution of emotions for each sentence. We can then plot the values for each emotion on a graph. For example, plotting the Love ratio for each sentence in the book "Madame Bovary" in Figure 6.1:

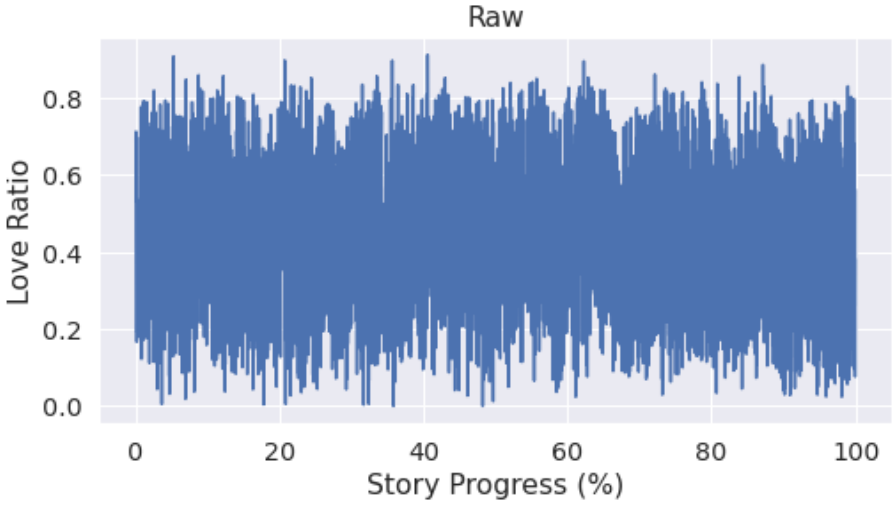


Figure 6.1: Madame Bovary by Gustave Flaubert (1856)

6.3.3 Smoothing

We can see that the raw plots are too noisy and difficult to interpret. Similar to Jockers [49], we compare various approaches to smoothing, so that the "emotional arcs" become more apparent:

- Binning - This involves dividing a text into equally sized bins, and constructing the arc from the averages of each of these bins. The main parameter to adjust is the number of bins. In Figure 6.2, bin smoothing is illustrated with a standard mean function and 10 bins.

- LOWESS - LOWESS (locally weighted scatterplot smoothing) [43] is usually used to fit a smooth line through a scatterplot to better represent the relationship between variables. In Figure 6.3, LOWESS smoothing is illustrated.

- Rolling Average - This involves computing a series of averages by moving a window of fixed size across the data. At each step, the average of the window is computed, before the window advances forward and the next average value is computed. The main parameter to adjust is the size of the moving window. In Figure 6.4, rolling average smoothing is illustrated, using the standard mean function and a window size of 1,000 sentences.

- Discrete Cosine Transform (DCT) - A Discrete Cosine Transform [18] expresses a series of data points as the sum of cosine functions of differing frequencies, and is commonly used for signal smoothing. It allows a complex, noisy signal to be decomposed into the aforementioned cosine signals of varying frequency. A low-pass filter can then be applied to filter out the unnecessary high frequency signals, leaving the lower-frequency signals. These can then be re-combined to reproduce a version of the original signal which is less noisy. This recombination is done via the "reverse" transformation. In Figure 6.5, DCT is illustrated using a low pass filter size of 10.

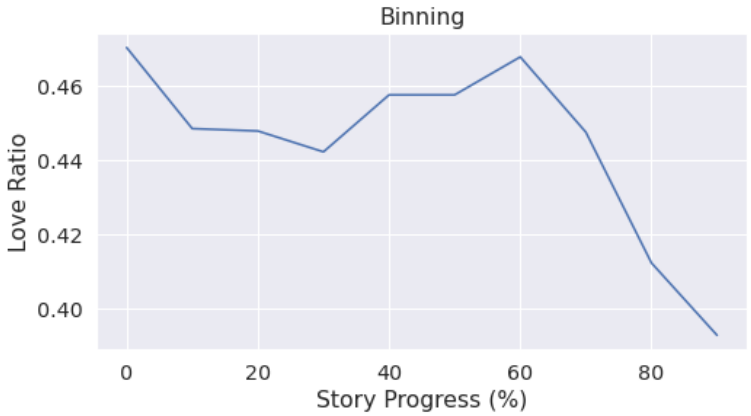


Figure 6.2: Madame Bovary (Love Ratios) – Binning (10 bins)

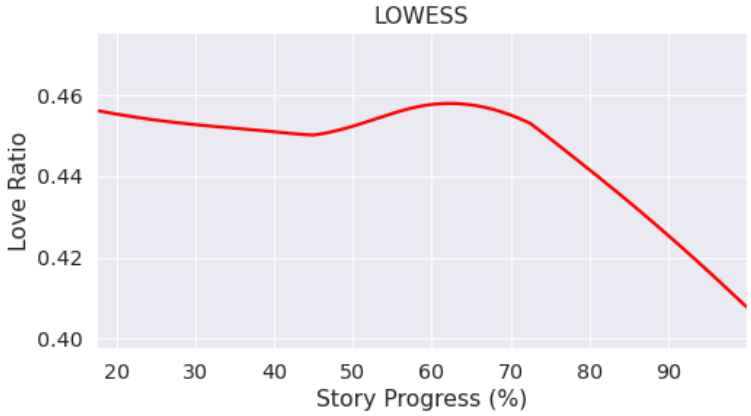


Figure 6.3: Madame Bovary (Love Ratios) – LOWESS Smoothing

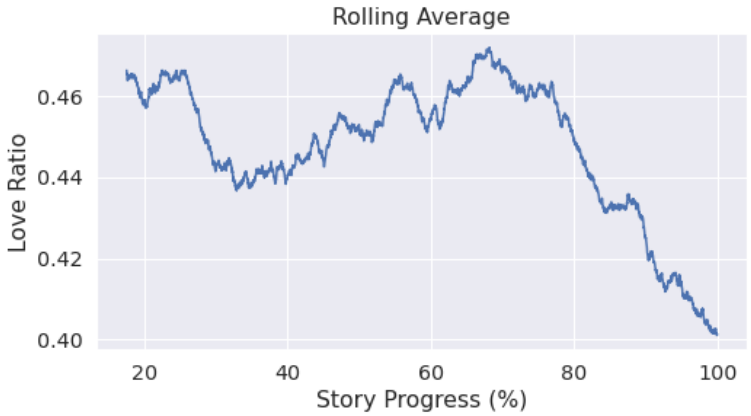


Figure 6.4: Madame Bovary (Love Ratios) – Rolling Average (windows size 1,000)

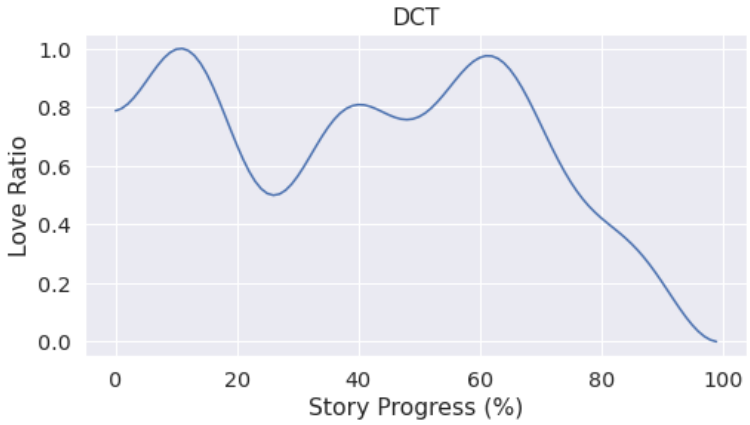


Figure 6.5: Madame Bovary (Love Ratios) – Discrete Cosine Transform (low pass filter size of 10)

The above smoothing operations are done via a combination of `pandas`, a Python library for data manipulation, and `scipy`, a Python library for scientific computing. We can see that smoothing is effective in extracting the overall emotional trajectory of a text. We can observe similar shapes using all four smoothing methods. As we seek a balance between detailed and more "big-picture" arcs, we decide to opt for DCT as our smoothing method. This is similar to the approach taken by Jockers and Vecchio et al [32].

6.4 Clustering

Clustering of emotional arcs has previously only been with respect to Valence, usually from the Writer's perspective. We differ from previous work by performing clustering from a Reader-Categorical-Mixed perspective, clustering on each of our five emotional Reactions (Love, Haha, Wow, Angry and Sad) individually.

We evaluate various clustering algorithms, ultimately deciding on using hierarchical agglomerative clustering (HAC) [62], as it is the most well-established clustering approach within this research area and also does not enforce globular clusters. HAC gives us the ability to evaluate clusters across different "cuts" as explained in Chapter 2.

In terms of distance metric, Dynamic Time Warping (DTW) has been demonstrated to be superior to Euclidean distance for clustering of time series (our emotional arcs can be considered to be time series) [24]. However as DTW is significantly more computationally expensive, most research has preferred usage of Euclidean distance [81]. This matches our own experience, and so we opt to use Euclidean distance due to the significant performance advantage . A more efficient version of DTW which performs approximate calculations exists, called FastDTW [81]. We experiment with using the `fastdtw` library for Python, but still found it too computationally slow. In terms of linkage criterion, we choose Ward's method [69], which has precedence with respect to hierarchical clustering for time series.

We use `sklearn` to perform our clustering operations. We take an exploratory approach, inspecting visually and experimenting with a wide variety of low pass filter sizes. We also vary the location of the "cuts", experimenting with different

levels of cluster granularity by varying the desired number of clusters to extract (as opposed to setting a maximum threshold value for the linkage criterion).

For the purpose of cross-emotion comparisons, we fix the low pass size to be 4 and the number of extracted clusters of 6. We find these to be good parameters as they provide sufficient detail on the emotional trajectories while having good clustering behaviour. We report these results in Appendix D.

We find that we can identify clusters for our Love Reaction that follow the aforementioned six basic Valence shapes of Reagan et al. [78]. These clusters are illustrated in Figures 6.6 and 6.7. The blue arcs are the clustered emotional arcs, the red arc is the mean arc, and the shaded red area indicates the standard deviation. Our investigations in Chapter 7 do demonstrate that Valence intensity is strongly correlated with Love ratio, so the identification of similar shapes does make sense.

6.5 Validating Individual Arcs

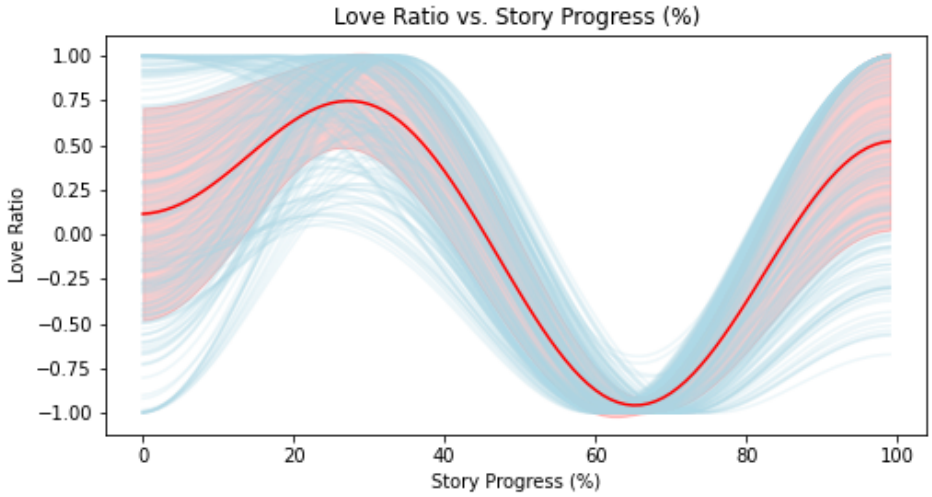
Validating that the constructed emotional arcs are an accurate representation of the shifting emotions of a text is important. Questions surrounding validity of such emotional arcs, the methods of their construction, and the inherent subjectivity of the task has generated much discussion [21] [90].

To the best of our knowledge, no ground-truth datasets for emotional arcs exist. Constructing a ground-truth dataset for evaluating performance is therefore an obvious next step. However, numerous problems arise:

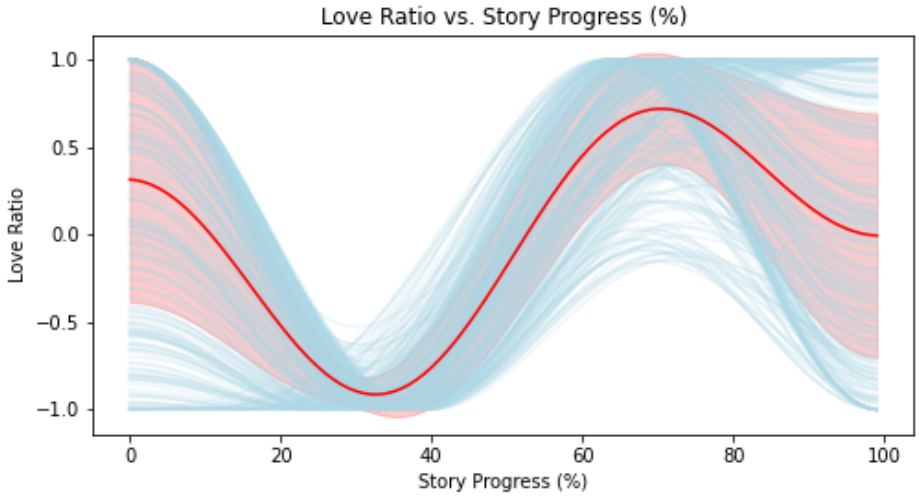
- Resource-intensive - A ground-truth dataset would either require manual human annotation of sentiment on a fine-grained level (e.g. per sentence), from which emotional arcs can be constructed. Alternatively, emotional arcs could be constructed directly. Either method would be extremely time-consuming, especially when dealing with long texts such as novels.

- Inter-annotator agreement - The previous problem is compounded by the fact that annotating text with evoked emotions is an inherently subjective task. Different people will have different, but equally valid emotional reactions. Therefore, having multiple human annotators would be necessary for determining the inter-annotator agreement, which is important for evaluation.

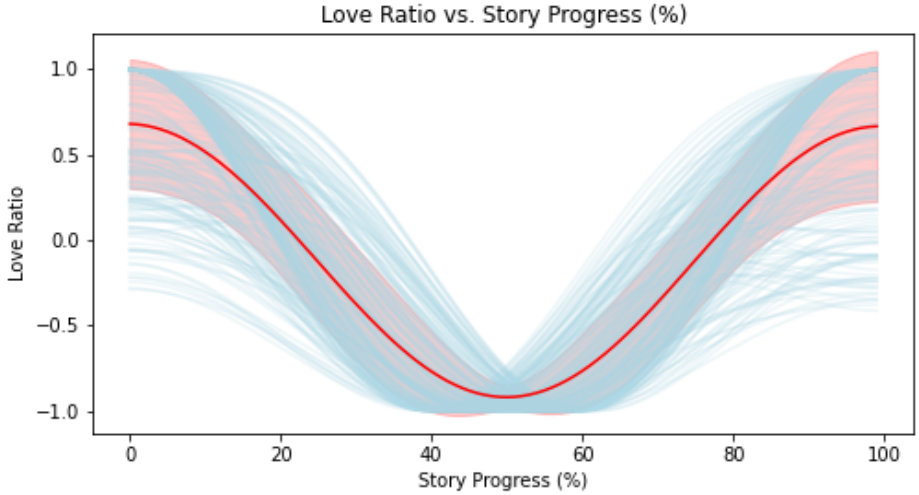
Due to time and financial constraints, we instead decide to leave this as an area for future work.



(a) "Cinderella" – Rise-Fall-Rise

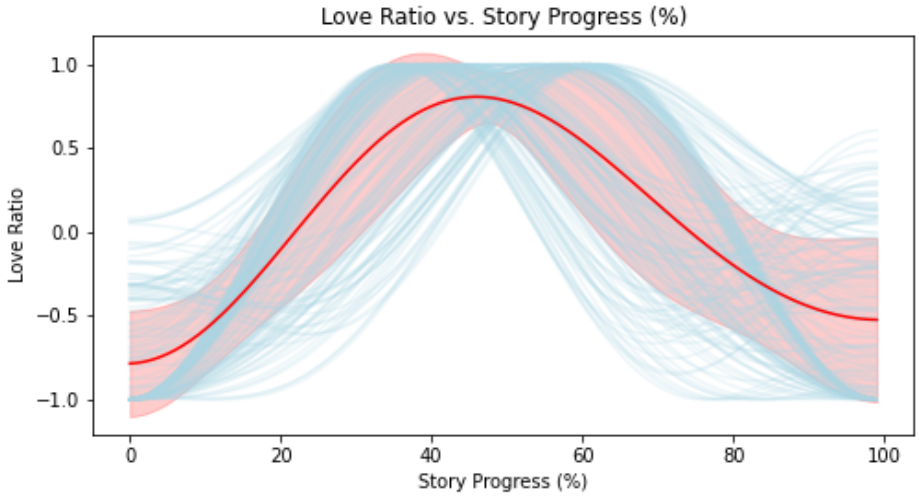


(b) “Oedipus” – Fall-Rise-Fall

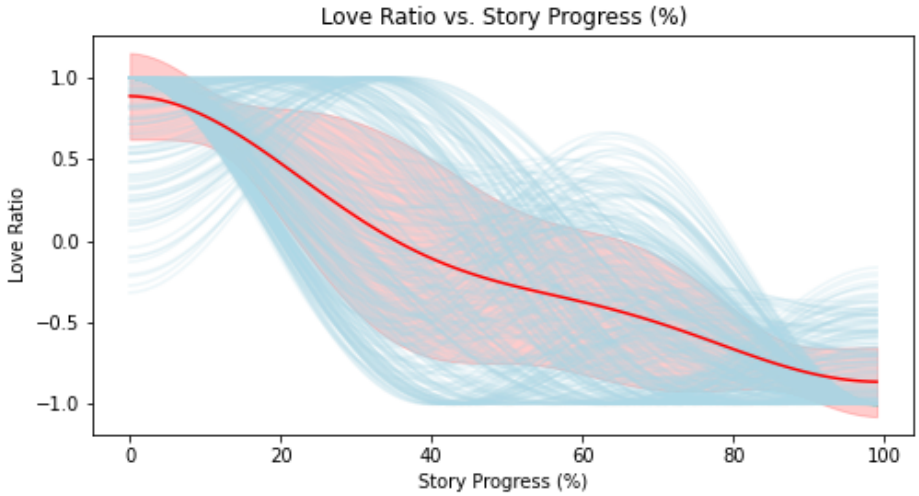


(c) “Man in a hole" – Fall-Rise

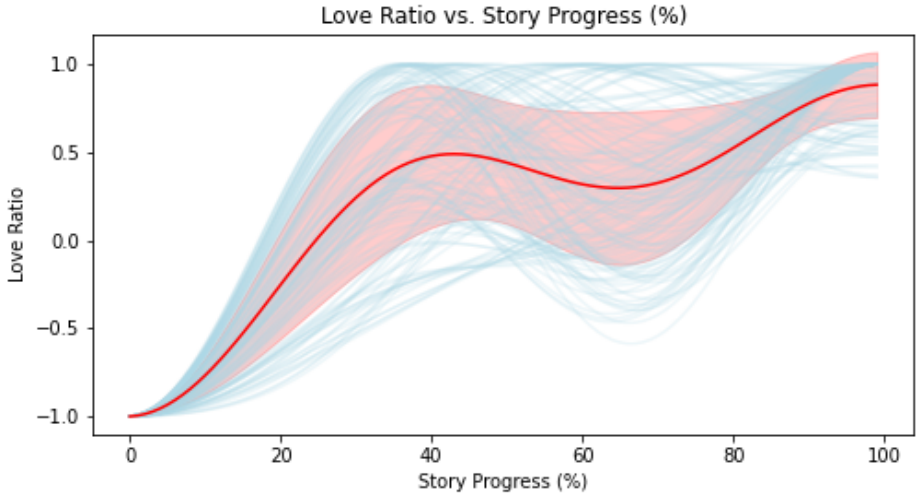
Figure 6.6: Love Ratio Clusters – 1-3



(a) “Icarus” – Rise-Fall



(b) “Riches to rags” – Fall



(c) “Rags to riches” – Rise

Figure 6.7: Love Ratio Clusters – 4-6

Chapter 7

Investigating Relationships

7.1 Introduction

The datasets and models we have built give us an opportunity to investigate the relationship between the different modes of sentiment analysis. We use scatterplots with lines drawn using LOWESS (Locally Weighted Scatterplot Smoothing) to illustrate the relationship between variables [43]. For visual clarity, the data has been divided into discrete bins, with the vertical blue lines indicating the central tendency and confidence interval. This binning only influences how the scatterplot is drawn. the regression is still fit to the original data. In addition to that, we use heatmaps to indicate the correlations between variables, calculated with Spearman's rank correlation coefficient [39].

7.2 EmoBank Dataset

7.2.1 Dimensional-Reader-Regression vs Dimensional-Reader-Regression

We use the EmoBank dataset to analyse the relationship between Dimensional-Reader-Regression and itself in terms of the intensity of each dimension of emotion. See Figures 7.1 and 7.2.

The most obvious relationship is the one between V Reader and D Reader, which occurs both ways and is monotonic. The Spearman's coefficient between the two variables is 0.40. The V Reader (x-axis) vs. D Reader (y-axis) and V Reader (x-axis) vs. D Reader (y-axis) graphs both indicate that a higher level of experienced pleasure is associated with a higher level of experienced control, and more displeasure is associated with more submissive feelings. We cannot determine if this relationship is causal, if it simply a result of the dataset construction, or in general, situations with high levels of each are more likely to occur. However, psychological research does validate our findings, as feeling more powerful is associated with more positive emotions, and powerlessness is associated with feeling more displeasure [50].

Another noteworthy finding is the relationship between V Reader and A Reader. Observing the scatterplot and LOWESS line of V Reader (x-axis) and A Reader (y-axis) shows that a greater deviation from a neutral Valence score of 3.0 is associated with higher feelings of Arousal. A neutral Valence score is hence associated with a

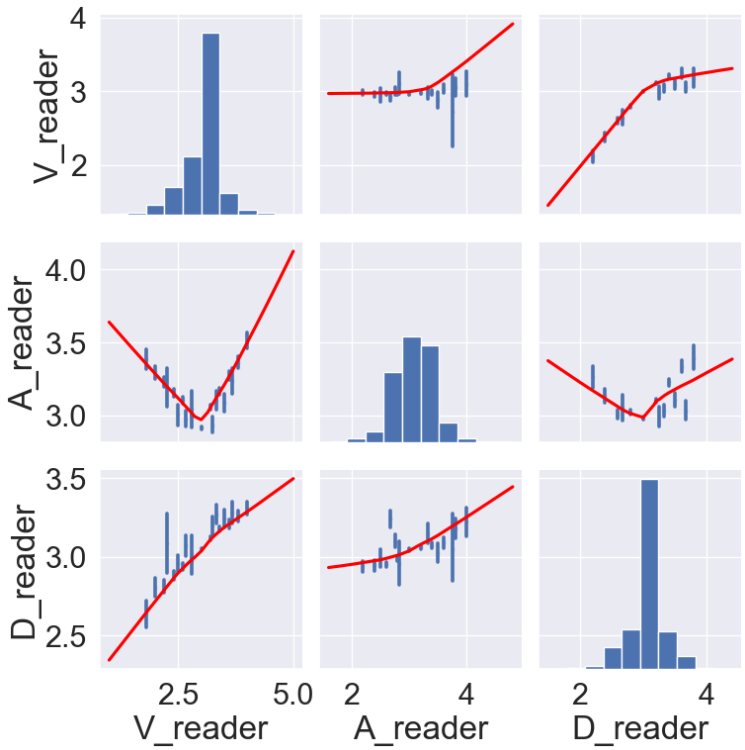


Figure 7.1: D-R-R vs. D-R-R Plots

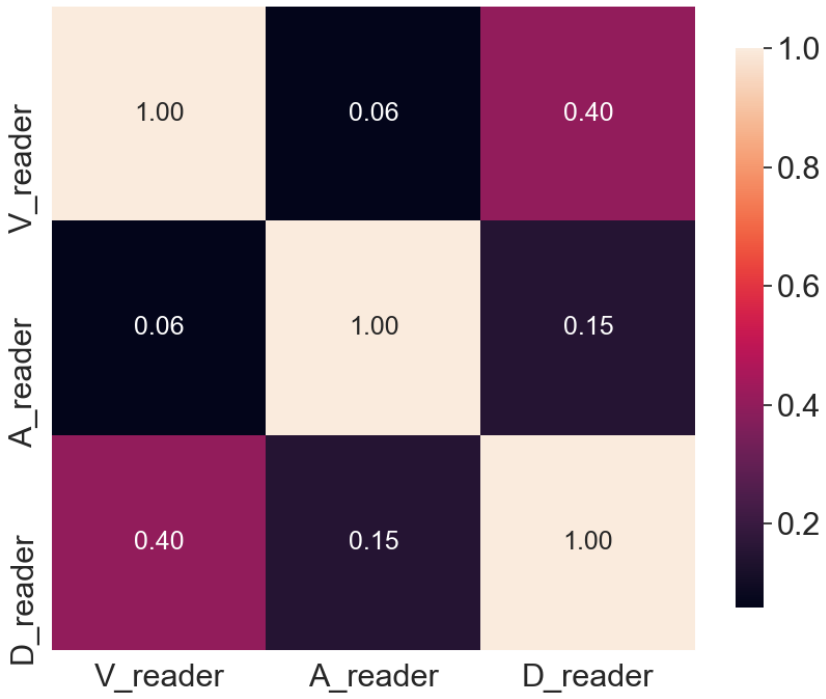


Figure 7.2: D-R-R vs. D-R-R Heatmap

neutral Arousal score. This suggests that the more intense the feelings of pleasure and displeasure, the more likely a reader is to feel energized. Observing A Reader (x-axis) vs V Reader (y-axis), calm Arousal values (below 3.0) are associated with neutral Valence, being equally associated with positive and negative Valence. On

the other hand, excited Arousal values (above 3.0) are more associated with higher Valence values. This is the same as saying that a higher energy level than calm indicates more strongly a pleasurable experience. The Spearman's coefficient is low – 0.06, which can be attributed to the presence of a non-monotonic relationship.

In addition to that, we observe the relationship between A Reader and D Reader. Observing the plots of A Reader (x-axis) and D Reader (y-axis) shows that an increase in Arousal is lightly associated with an increase in Dominance. Our guess is that being more excited may make you feel more dominant. The opposite relationship observed between D Reader (x-axis) and A Reader (y-axis) is similar to that of V Reader (x-axis) and A Reader (y-axis), whereby a deviation from neutral Dominance is associated with higher Arousal. This could be because feeling more submissive or dominant will make you feel more energetic. The Spearman's coefficient is low – 0.15, which can also be attributed to the presence of a non-monotonic relationship.

7.2.2 Dimensional-Writer-Regression vs Dimensional-Writer-Regression

We use the EmoBank dataset to analyse the relationship between Dimensional-Writer-Regression and itself in terms of the intensity of each dimension of emotion. See Figures 7.3 and 7.4.

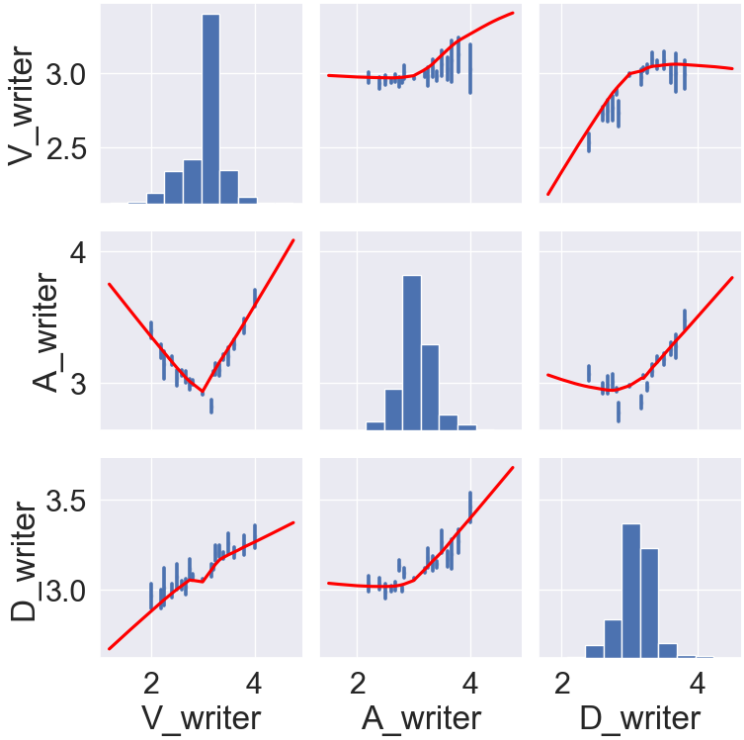


Figure 7.3: D-W-R vs. D-W-R Plots

We can observe similar but somewhat weaker relationships as when comparing Dimensional-Reader-Regression to itself. This indicates that the perspective does not significantly alter the emotional dimensions' relationships to each other. This makes intuitive sense as both Readers and Writers are people and will experience

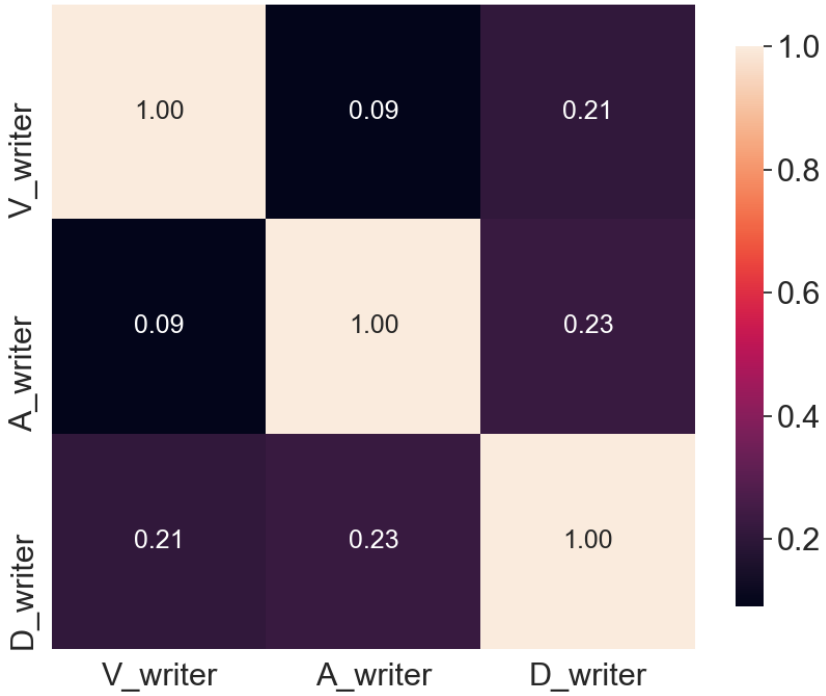


Figure 7.4: D-W-R vs. D-W-R Heatmap

emotions in similar ways more often than not. The only noticeably different trend is D Writer (x-axis) vs A Writer (y-axis), where submissive Dominance values are equally associated with calm and excited Arousal values.

7.2.3 Dimensional-Reader-Regression vs Dimensional-Writer-Regression

We again use the EmoBank dataset, but to analyse the relationship between the Dimensional-Reader-Regression mode and the Dimensional-Writer-Regression mode. This is in terms of the intensity of each dimension of emotion. See Figures 7.5 and 7.6.

Observing the plots on the diagonal, the V Writer and V Reader plot exhibits the strongest trend observed so far, with a Spearman's coefficient of 0.63. A higher Valence from either perspective is associated with a higher Valence from the opposite perspective. There is a similar but less clear trend with respect to Dominance from either perspective (Spearman's coefficient of 0.18). Arousal is flat below neutral but the relationship is positive after that (Spearman's coefficient of 0.24).

The off-diagonal plots indicate similar trends to the previous investigations. This makes intuitive sense as both perspectives exhibited similar trends.

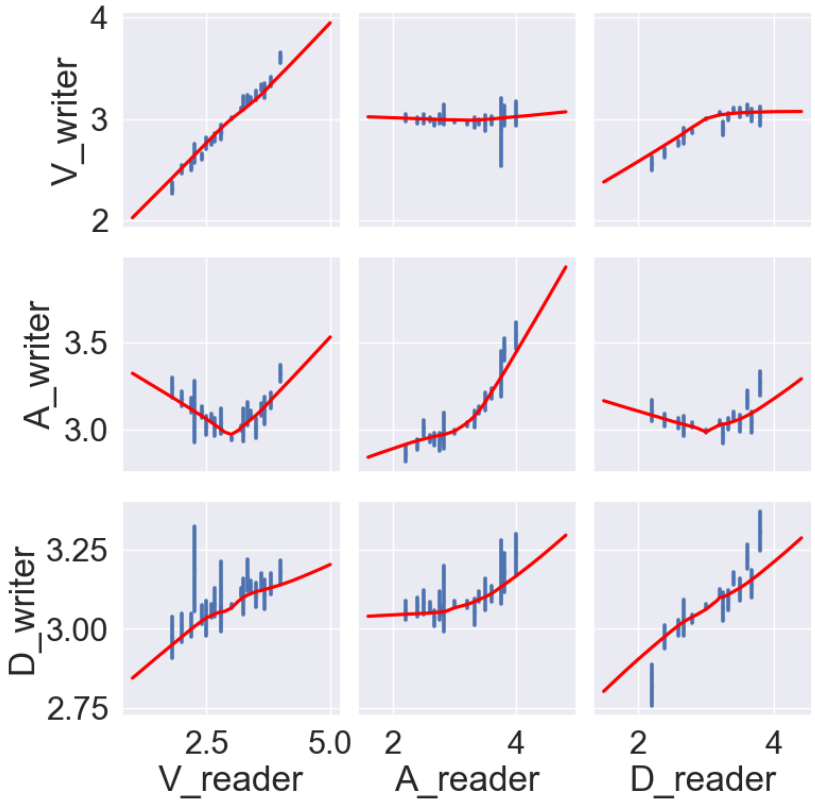


Figure 7.5: D-R-R vs. D-W-R Plots

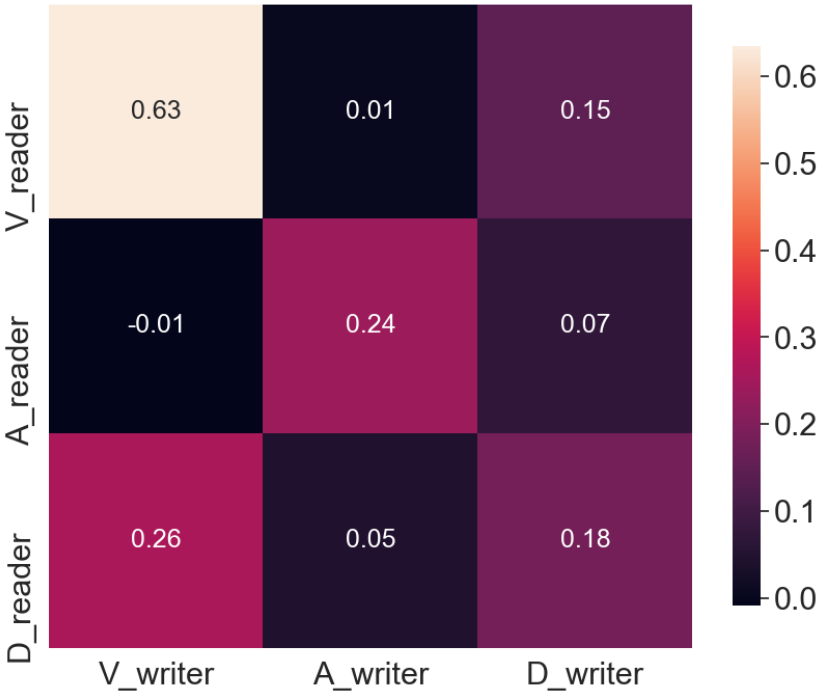


Figure 7.6: D-R-R vs. D-W-R Heatmap

7.3 Martinchek Facebook Dataset

7.3.1 Categorical-Reader-Mixed vs Categorical-Reader-Mixed

We use the Facebook dataset to analyse the relationship between Categorical-Reader-Mixed and itself in terms of the ratios of each category of emotion. We consider Facebook posts with at least 1 Reaction. See Figure 7.7 for the heatmap.

Two of the strongest relationships are between Love and Angry and Love and Sad. They both have strong negative relationships, with a Spearman's coefficient of -0.58 and -0.51 respectively. This is not surprising, as Love is a positive emotion and Angry and Sad are negative emotions. Haha and Sad also have a negative but not as strong relationship (Spearman's coefficient of -0.41).

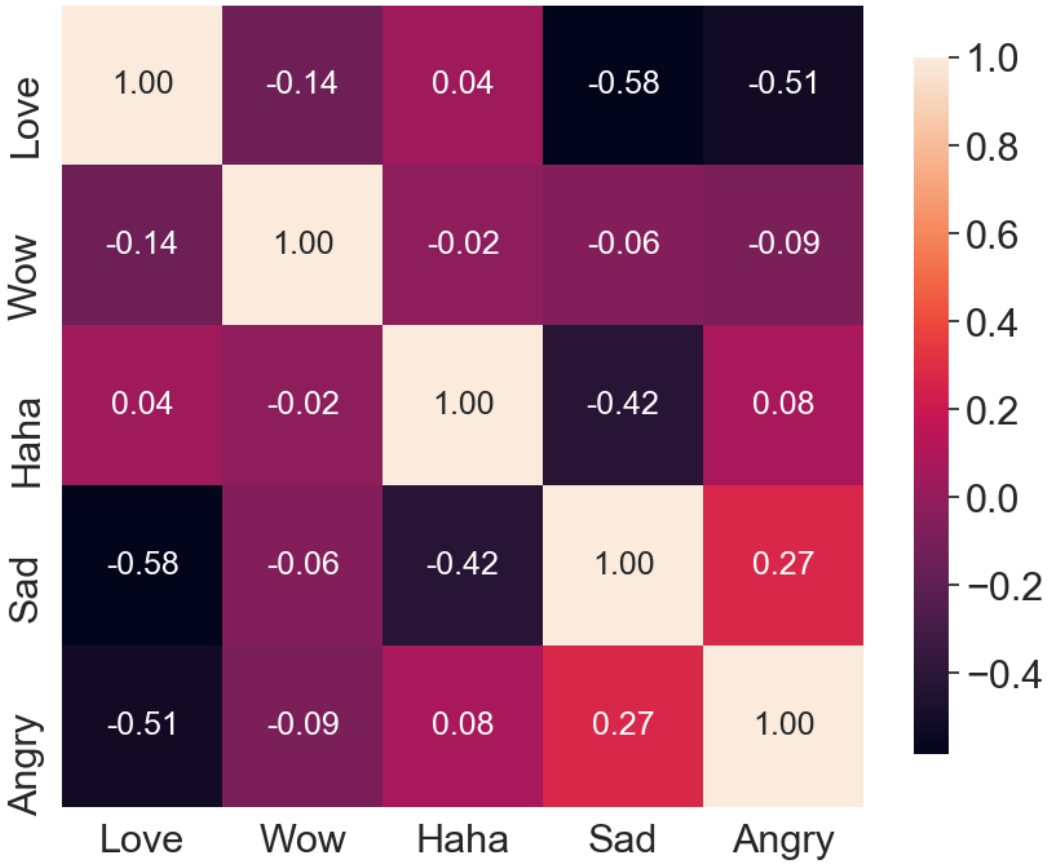
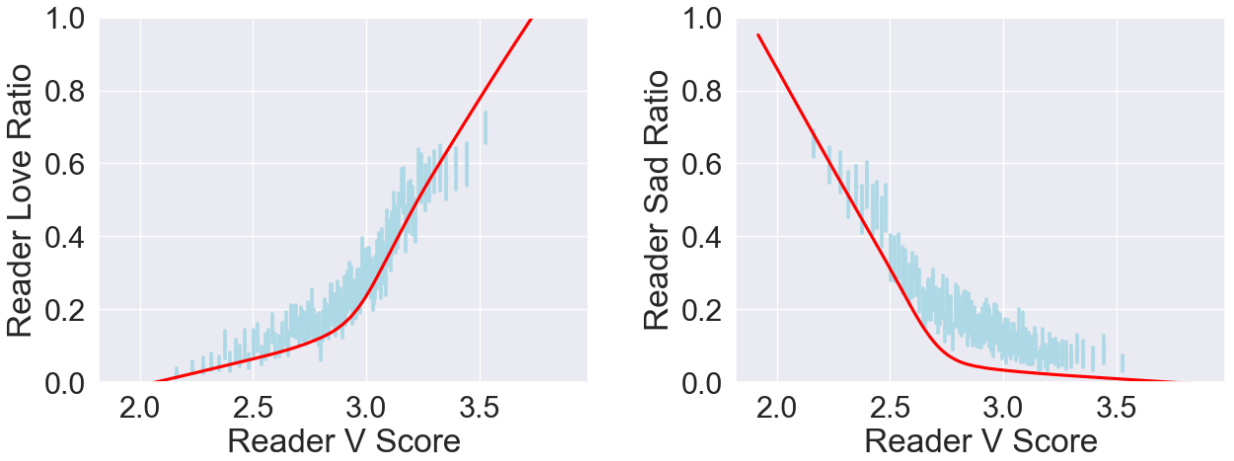


Figure 7.7: C-R-M vs. C-R-M Heatmap

In terms of positive relationships, the strongest is between Sad and Angry, likely because they are both associated with displeasure (Spearman's coefficient of 0.28).

7.3.2 Categorical-Reader-Mixed vs Dimensional-Reader-Regression

We now investigate the relationship between the Categorical-Reader-Mixed and Dimensional-Reader-Regression modes. We do this by using our base Dimensional-Reader-Regression model to perform predictions on a holdout Facebook dataset. The heatmap is illustrated in Figure 7.9. For brevity, we only show the plots for Love and Sad in Figure 7.8.

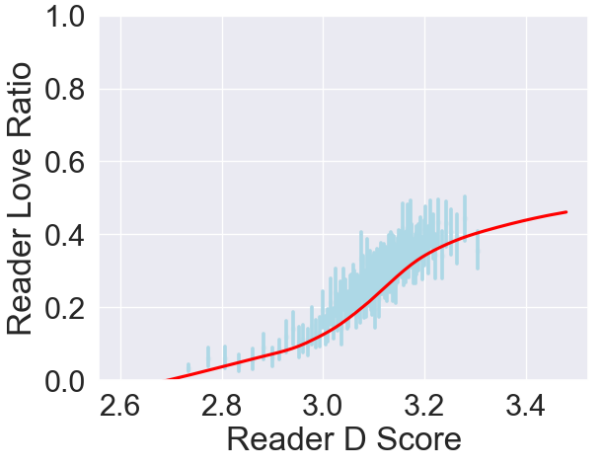


(a) Love vs. Valence

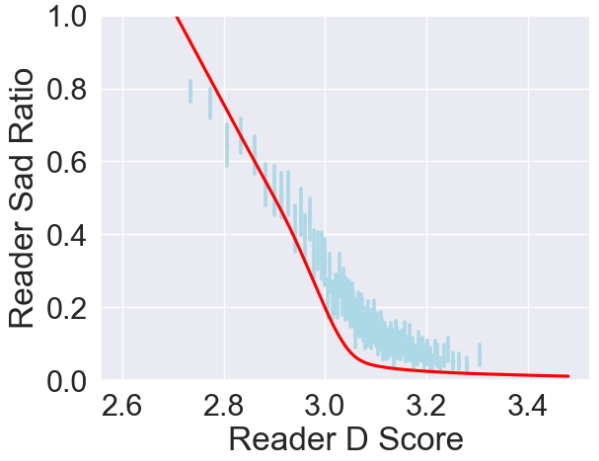
(b) Sad vs. Valence

(c) Love vs. Arousal

(d) Sad vs. Arousal



(e) Love vs. Dominance



(f) Sad vs. Dominance

Figure 7.8: Love and Sad vs. VAD

7.4 Using External Tools for Predictions

We were also interested in using external tools to perform predictions in a similar way and to observe resulting trends. Although this ultimately did not form a large

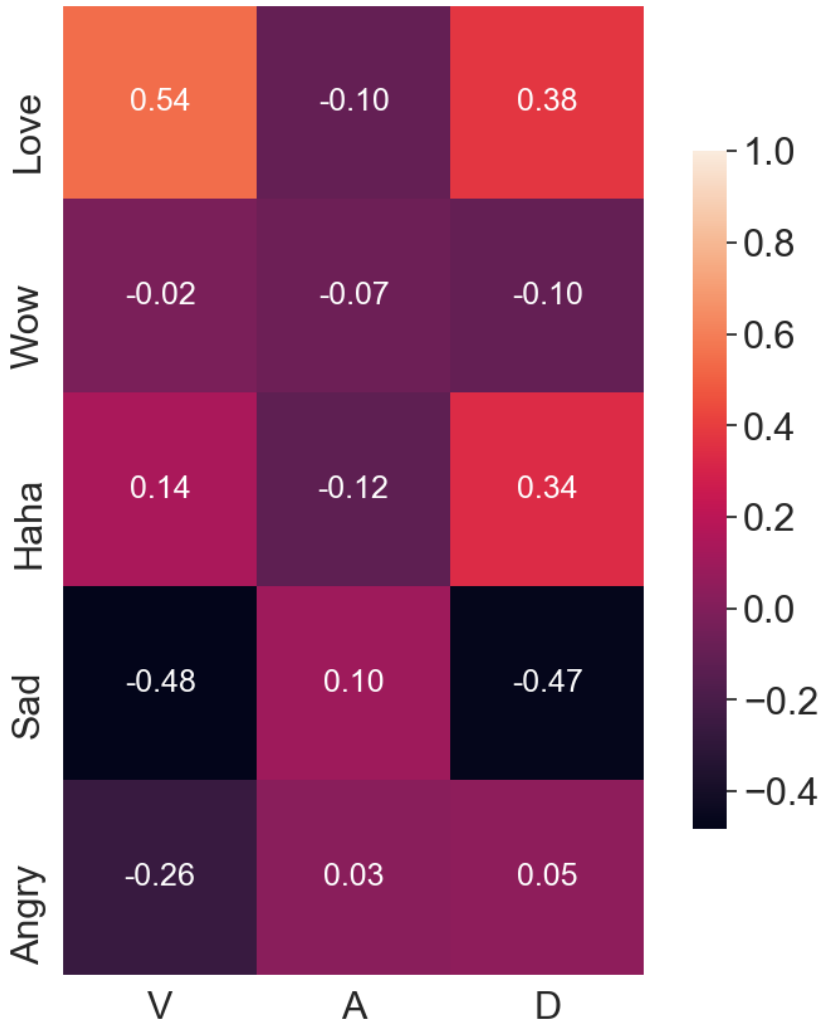
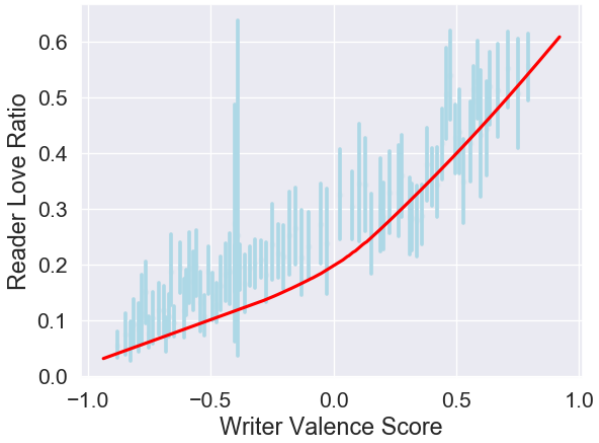


Figure 7.9: C-R-M vs. D-R-R Heatmap

part of our work, we did experiment with using VADER to predict Valence. VADER is a lexicon and rule-based sentiment analysis tool which is part of the `nltk` library, and was chosen because of its suitability for shorter texts [47]. The point of view for VADER is from the Writer's perspective. We apply it to a holdout Facebook dataset, as illustrated in Figure 7.10.

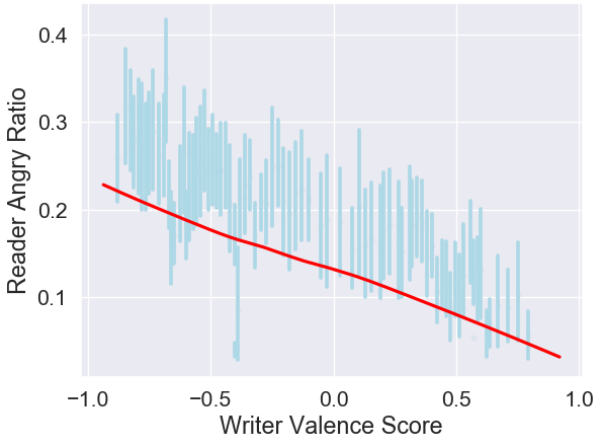
7.5 Further Work

We have analysed the relationships between emotions for both the Martinchek Facebook dataset and the EmoBank dataset. Using our Categorical-Reader-Mixed model and our Dimensional-Reader-Regression model, we perform predictions on a dataset consisting of holdout data from the Martinchek dataset, and perform similar analysis. However, there is still further work which can be done in this space. For example, we could conduct analysis between Categorical-Reader-Mixed and Dimensional-Writer-Regression modes of sentiment analysis. The latter can be constructed using a similar approach to our Dimensional-Reader-Regression model,

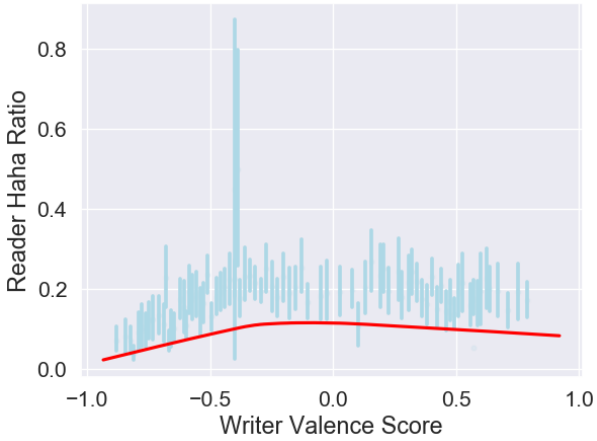


(a) Love vs. Valence

(b) Sad vs. Valence



(c) Angry vs. Valence



(d) Haha vs. Valence

(e) Wow vs. Valence

Figure 7.10: Reactions vs VADER Valence

using the EmoBank Reader dataset. Another example could be using IBM Watson's Tone Analyser to perform Categorical predictions from the Writer's perspective and analyse the relationship with Categorical predictions from the Reader's perspective. We could also vary how emotions are measured, for example in EmoBank, Valence is measured in terms of intensity. We could instead measure the probability of the Va-

lence being positive of negative. Many such variations exist, but our work showcases the potential of analysing the relationships between perspective, representation and measurement. Hopefully this forms a foundation for future work.

Chapter 8

Conclusion

We now conclude our project, summarising and evaluating the work we have done, listing the lessons learned and pointing out future areas of work. We do this with respect to both our main objective and our secondary objective.

8.1 Main Objective

Our main objective for the project was the development of a machine learning model which performs sentiment analysis from the point of view of the Reader, in a fashion which indicates the varying reactions of Readers.

8.1.1 Work Done

In Chapter 3, we identify Facebook as a source of data, using the Martinchek dataset to train our model. We evaluate several approaches to developing a model, including the Bag-of-Words approach, the Sequence Vectors approach, and the BERT approach. We identify RoBERTa as the best performing model, further improving performance by filtering out data with a low Reaction count.

In Chapter 4, we first clarify the various types of sentiment analysis. We identify EmoBank as a source of data, and used it train a model which predicts different but related affective features as compared to our original Reader model. We take a similar approach to model development as in Chapter 3, and identify RoBERTa as again being the best performing model approach. We then combine our model from Chapter 3 and our new model into an ensemble, and demonstrate improved performance. We then evaluate performance of our model on a classification task, and demonstrate competitive performance.

In Chapter 7, we use the datasets we obtained and our constructed models to perform an analysis of various emotional relationships.

8.1.2 Lessons Learned

We learn that BERT and its variants perform well in this task, significantly outperforming the other approaches we tried. To the best of our knowledge, it is the first time BERT-like approaches have been tried for this task. We also learn that filtering out Facebook posts with low Reaction counts from the dataset helps improve performance. In addition to that, we clarify various types of sentiment analysis,

proposing a three-dimensional framework (Perspective, Emotional Representation and Measurement Type) of categorising sentiment analysis approaches. We also learn that integrating affective features from one mode can help improve performance for another mode. To the best of our knowledge, this kind of approach has not previously been tried. Comparing with previous work, we learn that our model is still competitive when applied to a classification task, even when not explicitly trained for it.

8.1.3 Evaluation and Future Work

We believe that we largely succeed in achieving our main objective. We have evaluated the work done in each of Chapters 3, 4 and 7, reporting any results we attain. However, there still exists various points of improvement, which we now highlight.

One area of potential improvement is the fact that we only integrate features from one other mode of sentiment analysis in order to improve model performance via an ensemble. It would have been good to conduct a more comprehensive evaluation, using a larger variety and more combinations of modes of sentiment analysis to see how overall performance is affected. For example, we combined Reader-Categorical-Mixed and Reader-Dimensional-Regression models to perform Reader-Categorical-Mixed predictions. We could swap the Reader-Dimensional-Regression model with a Writer-Dimensional-Regression model to see how performance changes, or use the ensemble for other sentiment analysis mode predictions. We could also see how effective this approach is compared to standard ensembling techniques, such as stacking, boosting and bagging.

In our reported results, we used overall mean squared error across all Reactions to evaluate performance. It would have been good to evaluate performance on a per-Reaction level as well. Going back to our ensemble, we could analyse how the integration of different modes of sentiment analysis affects performance on a more precise level. For example, does integrating Valence features only help to improve performance for Love and Sad, but not Wow and Haha?

We performed some investigations on the relationships between Perspectives and Emotional Representations. Our initial investigations show promise, but more comprehensive work should be done, as discussed in Section 7.5.

The above points of improvement are all things which could be accomplished in future work. Another area of future work would be the establishment of a benchmark for Reader-Categorical-Mixed models. This could be accomplished by providing standard resources for evaluating performance. Such a benchmark would allow us to compare our results more directly to those of previous works, and would help better establish a picture of the state-of-the-art. Of course, such benchmarks could be created for each of the modes of sentiment analysis we proposed. This is also briefly discussed in Section 4.9.

Other interesting avenues of exploration include Reader sentiment analysis with respect to longer documents and images and video, and the relationship between entities in text and Reader response.

8.2 Secondary Objective

Our secondary objective was to explore the applicability of Reader-centric sentiment analysis by using it for specific tasks and applications.

8.2.1 Work Done

In Chapter 5, we developed a web application for providing positive news content to the reader, using our developed Reader model.

In Chapter 6, we use our Reader model to extract emotional arcs from fictional works. We first use our model to perform predictions on a per-sentence basis, before applying smoothing techniques to construct the arcs. We then use hierarchical agglomerative clustering to extract various clusters on a per-emotion basis.

8.2.2 Lessons Learned

We learn that our developed news application requires further work to be effective. We also learn that our model can be successfully used in the extraction of emotional arcs from fictional works. To the best of our knowledge, this is the first time a non-lexicon based approach has been used. We also learn that we can identify clusters of arcs with respect to categorical emotions. This is in contrast with previous work, which primarily only focused on extracting clusters of Valence arcs.

8.2.3 Evaluation and Future Work

We have evaluated the work done in each of Chapters 5 and 6, reporting any results we attain. However, there still exists various points of improvement, which we now highlight.

With respect to our news application, we observe poor performance. The reasons we suspect for this are 1) concept drift, and 2) lack of positive articles to choose from. We discuss these issues and how to counteract them in Sections 5.10.2 and 5.10.1.

In addition to that, our news application is utilitarian and designed more as a data collection tool than as a consumer product. Improving the application in that respect, for example by conducting more extensive testing and improving the design, would be a good next step. Additionally, our news application only makes use of predictions for the Love Reaction. Expanding the application so users can filter content based on the other Reactions as well would be an interesting avenue.

With respect to the work on emotional arcs, more extensive validation and the construction of ground truth datasets would be good areas of future work. This is further discussed in Section 6.5. Future work could also be done examining the relationship between certain emotional arcs and other factors like popularity and genre. Our model could also be applied to a wider range of media, such as speeches.

Overall, we also feel that although we found appropriate applications for the "Reader" aspect of our sentiment analysis model, we did not take full advantage of the "mixed" nature of our predictions. This would be another avenue of further exploration. In addition to that, more work should be done to see how significant the difference in results is between a Reader sentiment analysis model and a Writer

sentiment analysis model within the context of concrete applications like the ones we have explored.

Bibliography

[1] 2012-2016 Facebook Posts - dataset by martinchek. https://data.world/martinchek/2012-2016-facebook-posts.

[2] BERT Fine-Tuning Tutorial with PyTorch · Chris McCormick. https://mccormickml.com/2019/07/22/BERT-fine-tuning/advantages-of-fine-tuning.

[3] Conditions for large files - GitHub Help. https://help.github.com/en/github/managing-large-files/conditions-for-large-files.

[4] Dynos and the Dyno Manager | Heroku Dev Center. https://devcenter.heroku.com/articles/dynos.

[5] Facebook restricts APIs, axes old Instagram platform amidst scandals. https://social.techcrunch.com/2018/04/04/facebook-instagram-api-shutdown/.

[6] File:Clusters.svg. https://en.wikipedia.org/wiki/File:Clusters.svg.

[7] File:Hierarchical clustering simple diagram.svg. https://en.wikipedia.org/wiki/File:Hierarchical$_c lustering_s imple_d iagram.svg$.

[8] Pricing. https://newsapi.org.

[9] Pricing | Heroku. https://www.heroku.com/pricing.

[10] Slug Compiler | Heroku Dev Center. https://devcenter.heroku.com/articles/slug-compiler.

[11] SQLite Home Page. https://www.sqlite.org/index.html.

[12] Step 2.5: Choose a Model | ML Universal Guides. https://developers.google.com/machine-learning/guides/text-classification/step-2-5.

[13] Step 3: Prepare Your Data | ML Universal Guides. https://developers.google.com/machine-learning/guides/text-classification/step-3.

[14] Thankful Reaction. https://knowyourmeme.com/memes/thankful-reaction.

[15] To Scrape Or Not To Scrape? | Rocket Lawyer UK. https://www.rocketlawyer.com/gb/en/blog/to-scrape-or-not-to-scrape/.

[16] Newco Shift | What I Discovered About Trump and Clinton From Analyzing 4 Million Facebook Posts, Nov. 2016. https://shift.newco.co/2016/11/09/what-i-discovered-about-trump-and-clinton-from-analyzing-4-million-facebook-posts/.

[17] A. Agrawal, A. An, and M. Papagelis. Learning emotion-enriched word representations. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 950–961, 2018.

[18] N. Ahmed. How i came up with the discrete cosine transform. *Digital Signal Processing*, 1(1):4–5, 1991.

[19] C. O. Alm and R. Sproat. Emotional sequencing and development in fairy tales. In *International Conference on Affective Computing and Intelligent Interaction*, pages 668–674. Springer, 2005.

[20] J. C. D. A. I. C. A. C. J. G. T. M. E. M. M. d. R. Amigó, Enrique and D. Spina. Overview of replab 2013: Evaluating online reputation monitoring systems. *International Conference of the Cross-Language Evaluation Forum for European Languages*, pages 333–352, 2013.

[21] annieswafford. Problems with the Syuzhet Package, Mar. 2015. https://annieswafford.wordpress.com/2015/03/02/syuzhet/.

[22] H. Baden, McIntyre. The impact of constructive news on affective and behavioural responses. *Journalism Studies*, 20(13):1940–1959, 2019.

[23] A. Balahur, R. Steinberger, M. Kabadjov, V. Zavarella, E. Van Der Goot, M. Halkia, B. Pouliquen, and J. Belyaeva. Sentiment analysis in the news. *arXiv preprint arXiv:1309.6202*, 2013.

[24] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, 1994.

[25] C. Booker. *The Seven Basic Plots: Why We Tell Stories*. A&C Black, Jan. 2004. Google-Books-ID: qHJj9gOl0j8C.

[26] S. Buechel and U. Hahn. Emobank: Studying the impact of annotation perspective and representation format on dimensional emotion analysis. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 578–585, 2017.

[27] S. Buechel and U. Hahn. Readers vs. Writers vs. Texts: Coping with Different Perspectives of Text Understanding in Emotion Annotation. In *LAW@ACL*, 2017.

[28] R. A. Calvo and S. Mac Kim. Emotions in text: dimensional and categorical models. *Computational Intelligence*, 29(3):527–543, 2013.

[29] B. X. Y. H. C. Cambria, E; Schuller. New avenues in opinion mining and sentiment analysis. *IEEE Intelligent Systems*, 28(2):15–21, 2013.

[30] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun. Very deep convolutional networks for natural language processing. *arXiv preprint arXiv:1606.01781*, 2, 2016.

[31] N. S. Cows. Study Confirms Most People Share Articles Based Only On Headlines, Sept. 2018. https://www.patheos.com/blogs/nosacredcows/2018/09/study-confirms-most-people-share-articles-based-only-on-headlines/.

[32] M. Del Vecchio, A. Kharlamov, G. Parry, and G. Pogrebna. The data science of hollywood: Using emotional arcs of movies to drive business model innovation in entertainment industries. *Available at SSRN 3198315*, 2018.

[33] i. Design, Literature, W. |. February 18th, and . . Comments. Kurt Vonnegut Diagrams the Shape of All Stories in a Master's Thesis Rejected by U. Chicago. http://www.openculture.com/2014/02/kurt-vonnegut-masters-thesis-rejected-by-u-chicago.html.

[34] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[35] C. Dos Santos and M. Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78, 2014.

[36] P. Ekman. An argument for basic emotions. *Cognition and Emotion*, 6(3-4):169–200, May 1992. https://doi.org/10.1080/02699939208411068.

[37] M. Faruqui, J. Dodge, S. K. Jauhar, C. Dyer, E. Hovy, and N. A. Smith. Retrofitting word vectors to semantic lexicons. *arXiv preprint arXiv:1411.4166*, 2014.

[38] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):1–37, 2014.

[39] T. D. Gautheir. Detecting trends using spearman's rank correlation coefficient. *Environmental forensics*, 2(4):359–362, 2001.

[40] N. Godbole, M. Srinivasaiah, and S. Skiena. Large-scale sentiment analysis for news and blogs. *Icwsm*, 7(21):219–222, 2007.

[41] I. Goodman, V. Jain, and S. Pai. Predicting emotional reaction distributions from buzzfeed articles.

[42] E. Guzman, D. Azocar, and Y. Li. Sentiment analysis of commit comments in github: an empirical study. *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 352–355, 2014.

[43] F. E. Harrell Jr. *Regression modeling strategies: with applications to linear models, logistic and ordinal regression, and survival analysis*. Springer, 2015.

[44] S. Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

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

[46] J. Howard and S. Ruder. Fine-tuned language models for text classification. *arXiv preprint arXiv:1801.06146*, pages 1–7, 2018.

[47] C. J. Hutto and E. Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth international AAAI conference on weblogs and social media*, 2014.

[48] A. Jacovi, O. S. Shalom, and Y. Goldberg. Understanding convolutional neural networks for text classification. *arXiv preprint arXiv:1809.08037*, 2018.

[49] M. Jockers. » Revealing Sentiment and Plot Arcs with the Syuzhet Package Matthew L. Jockers. http://www.matthewjockers.net/2015/02/02/syuzhet/.

[50] S. L. Johnson, L. J. Leedom, and L. Muhtadie. The dominance behavioral system and psychopathology: Evidence from self-report, observational, and biological studies. *Psychological bulletin*, 138(4):692, 2012.

[51] N. Ketkar. Introduction to pytorch. In *Deep learning with python*, pages 195–208. Springer, 2017.

[52] E. Kim, S. Padó, and R. Klinger. Prototypical emotion developments in literary genres. In *Proceedings of the Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*, pages 17–26, 2017.

[53] S. M. Kim, A. Valitutti, and R. A. Calvo. Evaluation of unsupervised emotion models to textual affect recognition. In *Proceedings of the NAACL HLT 2010 Workshop on Computational Approaches to Analysis and Generation of Emotion in Text*, pages 62–70. Association for Computational Linguistics, 2010.

[54] M. Koppel and J. Schler. The importance of neutral examples for learning sentiment. *Computational Intelligence*, 22(2):100–109, 2006.

[55] F. Krebs, B. Lubascher, T. Moers, P. Schaap, and G. Spanakis. Social Emotion Mining Techniques for Facebook Posts Reaction Prediction. *arXiv:1712.03249 [cs]*, Dec. 2017. http://arxiv.org/abs/1712.03249.

[56] J. Kun. j2kun/imsdb_download_all_scripts, May 2020. https://github.com/j2kun/imsdb$_{d}ownload_{a}ll_{s}cripts$.

[57] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[58] K. H.-Y. Lin, C. Yang, and H.-H. Chen. What emotions do news articles trigger in their readers? In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 733–734, 2007.

[59] B. Liu. *Sentiment Analysis and Opinion Mining*. Morgan & Claypool Publishers, 2012.

[60] P. Liu, X. Qiu, and X. Huang. Recurrent neural network for text classification with multi-task learning. *arXiv preprint arXiv:1605.05101*, 2016.

[61] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[62] O. Maimon and L. Rokach. Data mining and knowledge discovery handbook. 2005.

[63] S. S. . S. McAdams. News, politics, and negativity. *Political Communication*, 32(1):1–22, 2015.

[64] A. Mehrabian. Pleasure-arousal-dominance: A general framework for describing and measuring individual differences in temperament. *Current Psychology*, 14(4):261–292, 1996.

[65] T. Mikolov, K. Chen, G. Corrado, J. Dean, L. Sutskever, and G. Zweig. word2vec. *URL https://code. google. com/p/word2vec*, 2013.

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

[67] S. Mohammad. From once upon a time to happily ever after: Tracking emotions in novels and fairy tales. *arXiv preprint arXiv:1309.5909*, 2013.

[68] S. M. Mohammad and P. D. Turney. Nrc emotion lexicon. *National Research Council, Canada*, 2013.

[69] F. Murtagh and P. Legendre. Ward's hierarchical agglomerative clustering method: which algorithms implement ward's criterion? *Journal of classification*, 31(3):274–295, 2014.

[70] F. Nielsen. Hierarchical clustering. In *Introduction to HPC with MPI for Data Science*, pages 195–211. Springer, 2016.

[71] B. Pang and L. Lee. Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2(1-2):1–135, 2008.

[72] A. H. Parizi, M. Kazemifard, and M. Asghari. Emonews: an emotional news recommender system. *Journal of Digital Information Management*, 14(6), 2016.

[73] R. Plutchik. *Emotions in the practice of psychotherapy: clinical implications of affect theories*. American Psychological Association, 2000.

[74] C. Pool and M. Nissim. Distant supervision for emotion detection using Facebook reactions. In *PEOPLES@COLING*, 2016.

[75] B. T. Raad, B. Philipp, H. Patrick, and M. Christoph. Aseds: Towards automatic social emotion detection system using facebook reactions. In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 860–866. IEEE, 2018.

[76] J. Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142. Piscataway, NJ, 2003.

[77] A. J. Reagan, C. M. Danforth, B. Tivnan, J. R. Williams, and P. S. Dodds. Sentiment analysis methods for understanding large-scale texts: a case for using continuum-scored words and word shift graphs. *EPJ Data Science*, 6(1):28, 2017.

[78] A. J. Reagan, L. Mitchell, D. Kiley, C. M. Danforth, and P. S. Dodds. The emotional arcs of stories are dominated by six basic shapes. *EPJ Data Science*, 5(1):31, 2016.

[79] M. Rice-Oxley. The good news is ... people like to read good news. *The Guardian*, 2018. https://www.theguardian.com/world/2018/feb/12/but-first-here-is-the-good-news-.

[80] J. A. Russell. A circumplex model of affect. *Journal of personality and social psychology*, 39(6):1161, 1980.

[81] S. Salvador and P. Chan. Fastdtw: Toward accurate dynamic time warping in linear time and space. 3rd wkshp. on mining temporal and sequential data. *ACM KDD '04*, 2004.

[82] S. Samothrakis and M. Fasli. Emotional sentence annotation helps predict fiction genre. *PloS one*, 10(11):e0141922, 2015.

[83] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[84] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[85] L. Stinson. Facebook Reactions, the Totally Redesigned Like Button, Is Here. *Wired*, Feb. 2016. https://www.wired.com/2016/02/facebook-reactions-totally-redesigned-like-button/.

[86] C. Strapparava and R. Mihalcea. SemEval-2007 Task 14: Affective Text. In *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*, pages 70–74, Prague, Czech Republic, June 2007. Association for Computational Linguistics. https://www.aclweb.org/anthology/S07-1013.

[87] C. Strapparava, A. Valitutti, et al. Wordnet affect: an affective extension of wordnet. In *Lrec*, volume 4, page 40. Citeseer, 2004.

[88] B. Stroube. Literary freedom: Project gutenberg. *XRDS: Crossroads, The ACM Magazine for Students*, 10(1):3–3, 2003.

[89] Y.-j. Tang and H.-H. Chen. Emotion modeling from writer/reader perspectives using a microblog dataset. In *Proceedings of the Workshop on Sentiment Analysis where AI meets Psychology (SAAIP 2011)*, pages 11–19, 2011.

[90] tedunderwood. Free research question about plot., Apr. 2015. https://tedunderwood.com/2015/04/01/free-research-question-about-plot/.

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

[92] J. Wang, L.-C. Yu, K. R. Lai, and X. Zhang. Dimensional sentiment analysis using a regional cnn-lstm model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 225–230, 2016.

[93] G. C. L. D. Wendy M. Johnston. The psychological impact of negative tv news bulletins: The catastrophizing of personal worries. *British Journal of Psychology*, 88(1), 2011.

[94] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[95] C. Wolff. c-w/gutenberg, May 2020. https://github.com/c-w/gutenberg.

[96] C.-H. Wu, Z.-J. Chuang, and Y.-C. Lin. Emotion recognition from text using semantic labels and separable mixture models. *ACM transactions on Asian language information processing (TALIP)*, 5(2):165–183, 2006.

[97] C. Zirn, M. Niepert, H. Stuckenschmidt, and M. Strube. Fine-grained sentiment analysis with structural features. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 336–344, 2011.

Appendix A

Training Details - Reader-Categorical-Mixed

In this chapter, we detail the model hyperparameters and architectures used. We use mean squared error loss and a softmax output layer. We also experimented with categorical cross entropy loss but obtained similar results. We perform 5-fold cross validation with a train-validation-test split of 80-10-10, performing early stopping, and shuffling our data beforehand. We did manual hyperparameter tuning.

A.1 Likes vs. No Likes Experiment

A.1.1 Hyperparameters

Vec. Encoding	Model	Epochs	Batch size	Features
One-hot	MLP	2	128	Most frequent 20k

A.1.2 Model Architecture

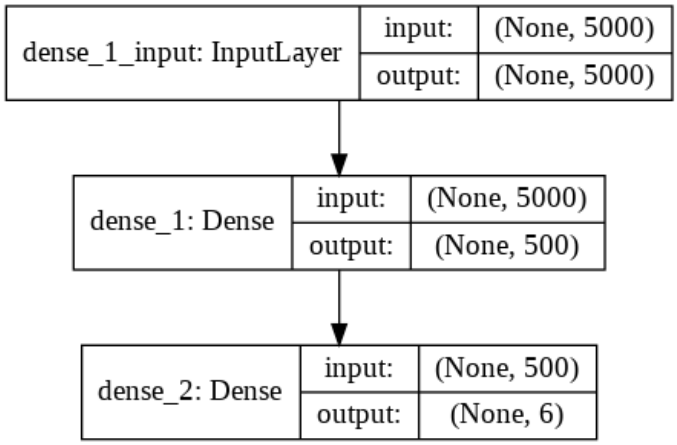


Figure A.1: MLP Diagram

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 500)               2500500
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 3006
=================================================================
Total params: 2,503,506
Trainable params: 2,503,506
Non-trainable params: 0
```

Figure A.2: MLP Summary

A.2 Bag-Of-Words Approach

A.2.1 Hyperparameters

Vec. Encoding	Model	Epochs	Batch size	Features
One-hot	MLP	2	128	Most frequent 20k
Frequency	MLP	2	128	Most frequent 20k
TF-IDF	MLP	1	128	Most frequent 20k

Vec. Encoding	Model	Optimiser	Optimiser Params.	N-grams
One-hot	MLP	Adam	default	1-gram, 2-gram
Frequency	MLP	Adam	default	1-gram, 2-gram
TF-IDF	MLP	Adam	default	1-gram, 2-gram

A.2.2 Model Architecture

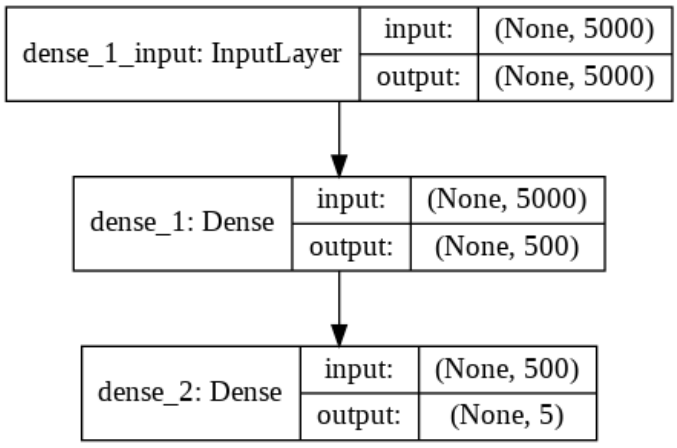


Figure A.3: MLP Diagram

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 500)               2500500
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 2505
=================================================================
Total params: 2,503,005
Trainable params: 2,503,005
Non-trainable params: 0
```

Figure A.4: MLP Summary

A.3 Sequence Vectors Approach

A.3.1 Hyperparameters

Embed. Type	Model	Epochs	Batch size	Dimension
Google word2vec	MLP	2	128	300
Facebook word2vec	MLP	2	128	100
Emotion-aware embeddings	MLP	2	128	300
Retrofitted FB word2vec	MLP	2	128	100

Embed. Type	Model	Epochs	Optimiser	Optimiser Params.
Google word2vec	MLP	6	Adam	default
Google word2vec	CNN	6	Adam	default
Google word2vec	RNN (LSTM)	6	Adam	default

A.3.2 Model Architecture

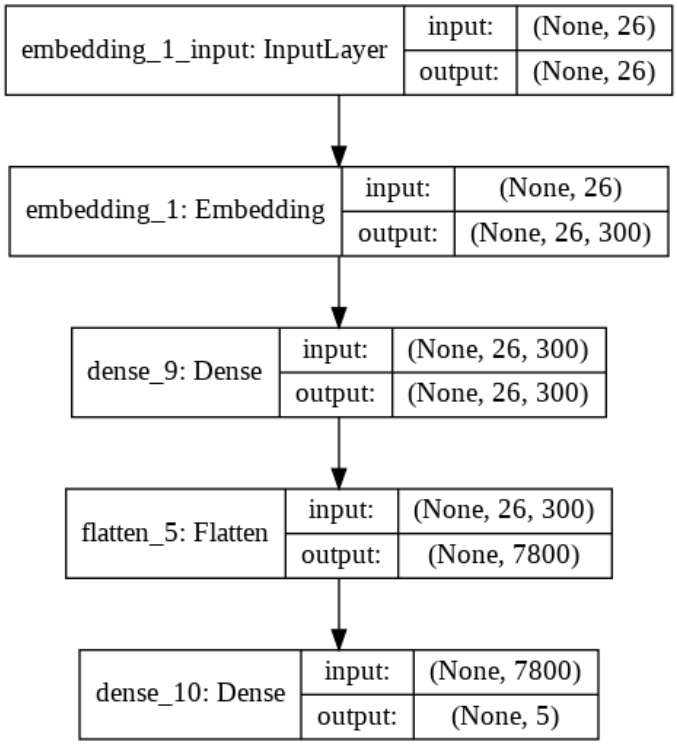


Figure A.5: Google word2vec and Emotion-aware embeddings MLP Diagram

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 26, 300)           12471900
_________________________________________________________________
dense_1 (Dense)              (None, 26, 300)           90300
_________________________________________________________________
flatten_1 (Flatten)          (None, 7800)              0
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 39005
=================================================================
Total params: 12,601,205
Trainable params: 129,305
Non-trainable params: 12,471,900
```

Figure A.6: Google word2vec and Emotion-aware embeddings MLP Summary

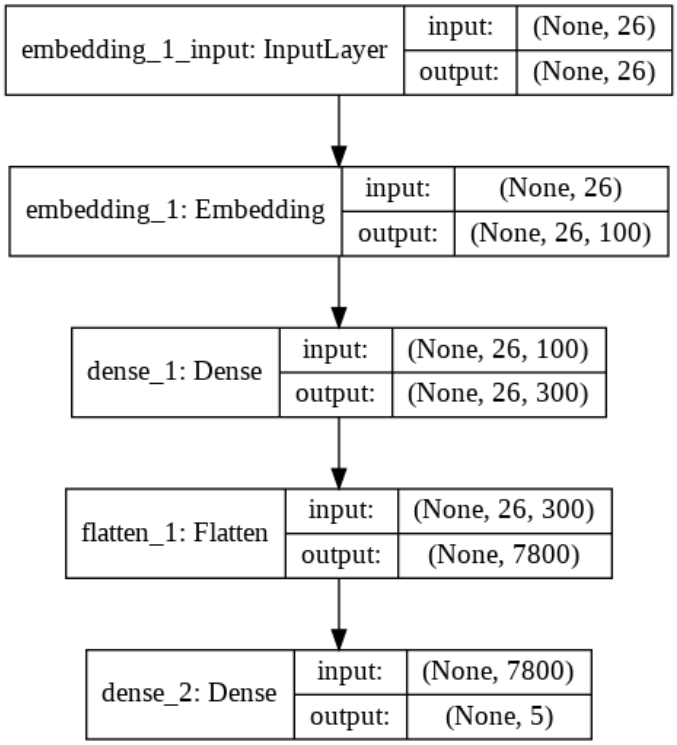


Figure A.7: Facebook word2vec and Retrofitted FB word2vec MLP Diagram

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 26, 100)           4157300
_________________________________________________________________
dense_1 (Dense)              (None, 26, 300)           30300
_________________________________________________________________
flatten_1 (Flatten)          (None, 7800)              0
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 39005
=================================================================
Total params: 4,226,605
Trainable params: 69,305
Non-trainable params: 4,157,300
```

Figure A.8: Facebook word2vec and Retrofitted FB word2vec MLP Summary

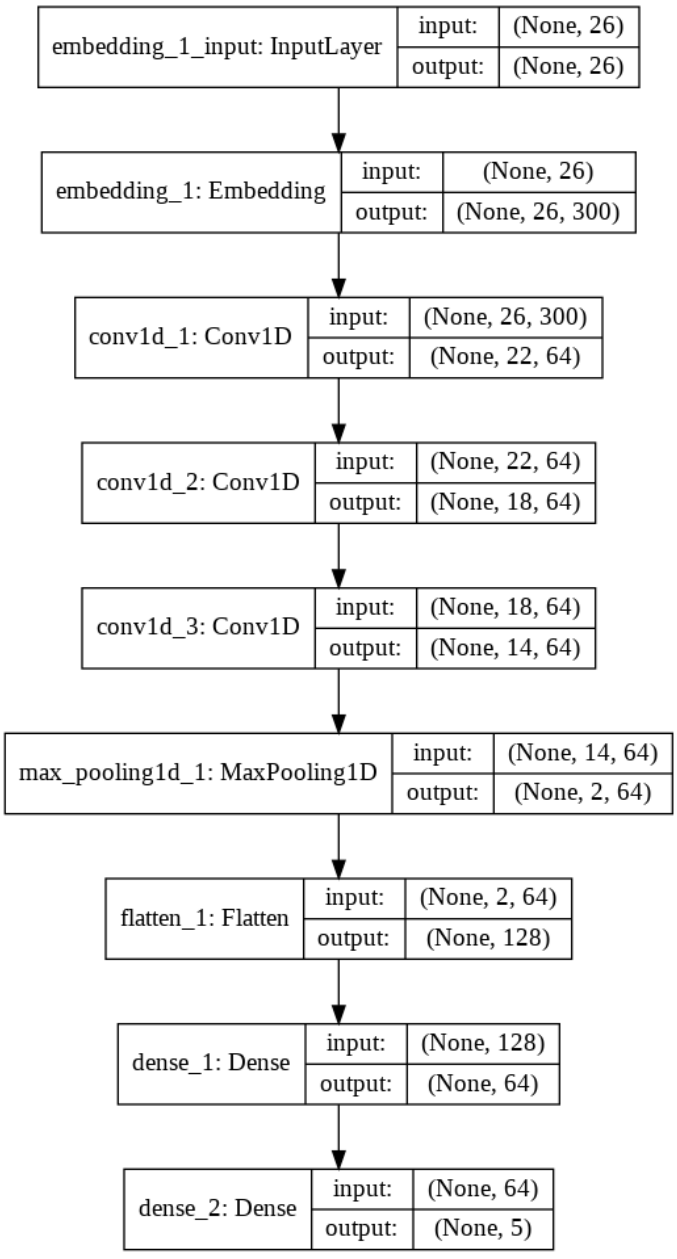


Figure A.9: Google word2vec and CNN Diagram

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 26, 300)           12471900
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 22, 64)            96064
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 18, 64)            20544
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 14, 64)            20544
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 2, 64)             0
_________________________________________________________________
flatten_1 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 325
=================================================================
Total params: 12,617,633
Trainable params: 145,733
Non-trainable params: 12,471,900
```

Figure A.10: Google word2vec and CNN Summary

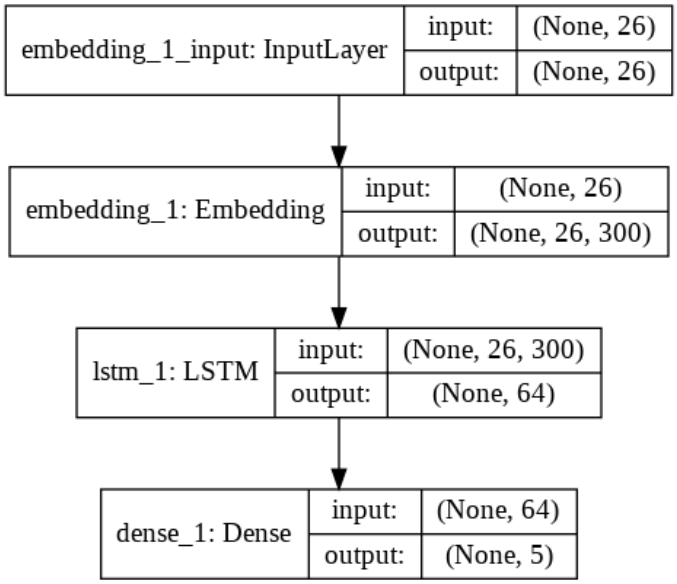


Figure A.11: Google word2vec and RNN (LSTM) Diagram

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 26, 300)           12471900
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                93440
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 325
=================================================================
Total params: 12,565,665
Trainable params: 93,765
Non-trainable params: 12,471,900
_________________________________________________________________
None
```

Figure A.12: Google word2vec and RNN (LSTM) Summary

A.4 BERT Approach

Model	Pretrained Model	Epochs	Batch size
DistilBERT	`distilbert-base-uncased`	2	32
BERT	`bert-base-uncased`	2	32
RoBERTa	`roberta-base`	2	32
ALBERT	`albert-base-v1`	2	32

Model	Optimiser	Learning Rate	Truncation Length
DistilBERT	AdamW	`lr=2e-5`, rest default	37 tokens
BERT	AdamW	`lr=2e-5`, rest default	37 tokens
RoBERTa	AdamW	`lr=2e-5`, rest default	37 tokens
ALBERT	AdamW	`lr=2e-5`, rest default	37 tokens

A.4.1 Model Architecture

Pretrained models are as provided by the HuggingFace `transformers` library, with modified downstream neural networks to support multi-dimensional regression.

DistilBERT

```
MyDistilBertForSequenceClassification(
  (distilbert): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0): TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
```

```
)
(1): TransformerBlock(
  (attention): MultiHeadSelfAttention(
    (dropout): Dropout(p=0.1, inplace=False)
    (q_lin): Linear(in_features=768, out_features=768, bias=True)
    (k_lin): Linear(in_features=768, out_features=768, bias=True)
    (v_lin): Linear(in_features=768, out_features=768, bias=True)
    (out_lin): Linear(in_features=768, out_features=768, bias=True)
  )
  (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  (ffn): FFN(
    (dropout): Dropout(p=0.1, inplace=False)
    (lin1): Linear(in_features=768, out_features=3072, bias=True)
    (lin2): Linear(in_features=3072, out_features=768, bias=True)
  )
  (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
)
(2): TransformerBlock(
  (attention): MultiHeadSelfAttention(
    (dropout): Dropout(p=0.1, inplace=False)
    (q_lin): Linear(in_features=768, out_features=768, bias=True)
    (k_lin): Linear(in_features=768, out_features=768, bias=True)
    (v_lin): Linear(in_features=768, out_features=768, bias=True)
    (out_lin): Linear(in_features=768, out_features=768, bias=True)
  )
  (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  (ffn): FFN(
    (dropout): Dropout(p=0.1, inplace=False)
    (lin1): Linear(in_features=768, out_features=3072, bias=True)
    (lin2): Linear(in_features=3072, out_features=768, bias=True)
  )
  (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
)
(3): TransformerBlock(
  (attention): MultiHeadSelfAttention(
    (dropout): Dropout(p=0.1, inplace=False)
    (q_lin): Linear(in_features=768, out_features=768, bias=True)
    (k_lin): Linear(in_features=768, out_features=768, bias=True)
    (v_lin): Linear(in_features=768, out_features=768, bias=True)
    (out_lin): Linear(in_features=768, out_features=768, bias=True)
  )
  (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  (ffn): FFN(
    (dropout): Dropout(p=0.1, inplace=False)
    (lin1): Linear(in_features=768, out_features=3072, bias=True)
    (lin2): Linear(in_features=3072, out_features=768, bias=True)
  )
  (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
)
(4): TransformerBlock(
  (attention): MultiHeadSelfAttention(
```

```
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
        (5): TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
      )
    )
  )
  (pre_classifier): Linear(in_features=768, out_features=768, bias=True)
  (classifier): Linear(in_features=768, out_features=5, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

BERT

```
MyBertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
```

```
(0): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(1): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(2): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
```

```
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(3): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(4): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
```

```
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(5): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(6): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
```

```
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(7): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(8): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
```

```
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(9): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(10): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(11): BertLayer(
  (attention): BertAttention(
```

```
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=5, bias=True)
)
```

RoBERTa

```
MyRobertaForSequenceClassification(
  (roberta): RobertaModel(
    (embeddings): RobertaEmbeddings(
      (word_embeddings): Embedding(50265, 768, padding_idx=1)
      (position_embeddings): Embedding(514, 768, padding_idx=1)
      (token_type_embeddings): Embedding(1, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
```

```
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(1): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(2): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
```

```
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(3): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(4): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
```

```
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(5): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(6): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
```

```
  )
)
(7): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(8): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(9): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
```

```
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (10): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (11): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
```

```
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (classifier): MyRobertaClassificationHead(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (out_proj): Linear(in_features=768, out_features=5, bias=True)
  )
)
```

ALBERT

```
MyAlbertForSequenceClassification(
  (albert): AlbertModel(
    (embeddings): AlbertEmbeddings(
      (word_embeddings): Embedding(30000, 128, padding_idx=0)
      (position_embeddings): Embedding(512, 128)
      (token_type_embeddings): Embedding(2, 128)
      (LayerNorm): LayerNorm((128,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): AlbertTransformer(
      (embedding_hidden_mapping_in): Linear(in_features=128, out_features=768, bias=True)
      (albert_layer_groups): ModuleList(
        (0): AlbertLayerGroup(
          (albert_layers): ModuleList(
            (0): AlbertLayer(
              (full_layer_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True
              (attention): AlbertAttention(
                (query): Linear(in_features=768, out_features=768, bias=True)
                (key): Linear(in_features=768, out_features=768, bias=True)
                (value): Linear(in_features=768, out_features=768, bias=True)
```

```
              (dropout): Dropout(p=0.1, inplace=False)
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            )
            (ffn): Linear(in_features=768, out_features=3072, bias=True)
            (ffn_output): Linear(in_features=3072, out_features=768, bias=True)
          )
        )
      )
    )
    (pooler): Linear(in_features=768, out_features=768, bias=True)
    (pooler_activation): Tanh()
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=5, bias=True)
)
```

Appendix B

Training Details - Reader-Dimensional-Regression

In this chapter, we detail the model hyperparameters and architectures used. We use mean squared error loss and a ReLU output layer. We perform 5-fold cross validation with a train-validation-test split of 80-10-10, performing early stopping, and shuffling our data beforehand. We did manual hyperparameter tuning.

B.1 Bag-Of-Words Approach

B.1.1 Hyperparameters

Vec. Encoding	Model	Epochs	Batch size	Features
One-hot	MLP	8	128	Most frequent 20k
Frequency	MLP	8	128	Most frequent 20k
TF-IDF	MLP	8	128	Most frequent 20k

Vec. Encoding	Model	Optimiser	Optimiser Params.	N-grams
One-hot	MLP	Adam	default	1-gram, 2-gram
Frequency	MLP	Adam	default	1-gram, 2-gram
TF-IDF	MLP	Adam	default	1-gram, 2-gram

B.1.2 Model Architecture

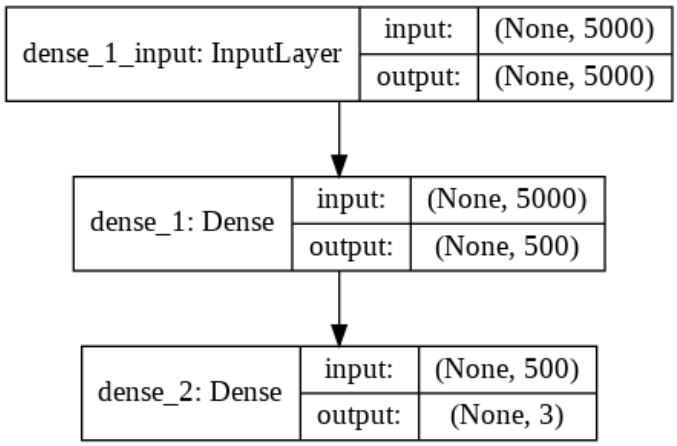


Figure B.1: MLP Diagram

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 500)               2500500
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 1503
=================================================================
Total params: 2,502,003
Trainable params: 2,502,003
Non-trainable params: 0
```

Figure B.2: MLP Summary

B.2 Sequence Vectors Approach

B.2.1 Hyperparameters

Embed. Type	Model	Epochs	Batch size	Dimension
Google word2vec	MLP	10	128	300
Facebook word2vec	MLP	10	128	100
Emotion-aware embeddings	MLP	10	128	300
Retrofitted FB word2vec	MLP	10	128	100

Embed. Type	Model	Epochs	Optimiser	Optimiser Params.
Google word2vec	MLP	10	Adam	default
Google word2vec	CNN	10	Adam	default
Google word2vec	RNN (LSTM)	10	Adam	default

B.2.2 Model Architecture

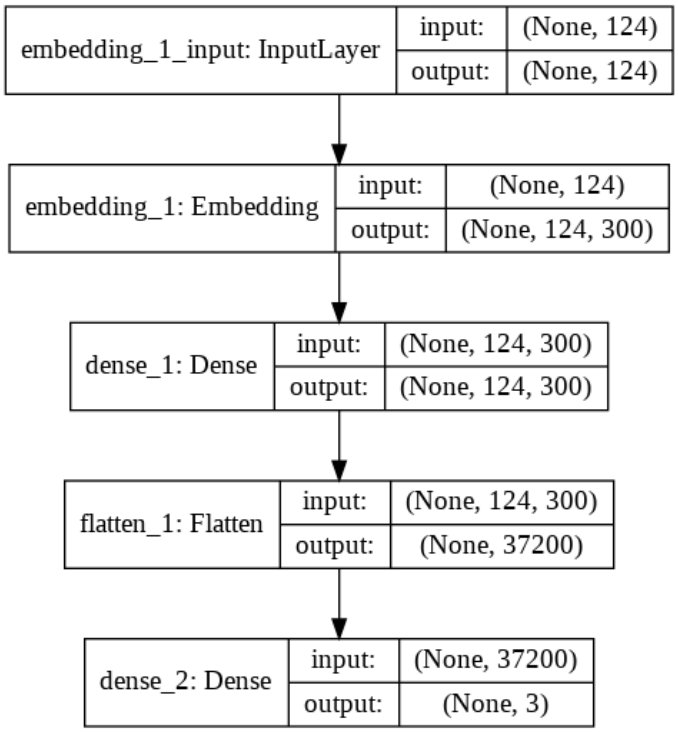


Figure B.3: Google word2vec and Emotion-aware embeddings MLP Diagram

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 124, 300)          4896300
_________________________________________________________________
dense_1 (Dense)              (None, 124, 300)          90300
_________________________________________________________________
flatten_1 (Flatten)          (None, 37200)             0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 111603
=================================================================
Total params: 5,098,203
Trainable params: 201,903
Non-trainable params: 4,896,300
```

Figure B.4: Google word2vec and Emotion-aware embeddings MLP Summary

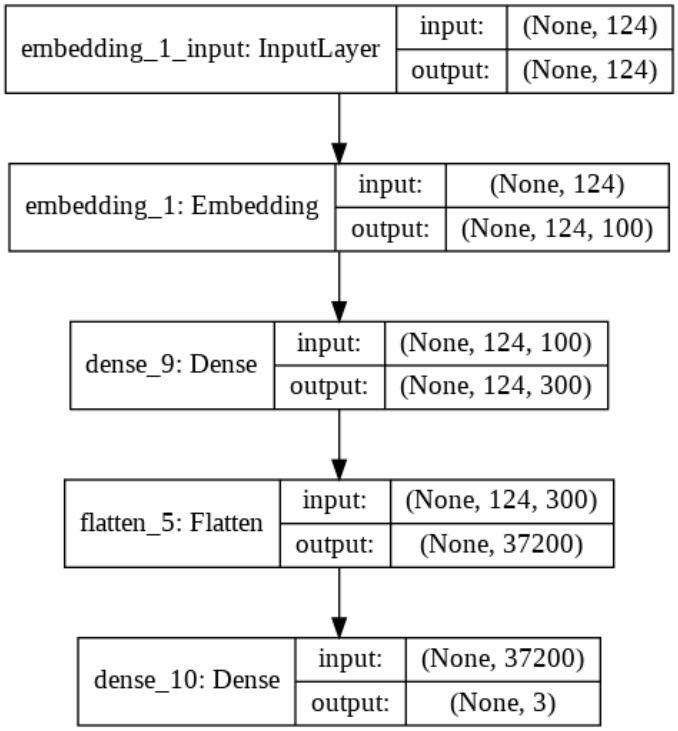


Figure B.5: Facebook word2vec and Retrofitted FB word2vec MLP Diagram

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 124, 100)          1632100
_________________________________________________________________
dense_1 (Dense)              (None, 124, 300)          30300
_________________________________________________________________
flatten_1 (Flatten)          (None, 37200)             0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 111603
=================================================================
Total params: 1,774,003
Trainable params: 141,903
Non-trainable params: 1,632,100
```

Figure B.6: Facebook word2vec and Retrofitted FB word2vec MLP Summary

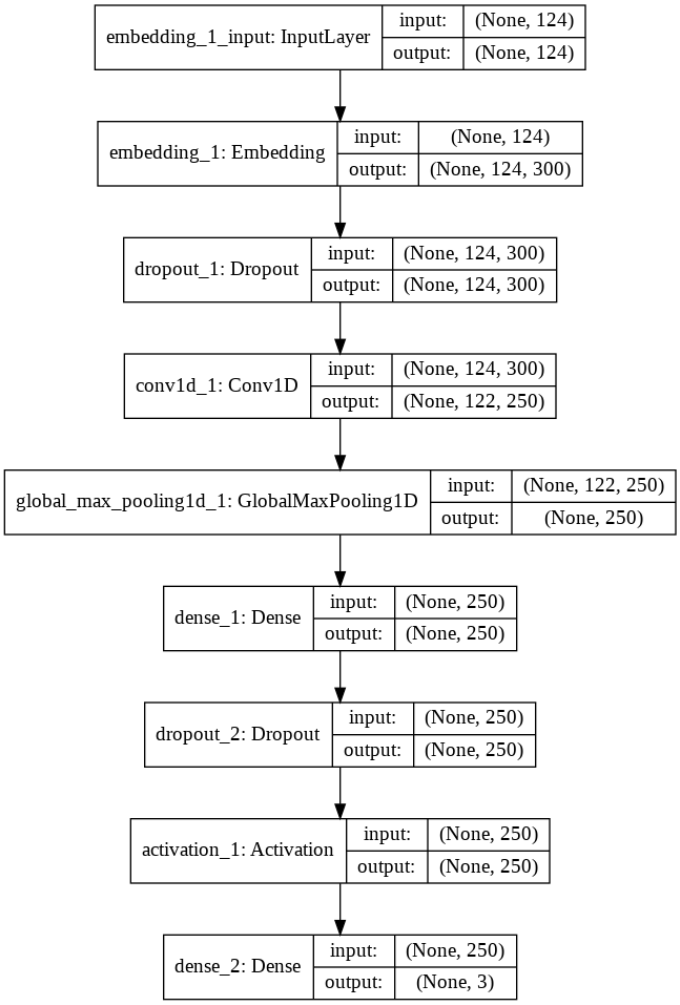


Figure B.7: Google word2vec and CNN Diagram

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 124, 300)          4896300
_________________________________________________________________
dropout_1 (Dropout)          (None, 124, 300)          0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 122, 250)          225250
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 250)               0
_________________________________________________________________
dense_1 (Dense)              (None, 250)               62750
_________________________________________________________________
dropout_2 (Dropout)          (None, 250)               0
_________________________________________________________________
activation_1 (Activation)    (None, 250)               0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 753
=================================================================
Total params: 5,185,053
Trainable params: 288,753
Non-trainable params: 4,896,300
```

Figure B.8: Google word2vec and CNN Summary

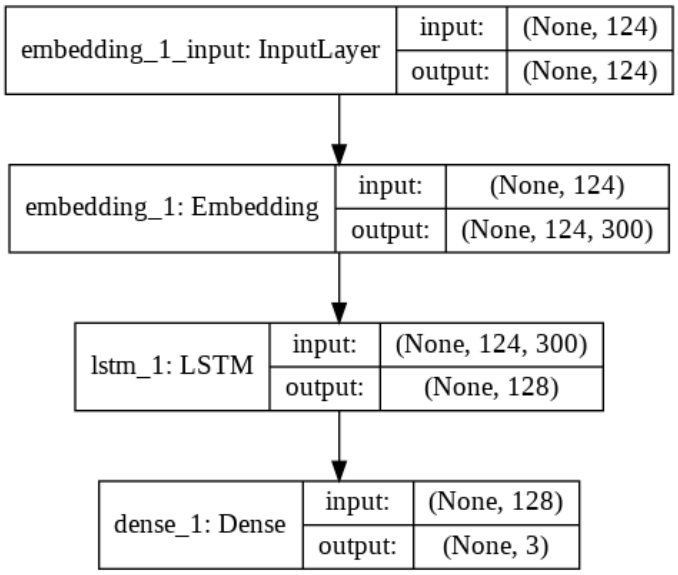


Figure B.9: Google word2vec and RNN (LSTM) Diagram

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 124, 300)          4896300
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               219648
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 387
=================================================================
Total params: 5,116,335
Trainable params: 220,035
Non-trainable params: 4,896,300
```

Figure B.10: Google word2vec and RNN (LSTM) Summary

B.3 BERT Approach

Model	Pretrained Model	Epochs	Batch size
DistilBERT	`distilbert-base-uncased`	4	32
BERT	`bert-base-uncased`	4	32
RoBERTa	`roberta-base`	4	32
ALBERT	`albert-base-v1`	4	32

Model	Optimiser	Learning Rate	Truncation Length
DistilBERT	AdamW	`lr=3e-5`, rest default	33 tokens
BERT	AdamW	`lr=3e-5`, rest default	33 tokens
RoBERTa	AdamW	`lr=3e-5`, rest default	33 tokens
ALBERT	AdamW	`lr=3e-5`, rest default	33 tokens

B.3.1 Model Architecture

Pretrained models are as provided by the HuggingFace `transformers` library, with modified downstream neural networks to support multi-dimensional regression.

DistilBERT

```
MyDistilBertForSequenceClassification(
  (distilbert): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0): TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
```

```
)
(1): TransformerBlock(
  (attention): MultiHeadSelfAttention(
    (dropout): Dropout(p=0.1, inplace=False)
    (q_lin): Linear(in_features=768, out_features=768, bias=True)
    (k_lin): Linear(in_features=768, out_features=768, bias=True)
    (v_lin): Linear(in_features=768, out_features=768, bias=True)
    (out_lin): Linear(in_features=768, out_features=768, bias=True)
  )
  (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  (ffn): FFN(
    (dropout): Dropout(p=0.1, inplace=False)
    (lin1): Linear(in_features=768, out_features=3072, bias=True)
    (lin2): Linear(in_features=3072, out_features=768, bias=True)
  )
  (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
)
(2): TransformerBlock(
  (attention): MultiHeadSelfAttention(
    (dropout): Dropout(p=0.1, inplace=False)
    (q_lin): Linear(in_features=768, out_features=768, bias=True)
    (k_lin): Linear(in_features=768, out_features=768, bias=True)
    (v_lin): Linear(in_features=768, out_features=768, bias=True)
    (out_lin): Linear(in_features=768, out_features=768, bias=True)
  )
  (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  (ffn): FFN(
    (dropout): Dropout(p=0.1, inplace=False)
    (lin1): Linear(in_features=768, out_features=3072, bias=True)
    (lin2): Linear(in_features=3072, out_features=768, bias=True)
  )
  (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
)
(3): TransformerBlock(
  (attention): MultiHeadSelfAttention(
    (dropout): Dropout(p=0.1, inplace=False)
    (q_lin): Linear(in_features=768, out_features=768, bias=True)
    (k_lin): Linear(in_features=768, out_features=768, bias=True)
    (v_lin): Linear(in_features=768, out_features=768, bias=True)
    (out_lin): Linear(in_features=768, out_features=768, bias=True)
  )
  (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  (ffn): FFN(
    (dropout): Dropout(p=0.1, inplace=False)
    (lin1): Linear(in_features=768, out_features=3072, bias=True)
    (lin2): Linear(in_features=3072, out_features=768, bias=True)
  )
  (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
)
(4): TransformerBlock(
  (attention): MultiHeadSelfAttention(
```

```
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
        (5): TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
      )
    )
  )
  (pre_classifier): Linear(in_features=768, out_features=768, bias=True)
  (classifier): Linear(in_features=768, out_features=3, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

BERT

```
Downloading: 100%
433/433 [00:00<00:00, 499B/s]

Downloading: 100%
440M/440M [00:18<00:00, 23.8MB/s]

MyBertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
```

```
  (token_type_embeddings): Embedding(2, 768)
  (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  (dropout): Dropout(p=0.1, inplace=False)
)
(encoder): BertEncoder(
  (layer): ModuleList(
    (0): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (1): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
```

```
)
(2): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(3): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(4): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
```

```
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(5): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(6): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
```

```
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(7): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(8): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
```

```
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(9): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(10): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
```

```
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (11): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=3, bias=True)
)
```

RoBERTa

```
Downloading: 100%
481/481 [00:00<00:00, 3.37kB/s]

Downloading: 100%
501M/501M [00:06<00:00, 72.4MB/s]

MyRobertaForSequenceClassification(
  (roberta): RobertaModel(
    (embeddings): RobertaEmbeddings(
      (word_embeddings): Embedding(50265, 768, padding_idx=1)
      (position_embeddings): Embedding(514, 768, padding_idx=1)
```

```
  (token_type_embeddings): Embedding(1, 768)
  (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  (dropout): Dropout(p=0.1, inplace=False)
)
(encoder): BertEncoder(
  (layer): ModuleList(
    (0): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (1): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
```

```
)
(2): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(3): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(4): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
```

```
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(5): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(6): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
```

```
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(7): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(8): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
```

```
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(9): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
)
(10): BertLayer(
  (attention): BertAttention(
    (self): BertSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (output): BertSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (intermediate): BertIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
  )
  (output): BertOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
```

```
            (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (11): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (classifier): MyRobertaClassificationHead(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (out_proj): Linear(in_features=768, out_features=3, bias=True)
  )
)
```

ALBERT

```
MyAlbertForSequenceClassification(
  (albert): AlbertModel(
    (embeddings): AlbertEmbeddings(
      (word_embeddings): Embedding(30000, 128, padding_idx=0)
      (position_embeddings): Embedding(512, 128)
      (token_type_embeddings): Embedding(2, 128)
      (LayerNorm): LayerNorm((128,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
```

```
    )
    (encoder): AlbertTransformer(
      (embedding_hidden_mapping_in): Linear(in_features=128, out_features=768, bias=True)
      (albert_layer_groups): ModuleList(
        (0): AlbertLayerGroup(
          (albert_layers): ModuleList(
            (0): AlbertLayer(
              (full_layer_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True
              (attention): AlbertAttention(
                (query): Linear(in_features=768, out_features=768, bias=True)
                (key): Linear(in_features=768, out_features=768, bias=True)
                (value): Linear(in_features=768, out_features=768, bias=True)
                (dropout): Dropout(p=0.1, inplace=False)
                (dense): Linear(in_features=768, out_features=768, bias=True)
                (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              )
              (ffn): Linear(in_features=768, out_features=3072, bias=True)
              (ffn_output): Linear(in_features=3072, out_features=768, bias=True)
            )
          )
        )
      )
    )
    (pooler): Linear(in_features=768, out_features=768, bias=True)
    (pooler_activation): Tanh()
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=3, bias=True)
)
```

Appendix C

Training Details - Ensemble Meta-Learner

In this chapter, we detail the model hyperparameters and architectures used. We use mean squared error loss and a softmax output layer. We perform 5-fold cross validation with a train-validation-test split of 80-10-10, performing early stopping, and shuffling our data beforehand. We did manual hyperparameter tuning.

C.1 Hyperparameters

Model	Epochs	Batch size	Optimiser	Optimiser Params.
MLP	20	128	Adam	default

C.2 Model Architecture

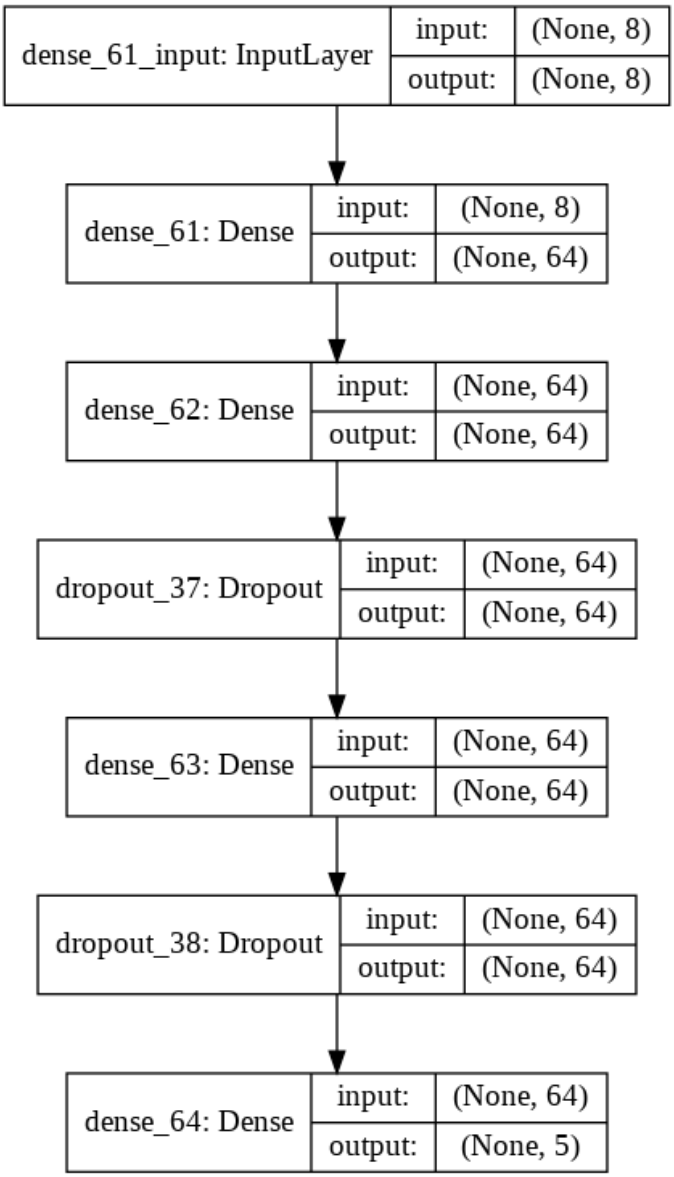
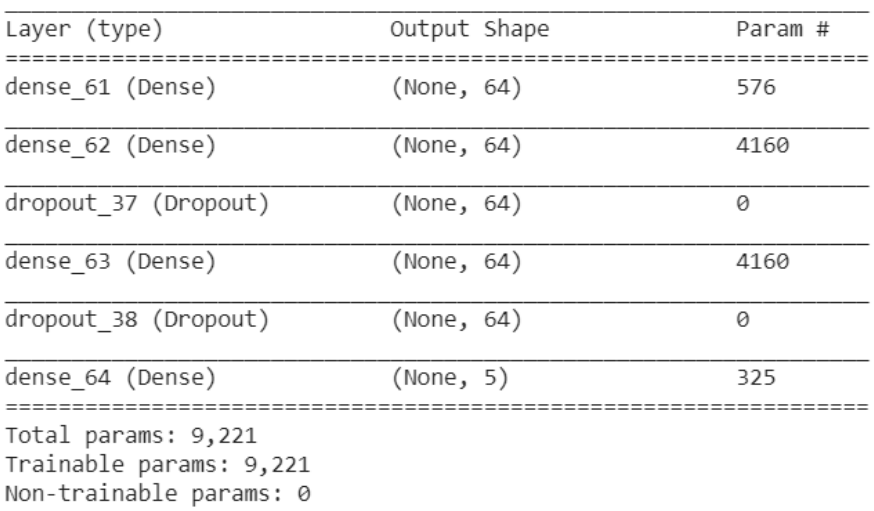


Figure C.1: Meta-Learner Diagram



Layer (type)	Output Shape	Param #
dense_61 (Dense)	(None, 64)	576
dense_62 (Dense)	(None, 64)	4160
dropout_37 (Dropout)	(None, 64)	0
dense_63 (Dense)	(None, 64)	4160
dropout_38 (Dropout)	(None, 64)	0
dense_64 (Dense)	(None, 5)	325

Total params: 9,221
Trainable params: 9,221
Non-trainable params: 0

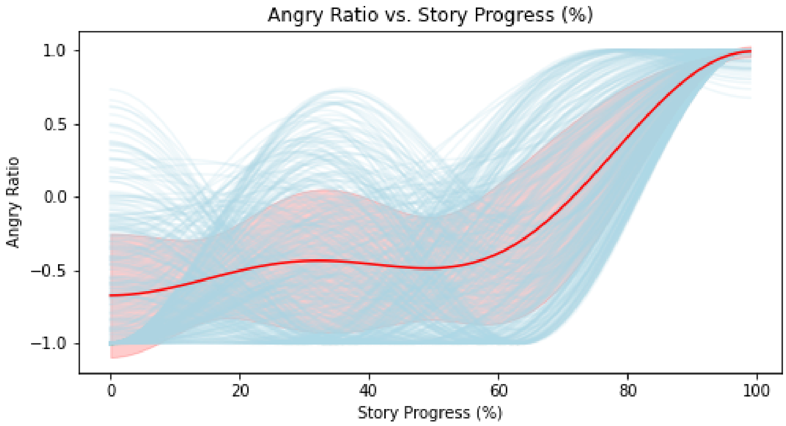
Figure C.2: Meta-Learner Summary

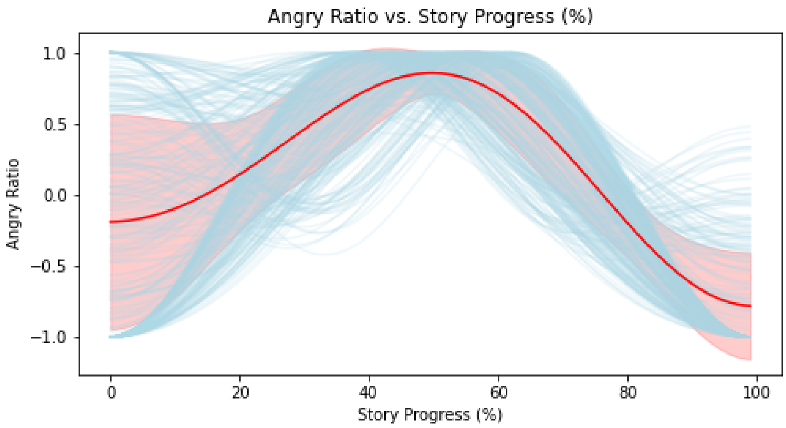
Appendix D

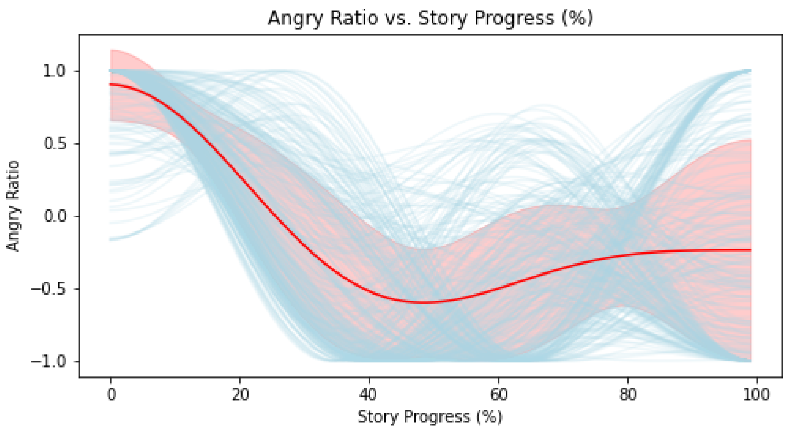
Emotional Arc Clusters

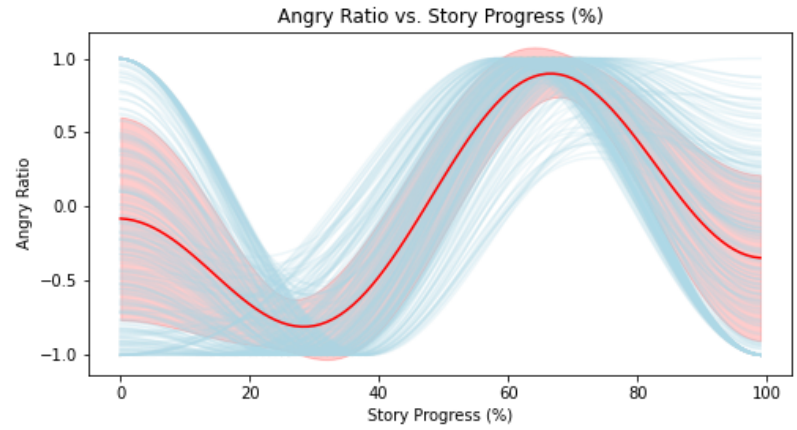
The follow clusters are for the corpus of Project Gutenberg books and IMSDB movie scripts. DCT smoothing was applied, with a low pass filter size of 4 and setting the number of desired clusters to 6. Blue arcs are the clustered arcs, red arcs are the mean arc, shaded red areas are the standard deviation.

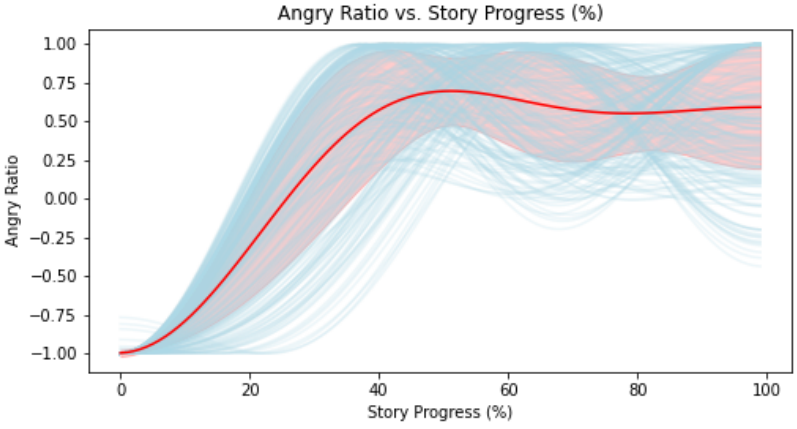
D.1 Angry

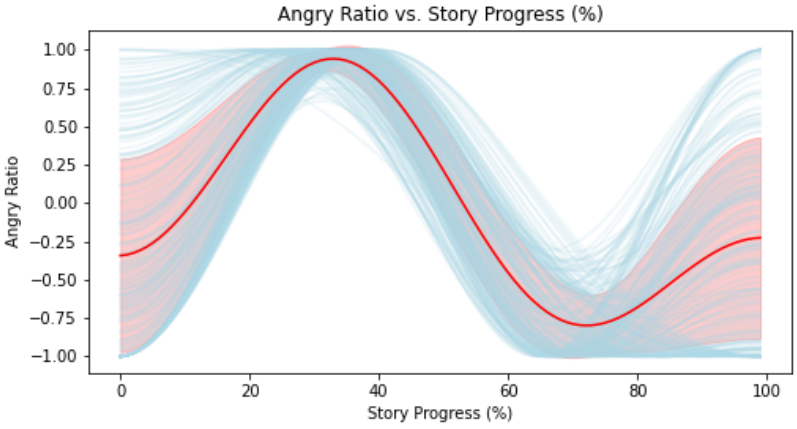

Angry Ratio vs. Story Progress (%)
Angry Ratio
Story Progress (%)


Angry Ratio vs. Story Progress (%)
Angry Ratio
Story Progress (%)

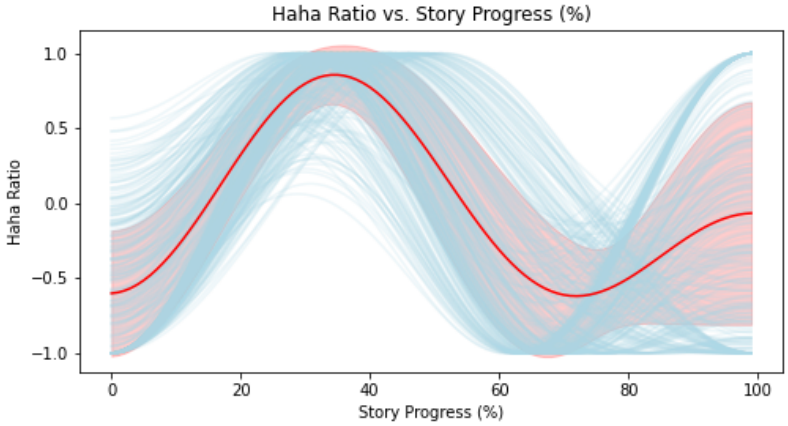

Angry Ratio vs. Story Progress (%)
Angry Ratio
Story Progress (%)

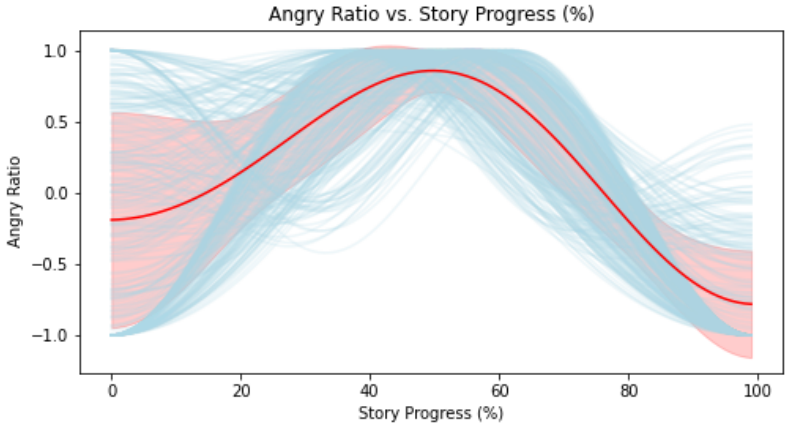

Angry Ratio vs. Story Progress (%)
Angry Ratio
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
80
100
Story Progress (%)

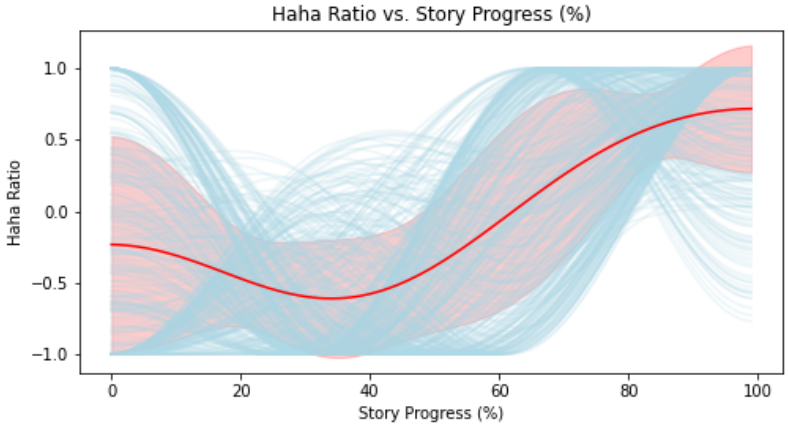

Angry Ratio vs. Story Progress (%)
Angry Ratio
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80
100
Story Progress (%)

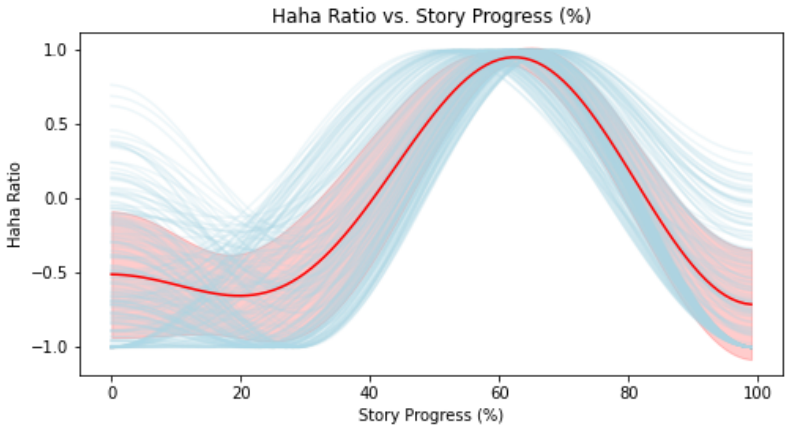

Angry Ratio vs. Story Progress (%)
Angry Ratio
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80
100
Story Progress (%)

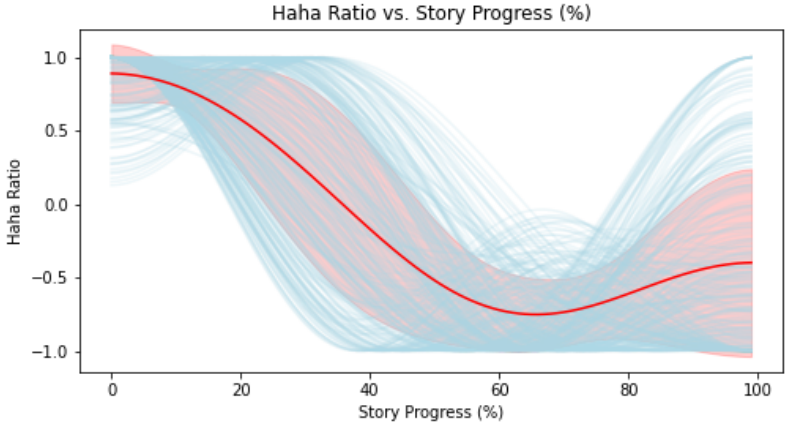
D.2 Haha

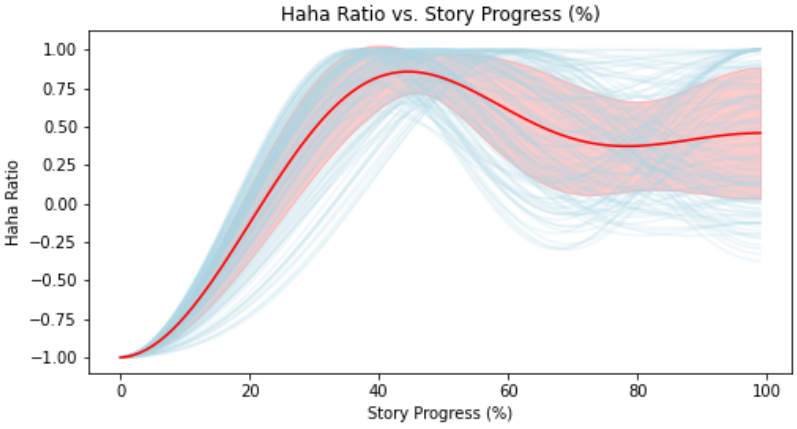

Haha Ratio vs. Story Progress (%)
Haha Ratio
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
80
100
Story Progress (%)


Angry Ratio vs. Story Progress (%)
Angry Ratio
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
80
100
Story Progress (%)

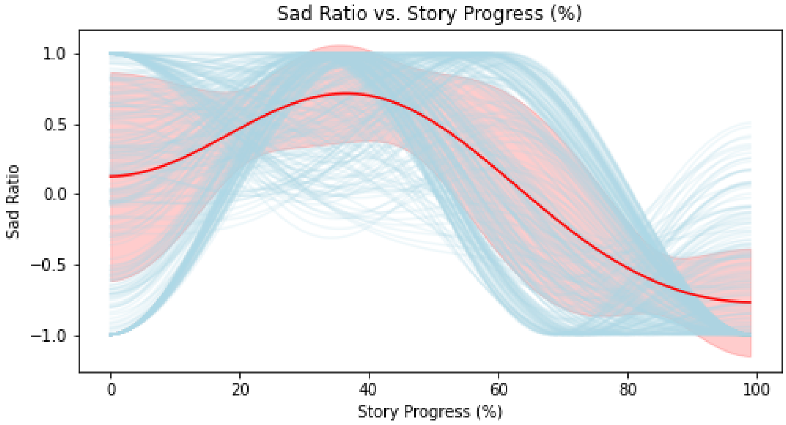

Haha Ratio vs. Story Progress (%)
Haha Ratio
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
80
100
Story Progress (%)


Haha Ratio vs. Story Progress (%)
Haha Ratio
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
80
100
Story Progress (%)

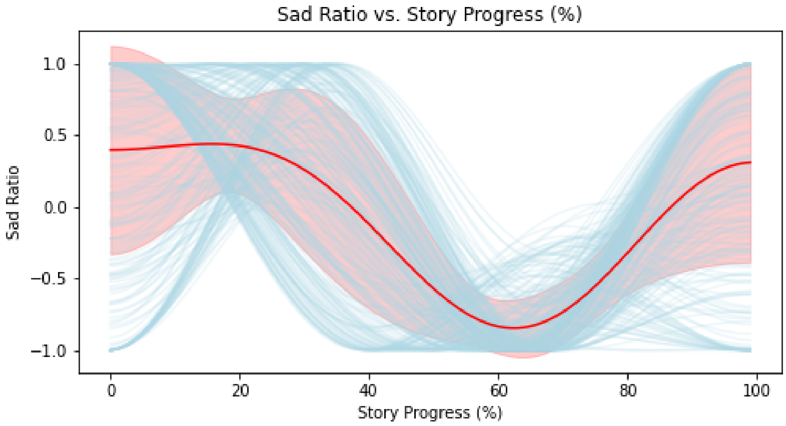

Haha Ratio vs. Story Progress (%)
Haha Ratio
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
80
100
Story Progress (%)


Haha Ratio vs. Story Progress (%)
Haha Ratio
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80
100
Story Progress (%)

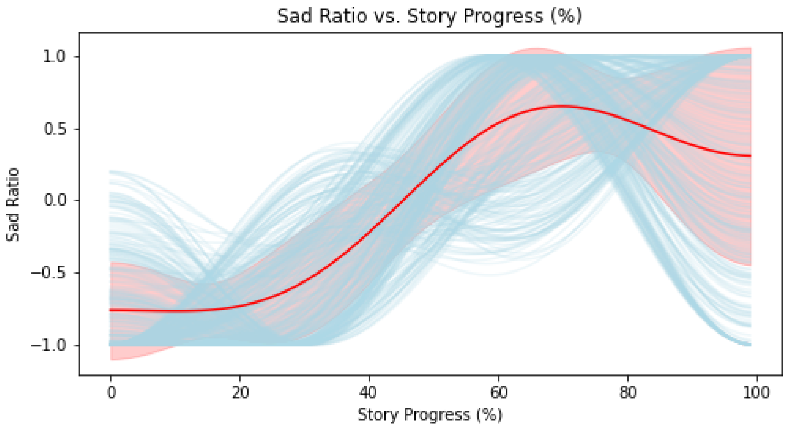
D.3 Sad



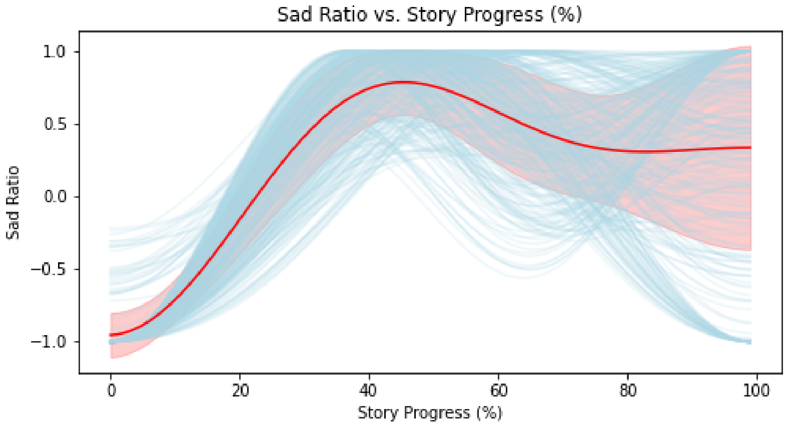
Sad Ratio vs. Story Progress (%)
1.0
0.5
0.0
−0.5
−1.0
Sad Ratio
0
20
40
60
80
100
Story Progress (%)

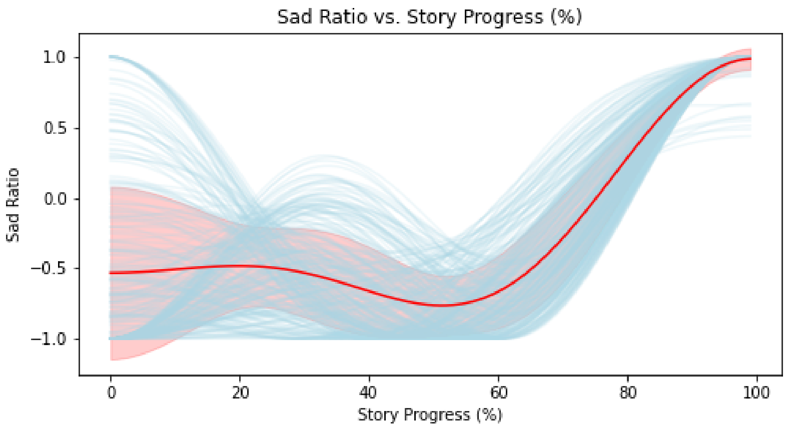


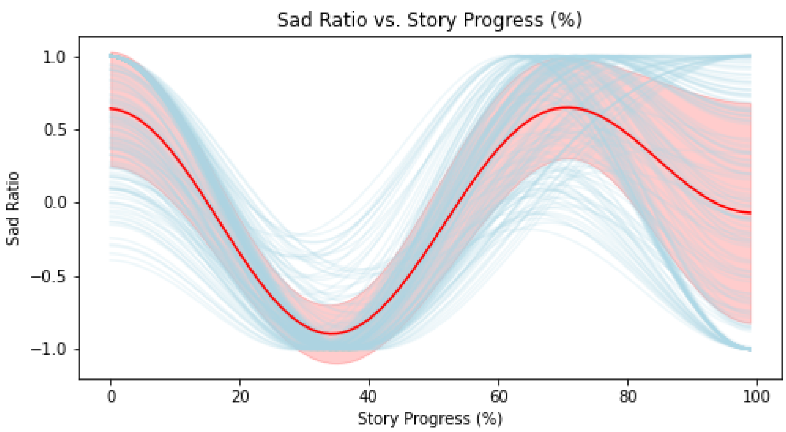
Sad Ratio vs. Story Progress (%)
1.0
0.5
0.0
−0.5
−1.0
Sad Ratio
0
20
40
60
80
100
Story Progress (%)



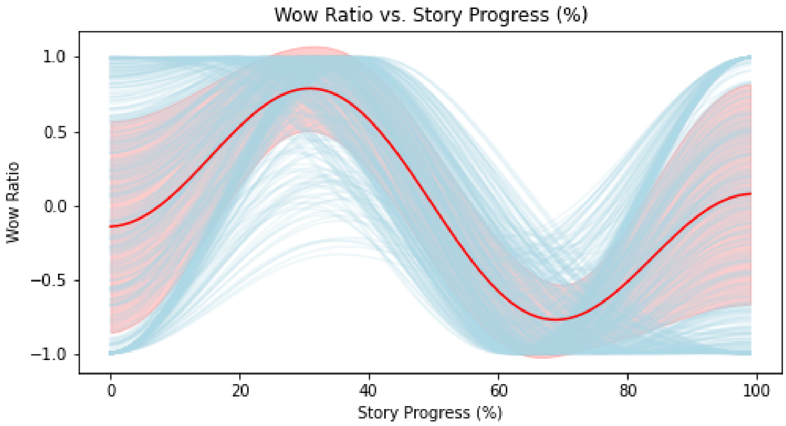
Sad Ratio vs. Story Progress (%)
1.0
0.5
0.0
−0.5
−1.0
Sad Ratio
0
20
40
60
80
100
Story Progress (%)


Sad Ratio vs. Story Progress (%)
1.0
0.5
0.0
−0.5
−1.0
Sad Ratio
0
20
40
60
80
100
Story Progress (%)

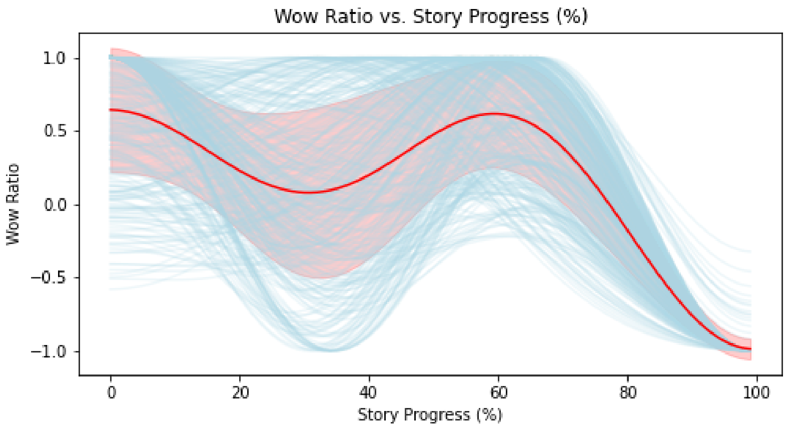

Sad Ratio vs. Story Progress (%)
1.0
0.5
0.0
−0.5
−1.0
Sad Ratio
0
20
40
60
80
100
Story Progress (%)


Sad Ratio vs. Story Progress (%)
1.0
0.5
0.0
−0.5
−1.0
Sad Ratio
0
20
40
60
80
100
Story Progress (%)

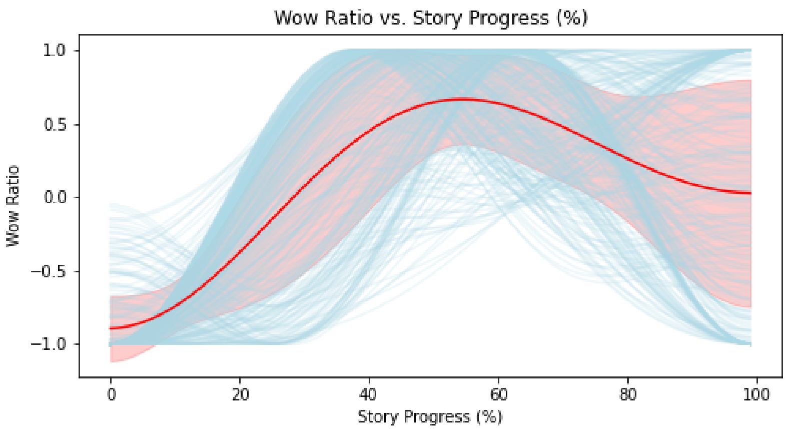
D.4 Sad



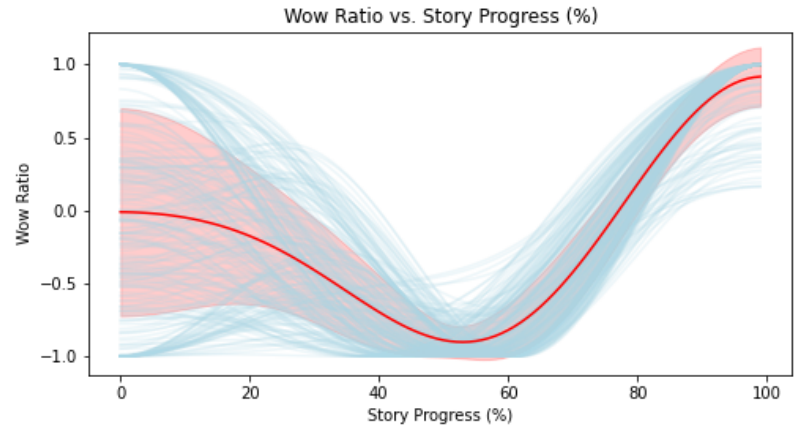
Wow Ratio vs. Story Progress (%)
Wow Ratio
Story Progress (%)
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
80
100

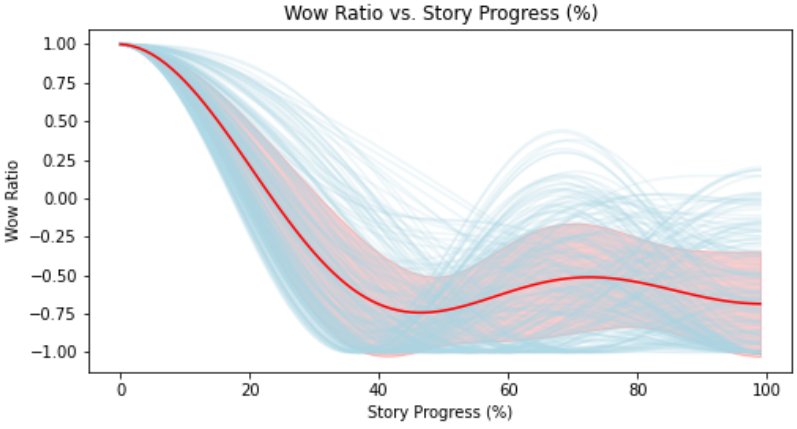


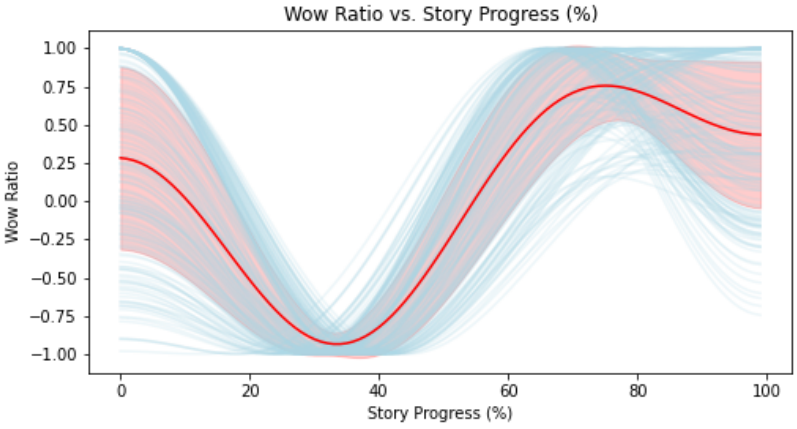
Wow Ratio vs. Story Progress (%)
Wow Ratio
Story Progress (%)
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
80
100



Wow Ratio vs. Story Progress (%)
Wow Ratio
Story Progress (%)
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
80
100


Wow Ratio vs. Story Progress (%)
Wow Ratio
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
80
100
Story Progress (%)


Wow Ratio vs. Story Progress (%)
Wow Ratio
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80
100
Story Progress (%)


Wow Ratio vs. Story Progress (%)
Wow Ratio
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80
100
Story Progress (%)

D.5 Love

As in Section 6.4.