UNIVERSITY OF WARWICK

Sentiment Analysis of Political News Articles

FINAL REPORT

Author
Matthew WIGHT

Supervisor
Sara KALVALA

ABSTRACT

The volume of news published daily has increased greatly with the advancement of information and technology. Despite the positive implications of this, it presents challenges for readers as extreme views and sentiments are more common in news. Therefore, classifying the sentiment conveyed by these news articles is an important, although challenging, task. We experiment with several approaches to extract opinions from text paragraphs, including a novel approach which makes use of an autoencoder neural network and a binary classifier. This approach detects sentiment in news articles at a document level at a rate of 95% to 98% on our novel datasets.

KEYWORDS

Opinion mining, Sentiment analysis, Data mining, Document-level sentiment analysis, Neural networks, Machine learning, News articles

April 29, 2023

Contents

Motivation

In an increasingly globalised society, mass media serves an important role in informing people around the world. News publishers are often a catalyst for political discussion and the opinions they convey are likely to be the first and most prominent about any given subject.

The rise of Web 2.0 has had a dramatic impact on the way we consume news, and our political landscape has changed to reflect that. Thanks to social media, it is easier than ever for professional and amateur journalists alike to broadcast their views to a large and highly interconnected audience. Unsurprisingly, this had led to a massive increase in the number of people and organisations publishing news articles, and the number of articles produced every day.

While this exponential rise in the volume of published news has some clear positive implications such as freedom of speech and diverse representation in media, it also presents a host of issues for readers. Social media lends itself especially well to the propagation of 'fake news' and extreme sentiment. Previous research has found that misinformation is often republished as news, whereas true statements of fact are not (Shin et al.,2018).

For most readers, detecting bias in news articles can be a difficult task, especially if a reader isn't familiar with a given publisher or author. While there are a plethora of text classification systems that have been used to detect emotions in news content, what is missing from this area is dynamic analysis of newly released articles at the document level.

This project has aimed to help users quickly gain insights into news article sentiment through software. The software that has been developed can be used to detect emotions in a given text or to analyse broader patterns such as the sentiment differences between different writers and publishers.

Background

Sentiment analysis is an area of computer science and data analytics that is already very popular for both research and business purposes. It has been used in combination with ideas from business and finance as well as sociological research. For example, the CyberEmotions project was a multi-million-pound research project funded by the European Commission that ran from 2009 to 2013, aiming to understand what it called 'collective emotions formation'. Essentially, this is the process by which online communities come to form shared feelings and ideas—a process which can be measured and analysed through the use of mathematical models and software. Cinelli et al. (2021) found that some social media platforms by their design led to the formation of echo chambers with a shared highly emotive outlook on given topics. This was achieved with the help of Media Bias/Fact Check, an independent organisation that labels the political leaning of a range of news publishers. The labels are decided subjectively by members of the organisation rather than through computational analysis of published materials.

Another application of sentiment analysis is automated polling to draw conclusions about public opinion on political matters. In a blog post published shortly before the 2010 UK general election, BBC technology correspondent Rory Cellan-Jones discussed the work done by sentiment analysis firm Lexalytics to monitor the discussion on social media during a party leaders' debate. It looked at social media activity to track sentiment towards the debaters. Furthermore, in 2012, political newspaper Politico partnered with Facebook to gain access to some user activity data, such as engagement statistics around various candidates in the US Republican primary elections. Facebook uses the automated tool *Linguistic Inquiry and Word Count* to identify and aggregate positive and negative emotions in text.

Gunter et al. (2014) identified several limitations of currently existing sentiment analysis techniques. One of these was the issue of deciding on a sentiment label for a given text: if a passage of text speaks with both positive and negative sentiment about a subject, how should it be labelled? This is especially common in longer pieces of written text, such as news articles. To validate the accuracy of a sentiment classifier it is common to have human readers evaluate the same texts, but a weakness of this is that human coders can provide inconsistent results.

Review of Existing Literature

There is already a body of work dedicated to analysing sentiment in news articles, mainly focused on financial news—with the aim of forecasting future financial events. These differ from political news articles mainly because the content aims to be solely factual or speculative rather than persuasive. Because of this, the techniques used to analyse both types of content are often slightly different.

Sentiment analysis can be carried out at a variety of different levels of scope in a text:

- **Word-level analysis**: considers the sentiment that can be inferred from a single word
- **Sentence-level analysis**: considers an entire sentence. Can take language features such as word order and negation into account
- **Aspect-level analysis**: splits sentences into phrases containing a single target, then computes a sentiment value for each
- **Document-level analysis**: calculates a sentiment value for an entire document.

The majority of existing literature in sentiment analysis focuses on shorter texts such as social media posts and online user reviews. These texts are generally easier to analyse because sentiment is less likely to change throughout the text, and the number of targets is likely to be fewer compared to a longer text. However, to gain meaningful insight into news article sentiment, it would be helpful to look at the text as a whole, either by processing larger portions of text or by aggregating sentence-level data.

Document-level sentiment analysis can be especially challenging because a single document will often contain many conflicting sentiments directed towards multiple targets. The number of words is much larger and more 'noise words' appear in each text, which can negatively affect accuracy. Often it can be difficult even for a human to label a document as either positive or negative.

At any level of sentiment analysis, the techniques used can be split into three main categories:

- **Lexicon-based**: using word-level analysis, construct a thesaurus by assigning a sentiment value or classification to each word in the vocabulary.
- **Machine learning-based**: first, label in advance each entry in a corpus with a sentiment score. Then, extract relevant features from the texts and use these features to fit a machine learning model to give predictions.
- **Hybrid approach**: these use the strengths of both methods, first taking into account linguistic features using the lexicon approach before applying machine learning methods.

Behdenna et al. (2018) found that the pure machine learning approach was the most dominant in previous studies. This is a supervised approach whereby learning is conducted on prelabelled data points. Therefore, one of the challenges of applying this method to news articles is finding an existing labelled dataset (or manually populating a new one).

Lexicon-Based Analysis Algorithms

Lexicon-based approaches are typically composed of two steps: first, a thesaurus is constructed, mapping each term to a sentiment score. The second step is calculating the scores for all the words in a document and aggregating them into a single sentiment score for the whole news article.

Building an emotional dictionary

Rao et al. (2013) proposed a method to build an *emotional dictionary* which would classify words into one of several categories (such as anger, amusement, sadness or surprise) using a training dataset of news articles. For each word w_j in the dictionary and each emotion class e_k, $k \in \{1, ..., E\}$, we can calculate a score θ_{jk} for the probability of the k-th emotion on the j-th word by:

$$\theta_{jk} = \frac{\sum_{i=1}^{N} \epsilon_i \sigma_{ij} r_{ik}}{\sum_{k=1}^{E} \sum_{i=1}^{N} \epsilon_i \sigma_{ij} r_{ik}}$$

where N is the number of documents, E is the number of emotion labels, ϵ_i is the probability of the i-th document, σ_{ij} is the probability of the j-th word in the i-th document, and r_{ik} is the normalised rating of the i-th document with respect to the k-th emotion label. Then by manually rating each document in the training set with respect to each emotion label, an emotional dictionary can be derived.

Once this dictionary has been produced, the process of predicting sentiment in a news article is very simple. The predicted probability $\hat{P}(e|d)$ of document d conveying emotion e is calculating by summing:

$$\hat{P}(e|d) = \sum_{w} P(w|d) P(e|w)$$

where $P(w|d)$ is the probability of word w appearing in d and $P(e|w)$ is the dictionary score of emotion e given w. An advantage of this approach is that the number of sentiment categories E can easily be reduced or expanded. However, other models have achieved better accuracy when classifying document sentiment.

SentiWordNet dictionary

Baccianella et al. (2010) introduced SentiWordNet 3.0, the most recent version of the SentiWordNet model. It is an automatically labelled instance of WordNet, a collection of synsets (sets of synonyms). The sentiment labels are generated by a semi-supervised learning method whereby a small 'seed' set of terms are manually labelled as positive or negative, then using a bag-of-words model, the rest of the words are labelled as positive or negative (or neutral) based on their perceived similarity to other words. Since the synsets all contain only synonyms, the labels can be assigned to the synsets rather than the actual terms, improving the computational efficiency of the algorithm.

Once the synset dictionary is constructed, a simple aggregation method can be used to evaluate whole documents (such as the summing formula outlined earlier).

Text Vectorisation Algorithms

Feature extraction is a core component of almost any machine learning architecture. A variety of text vectorisation algorithms have previously been devised in order to embed textual data into high-quality feature vectors that can be analysed by ML models.

N-grams

Broder et al. (1997) defined a method for measuring the resemblance of two text documents to each other by comparing contiguous subsequences in the documents (called 'shingles' or 'n-grams'—subsequences of size n). These are usually composed of either characters or words; in some cases, they are composed of more complex parts of speech. For example, the phrase *'the quick brown fox'* can be decomposed at word-level into three bigrams or two trigrams:

$$[\textit{the quick},\ \textit{quick brown},\ \textit{brown fox}]$$

$$[\textit{the quick brown},\ \textit{quick brown fox}]$$

Given a document D we define the set $S(D, n)$ as the set of all n-grams in D. Then, the resemblance r of documents A and B for a subsequence length n is given by:

$$r(A, B) = \frac{|S(A) \cup S(B)|}{|S(A) \cap S(B)|}$$

Similar documents can then be clustered together. N-gram models (called *bag of words* or *BOW* models) have been used to good effect in text classification (Pang & Lee, 2002). First, a set $\{f_1, ..., f_m\}$ of m predefined features are selected, where each feature is an n-gram that could appear in the text. Then each document is transformed into a vector

$$\vec{d} = (n_1(d), ..., n_m(d))$$

where $n_i(d)$ is the number of times feature f_i appears in document d.

After vectorisation, we can assign a class $c*$ to the input document d such that $c^* = argmax_c P(c|d)$. The Naive Bayes' classifier uses Bayes' rule,

$$P(c|d) = \frac{P(c)P(d|c)}{P(d)}$$

to compute this. We assume that the document's features are conditionally independent, allowing us to decompose the term $P(d|c)$:

$$P_{NB}(c|d) = \frac{P(c)(\prod_{i=1}^{m} P(f_i|c)^{n_i(d)})}{P(d)}$$

Naive Bayes' classifiers have performed well on text data in the past—for example, when classifying sentiment in movie reviews. In reality, however, the independency assumption almost never holds for real-life data (Lewis, 1998), and more sophisticated models are required to achieve truly accurate results on complex datasets.

As a whole, the n-gram approach tends to be efficient and robust (Bofang Li et al., 2016). The basic idea can also be expanded upon by assigning weights to more important words, making the classifier more sophisticated. However, because it ignores word order, some meaning is lost. Additionally, the traditional BOW representation takes each individual word as a unit, ignoring the actual semantic contents of the words—this tends to make the classifier poor at

generalising because the model isn't able to identify the complex patterns that convey sentiments in texts.

Another weakness of traditional n-gram representations of text is that syntax is ignored entirely. Sidorov (2019) describes the idea of syntactic n-grams, which are obtained through constructing and traversing the syntax trees of text documents. It found that these n-grams could be used to identify the author of a text with very high accuracy. However, previous syntactic processing is required to generate the syntactic n-grams for each document, increasing the overall processing time. For models trained on a relatively small amount of data, this limitation is not serious, but when looking to identify more complex relations over larger datasets, the overall processing time would be much longer.

Skip-grams

One of the biggest advancements in this area is the skip-gram model. Essentially, a skip-gram is an n-gram of nearly-contiguous items, where up to k items in the subsequence can be skipped. Given a word, skip-gram predicts words that are likely to appear around it. From our previous example, the word 'quick' could be represented as a skip-gram containing 'the', 'brown' and 'fox'. All the words in the dictionary can then be sorted in a predetermined order (e.g. alphabetically) and represented as a single vector of continuous numerical values, learned by a single-layer neural network. Using a softmax function, the vector scores can be converted to a probability distribution (Hagiwara, 2019).

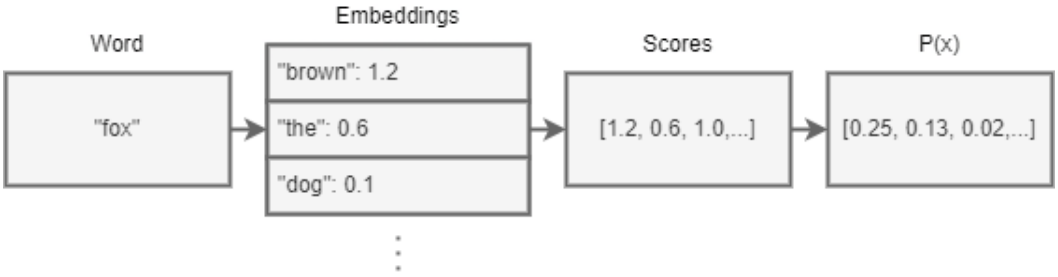

Figure 1: Example of the process of creating a skip-gram representation from a word.

Most implementations of the skip-gram model choose to omit words that appear very commonly in the training dataset (such as 'the') in order to improve the quality of the vectors.

When trained on a sufficient volume of high-quality data, word vectors made from skip-grams can carry a significant amount of meaning. Words with similar meanings and uses tend to have similar skip-gram vector representations, making them very useful for the analysis of semantic notions in text documents (Mikolov et al., 2013).

Word2Vec

Mikolov et al. (2013) described an efficient n-gram and skip-gram based model (called *Word2Vec*) by which words with a high level of syntactic or semantic similarity are effectively grouped together. Vectors generated by Word2Vec have been found to be of higher quality (that is, they achieve better accuracy on text classification problems in most cases due to conveying more meaning). Additionally, simple algebraic techniques can be performed on the vectors, illustrating multiple degrees of similarity: for example, $vector('king') - vector('man') + vector('woman')$ results in a vector similar to $vector('queen')$. This property is called additive compositionality.

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Figure 2: Examples of semantic and syntactic relationships (Mikolov et al., 2013).

The paper proposed two model architectures for computing the word vectors in this way, using either a continuous BOW (*CBOW*) approach or a continuous skip-gram approach. Where the skip-gram architecture transforms an input word into a vector representation of potential context words, CBOW works in the opposite direction, predicting the main word from the context words.

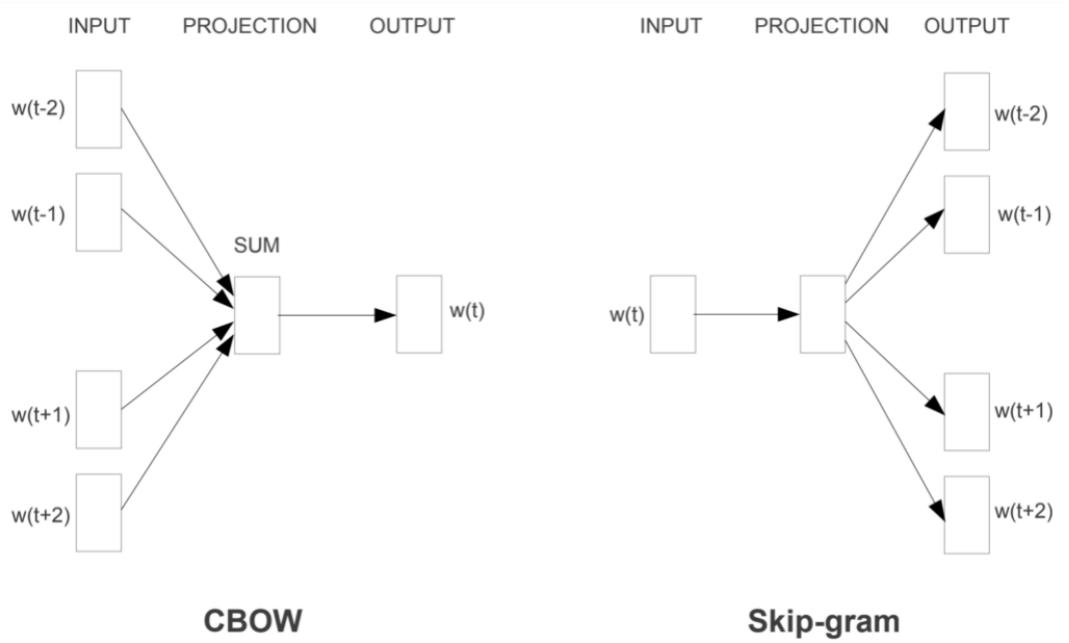

Figure 3: Comparison of predictions made by the two architectures (Riva, 2023).

It was found that the skip-gram architecture works better with smaller datasets and better represents less common words. In contrast, CBOW has faster training times and better represents

the most common words.

The aim of the Word2Vec model is to maximise vector similarity between words that appear in the same context and minimise similarity between words that appear in different contexts. However, calculating similarity between every pair of words in a vocabulary would be very computationally expensive. Instead, a subset of non-context ('negative') words are chosen at random. The aim of the model is to maximise the log probability of context word w_O given its input word w_I, where $log\, P(w_O|w_I)$ is given by:

$$log\, \sigma({v'_{w_O}}^T\, v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)}[log\, \sigma(-{v'_{w_i}}^T\, v_{w_I})]$$

where P is the sampling probability distribution (where less frequent words are more likely to be chosen) and σ is the sigmoid function $\sigma(x) = \frac{e^x}{e^x+1}$ (Karimi, 2023)(Lau & Baldwin, 2016).

Word2Vec has been shown to produce high-quality word embeddings. However, it is not known exactly why this is the case and no formal proof exists to explain its efficacy—the explanation given is only based on the intuition that similar words tend to appear in similar contexts (Goldberg & Levy, 2014).

Doc2Vec

Doc2Vec is an extension of Mikolov et al's architecture which is able to convert text documents of any length into a single vector. Each paragraph in the training dataset is randomly assigned a paragraph ID token vector, and each word in the paragraph is transformed using Word2Vec into a word vector. The paragraph token vector and word vectors are then combined (through averaging or concatenation) to produce a vector for the entire paragraph. This vector can be used by a conventional machine learning model to make predictions.

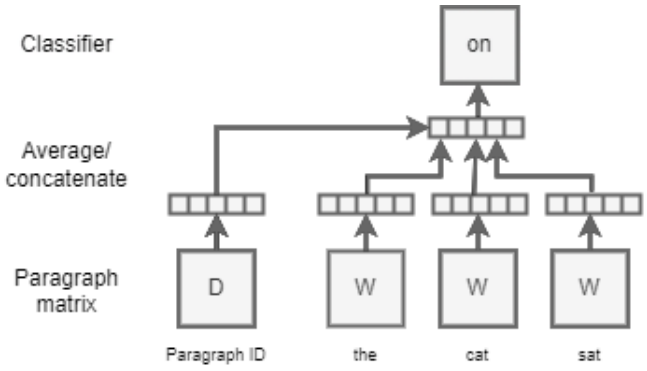

Figure 4: Illustration of Doc2Vec vectorisation model.

Doc2Vec paragraph vectorisation can be trained on unlabelled data and thus will work effectively even if there is not a large amount of labelled data.

There are two main advantages of using Doc2Vec for training paragraph vectors. Firstly, they inherit the semantics of the words which were learned by the Word2Vec word vectors. The second advantage is that they are able to take word order and context into account, in the same way that n-gram models do. Additionally, Doc2Vec vectors are typically much smaller than BOW n-gram vectors, despite usually being as or more effective in representing the text (Le & Mikolov, 2014).

Alternative BOW models

Kenter et al. (2016) proposed a bag of words model called Siamese CBOW. This model was developed explicitly for the purpose of being averaged, allowing for more effective representations of whole sentences. It is based on the Siamese neural network architecture, where two identical sub-networks with the same parameters are given different inputs to identify similarities between the inputs. Siamese CBOW was tested on 20 corpus datasets for sentence vectorisation and outperformed both the CBOW and skip-gram architectures for Word2Vec on 16 of them. However, Doc2Vec still proved to be more effective in the majority of cases.

TF-IDF

Another method of text vectorisation at the document level is TF-IDF (*text frequency, inverse document frequency*) (Li-Ping Jing et al., 2002). The result of TF-IDF vectorisation is a matrix of weights W, where $w_{i,j} \in W$ is the weight of term i in document j. We can calculate $w_{i,j}$ for any i, j by:

$$w_{i,j} = TF(i,j) \times IDF(i) = log\left(\frac{N}{DF(i)}\right)$$

where $TF(i,j)$ is text frequency (the number of times term i appears in document j), $DF(i)$ is document frequency (the number of times term i appears in any document in the dataset) and N is the number of documents in the dataset. $IDF(i)$ is the inverse document frequency of term i. TF-IDF is simpler than Doc2Vec but has been shown to sometimes perform better when used to train machine learning models (Singh, 2021).

Keras API text vectors

Keras is a deep learning API for Python. It features a variety of powerful deep learning tools, including a text vectoriser. The vectoriser is implemented in the form of a layer which can be integrated directly into a neural network. It has a built-in text standardisation process and also supports custom text preprocessing implementation. This layer conducts all the necessary steps to convert raw texts into text vectors: first, preprocessing, tokenisation and detokenisation steps are applied. Then, an indexed vocabulary is generated from the training dataset. From this, the processed texts can be transformed into vectors, either by a simple n-gram technique or by TF-IDF (Frassetto, 2020).

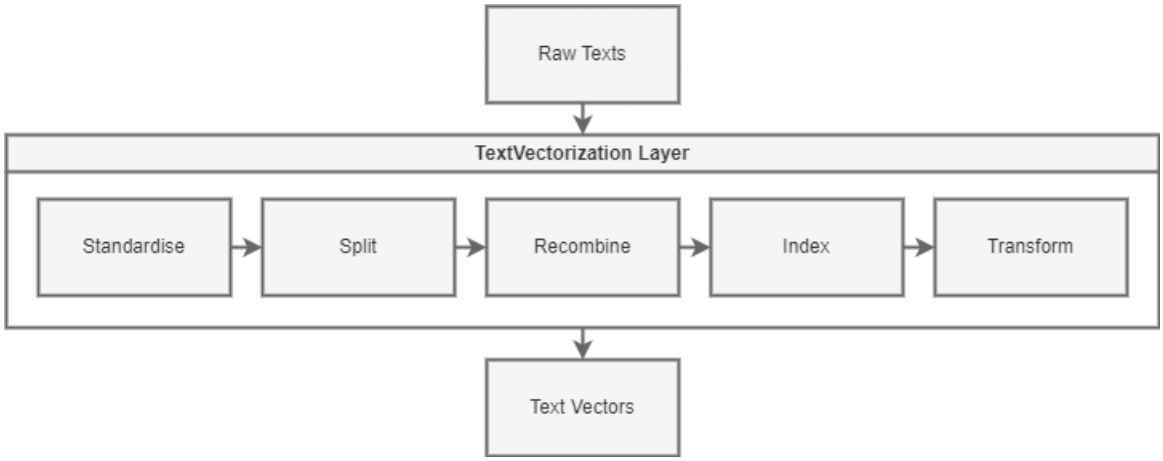

Figure 5: Basic overview of the flow of Keras' TextVectorization layer.

The Keras TextVectorization layer is powerful because it is well-integrated with the rest of the Keras API: the outputs of this layer can be plugged directly into other neural network layers in Keras (such as a linear network layer which can be used to make predictions).

Aspect-Based Sentiment Analysis

Aspect-based sentiment analysis (ABSA) can refer to two different approaches. The first, more often known as target-level analysis, finds the overall sentiment associated with an entity. The second approach deals with different aspects associated with an entity (such as quality or price of a product).

Zitnik et al. (2022) presented *SentiCoref 1.0*, an annotated corpus of named entities and their coreferences found in a document, along with a target-based sentiment label for each entity. The named entities were identified automatically from a set of 837 news articles and annotated manually according to perceived sentiment in the context of the article. The results of the aspect-level analysis were then applied to document-level by aggregating the sentiment scores for all the entities in an article. The power of first decomposing the document into entities is that we can control how significantly different targeted statements affect the overall sentiment of the document. For example, Zitnik et al. considered giving extra weight to the sentiment labels of the 'most important' named entities in a document—those which were mentioned the most frequently. This method saw an increase in accuracy over another approach which weighted all entities equally.

ABSA is a popular approach to sentiment analysis, especially in news articles, because it allows for a fine-grained level of understanding to be extracted from the texts. However, it has some weaknesses. By looking at aspects within a text document, the number of outputs from an ABSA model is much larger compared to other sentiment classifiers (since each aspect is associated with a sentiment class). ABSA also tends to be more reliant on supervision and manual labelling of sentiment in texts in order to train machine learning models (Zitnik et al., 2022). For this reason, it would be interesting to compare the efficacy of a document-level or sentence-level classifier with a more fine-grained classifier.

Topic-aware sentiment analysis

A somewhat related idea is topic-aware sentiment analysis. This involves separating training data by topic (in the case of news articles, topics could include local news or sports). Akhmetov et al. (2022) experimented with this idea and found that in topics with a large number of samples, the topic-aware model performed slightly better than an equivalent model without topic-awareness. However, the performance improvements over the entire corpus were negligible and the topic-aware model performed worse on topics with few samples.

For the purposes of this project, all political articles could be considered to already be grouped under a single topic. However, based on the previous research, splitting the training data into smaller subsets could be a means of improving performance.

Using Recurrent Neural Networks to Analyse Text

While there is a lack of research on document-level sentiment analysis of news articles using neural networks, the same techniques have been used to analyse different types of texts. Li & Qian (2016) used recurrent neural networks to analyse sentiment in movie reviews. Whereas the information in a traditional feedforward network only travels forward, RNNs contain loops, functionally giving the network a 'memory' where the outputs are dependent on previous inputs. RNNs are designed to model time series, but they are also especially effective on sequential data such as text—the ordering of words in a sentence can clearly have an impact on its overall meaning.

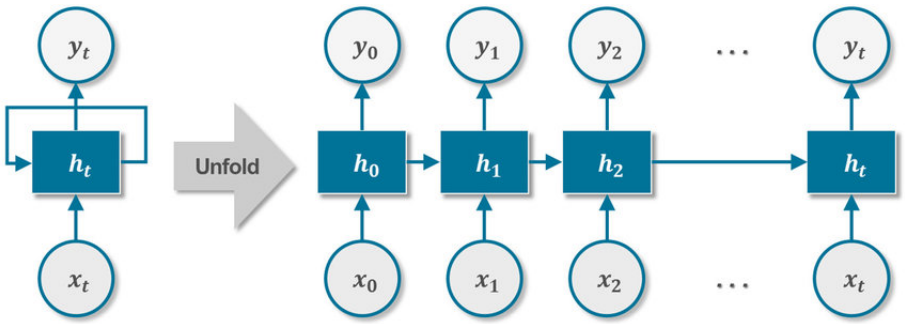

Figure 6: The basic architecture of a recurrent neural network (Son & Kim, 2020).

In practice, RNNs do not always learn dependence information in text, especially when dependent features are far apart in the text. This is due to the vanishing gradient problem, where weight parameters converge to zero, preventing learning from occurring. For longer texts, long short-term memory (LSTM) units are usually much more effective (Graves, 2008). LSTMs feature input, output and forget gates inside each neuron, allowing the network to selectively remember or forget different parts of the input sequence.

The neural network can be trained using labelled training data: a set of vectorised text documents, each labelled with a 'positive' or 'negative' sentiment label. The weights and biases of network are adjusted according to the loss function and optimiser specified during training until the model is able to accurately make predictions about the sentiment of unseen texts.

Bidirectional LSTM network

An extension of LSTM is bidirectional LSTM (biLSTM). A biLSTM model consists of two LSTM networks working in opposite directions and the outputs of the two networks are combined into a single output. This architecture allows context to be captured from both ends of the sequence (Cornegruta et al., 2016). Essentially, the amount of potentially relevant information available to the network is increased.

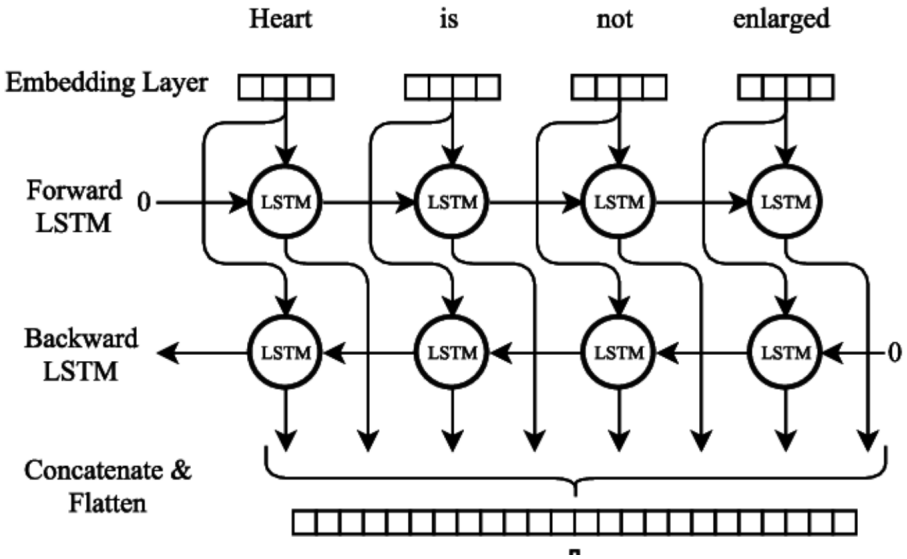

Figure 7: Architecture of a bidirectional LSTM neural network (Cornegruta et al., 2016).

Rhanoui et al. (2019) proposed adding a convolutional layer to the biLSTM model (to form a CNN-biLSTM). A convolutional neural network (CNN) is a type of neural network architecture that is able to detect information in different locations with very high accuracy. While CNNs are typically used for processing image data, they can also be used to analyse sequential and text data. A convolutional network layer can be used to extract features, facilitating better learning in later parts of the model. Rhanoui's proposed model saw a modest increase in its accuracy of document-level sentiment classification when a convolutional layer was added to the biLSTM network.

Autoencoders for Dimensionality Reduction

An autoencoder is a type of neural network composed of two parts: an encoder and a decoder. Between the two components is a bottleneck 'latent' layer. It is designed to construct efficient low-dimensional representations of a high-dimensional input (encoding), then reconstruct the original input from the latent representation (decoding).

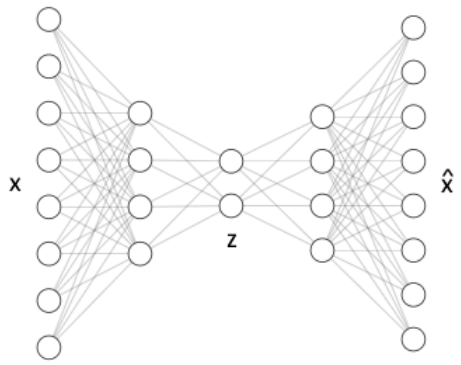

Figure 8: Structure of an autoencoder neural network, with input x, reconstruction $\hat{x}$ and latent layer z.

Autoencoders have a large number of applications, such as compression and denoising (Roy, 2020). In text analysis, they are often used as a pre-processing step to reduce the dimensionality of the input data. They are often more effective at capturing important correlations than other methods due to their ability to learn. As a result, classifiers that use autoencoders are often able to achieve a very high level of accuracy (Liang et al., 2017)

Autoencoders have previously been used for sentiment analysis. Zhai & Zhang (2016) first trained a linear classifier on labelled data, then used an autoencoder to tackle the issue of analysing high-dimensional data (the data has many dimensions due to the very large number of words in the vocabulary of the texts). Compared to other methods of dimensionality reduction (such as principal component analysis, which chooses the subset of features with the highest variance), autoencoder representations typically have more expressive power because they can be non-linear.

One weakness of autoencoders used for dimensionality reduction is that the set of features they learn are often similar to each other. In the case of text analysis, this usually corresponds to a small set of frequently used words. Chen & Zaki (2017) hypothesised that this is caused by the autoencoder greedily learning relatively trivial features during the training process. To overcome this, they designed *KATE*, an autoencoder model with an added hidden layer that would force neurons to compete to respond to an input. It was found to outperform the non-competitive model as well some other models when performing classification tasks.

Project Management and Development Methodology

Methodology

At the beginning of this project, a variety of development methodologies were considered. I felt it was important to have a clear organisation style and development philosophy outlined before the start of development in order to maximise the quality and consistency of the code.

The main development methodology used was an agile approach, adapted to suit the needs of this project. Constant communication, adaptability to changing requirements and consistent delivery are all very achievable because the entire process is organised by a single developer. Development processes and environments were adjusted to emphasise individual strengths: for example, a large majority of the system was written in a custom environment of Anaconda, a distribution for the Python programming language. This served to make the development environment more consistent throughout the whole project, reducing the amount of research that needed to be conducted into different libraries, tools and platforms.

Due to the strict deadlines and the need for a clear specification to be set from the beginning, a broad timeline was made. Estimating the amount of work required is a very difficult task, especially for inexperienced developers. As a result, some tasks took longer to complete than initially estimated—this was anticipated during planning, so some slack time was incorporated into the schedule. The 'backend implementation' section of the project in particular took longer than expected, and a large portion of the slack was used on this. Conversely, the implementation of the data visualisation features was completed faster than expected, which left some additional development time for other parts of the project. For example, the efficacy of the autoencoder model was vastly improved thanks to adjustments made very close to the end of the project's development, using time that had initially been allotted to data visualisation.

Project Management Tools

Because the development team consists of just a single developer, it was important to ensure that the workload would be manageable. A Kanban board was set up using Trello in order to monitor the completion of tasks and optimise workflow. The use of Kanban also helped to implement the agile methodology that was chosen at the start of the project.

The flow of development was further optimised using GitHub for version control. GitHub was helpful in project organisation and tracking changes made to the project throughout development thanks to its smart code repositories. To make the system more portable, Docker was used to containerise the components of the system.

Architectural and Design Patterns

Initially, a pipeline architectural pattern was considered:

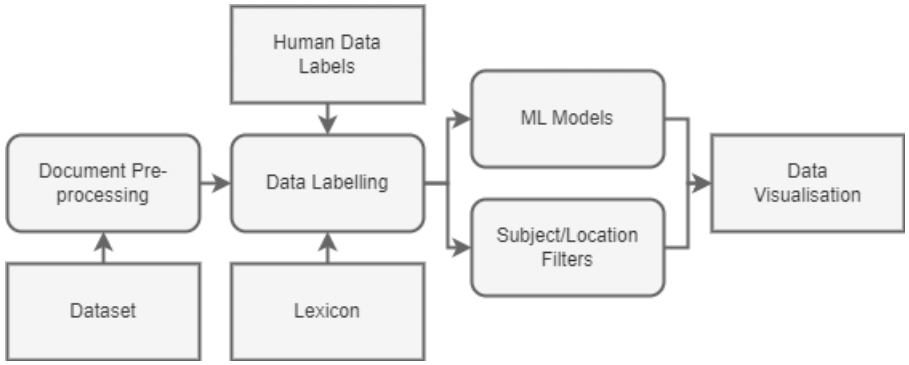

Figure 9: Overview of the system architecture using a pipeline architecture.

A pipeline architecture was appealing due to it being simple to understand and construct. However, it would make the system less capable of interfacing with user input (for example, the pipeline would not be ideal for predicting sentiment for a single input document). Additionally, with the use of multiple different data sources, the strict regularisation standards required for the different components and datasets would present further issues.

Instead, a repository architectural pattern was used. This is a sensible option for this project because it centres on a large store of data that can be used by many different components. In this case, the data store is the collection of training and testing data given to the machine learning models (as well as the results of the models' predictions). Each subsystem is independent of the others, interacting only with the dataset.

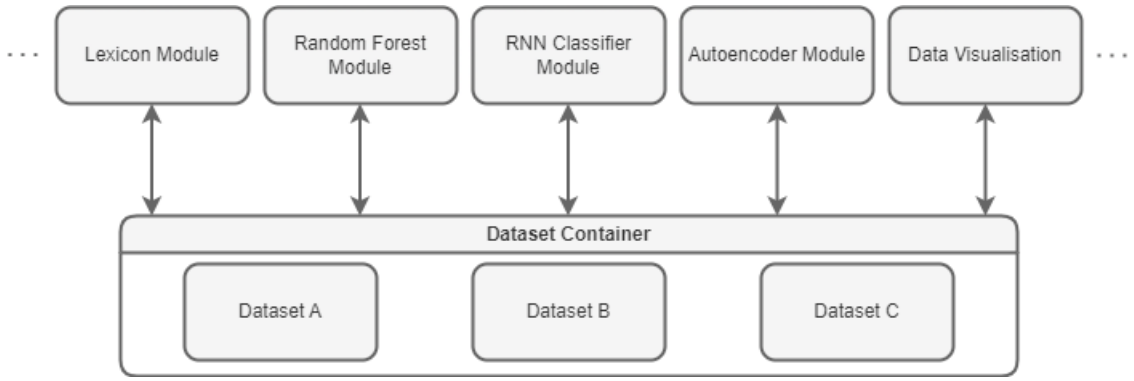

Figure 10: Overview of the system architecture using a repository architecture.

This is an efficient means of sharing large amounts of data, which is ideal for this project. Changes made by one component of the system are easily propagated to the rest— so for example, changes to the sentiment predictions or data sources can be seen immediately in the data visualisation component.

One of the main weaknesses of this design is having to enforce standards in the data, making evolution more difficult. This is not a significant concern for this project because it is not intended to evolve after its completion.

Consideration of Legal and Ethical Issues

All of the data used for this project is extracted from publicly available news articles. No personal data is processed, so ethical consent is not required.

Nevertheless, it is important to understand the ethics surrounding sentiment analysis and the potential impacts of research in the area. Lang (1984) found that statements about public opinion made in media can drive sentiment as well as reflect it (and this is likely to also apply to opinions found in news publications). Computational techniques for sentiment polling are largely experimental and results vary significantly across different methods (Anstead & O'loughlin, 2012). Therefore, there should be methodological transparency when presenting results obtained through such means.

Overall, both the scientists carrying out sentiment analysis and any media publishers that report the results have a responsibility to explain the methodology and limitations of the process.

There are also some legal issues pertaining to this project that must be considered; the main being the terms of use of the websites of some online newspapers. For example, the New York Times' terms of service page states that using automated tools to data mine or scrape content on the website is prohibited and may potentially lead to criminal penalties. Therefore, it is important to ensure that the use of any data taken from newspapers in this project is allowed by the publisher.

Design of Algorithms and Models

This section will outline the main techniques tested in the development of the solution, with a justification of the choices made.

Natural Language Processing

To gain insight into the meaning of a text document, an algorithm needs to be designed with the rules, conventions and irregularities of the language in mind. These differ for every spoken language (but this paper focuses only on modern English as it is used by English-speaking news publishers).

Several simple processes can be applied to a document to reduce noise and make analysis easier. Firstly, punctuation and white spaces are removed because these language features largely do not affect sentiment. Then, the document is tokenised: it is converted from a contiguous block of words separated by spaces into an ordered list of word tokens. Some tokenisation methods will also label each token with a part-of-speech tag (such as 'verb' or 'proper noun').

Superfluous words can be removed from a document to further distil the text into a form that is more dense with meaning. These words (mainly articles and pronouns such as 'the' or 'you') are called *stop words*. There is no universal list of words considered to be stop words, but a range of corpora are available for programmers to use.

Most language processing algorithms will be more successful if the size of the vocabulary across the dataset is smaller. After removing stop words, we can consider other means to remove unnecessary details from a text. Stemming is a technique by which inflected forms of a word are reduced to their 'stem'. The different forms of the same word all have nearly the same meaning, so the size of the vocabulary is reduced without significant loss of information.

Lemmatisation is an alternative to stemming; it is more thorough in that it takes the context of a word into account. As a result, lemmatisation is usually more computationally expensive.

In order to pre-process the news articles in preparation for analysis, each paragraph is tokenised, lemmatised and detokenised. Accents, non-alphabetic characters and stop words are removed.

```
CHARFILTER = re.compile('[a-zA-Z]+')
STOPWORDS = nltk.corpus.stopwords.words('English')

def textPrep(text): # preprocess text ignoring case
    return textPrepCaseSensitive(text.lower())

def textPrepCaseSensitive(text): # preprocess text (case sensitive)
    # strip accents from text
    text = unidecode(unidecode(text, "utf-8"))
    # tokenise the text into word tokens
    tokens = nltk.word_tokenize(text)
    # remove stop words
    tokens = [t for t in tokens if t not in STOPWORDS]
    # remove any tokens with invalid characters
    tokens = [t for t in tokens if CHARFILTER.match(t)]
    # join tokens back into text
    text = TreebankWordDetokenizer().detokenize(tokens)
    return text
```

We can test how these steps work by feeding some article snippets into the function:

```
input:
The healthcare system is groaning under demand.
output:
healthcare system groaning demand

input:
DeSantis reacted to Trump's indictment by stating that he would not extradite
    him from Florida to New York, which nobody had asked him to do.
output:
desantis reacted trump indictment stating would extradite florida new york
    nobody asked
```

The aim of the pre-processing function is to denoise simplify the input data, making it easier to analyse. We can see from the examples that there is not much loss of sentiment information, but superfluous details are removed.

Feature Extraction and Selection

- **Feature extraction**: deriving from raw data a set of informative values suitable for modelling. For example, latitude and longitude can be extracted from geospatial data. In text analysis, this usually involves quantifying vocabulary, word frequency and sentence structure.
- **Feature selection**: selecting a subset of the derived features to be used for machine learning. Feature selection aims to simplify the model by reducing the dimensionality of the data.

In analysis of text, feature extraction is generally achieved through converting texts into vectors (*text vectorisation*—the process of converting words into numbers). The way in which the text is vectorised can have a significant impact on the efficacy of machine learning models using the vectors as training data. High-quality word vectors are generally characterised by having words with similar meanings grouped closely together.

The Word2Vec model is trained using a neural network language model (NNLM). The network essentially predicts a word in a text based on the words earlier and later in the sequence. Vectors generated using Word2Vec are generally of high quality and produce better results than similar models such as n-grams (Mikolov et al., 2013).

Doc2Vec is an extension of Word2Vec that is able to vectorise sentences, paragraphs and documents of variable length(Lau & Baldwin, 2016). Like Word2Vec, it creates vector representations of every word in a corpus. After this step, it vectorises documents by concatenation of the word vectors (Le & Mikolov, 2014). Essentially, the vectorised document is a matrix consisting of the word vectors of the words in the document, as well as a vector referring to the document itself. Doc2Vec has been found to perform better on document-level analysis tasks than an aggregated Word2Vec approach.

Document-level analysis was chosen as the primary scope of analysis for this project because it allows for insights on a broader level. This will help readers to identify differences in the sentiment expressed by different authors and publishers, looking at broad trends rather than a single expression of sentiment.

TF-IDF can also be used to generate vector representations of text documents. It has previously achieved a high level of accuracy in some text classification tasks, especially when appropriate preprocessing measures are utilised (Li-Ping Jing et al., 2002). There is a clear connection between the vocabulary used in a section of text and the sentiment it expresses (most sentences containing the word *'good'* likely express at least a degree of positive sentiment), so it is not surprising that that text sentiment can be estimated by analysing just the words used—even with no consideration of syntax.

For the purposes of this project, Doc2Vec is likely to be very effective and relatively easy to implement using the Gensim Python library. Large Word2Vec and Doc2Vec architectures typically take days to train unless large scale parallel training is used (Mikolov et al., 2013), so using one of Gensim's pre-trained architectures vastly reduces computational overhead and development time for the system. TF-IDF vectorisation can be implemented from scratch or using Scikit-learn's TF-IDF vectoriser. Each method has different strengths and applications, so both will be tested when training the sentiment classifier machine learning models.

Doc2Vec	TF-IDF
Longer training time	No training time
Groups similar words	Assigns weights to words in text
Considers logical order of words	Considers frequency of words
More compact representation	High dimension representation

Figure 11: Comparison of characteristics of Doc2Vec and TF-IDF.

Scikit-learn TF-IDF

The Scikit-learn Python library contains a TF-IDF vectoriser object built specifically for the purpose of vectorising text documents. The vectoriser can be fitted with training data, then used to transform the documents into TF-IDF representations. A pre-processing function, such as the one described in the previous section, can be given to the model in order to improve the quality of the text vectors.

```
# load some training data (raw text from news articles)
df = pd.read_csv('dataset/guardian.csv')
docs = df['raw']
# define TF-IDF vectoriser
v = TfidfVectorizer(preprocessor=lexmodels.textPrep)
tfidf = v.fit_transform(docs)
```

Once the vectoriser model has been fitted, we can apply TF-IDF to some new text data taken from a news article:

```
text:
Young people, like me, may be economically Liberal but we're overwhelmingly
    socially centrist, if not progressive.
TF-IDF vector:
  (0, 3554)     0.3552006205673242
  (0, 2445)     0.4032025726387211
  (0, 2284)     0.21459538125199684
  (0, 2222)     0.44959579196697275
  (0, 1946)     0.27755526018695276
  (0, 1831)     0.2075557710122551
  (0, 1820)     0.3752598487243967
  (0, 472)      0.44959579196697275
```

Each entry in the vector corresponds to a word in the input text. The value on the right is the TF-IDF value calculated for that word. There are fewer items in the vector than words in the input text because stop words have been removed.

Word2Vec Implementation

Python library Gensim features an implementation of the Word2Vec and Doc2Vec models, along with a list of pre-trained models. The quality of the vector representations can be tested using some of the built-in features that come with it, such as the 'most-similar-to' functionality. The similarity measure used is cosine similarity: $S_C(A, B) = \frac{A \cdot B}{||A||\,||B||}$.

```
Most Similar words to great:
    ('terrific', 0.7989331483840942)
    ('fantastic', 0.7935212254524231)
    ('tremendous', 0.7748856544494629)
    ('wonderful', 0.7647868990898132)
    ('good', 0.7291510105133057)
    ...
```

The vectors can also be plotted on a graph to compare the distances between related and unrelated terms. In the figure below, similar words are clearly grouped together and words with no clear semantic links have larger distances between them.

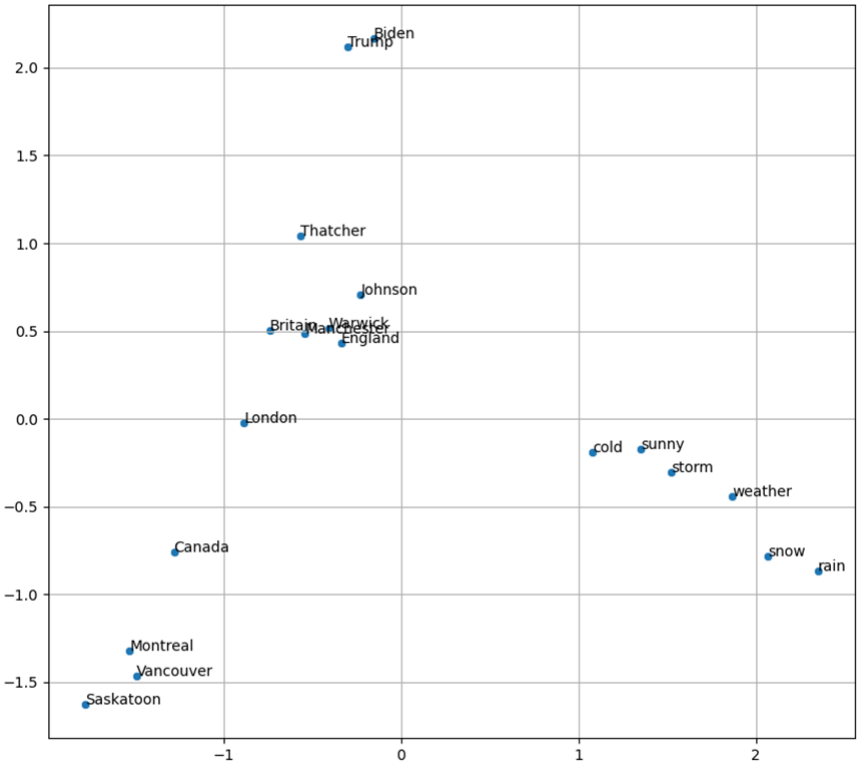

Figure 12: A 2D visualisation of a selection of word vector representations.

The most accurate Word2Vec models take over a day to train (Mikolov et al., 2013). As a result, using a pre-trained Word2Vec model is much more computationally efficient. Some of the existing models were trained on Google News data, which is likely to be representative of a variety of news publishers and suitable for use in this project.

Data Handling and Acquisition

Before supervised learning can begin, a labelled dataset of text documents is required. This section will review the suitability of existing datasets, then consider methods for constructing datasets with new data.

Existing datasets

Bhutani et al. (2021) utilised sentiment analysis to detect fake news articles on a dataset created by Shu et al. (2018). This dataset is not suitable for the purposes of this project because the labels are a measure of the factual correctness of the texts (which is outside the scope of the project). However, some of the preprocessing, learning and analysis techniques could be applicable.

NewsMTSC (Hamborg & Donnay, 2021) is a dataset for aspect-level analysis of content in news articles. Each entry contains a selected sentence from a political news article, along with a list of targeted sentiment labels. While this source could be used to train a machine learning model, the model wouldn't be effective at analysing entire documents (especially because only optimal sentences are selected from the source articles). Aspect-level analysis has previously proven an effective means of deriving meaningful results from a small amount of text, but this project will mainly focus on identifying broader patterns in larger and noisier items of text. As a result, this dataset was not chosen.

SEN (Sentiment Analysis of Entities in News headlines) (Baraniak & Sydow, 2021) is another human-labelled dataset for aspect-level sentiment analysis. It differs from NewsMTSC because each of its entries features just a single target—so the data labels would also be suitable for sentence-level analysis. Although this project aims to conduct document-level analysis on news articles, the SEN data could be of some value for training and validation purposes. Another issue with the use of this dataset is that it consists of news headlines rather than sentences taken from article bodies. Headlines are often stylistically different from the rest of an article, so a model trained on headlines could struggle when presented with full news articles.

headline	entity	majority_label
Russia and Poland Feud Over Putin Remarks on World War II	Putin	neg
Trump Moves to Lift Visa Restrictions on Polish Citizens Who Visit U.S.	Trump	neutr
Trump-Backed U.S.-British Trade Deal Faces Hurdles	Trump	neutr
Trump Praises Lame-Duck U.K. Leader Theresa May	Trump	neutr

Figure 13: A sample of the SEN dataset (Baraniak & Sydow, 2021).

While this dataset is not perfectly suited to the task, it was important to include a fully human-labelled dataset to ensure that our model is capable of identifying actual sentiment and not just features of the lexicon-based labelling algorithm.

Populating datasets with web scraping

Due to the lack of suitable datasets, a novel dataset was required for this project. The first step of assembling the dataset is obtaining the raw text documents. Sinclair (2005) and Pustejovsky & Stubbs (2012) set out a list of guidelines for the creation of a corpus. Among these are:

- Language samples should consist of entire documents where possible.
- Any information about a text other than the actual strings should be stored separately from the raw text.
- The corpus should aim to be homogeneous and to avoid rogue texts.
- The corpus should ideally be representative and balanced.

After considering these criteria, I decided to use data from the Opinion section of The Guardian UK Online site as a starting point for dataset construction. This source is ideal for several reasons: firstly, the HTML formatting of articles is consistent, with article contents split into paragraphs of roughly equal size (making the resulting data more homogeneous), and external information such as author name stored apart from the body of the article. The URL of each article on the website is easy to obtain through web scraping; this is not true for some other publishers due to irregular webpage formatting. Furthermore, articles are archived well and enough articles can be scraped to make the dataset representative of the source.

In order to scrape the text data from the Guardian website, the site's format must be examined. All of The Guardian's editorial and columnist opinion articles can be accessed from the Opinion section of `theguardian.com`. The articles are sorted by date of publishing and grouped into pages, each page containing approximately 25 articles. The URL of a large number of articles is obtained by scraping these pages, then the raw text documents are stored in a CSV file. There are well over 100 thousand articles available to read in this section—more than enough to create a sizeable training dataset.

Not all of the articles in the Opinion section are relevant or usable—for example, the text classifier is not able to parse picture data, so opinion-based cartoons must be discarded. Furthermore, many opinion articles on the site discuss issues not related to politics. The inclusion of non-political articles in the training dataset would likely reduce the accuracy of the sentiment prediction model on political articles, so these should be discarded too. Non-political articles are detected by scanning each article's headline for one of a list of key terms (references to people, organisations and ideas present in political discussion, such as 'Labour', 'Starmer' or 'health service').

A negative consequence of using this method of obtaining raw texts from The Guardian is that articles written for the US and Australia editions of The Guardian are stored on this webpage, not just UK articles. While this does not clearly affect the quality of the training data, it could lead to issues when training the text classifier because the range of vocabulary is likely to be larger. Essentially, the training dataset could be more complex and more difficult to analyse. Despite this, some of the models showcased later in this report were still able to achieve very accurate results when trained on this dataset.

Beautiful Soup, a Python library for extracting HTML data, is used to obtain the raw text from the source. The code for this is shown below:

```
def scrapeArticle(url):
    # request article url and parse html
    res = requests.get(url)
    soup = BeautifulSoup(res.content, 'html.parser')
    # find all the raw text article contents
    body = soup.find_all('p',class_='dcr-94xsh')
    pars = [p.text for p in body]
    # find the author name
    authors = soup.find_all(rel='author')
    if authors == []:
        authors = ['N/A' for _ in body]
    else:
        authors = [authors[0].text for _ in body]
    return (pars, authors)

def scrapePage(page_num):
    # request the page url
    url = f'https://theguardian.com/uk/commentisfree/all?page={page_num}'
    res = requests.get(url)
    # parse the page html
    soup = BeautifulSoup(res.content, 'html.parser')
    # find all article objects on the page
    articles = soup.find_all('a',class_='fc-item__link')
    # extract link url for each article object
    links = [r.attrs['href'] for r in articles]
    texts = []
    authors = []
    for link in links:
        # scrape raw text from article url
        (new_texts, new_authors) = scrapeArticle(link)
        texts += new_texts
        authors += new_authors
    return (texts, authors)
```

Here are some text samples extracted this way:

> "The healthcare system is groaning under demand. But amid the enthusiasm for shiny buildings and clever machines, we must not forget the timeless elements of good healthcare."

> "Maybe it has to do with too many managers managing a dwindling number of clinicians. Or muting the clinician's voice under layers of hierarchy so that resistance feels useless."

Most entries consist of several paragraphs taken from one of the selected news articles. By brief manual inspection, some entries appear to express a clear positive or negative sentiment, with others being more neutral or mixed. There is no additional formatting data or external context in the text. The CSV file containing the new dataset also stores some metadata for each document such as author name and length in words.

In order to ensure the model would be capable of analysing articles from multiple publishers, not just The Guardian, news articles from Politico were also scraped and integrated into a separate dataset. Initial testing of the machine learning models was conducted on the Guardian dataset, and once a satisfactory level of accuracy was reached with this, the Politico dataset was added to testing.

Lexicon-based sentiment label generation

Once the raw text data has been obtained, labels must be generated for each entry. In order for a representative sample, the dataset must be large (around 2,000 entries were used for testing), making manual labelling of the entire dataset highly time-consuming and thus infeasible for this project. Instead, a procedural method is used to derive a set of approximate sentiment labels.

In this system, a lexicon model is used to label the newly extracted documents. An ideal model should consider a range of features in the texts in order to prevent a machine learning model from overfitting on a small set of features which are not necessarily representative of the actual conveyed sentiment. Selecting a set of high quality features to form the basis of the labels will also help the model to generalise better to a larger range of input data, helping it to perform better in more situations. The dataset could also feature some number of human-labelled documents to aid this further.

A substantial amount of research has previously been conducted on lexicon-based analysis of news articles. SentiWordNet 3.0 (Baccianella et al., 2010) is a lexical resource designed for sentiment analysis. It is composed of synsets (sets of one or more synonyms). Each synset is associated with a positive, negative and objective sentiment score. The positive and negative scores reflect positive and negative sentiment, whereas the objective score represents the objectivity of the words in the synset. An objective score of 1 is given to words that express objective fact, whereas a score of 0 is given to words expressing wholly subjective opinions (and partially subjective terms are scored in between).

To generate the sentiment labels, the texts are first tokenised and part-of-speech tagged. Some types of words typically carry more sentiment than others (typically adjectives), so we could focus on just this subset of the tokens (some existing research focuses on other parts of speech too, such as verbs and adverbs). It would also be helpful to consider other word categories such as those that convey negation, since these clearly affect sentiment. After the important words are extracted from the document, an existing lexicon can be used to calculate the sentiment for each word, additionally accounting for extra dependencies and adverbs (*'very bad'* suggests different sentiment to *'not bad'*). The sentiment values of the extracted words and phrases can be composed into a single value (e.g. by summation), then normalised to obtain a sentiment score $s(doc) \in [0, 1]$. These sentiment labels can be used for training machine learning models (the values are continuous, but can be rounded for use by a binary classifier).

The dataset's assigned sentiment labels may be unevenly distributed. This could affect the accuracy of the results. Here is the distribution of labels given to documents in an example dataset, taken from Reuters News.

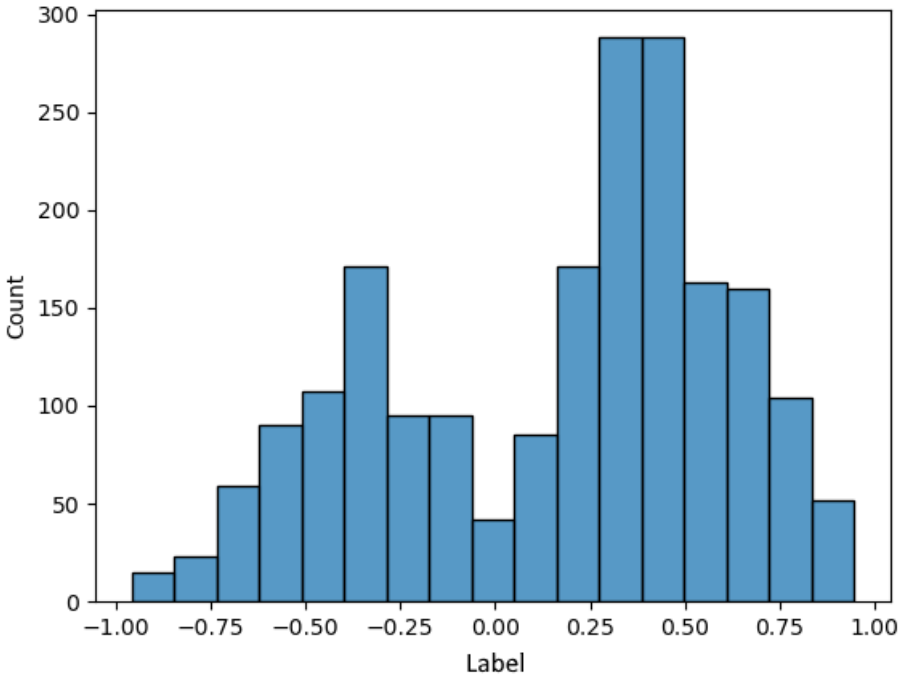

Figure 14: Distribution of sentiment labels on an example dataset.

This dataset's sentiment leans positive. This can be treated by removing positively labelled documents from the set at random until the distribution is even. Overall, this step helps to mitigate bias in the training set.

One issue to be considered when using computer labelled datasets such as this one is that a machine learning model might not perform any better outside of the testing environment than the lexicon approach used to label the data. In order to tackle this, multiple datasets and variations of the lexicon model can be used in order to force the model to learn a range of features. Semi-supervised learning can also be used: after initial training on labelled data, some models can use unlabelled data to further improve performance (Zhai & Zhang, 2021).

Machine Learning Models

Machine learning models aim to find a correlation between the features of the input data (the raw texts) and the outputs (the sentiment labels). We already know how to convert the qualitative text information into quantitative data through vectorisation (described in the previous section of this report). The challenge of this part of the system is to derive an accurate estimation of a label from an input vector with high dimension. This section will overview the different techniques and models that were tested for this project, as well as other models that could be considered.

Statistical models

Statistical sentiment classification models were implemented using the Scikit-learn Python development platform. Before training, the dataset undergoes pre-processing steps.

First, the continuous sentiment labels are binarised (to represent 'positive' and 'negative' sentiment):

```
# load dataset
df = pd.read_csv('dataset/guardian.csv')
# define number of texts for each label
n_texts = 500
# binarise labels
df['bin'] = df['label'].map(lambda y: 0 if y<0.5 else 1)
df_pos = df[df['bin']==1].sample(n_texts) # only positive texts
df_neg = df[df['bin']==0].sample(n_texts) # only negative texts
df = df_pos.append(df_neg).reset_index(drop=True) # combine positive + negative
    texts
df = df.sample(frac=1).reset_index(drop=True) # shuffle dataset
```

The raw text documents are tokenised and stemmed:

```
def prep_text(text): # pre-process the input text document
    text = text.lower() # make all characters lower case
    tokens = text.split() # tokenise the document
    tokens = [t for t in tokens if t not in stopwords] # remove stop words
    tokens = [stemmer.stem(t) for t in tokens] # stem tokens
    tokens = [t for t in tokens if charfilter.match(t)] # remove invalid words
    text = ' '.join(tokens)
    return text
# pre-process all the documents in the dataset
articles = [prep_text(d) for d in df['raw']]
```

The text documents are then vectorised using TF-IDF:

```
# define the TF-IDF vectoriser parameters
tfidf = sklearn.feature_extraction.text.TfidfVectorizer(
    max_features=1000,
    ngram_range=(1,3)
)
tfidf_x = tfidf.fit_transform(articles).toarray()
tfidf_y = df['bin']
```

Finally, each model is trained on the processed dataset and is evaluated on accuracy on test data:

```
def trainPredictModel(model, xs, ys, train_split=0.9, name=''):
    # split data into testing and training subsets
    n = round(len(xs)*train_split)
    x_train = xs[:n]
    x_test = xs[n:]
    y_train = ys[:n]
    y_test = ys[n:]
    # fit model and make predictions
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    # measure accuracy
    acc = str(sklearn.metrics.accuracy_score(y_test, y_pred))
    print(' '.join([name,'accuracy :',acc]))
    return model
```

The accuracy report shows the following results:

```
logistic regression accuracy : 0.64
random forest accuracy : 0.61
SVM accuracy : 0.65
```

I also used Doc2Vec vectorisation on the data and used the same models to compare efficacy:

```
article_tokens = [a.split(' ') for a in articles] # split back into tokens
tagged_articles = [doc2vec.TaggedDocument(d,[i]) for i,d in enumerate(
    article_tokens)]
# define and train Doc2Vec model
dtv = doc2vec.Doc2Vec(vector_size=20, window=2, epochs=50)
dtv.build_vocab(tagged_articles)
dtv.train(tagged_articles, total_examples=dtv.corpus_count, epochs=dtv.epochs)
# predict Doc2Vec vectors
dtv_x = [dtv.infer_vector(x) for x in article_tokens]
dtv_y = df['bin']
```

The accuracy report is shown below:

```
logistic regression accuracy : 0.62
random forest accuracy : 0.60
SVM accuracy : 0.59
```

	Logistic regression	Random forest	Support vector machine
TF-IDF	64%	61%	65%
Doc2Vec	62%	60%	59%

Figure 15: Accuracy results for three models over TF-IDF and Doc2Vec vectors.

The results achieved on the TF-IDF vectors were slightly better than the Doc2Vec vectors. Overall, the accuracy report suggests that this method is better than guessing sentiment labels at random, but accuracy is still low.

The text vectors can be visualised using t-SNE (*t-distributed stochastic neighbour embedding*), a method used to represent high-dimensional data in two dimensions (Hinton & van der Maaten, 2008). The result of this is shown below: the colours of the points on the graph represent the actual sentiment labels and the predicted labels (green is positive and red is negative).

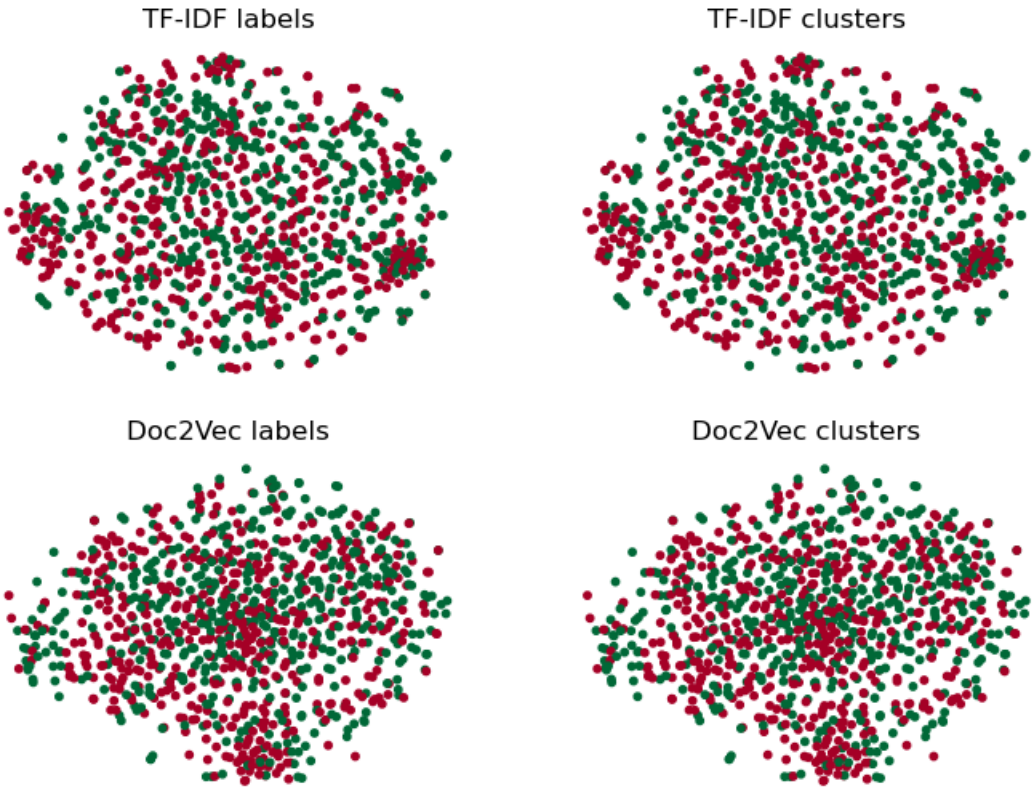

Figure 16: t-SNE visualisation of the TF-IDF and Doc2Vec text vectors.

This visualisation illustrates that the vector data is relatively unstructured—there doesn't appear to be any clear and simple connection between the values of the vectors and the sentiment label. In order to improve the accuracy of the classifier, we could either develop a more sophisticated prediction model or attempt to further denoise the data.

Feedforward neural networks

Like with the statistical models, the dataset is loaded, processed and vectorised. TF-IDF vectorisation was used for this section because it appeared to help the earlier models achieve higher accuracy.

A simple two-layer feedforward neural network is defined, compiled and trained:

```
# define the feedforward network model
ffn_1 = models.Sequential([
    layers.Dense(32),                       # generic dense layer
    layers.Dense(1,activation='sigmoid')    # output layer
])
# compile the model with suitable learning rate + loss function
ffn_1.compile(
    optimizer=optimizers.Adam(1e-4),
    loss='binary_crossentropy',
    metrics=['accuracy']
)
# train the model on our data
ffn_1_history = ffn_1.fit(
    x_train,y_train,
    epochs=10,
    validation_data=(x_val,y_val))
```

After ten epochs of training, this model achieved 64% accuracy:

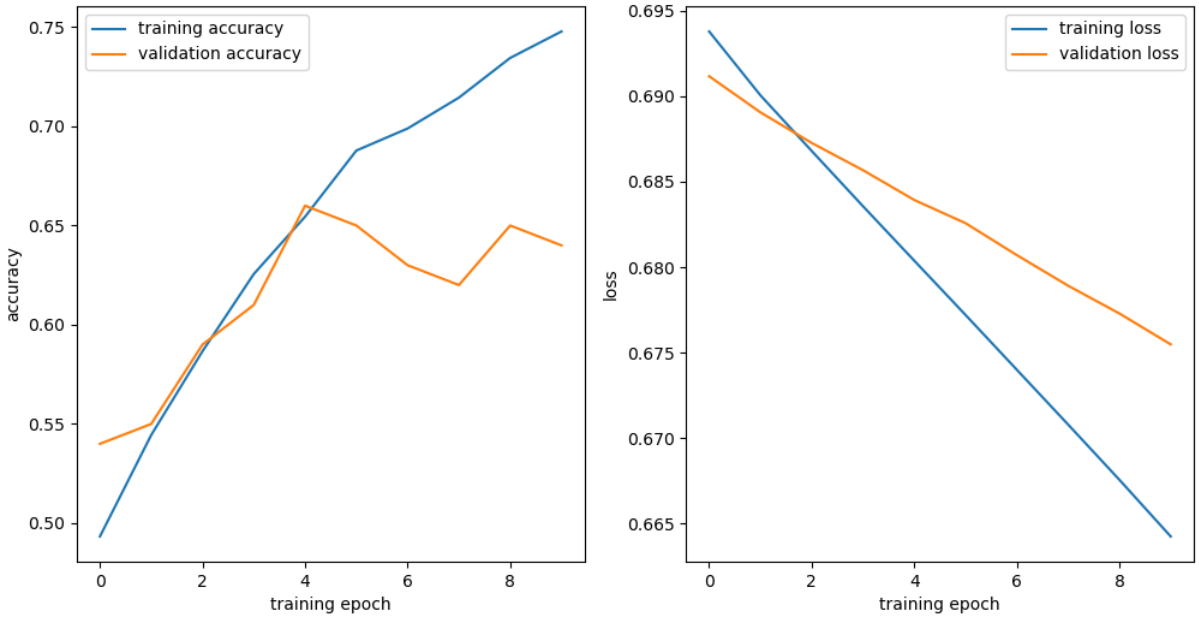

Figure 17: Graph displaying accuracy and loss of the FFN throughout training.

The neural network appears to be about as accurate as the statistical models. The loss graph shows that the quality of the network's predictions continued to improve throughout the entire training process, which suggests that further training could improve accuracy further. The network was then trained for another 100 epochs, improving the accuracy on the test set to 70%:

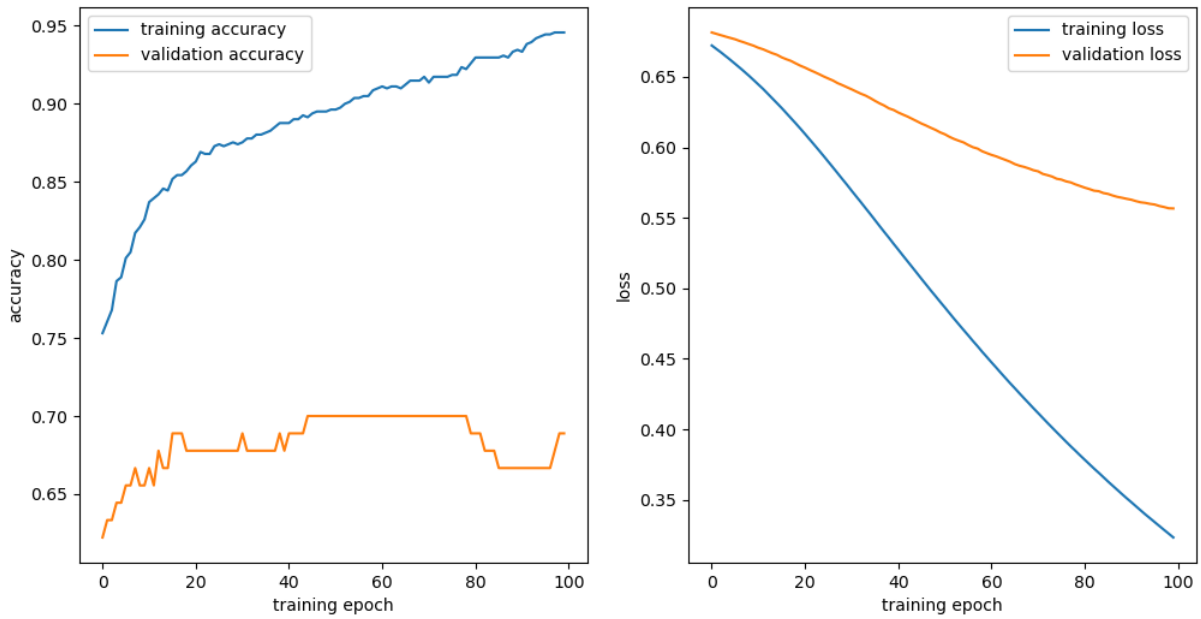

Figure 18: Model accuracy and loss over epochs 11-110.

The training graphs reveal that the model performs significantly better on the training data than the validation data; this is likely due to overfitting. One technique used to treat overfitting in neural networks is regularisation. Cortes et al. (2009) found that L2 regularisation is an effective technique for improving ANN performance. It introduces a penalty to the loss function in the form:

$$\min_{w} L(y, \hat{y}) + \lambda ||w||_2^2$$

where w is the vector containing a neuron's weights, $L(y, \hat{y})$ is the loss function, $||w||_2$ is the L_2 norm of w and λ is a manually specified hyperparameter. The L_2 norm is given by:

$$||w||_2 = \sqrt{\sum_i |x_i|^2}$$

This penalty discourages the network from learning more complex features, which helps to prevent overfitting. The neural network model defined earlier was adapted to use L_2 regularisation.

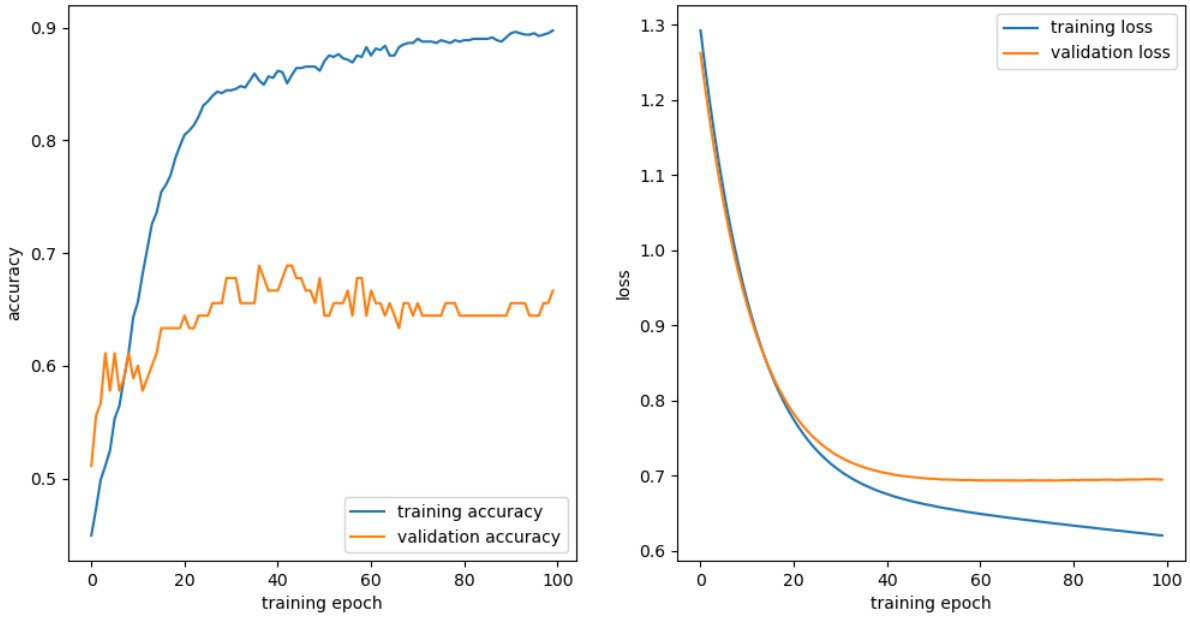

Figure 19: L_2 regularised model accuracy and loss over 100 epochs.

This training visualisation shows that the regularisation measure has reduced overfitting. However, the accuracy and loss were not improved from the previous model. In fact, the accuracy only reached 67%, down from 70% previously.

Dropout regularisation was tested as an alternative to L_2 regularisation. The dropout method removes connections from the network at random during training to avoid overfitting (Wager et al., 2013).

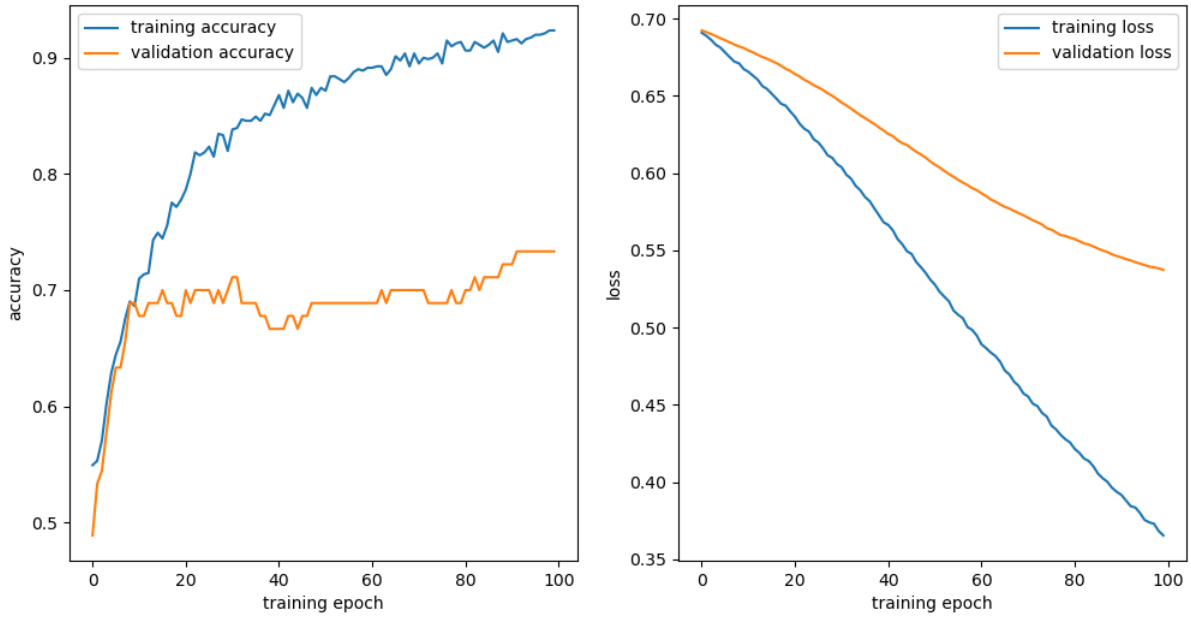

Figure 20: Dropout regularised model accuracy and loss over 100 epochs.

The gap between the training and validation loss during training implies that some overfitting is still present, but accuracy is improved somewhat compared to the alternative model.

Both regularisation techniques were then combined:

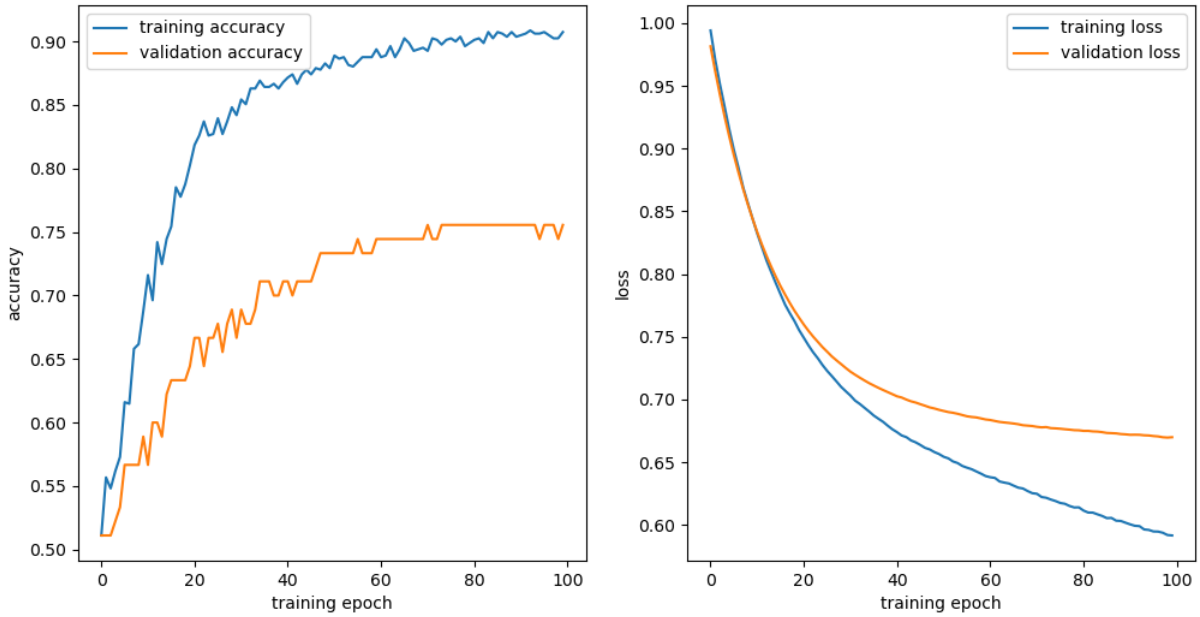

Figure 21: Dropout and L_2 regularised model accuracy and loss.

The combined approach appeared to be the most effective, achieving 74% accuracy in one instance of training.

Recurrent neural networks

Some tests were conducted using TF-IDF vectorisation and a simple RNN. The model had very low accuracy and no improvements were seen during training. This is to be expected because TF-IDF vectors lose the sequential information that makes recursive networks so powerful. The RNN is able to remember context (previous input values), but the order of the TF-IDF vector has no useful meaning. This is because TF-IDF only considers the frequency of words in the texts, not the order. Instead, another vectorisation method which does consider word order should be used (such as Doc2Vec).

To test the simple, single-hidden-layer recurrent neural network, the documents in the dataset were vectorised using Doc2Vec. The RNN model was then defined and compiled:

```
# define the simple RNN model
rnn_1 = models.Sequential([
    layers.Reshape((1,20)),                    # format data
    layers.SimpleRNN(32),                      # hidden layer
    layers.Dense(1,activation='sigmoid')       # output layer
])
compileAndTrainModel(rnn_1,dtv_x,bin_y)
```

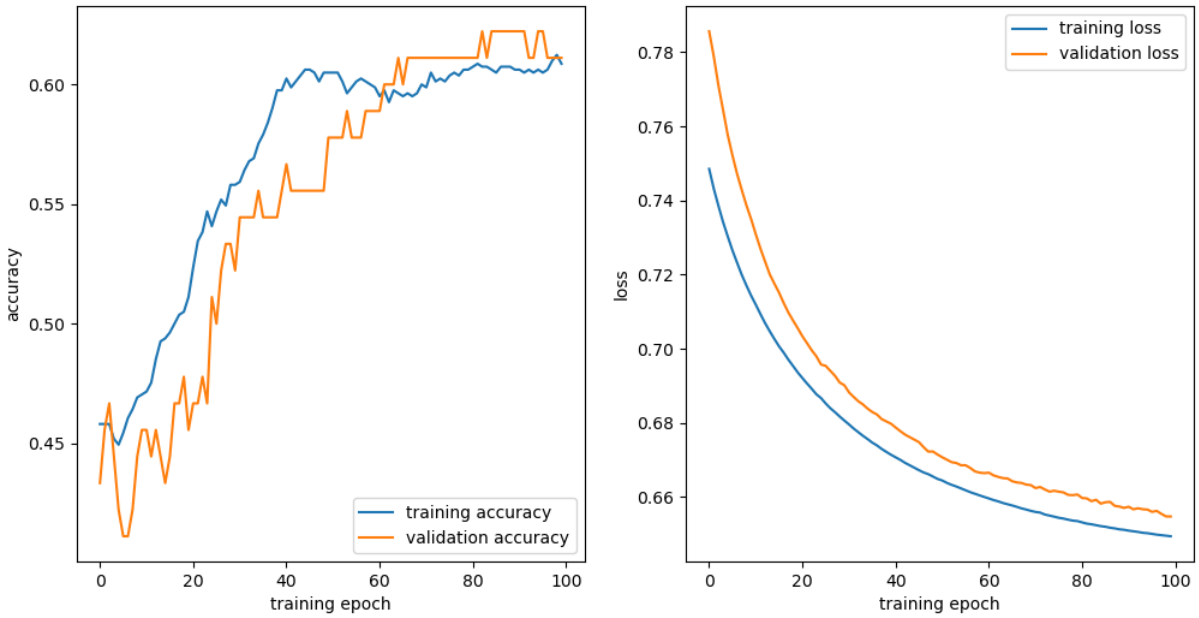

Figure 22: Simple recurrent neural network training history.

With the Doc2Vec vectors, the model is capable of learning. However, the accuracy remains low. The trends in the loss graph suggest underfitting, so a more complex model is likely needed.

Mishra et al. (2020) found that deep RNNs—RNNs with multiple hidden layers—are often more effective at tackling classification problems. A deep RNN sentiment classifier was implemented:

```
# define a deep vertical RNN
rnn_2 = models.Sequential([
    layers.Reshape((1,20)),                          # format data input
    layers.SimpleRNN(32,return_sequences=True),      # hidden layer 1
    layers.Dropout(0.5),                             # regulariser
    layers.SimpleRNN(32),                            # hidden layer 2
    layers.Dense(1,activation='sigmoid')             # output layer
])
compileAndTrainModel(rnn_2,dtv_x,bin_y)
```

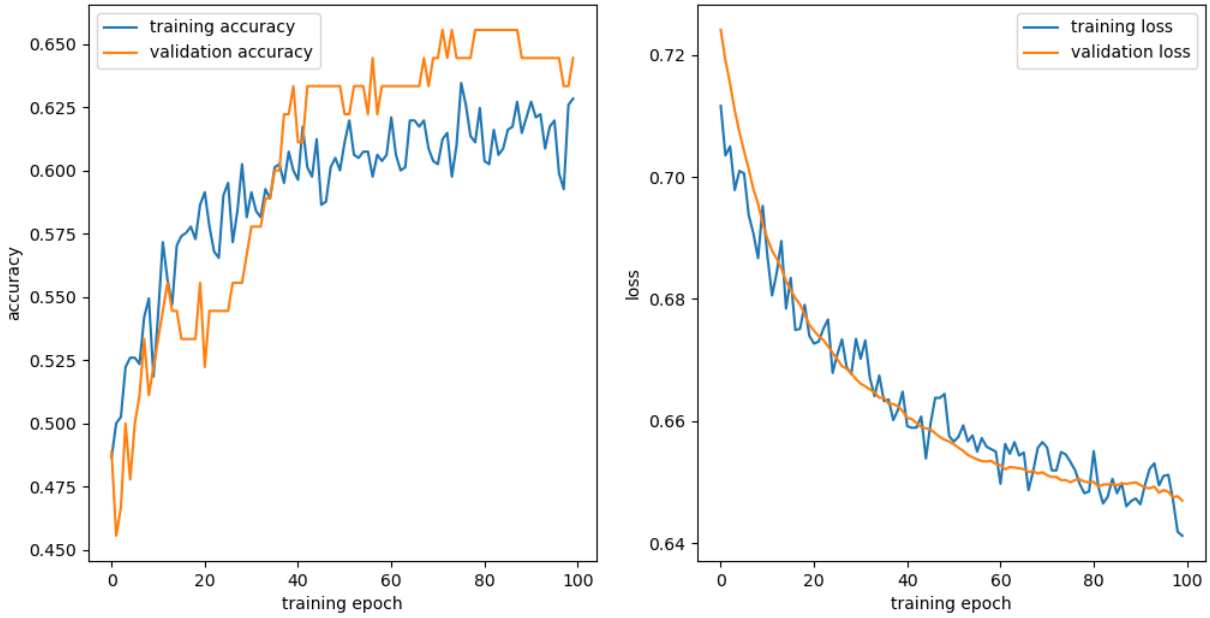

Figure 23: Deep recurrent neural network training history.

Vector embedding and bidirectional LSTM neuron layers were then added to the model:

```
# define a bidirectional LSTM model with embedding
disc = layers.Discretization(num_bins=20) # converts vector values to naturals
    to allow for embedding
disc.adapt(dtv_x)
rnn_3 = models.Sequential([
    disc,                                            # prepare for embedding
    layers.Embedding(20,64),                         # vector embedder
    layers.Bidirectional(                            # hidden layer 1
        layers.LSTM(32,return_sequences=True)),      # forward & backward layer
    layers.Dropout(0.5),                             # regulariser
    layers.Bidirectional(                            # hidden layer 2
        layers.LSTM(32)),                            # forward & backward layer
    layers.Dense(1,activation='sigmoid')             # output layer
])
compileAndTrainModel(rnn_3,dtv_x,bin_y)
```

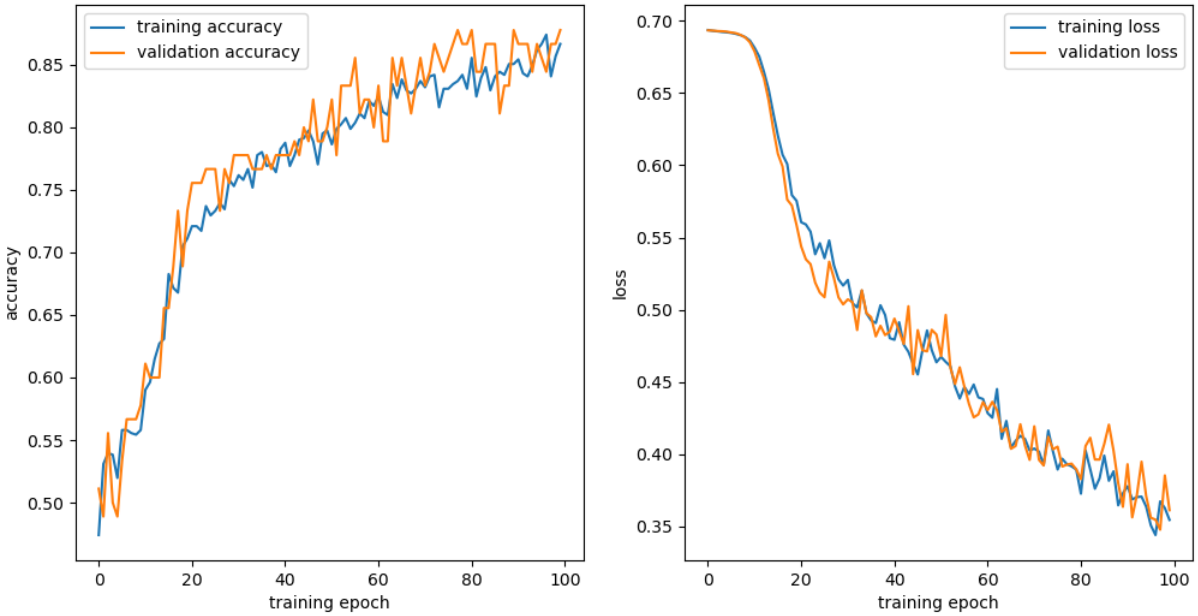

Figure 24: Bidirectional LSTM recurrent neural network training history.

The accuracy of this iteration of the model is much better than before. However, the discretisation layer (which converts the real vectors into positive integer vectors for the embedding layer) loses some information. Instead we could use the built-in vectoriser provided by Keras, which outputs ready-to-embed positive integer vectors. The architecture of such a model would look like this:

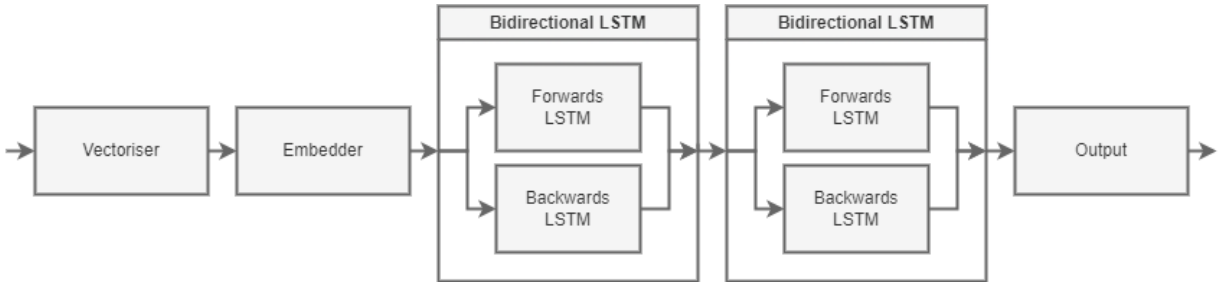

Figure 25: Architecture of the new network.

Since the Keras vectoriser outputs integer vectors, the size of the vectors needs to be larger. As a result, training is slower—so fewer epochs were run when training this model.

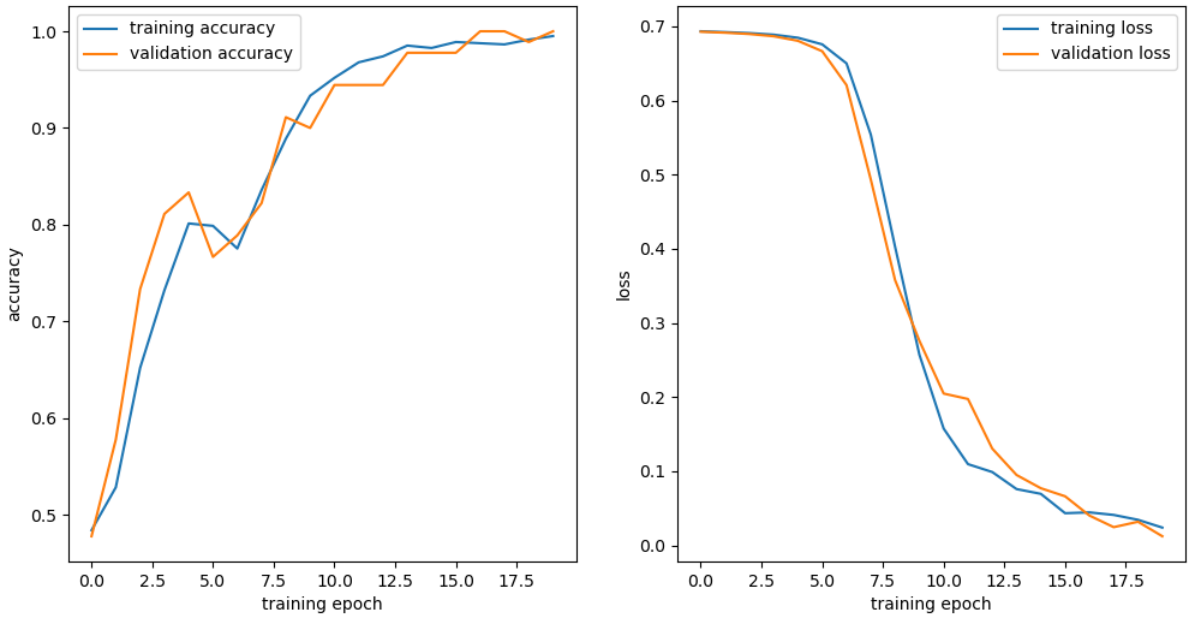

Figure 26: Bidirectional LSTM training on alternative vectors.

With the new vectoriser, the model is even more effective than before. It was able to predict the sentiment label of all 100 test cases correctly.

Because this project aims to look at broad patterns and trends in news publications, it would be helpful to look at other sources. According to the publisher's website, Politico aims to give insights on political issues without endorsements or partisan takes. Therefore, it would be interesting to find out whether sentiment is even across its different articles. The bidirectional LSTM model was trained using a dataset of Politico articles, but accuracy on test data was found to be lower (75-80%). This could be due to stylistic differences in Politico's articles, such as its more neutral tone and vocabulary.

In order to improve the accuracy of the classifier, alternative methods will be used to vectorise the text data.

Autoencoders

Previous research has successfully used autoencoders for dimensionality reduction. An autoencoder typically consists of an encoder, a decoder and a bottleneck *latent layer*. The encoder transforms high-dimension feature vectors into a low-dimension latent space. The decoder aims to reconstruct the latent representations back into the original features with as little loss of information as possible.

In the proposed autoencoder model, pre-labelled texts are split into positive and negative groups. The autoencoder itself takes vectorised text documents as input.

The training of this model differs from most other autoencoders because positive text vectors are fed into the encoder as input, and negative vectors are given to the decoder output for loss computation. Therefore, the latent output of the encoder can represent an interpolated encoding between texts with positive and negative text features. However, by maximising loss during training, the encoder can be trained to differentiate between positive and negative inputs—since the output of the autoencoder on a positive text vector will be further away from a negative vector. As a result, any information that does not appear to directly correlate with the sentiment label is lost. The low-dimension, highly discriminative feature vectors output by the the encoder can then be given to a classifier for more accurate classification.

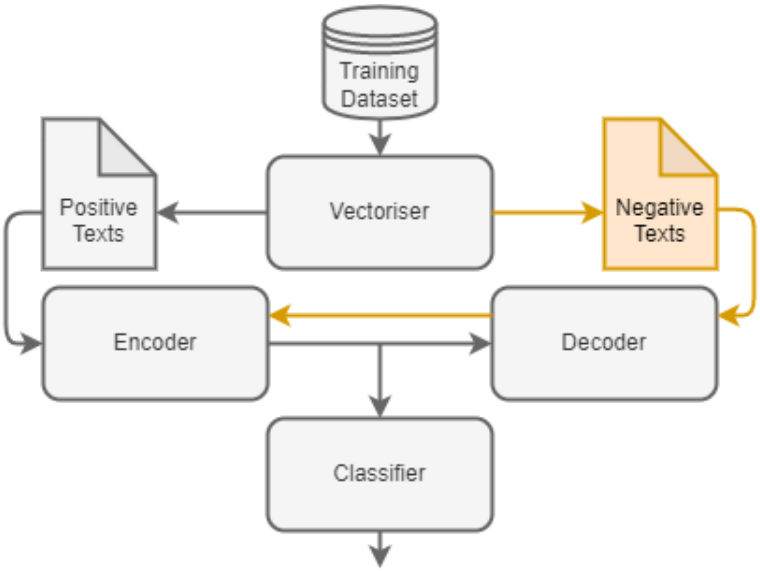

Figure 27: Basic architecture of the autoencoder classification model.

This architecture could be considered a form of semi-supervised learning: while, like a typical autoencoder, labels are not given to the model, the sentiment label of each document is implied by the direction in which it is fed into the machine. Post-training, the encoded outputs given by the latent layer can be used for supervised or unsupervised learning.

Using the previously described technique t-SNE for dimensionality reduction, we can visualise the latent representation of the training inputs. The low-dimension vector clearly differentiates most of the input documents correctly. Predictions were made using a simple linear regression model, which was able to classify a large proportion of the test inputs correctly.

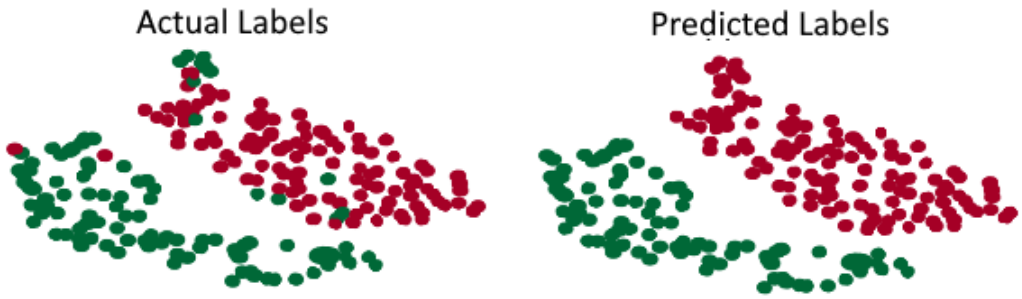

Figure 28: Graph showing t-SNE clustering of positive (green) and negative (red) documents.

	TF-IDF + random forest	Doc2Vec + bidirectional LSTM	TF-IDF + autoencoder + random forest
The Guardian dataset	64%	99%	98%
Politico dataset	62%	76%	95%

Figure 29: Accuracy results for three models over TF-IDF and Doc2Vec vectors.

The autoencoder model was also trained and tested on the SEN dataset (Baraniak & Sydow, 2021) and achieved 78% accuracy on the test data. The lower accuracy for this data could be due to the difference in format and style of text data—since the input data is made up of news headlines rather than document bodies.

Data Insights

Once the model is trained, we can classify sentiment on unseen articles. After analysing a sizeable test sample of articles, various methods can be used to gain better understanding into the significance of the results.

Filtering the input data

In order to detect trends and biases in the media analysed by the system, it would be useful to be able to filter and sort the input documents by subject. This could also help to mitigate one of the limitations of sentiment analysis: the absolute sentiment value of a text is very subjective and algorithm-dependent. By comparing sentiment across subjects, readers can instead look at the relative sentiment scores instead.

SpaCy's natural language processing pipeline for Python features built-in named-entity recognition (NER). Named entities are real-world objects such as people, organisations or geopolitical entities, usually denoted by proper nouns. Each named entity is also given a tag depending on the type of object it refers to. By looking at the usage of the named entities in a text document, we can get a sense of the main topic of discussion.

We designed a simple module to attach subject information to each document in the news article datasets. Every named entity instance in a document tagged as an organisation, person or nationality or political entity is counted, then the most frequently mentioned entity is given as the main subject of the article.

Document text	Document subject
"The goal of Russia in a country like Georgia is to cause social unrest, instability, and chaos. And Georgia is already aflame."	Georgia
"Or they sow so much doubt about the popular vote that they decide the outcome."	<None>

Figure 30: Subject tags on sample documents taken from the dataset.

We can also look at different named entity tags to extract other information. For example, we can assign a 'country' tag to each document by looking at named entities tagged as geopolitical entities.

One cause of inaccuracy in determining the main subject of an article could be inflected forms of words. For example, 'Russians' and 'Russia' would be counted separately. To mitigate this, the entity strings can be formatted (for example by stemming or making all characters lower case). Given more development time, we could also consider some substantially more complex text summarisation algorithms to extract subjects from the text documents.

The text documents can be given other metadata labels to help readers filter the data, such as author, date published or article length in words. These labels were all relatively easy to extract either at the web scraping stage or after processing the data. Some of these features might not appear to be particularly interesting or relevant to the project aims; nevertheless, by analysing them we might uncover unexpected patterns or information that could aid troubleshooting.

Data visualisation

Data visualisation is a critical aspect of data science and analysis. It helps developers convey information to colleagues and decision-makers. Visualisation is especially important in this project because the analysis methods are complex and difficult to explain in simple terms. Therefore, presenting the results visually will help readers come to conclusions based on the analysis.

Tableau is a data visualisation program that is able to connect seamlessly to a variety of data sources (including CSV files, which are the main format used to store data in this project). Information in the database can be updated live, with updated results showing in the graphs automatically. Tableau is easy to work with and is capable of generating high-quality figures which convey information clearly.

Using the metadata included in the results dataset, we can create a variety of charts and graphs. For example, we can visualise the average sentiment of articles from a given publisher by country for The Guardian or The Telegraph.

From these graphs we can make insights quickly and easily. For example, the Guardian articles sampled spoke mostly positively about countries in mainland Europe, whereas The Telegraph had a more mixed sentiment. In articles discussing Rwanda, The Guardian was very negative whereas The Telegraph was neutral or mixed. When combined with context and some analysis by humans, we can understand some of the broad patterns in the dataset.

Other insights we can gather are average sentiment by author, sentiment by subject and levels of polarisation in the news articles (for results that were given a continuous sentiment score). By collecting additional metadata about the entries in the dataset, we can find more ways to analyse the results.

Data visualisation tools can also be used to assess the performance of the model in finer detail. For example, the predictions can be broken down by subject and plotted on a graph to visualise the model's accuracy. By doing this we can see a strong correlation between predicted sentiment y and actual sentiment $\hat{y}$—unsurprising given the high accuracy of the model on the dataset. However, some weaknesses can be identified: for example, subjects with extreme sentiment are sometimes labelled with less precision than other labels. We can see that subjects with all-negative or all-positive actual labels are labelled more neutrally than they should be. This is a finding that would not be obvious from just the accuracy value.

Interestingly, the model appears to struggle with subjects that are evenly split between positive and negative. These are subjects that are highly polarised or mainly neutral. This highlights a weakness of the autoencoder model: because the input documents are split into positive and negative, the model only considers two options. Therefore, it can not identify features in 'neutral', 'slightly positive' or 'slightly negative' sentiment in documents, and any input documents with neutral or conflicting sentiment will be classified less accurately.

The Doc2Vec RNN model that doesn't feature an autoencoder is capable of processing data with continuous sentiment labels instead of just binary ones. A visual representation of the results made in tableau is available to view in an appendix. The red circles (representing the labelled Politico documents) are distributed much more sparsely. This follows from the fact that the RNN model was less effective at evaluating the Politico dataset. Overall, however, there is still a clear positive correlation between the predictions and the actual sentiment scores.

Looking at the blue circles (the Guardian dataset), the results follow a subtle sigmoid curve. Essentially, the predicted scores are slightly exaggerated on documents where extreme sentiment is expressed. This could either be considered an inaccuracy, or a reflection of the additional discriminatory features that may have been detected by the model. Regardless, the magnitude of the sentiment score is a subjective measure rather than a measure of any objective feature in the documents. For measuring the efficacy of the system, the strength of correlation between y and $\hat{y}$ is more useful than its gradient.

Results and Evaluation

One of the main difficulties of designing this system was judging the accuracy and quality of results obtained from the various supervised models. This is a direct result of the dearth of high-quality datasets that have been constructed with this specific classification task in mind. With this in mind, it was important to inspect the results in a variety of ways in order to ensure efficacy.

Testing on a Labelled Dataset

The first and most empirical method of evaluating the accuracy of the classifier was by making predictions on a labelled testing set. The dataset used for training was split into three parts: training, validation and testing data. The training dataset is used to fit the model and the validation set is used at each training epoch to calculate model loss, which is then used in backpropagation calculations to adjust weights and biases in the network. The testing set is used at the end of training to evaluate how well the model was able to generalise and produce accurate predictions on unseen data. The testing set is sampled randomly from the entire dataset to avoid bias in the order of the dataset entries.

All of the models tested can be compared by measuring their accuracy on the testing set. Some of the models were tested with both TF-IDF and Doc2Vec vectorisation methods, which had an impact on the accuracy. The results on the Guardian dataset have been included as an appendix, but the most notable of these are as follows:

Model	TF-IDF accuracy	Doc2Vec accuracy
Embedded bidirectional LSTM	-	86%
LSTM + integer vectors	99%	-
Autoencoder + random forest	98%	-

Figure 31: Results given by accuracy report for three models on texts obtained from The Guardian.

We found that both the LSTM model and the autoencoder model delivered very high accuracy on the testing data. Essentially, this is the highest level of accuracy we would likely expect from any model for this specific task. Since both of these models have a similar level of accuracy on this dataset, there is not much to separate them. However, by testing on other datasets we found differences in their performance:

Model	TF-IDF accuracy	Doc2Vec accuracy
Embedded bidirectional LSTM	-	53%
LSTM + integer vectors	76%	-
Autoencoder + random forest	95%	-

Figure 32: Results given by accuracy report for the same three models on Politico news articles.

In the Politico dataset, the discriminative features appear to be much harder to identify: the baseline bidirectional LSTM model achieved a much lower accuracy, suggesting that this classification task is more challenging. The improved LSTM model achieved only 76% accuracy, which is not high enough to be used in business or research. In this task, the autoencoder model showcases its strength, where it is able to perform well even where other models struggled. The full results obtained on this data are shown in the corresponding appendix.

The same tests were conducted on the SEN dataset. The end results were similar. In contrast to the first two datasets, all of the entries in the SEN dataset are human-labelled. Another factor that made this data harder to learn from was the smaller training set size (400 documents total compared to the 2,000 available with the previous two datasets). Most of the models that were tested struggled on this data, but the autoencoder model was again able to achieve a high level of accuracy (98%), showing that it is flexible and can easily identify sentiment in the testing set, even when given a relatively modest amount of training data.

Comparing to Existing Results

Another way of evaluating the models designed and tested in this project is by comparing the results to those found in previous work. The accuracy achieved on the novel datasets from The Guardian and Politico can not be compared to other models because no other models have been tested on them. However, we can test the models on commonly used datasets, including datasets of different types of text documents, in order to evaluate their ability to generalise.

Pang & Lee (2002) created a dataset of 2,000 IMDb reviews labelled as positive or negative. The novel autoencoder and random forest classification model was able to correctly predict the sentiment of all 200 documents in the randomly sampled test set. This suggests that the model was able to identify features of positive and negative texts that are not unique to just news articles. Using a t-SNE plot as before shows that a set of highly discriminative features were found to differentiate positive and negative documents:

Figure 33: t-SNE clustering of the autoencoder vectorised positive (green) and negative (red) IMDb reviews.

The IMDb Movie Reviews dataset is a sentiment analysis dataset of 50,000 movie reviews labelled as positive or negative (Maas et al., 2011). This is essentially an improved version of Pang & Lee's dataset and is often used as a benchmark for review sentiment analysis models.

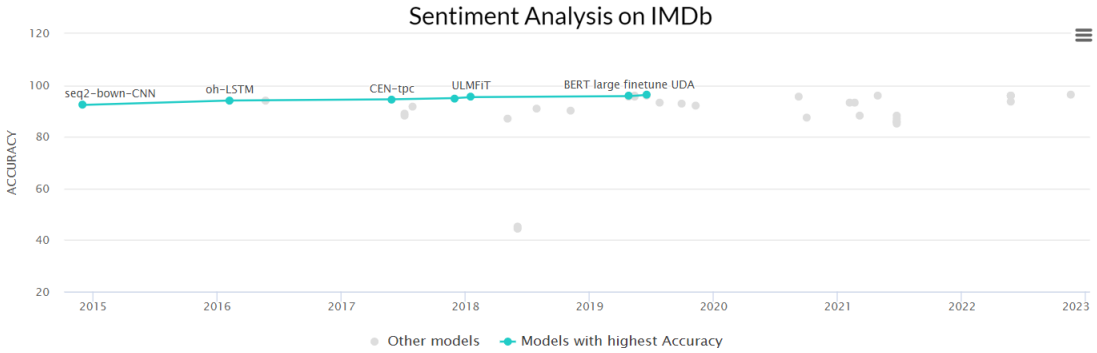

Figure 34: Leaderboard of the most accurate classifiers on the IMDB Movie Reviews dataset.

Most of the models used in previous research were able to achieve a high level of accuracy (more than 90%) on this dataset. The best-performing existing model is *XLNet* (Zhilin Yang et al., 2020), which was 96.2% accurate on the test data. The autoencoder and random forest classifier model developed for this project was able to achieve a comparable 96% accuracy. Interestingly, XLNet also makes use of an autoencoder neural network model as a means of dimensionality reduction.

Islam et al. (2017) conducted research on the document-level sentiment analysis of news articles using a lexicon-based approach. It achieved 91% classification accuracy on its chosen dataset consisting of articles from The Independent, The Telegraph and The Daily Star. It would be interesting to compare the efficacy of this model to that of our own classifier but the documents and labelled used for testing were not provided in the study. In addition, the number of samples (56) was relatively small, so the results are not as significant as other studies.

Samuels & Mcgonical (2020) conducted a similar study using a BBC News dataset from Greene & Cunningham (2006). However, this dataset has no sentiment labels and no measure of accuracy was given by the study.

Overall, the efficacy of our novel autoencoder-based model appears to be high compared to other models approaching similar tasks, including more general sentiment analysis problems such as classification of movie review sentiment. However, due to the lack of labelled data, it is difficult to ascertain exactly how well it performs compared to other approaches to document-level sentiment analysis of political news articles specifically.

Using the System

While all the main findings of the project are included in this report, it might interest readers to recreate the results, or to test the model on new data. The code used to produce the results is available as an attachment to this submission. Inside the zip file there are two folders: a `dev` folder and a `main` folder. The `dev` folder contains various files, resources and prototypes that were developed and used during the development of the system. Most of the Python files contain helpful comments and can be executed individually to test results. The `main` folder contains a smaller set of code files that can be executed in order to reproduce the results presented in this report. In `main` there is code to activate a virtual environment and a Docker container to make the system easier to use.

With Docker installed, setting up the container is simple: navigate to `main`, then in the command prompt enter:

```
$   docker build -t pol_sent --rm .
```

This will build the Docker image, ready to be used. This process will likely take several minutes, but only needs to be executed once. Once this is complete, run the following in the command prompt:

```
$   docker-compose run --rm app
```

The Python environment will now instantiate, allowing us to execute individual Python files (`python <file-path>.py`) or run Bash scripts to execute multiple parts sequentially.

An example run of a subset of the Python scripts might look like this:

```
/app#   python data_proc/politico_reader.py dataset/mydata.csv
/app#   python data_proc/subject_finder.py dataset/mydata.csv
/app#   python report_demo/aenet.py dataset/mydata.csv
```

The end result will be an annotated dataset of Politico news articles, each labelled with a lexicon-based and an autoencoder-based sentiment label as well as a topic-of-discussion label, stored in `dataset/mydata.csv`.

Alternatively, the files can all be run without the use of Docker. The version of Python used for development was Python 3.9 and the required libraries for the project are all available in `main/requirements.txt`. All of the code should function properly in any environment as long as the required modules are installed and the correct version of Python is used. For ease of use, a virtual environment is included in the `venv` folder for all the requirements to be installed to. Note that some additional materials may need to be downloaded during runtime, such as the stop words corpus provided by NLTK.

To view, adjust and create new data visualisations, readers can use the files included in the `datavis` folder. This requires an installation of Tableau Desktop. Contained in these files are a selection of visual aids to help understand the results, as discussed in the data visualisation section of this report. Results can be updated in the program by clicking the 'refresh data source' button, and graphs can be made and adjusted relatively easily with help from the Tableau user interface.

Due to its very large size, the file containing the Word2Vec model was not included. An alternative version can be generated by running the following command:

```
/app#   python data_proc/wtv.py -g
```

Conclusion and Reflection

Main Achievements

While sentiment analysis is currently a very popular area of research, the amount of existing work around document-level sentiment classification of news articles is relatively small. On the datasets tested, the performance of the model presented in this report is as good or better than all of the comparable models we tested it against.

The main unique approach taken in this project was the novel application of autoencoders. Autoencoders have been used perviously for denoising text vector data, but by training the network to encode from positive documents and decode into negative documents, we were able to produce a latent representation which discriminated very well between the two types of documents. While this model has fewer applications than other methods (since it is more difficult to consider labels that are not binary positive or negative), the impressive performance makes up for this in some ways.

The model was also shown to be relatively robust as it performed well on a variety of datasets, even when the datasets consisted of other forms of opinionated text data than political news articles. The performance remained consistent across tests when the dynamically constructed training datasets were rebuilt using newly published articles, suggesting that the model is not reliant on a small set of obvious features in the texts.

Limitations and Difficulties

The system designed in this project suffers from many of the same shortcomings as most other sentiment classifiers. Assigning an objective value to such a subjective notion as sentiment opens the system up to obvious criticisms, since humans are likely to disagree on what they perceive to be sentiment in different texts. This is also reflected in the results of the algorithms and models designed by different researchers, where the methods are largely experimental and results often vary significantly between these different methods (Anstead & O'loughlin, 2012). Furthermore, readers should be aware that computational methods are at risk of being biased, regardless of how well they perform in tests. This is due to the way in which accuracy is calculated: since test data consists of human-generated labels, it is subject to the biases of the person or group labelling the samples.

Another limitation of sentiment analysis algorithms is in their ability to handle sarcasm and satirical news articles. No testing was conducted on such articles in this project, but the potential difficulties of this are clear. Where a human reader would likely have additional context (such as knowledge of the publisher or author) to supplement the actual information conveyed in the text, pure text analysis algorithms must rely solely on the raw text.

Probably the most challenging aspect of this project was acquiring a selection of high-quality datasets that would be suited to the task of document-level news sentiment analysis. Most document-level analysis conducted by previous studies target shorter, less formal and generally more emotive texts such as social media posts and user reviews. Although similar concepts can be used to analyse both this type of document as well as news articles, the datasets provided in the existing studies were not suitable to be used in this project. Some research has also focused on sentiment analysis of news articles, but these tended to focus on aspect-level analysis, which was not the aim of this project and which utilises substantially different training and testing data to that required by document-level classifiers.

After struggling to find a suitable pre-existing dataset for the project, the decision was made to construct a new one from online news sources. This presented two main challenges during development: the legality of scraping the articles, and the difficulty of accounting for different HTML formats in the data sources. Many online newspapers, especially those which require a subscription to access, prohibit the use of automated software to mine their content. The New York Times terms of service explicitly states that scraping articles from its website could lead to legal action. As a result, the variety of news sources available to analyse is much less than the newspapers which a human could read. The other challenge that came with the development of a web-scraped dataset was the inconsistent and difficult to navigate format of many online news sites. For example, both CNN and Fox News use different container names and classes for each article on their online sites, making their articles difficult to access automatically. Furthermore, most news publishers do not keep a comprehensive archive of past articles in an easy to access place. Therefore, only a limited number of news articles could be scraped and used in the training dataset.

Improvements on the Project

A current limitation of the system is that only a small number of news publishers were read by the web scraping modules to construct the novel datasets (The Guardian UK, The Telegraph, The Mail Online and Politico were all used as data sources). A simple improvement that could be made would be to use Google News, which aggregates local and national newspapers, as an additional data source. This could help to make the training data more varied and more representative of the news articles that an average person might see online.

Usability was also a challenge when designing the system. Because the majority of the project was focused on research, the bulk of development time was spent designing and implementing the machine learning algorithms and models at the expense of system usability. As a result, most of the components of the system are interfaced with via console input or CSV file. Given more development time, it would be relatively trivial to produce an executable file or web application showcasing the functionality of the models.

Additionally, there are a variety of optimisation steps that could be taken to reduce training time or to improve performance. For example, a more comprehensive text pre-processing module could help to reduce the amount of information given to the machine learning models during testing by further trimming unnecessary details from the text—this could include a more sophisticated lemmatisation process, or removing sentences with neutral or ambiguous sentiment. The datasets created could also be optimised further by removing low-quality entries (such as particularly short/long paragraphs).

Potential for Future Work

A limitation discussed earlier in this chapter is that the classifier model ignores the context surrounding the news articles. Further research could look into methods to factor this into the model's computation. For example, metadata such as date of publishing, author name or headline could be looked at. Additionally, other forms of media (images and audio and video clips) could be analysed to give the system a fuller understanding of the sentiment conveyed by the articles.

Another difficulty was acquiring a large, high-quality dataset for training and testing. An interesting way to tackle this would be to automatically generate new texts or text vectors based smaller sets of positive or negative text documents. Variational autoencoders (VAEs) have been used in the past in order to generate text (Kingma & Welling, 2013). Essentially, a VAE is an autoencoder combined with variational interference (Nicola, 2018). The VAE defines a probability distribution on the latent representation defined by the autoencoder, allowing for new latent samples to be created and then decoded. With a suitably powerful autoencoder (perhaps the one presented in this report), new high-quality training data could be produced.

An alternative approach to generating new texts is to use a generative adversarial network (GAN). GANs are often preferred over VAEs for text generation because they tend to produce more realistic and diverse texts than VAEs. GANs are also generally better at modelling the relations in language between words (Rosa & Papa, 2021).

There is also potential for further research around analysing other qualities of the news articles. For example, it might be interesting to attempt to detect *fake news* by analysing the language and structure of the texts. This would likely be more challenging than sentiment analysis, but potentially equally as useful. Data mining approaches have been used in the past to detect fake news to some level of success (Shu et al., 2017), but the amount of research into linguistic-based fake news detection is relatively small. The concepts presented in this report could be used to explore the problem of detecting fake news in more depth.

Author Self-Assessment

- *What is the technical contribution of this project?* The main unique aspect of this project is its novel use of autoencoders to perform document-level sentiment analysis on news articles. The sentiment classifier is robust and performed as well or better than similar models on the test data.
- *Why should this contribution be considered relevant and important for the subject of your degree?* Sentiment analysis and natural language processing are very popular areas of computer science, with a variety of important applications that have been discussed in this report. This project has expanded on some previously seen ideas and there is potential for these ideas to be expanded upon further in the future.
- *How can others make use of the work in this project?* The results in and of themselves are interesting to look at, although as discussed throughout they should not be used as a sole means of understanding sentiment in news articles. In addition to reproducing the results and producing new results using the code provided, readers could expand upon the models presented in this report to build an even more sophisticated sentiment classifier (or a similar model that could perform a different task).
- *Why should this project be considered an achievement?* At the beginning of this project, I had very little knowledge of machine learning or natural language processing. Since then, I have succeeded in designing and implementing a sophisticated machine learning sentiment classifier model which makes use of some novel ideas and appears to perform very well in various tests.
- *What are the limitations of this project?* There are some datasets which the classifier seems to struggle on, such as the SEN dataset of news headlines. It also lacks the fine-grained classification power of aspect-based models. Additionally, the autoencoder model is limited to binary classification due to its architecture, meaning it can only take two types of input (positive and negative articles).

Acknowledgements

The author thanks project supervisor Sara Kalvala for excellent guidance and supervision throughout the project. Appreciation goes to friends and family for in proof-reading this report.

Appendices

Data Visualisation Figures

The Guardian sentiment map

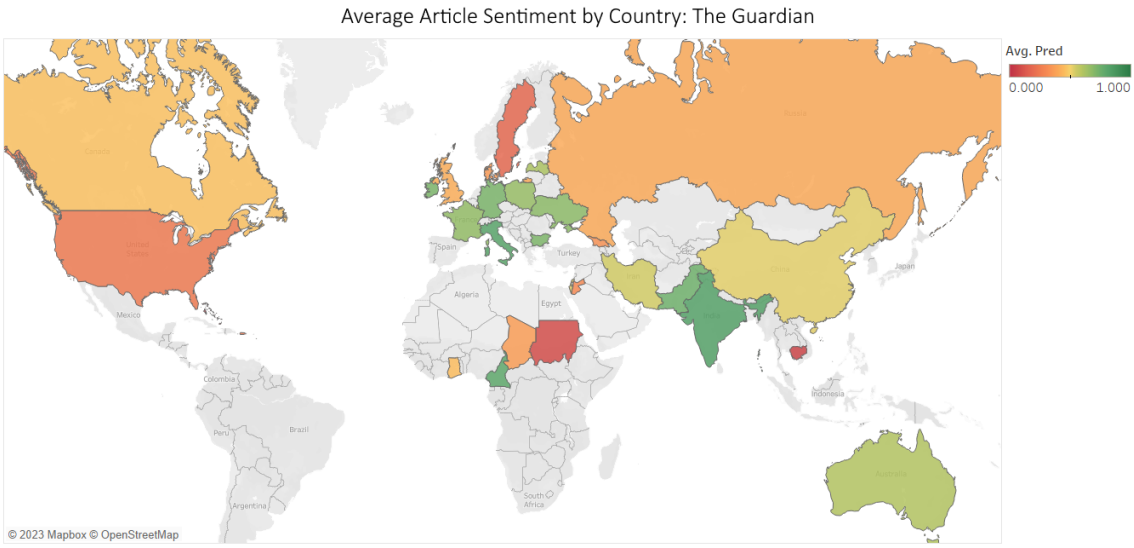

Figure 35: Map of sentiment in The Guardian articles, filtered by geographical location.

The Telegraph sentiment map

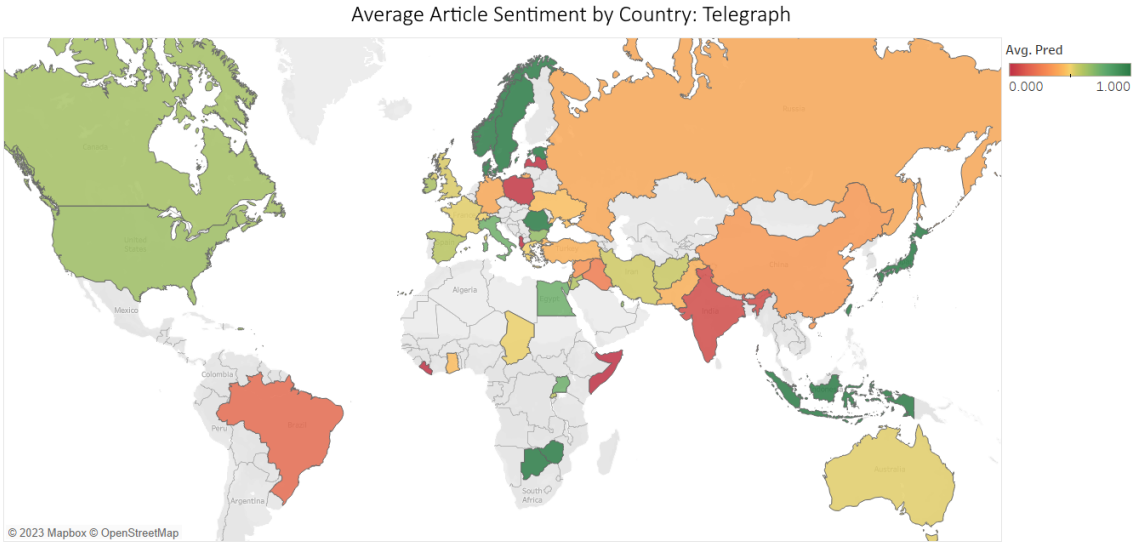

Figure 36: Map of sentiment in Telegraph articles, filtered by geographical location.

Binary accuracy evaluation

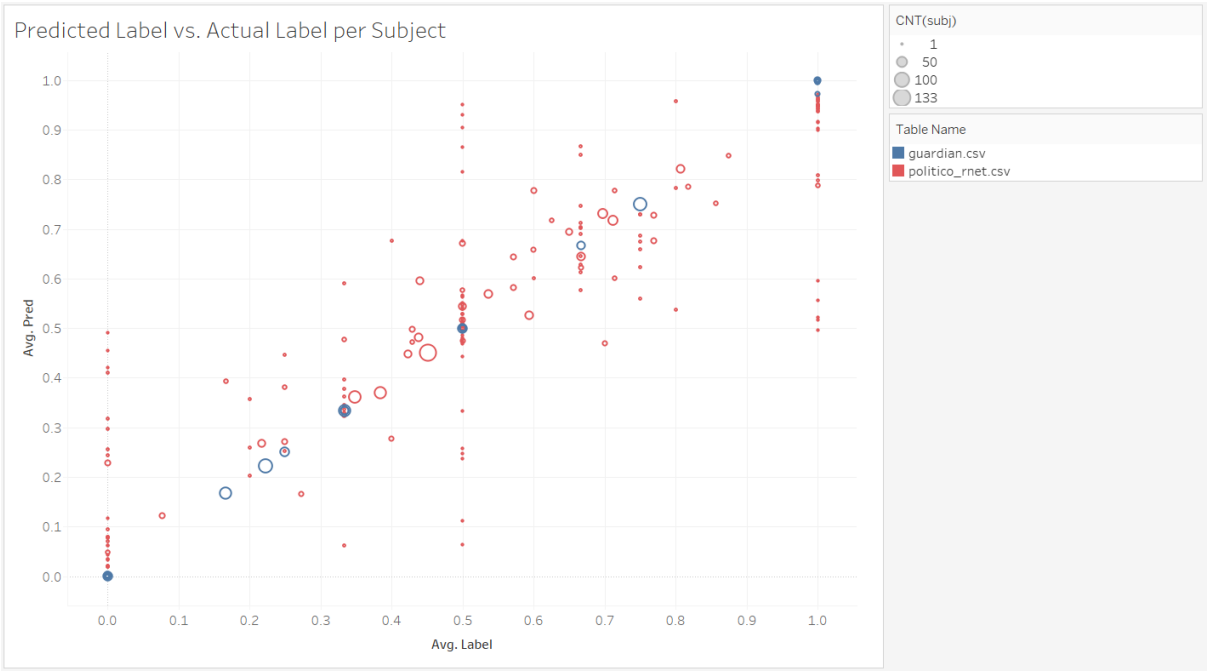

Figure 37: Graph showing the correlation between predicted and actual sentiment label for each subject in two datasets.

Continuous accuracy evaluation

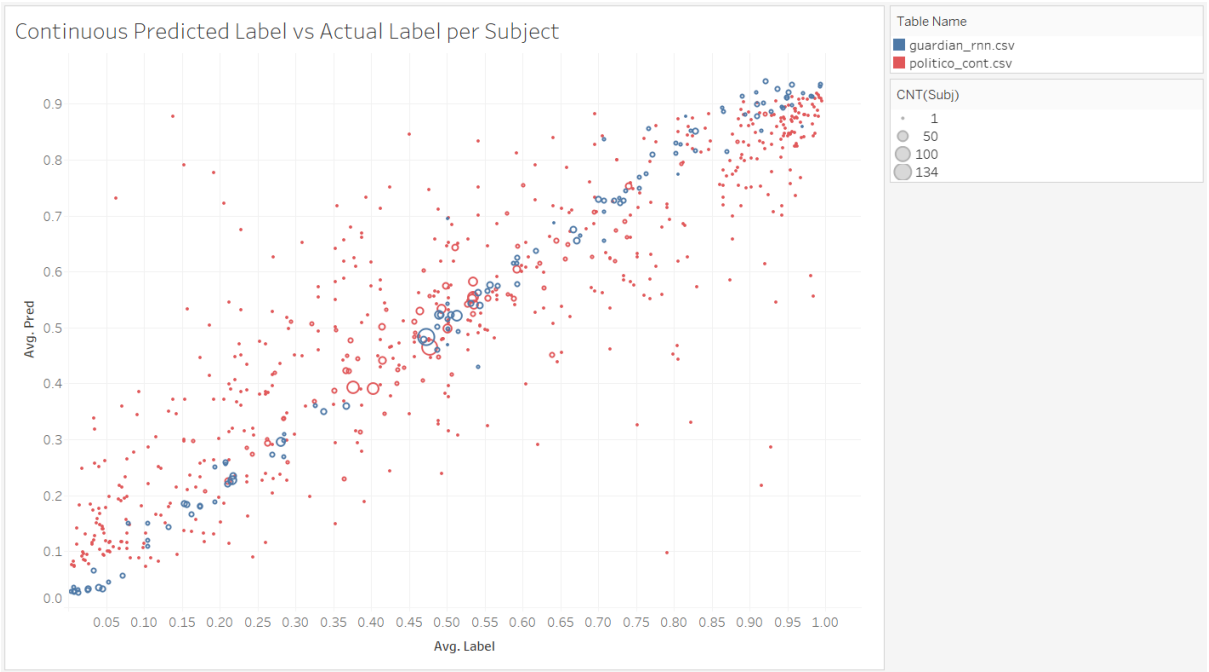

Figure 38: Graph showing the predicted and actual label using the continuous label predictor.

Model Accuracy Report Results

The Guardian dataset

Model	TF-IDF accuracy	Doc2Vec accuracy
Logistic regression	64%	62%
Random forest	61%	60%
Support vector machine	65%	59%
Vanilla FFN	70%	-
L_2 regularised FFN	67%	-
dropout regularised FFN	71%	-
L_2 + dropout FFN	74%	-
Shallow RNN	50%	61%
Deep RNN	-	63%
Embedded bidirectional LSTM	-	86%
LSTM + integer vectors	99%	-
Autoencoder + random forest	98%	-
Autoencoder + deep RNN	94%	-

Figure 39: Accuracy results for each model over TF-IDF and Doc2Vec vector representations of the Guardian dataset.

Politico dataset

Model	TF-IDF accuracy	Doc2Vec accuracy
Logistic regression	75%	51%
Random forest	69%	57%
Support vector machine	76%	58%
Vanilla FFN	74%	-
L_2 regularised FFN	72%	-
dropout regularised FFN	73%	-
L_2 + dropout FFN	73%	-
Shallow RNN	49%	61%
Deep RNN	-	64%
Embedded bidirectional LSTM	-	53%
LSTM + integer vectors	76%	-
Autoencoder + random forest	95%	-
Autoencoder + deep RNN	71%	-

Figure 40: Accuracy results for each model over TF-IDF and Doc2Vec vector representations of the Politico dataset.

SEN headlines dataset

Model	TF-IDF accuracy	Doc2Vec accuracy
Logistic regression	75%	45%
Random forest	70%	55%
Support vector machine	73%	48%
Vanilla FFN	65%	-
L_2 regularised FFN	60%	-
dropout regularised FFN	53%	-
L_2 + dropout FFN	58%	-
Shallow RNN	50%	47%
Deep RNN	-	47%
Embedded bidirectional LSTM	-	48%
LSTM + integer vectors	76%	-
Autoencoder + random forest	98%	-
Autoencoder + deep RNN	78%	-

Figure 41: Accuracy results for each model over TF-IDF and Doc2Vec vector representations of the SEN dataset.

References

[1] Gensim: topic modelling for humans. URL https://radimrehurek.com/gensim/auto_examples/tutorials/run_doc2vec_lee.html.

[2] Papers with code - imdb movie reviews dataset. URL https://paperswithcode.com/dataset/imdb-movie-reviews.

[3] Stock sentiment analysis using autoencoders. URL https://www.nbshare.io/notebook/84089365/Stock-Sentiment-Analysis-Using-Autoencoders/.

[4] Terms of service. URL https://help.nytimes.com/hc/en-us/articles/115014893428-Terms-of-Service.

[5] Politico: write for us. URL https://www.politico.com/write-for-us.

[6] Methodology - media bias/fact check, 2017. URL https://mediabiasfactcheck.com/methodology/.

[7] Data analytics with nlp & text analytics | lexalytics, 2019. URL https://www.lexalytics.com/.

[8] Word2vec, 2019. URL https://code.google.com/archive/p/word2vec/.

[9] Cordis | european commission, 2023. URL https://cordis.europa.eu/project/id/231323/EN.

[10] M. Adil, R. Ullah, S. Noor, and N. Gohar. Effect of number of neurons and layers in an artificial neural network for generalized concrete mix design. *Neural Computing and Applications*, 09 2020. doi: 10.1007/s00521-020-05305-8.

[11] I. Akhmetov, A. Gelbukh, and R. Mussabayev. Topic-aware sentiment analysis of news articles. *Computación y Sistemas*, 26, 03 2022. doi: 10.13053/cys-26-1-4179.

[12] N. Anstead and B. O'Loughlin. Semantic polling: the ethics of online public opinion, 05 2012. URL https://eprints.lse.ac.uk/46944/.

[13] N. Anstead and B. O'Loughlin. The emerging viewertariat and bbc question time. *The International Journal of Press/Politics*, 16:440–462, 07 2011. doi: 10.1177/1940161211415519.

[14] S. Baccianella, A. Esuli, and F. Sebastiani. Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining, 2010. URL http://lrec-conf.org/proceedings/lrec2010/pdf/769_Paper.pdf.

[15] K. Baraniak and M. Sydow. A dataset for sentiment analysis of entities in news headlines (sen). *Procedia Computer Science*, 192:3627–3636, 2021. doi: 10.1016/j.procs.2021.09.136.

[16] B. Barnhart. The importance of social media sentiment analysis (and how to conduct it), 03 2019. URL https://sproutsocial.com/insights/social-media-sentiment-analysis/.

[17] S. Behdenna, F. Barigou, and G. Belalem. Document level sentiment analysis: A survey. *EAI Endorsed Transactions on Context-aware Systems and Applications*, 4:154339, 03 2018. doi: 10.4108/eai.14-3-2018.154339.

[18] B. Bhutani, N. Rastogi, P. Sehgal, and A. Purwar. Fake news detection using sentiment analysis, 08 2019. URL https://ieeexplore.ieee.org/document/8844880.

[19] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29:1157–1166, 09 1997. doi: 10.1016/s0169-7552(97)00031-7.

[20] R. Cellan-Jones. Bbc - dot.rory: Sentiment, social media, and the leaders, 04 2010. URL https://www.bbc.co.uk/blogs/thereporters/rorycellanjones/2010/04/sentiment_social_media_and_the.html.

[21] Y. Chen and M. J. Zaki. Kate. *Knowledge Discovery and Data Mining*, 08 2017. doi: 10.1145/3097983.3098017.

[22] L. Chierotti. Harvard professor says 9503 2018. URL https://www.inc.com/logan-chierotti/harvard-professor-says-95-of-purchasing-decisions-are-subconscious.html.

[23] M. Cinelli, G. D. F. Morales, A. Galeazzi, W. Quattrociocchi, and M. Starnini. The echo chamber effect on social media. *Proceedings of the National Academy of Sciences*, 118, 03 2021. doi: 10.1073/pnas.2023301118. URL https://www.pnas.org/doi/10.1073/pnas.2023301118.

[24] L. Com and G. Hinton. Visualizing data using t-sne laurens van der maaten. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL https://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf?fbcl.

[25] S. Cornegruta, R. Bakewell, S. Withey, and G. Montana. Modelling radiological language with bidirectional long short-term memory networks. *arXiv:1609.08409 [cs, stat]*, 09 2016. URL https://arxiv.org/abs/1609.08409.

[26] C. Cortes, M. Mohri, and A. Rostamizadeh. L2 regularization for learning kernels. *arXiv:1205.2653 [cs, stat]*, 05 2012. URL https://arxiv.org/abs/1205.2653#.

[27] G. H. de Rosa and J. P. Papa. A survey on text generation using generative adversarial networks. *Pattern Recognition*, 119:108098, 11 2021. doi: 10.1016/j.patcog.2021.108098.

[28] R. V. Dongen. Mitt, paul winning facebook primary, 01 2012. URL https://www.politico.com/story/2012/01/mitt-paul-winning-facebook-primary-071345.

[29] S. Frassetto. You should try the new tensorflow's textvectorization layer., 08 2020. URL https://towardsdatascience.com/you-should-try-the-new-tensorflows-textvectorization-layer-a80b3c6b00ee.

[30] Y. Goldberg and O. Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method, 2014. URL https://arxiv.org/abs/1402.3722.

[31] B. Gunter, N. Koteyko, and D. Atanasova. Sentiment analysis: A market-relevant and reliable measure of public feeling? *International Journal of Market Research*, 56:231–247, 03 2014. doi: 10.2501/ijmr-2014-014.

[32] F. Hamborg and K. Donnay. Newsmtsc: A dataset for (multi-)target-dependent sentiment classification in political news articles. *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, 2021. doi: 10.18653/v1/2021.eacl-main.142.

[33] IBM. What are recurrent neural networks? | ibm. URL https://www.ibm.com/topics/recurrent-neural-networks.

[34] IBM. Data visualization, 2019. URL https://www.ibm.com/topics/data-visualization.

[35] M. U. Islam, F. B. Ashraf, A. I. Abir, and M. A. Mottalib. Polarity detection of online news articles based on sentence structure and dynamic dictionary. *2017 20th International Conference of Computer and Information Technology (ICCIT)*, 12 2017. doi: 10.1109/iccitechn.2017.8281777.

[36] A. Karimi. Nlp's word2vec: Negative sampling explained, 03 2023. URL https://www.baeldung.com/cs/nlps-word2vec-negative-sampling.

[37] K. Kawakami and A. Graves. Supervised sequence labelling with recurrent neural networks, 2008. URL https://pdfs.semanticscholar.org/a97b/5db17acc731ef67321832dbbaf5766153135.pdf.

[38] T. Kenter, A. Borisov, and M. de Rijke. Siamese cbow: Optimizing word embeddings for sentence representations. *arXiv:1606.04640 [cs]*, 06 2016. URL https://arxiv.org/abs/1606.04640#.

[39] K. LANG and G. E. LANG. The impact of polls on public opinion. *The ANNALS of the American Academy of Political and Social Science*, 472:129–142, 03 1984. doi: 10.1177/0002716284472001012.

[40] J. H. Lau and T. Baldwin. An empirical evaluation of doc2vec with practical insights into document embedding generation. *arXiv:1607.05368 [cs]*, 07 2016. URL https://arxiv.org/abs/1607.05368.

[41] Q. Le and T. Mikolov. Distributed representations of sentences and documents, 06 2014. URL http://proceedings.mlr.press/v32/le14.html?ref=https://githubhelp.com.

[42] B. Li, Z. Zhao, T. Liu, P. Wang, and X. Du. Weighted neural bag-of-n-grams model: New baselines for text classification, 12 2016. URL https://aclanthology.org/C16-1150/.

[43] D. Li and J. Qian. Text sentiment analysis based on long short-term memory, 10 2016. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7778967.

[44] H.-K. H. Li-Ping Jing and H.-B. Shi. Improved feature selection approach tfidf in text mining. *Proceedings. International Conference on Machine Learning and Cybernetics*, 11 2002. doi: 10.1109/icmlc.2002.1174522.

[45] H. Liang, X. Sun, Y. Sun, and Y. Gao. Text feature extraction based on deep learning: a review. *EURASIP Journal on Wireless Communications and Networking*, 2017, 12 2017. doi: 10.1186/s13638-017-0993-1.

[46] C.-Y. Liou, W.-C. Cheng, J.-W. Liou, and D.-R. Liou. Autoencoder for words. *Neurocomputing*, 139:84–96, 09 2014. doi: 10.1016/j.neucom.2013.09.055. URL https://www.sciencedirect.com/science/article/pii/S0925231214003658.

[47] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis, 06 2011. URL https://aclanthology.org/P11-1015/.

[48] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space, 2013. URL https://arxiv.org/pdf/1301.3781.pdf.

[49] D. Mishra, B. Naik, R. M. Sahoo, and J. Nayak. Deep recurrent neural network (deep-rnn) for classification of nonlinear data. *Advances in intelligent systems and computing*, pages 207–215, 01 2020. doi: 10.1007/978-981-15-2449-3_17.

[50] G. Nicola. Text generation with a variational autoencoder, 2013. URL https://nicgian.github.io/text-generation-vae/.

[51] I. Nusrat and S.-B. Jang. A comparison of regularization techniques in deep neural networks. *Symmetry*, 10:648, 11 2018. doi: 10.3390/sym10110648.

[52] B. Pang, L. Lee, and S. Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques, 2019. URL https://arxiv.org/abs/cs/0205070.

[53] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio. How to construct deep recurrent neural networks. *arXiv:1312.6026 [cs, stat]*, 04 2014. URL https://arxiv.org/abs/1312.6026.

[54] J. Pustejovsky and A. Stubbs. *Natural Language Annotation for Machine Learning: A Guide to Corpus-Building for Applications*. "O'Reilly Media, Inc.", 10 2012. URL https://books.google.co.uk/books?hl=en&lr=&id=A57TS7fs8MUC&oi=fnd&pg=PR2&dq=pustejovsky+and+stubbs+2012&ots=SKirrJ3zvH&sig=QnbCW8yOQbNeFOq7dtc5F9svca8#v=onepage&q=pustejovsky%20and%20stubbs%202012&f=false.

[55] Y. Rao, J. Lei, L. Wenyin, Q. Li, and M. Chen. Building emotional dictionary for sentiment analysis of online news. *World Wide Web*, 17:723–742, 06 2013. doi: 10.1007/s11280-013-0221-9. URL https://link.springer.com/article/10.1007/s11280-013-0221-9.

[56] M. Rhanoui, M. Mikram, S. Yousfi, and S. Barzali. A cnn-bilstm model for document-level sentiment analysis. *Machine Learning and Knowledge Extraction*, 1:832–847, 07 2019. doi: 10.3390/make1030048.

[57] M. Riva. Word embeddings: Cbow vs skip-gram | baeldung on computer science, 03 2021. URL https://www.baeldung.com/cs/word-embeddings-cbow-vs-skip-gram.

[58] A. Roy. Introduction to autoencoders, 12 2020. URL https://towardsdatascience.com/introduction-to-autoencoders-7a47cf4ef14b.

[59] A. Samuels and J. Mcgonical. News sentiment analysis. *arXiv:2007.02238 [cs]*, 07 2020. URL https://arxiv.org/abs/2007.02238#.

[60] F. P. Shah and V. Patel. A review on feature selection and feature extraction for text classification. *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, 03 2016. doi: 10.1109/wispnet.2016.7566545.

[61] J. Shin, L. Jian, K. Driscoll, and F. Bar. The diffusion of misinformation on social media: Temporal pattern, message, and source. *Computers in Human Behavior*, 83:278–287, 06 2018. doi: 10.1016/j.chb.2018.02.008. URL https://www.sciencedirect.com/science/article/pii/S0747563218300669#bib50.

[62] K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu. Fake news detection on social media: A data mining perspective. *ACM SIGKDD Explorations Newsletter*, 19:22–36, 09 2017. doi: 10.1145/3137597.3137600.

[63] K. Shu, D. Mahudeswaran, and H. Liu. Fakenewstracker: a tool for fake news collection, detection, and visualization. *Computational and Mathematical Organization Theory*, 25:60–71, 10 2018. doi: 10.1007/s10588-018-09280-3.

[64] G. Sidorov. Syntactic n-grams in computational linguistics, 2019. URL https://link.springer.com/content/pdf/10.1007/978-3-030-14771-6.pdf.

[65] L. Singh. Clustering text: A comparison between available text vectorization techniques. *Advances in intelligent systems and computing*, pages 21–27, 01 2022. doi: 10.1007/978-981-16-1249-7_3.

[66] H. Son and C. Kim. A deep learning approach to forecasting monthly demand for residential–sector electricity. *Sustainability*, 12:3103, 04 2020. doi: 10.3390/su12083103.

[67] R. C. Staudemeyer and E. R. Morris. Understanding lstm – a tutorial into long short-term memory recurrent neural networks. *arXiv:1909.09586 [cs]*, 09 2019. URL https://arxiv.org/abs/1909.09586.

[68] S. Wager, S. Wang, and P. S. Liang. Dropout training as adaptive regularization, 2013. URL https://proceedings.neurips.cc/paper/2013/hash/38db3aed920cf82ab059bfccbd02be6a-Abstract.html.

[69] Y. Wang, H. Yao, and S. Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, 04 2016. doi: 10.1016/j.neucom.2015.08.104.

[70] C. Wu, F. Wu, S. Wu, Z. Yuan, J. Liu, and Y. Huang. Semi-supervised dimensional sentiment analysis with variational autoencoder. *Knowledge-Based Systems*, 165:30–39, 02 2019. doi: 10.1016/j.knosys.2018.11.018.

[71] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. Le. Xlnet: Generalized autoregressive pretraining for language understanding, 2019. URL https://arxiv.org/pdf/1906.08237v2.pdf.

[72] A. Younus, M. A. Qureshi, S. K. Kingrani, M. Saeed, N. Touheed, C. O'Riordan, and P. Gabriella. Investigating bias in traditional media through social media. *Proceedings of the 21st international conference companion on World Wide Web - WWW '12 Companion*, 2012. doi: 10.1145/2187980.2188168.

[73] S. Zhai and Z. Zhang. Semisupervised autoencoder for sentiment analysis. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30, 02 2016. doi: 10.1609/aaai.v30i1.10159.

[74] S. Žitnik, N. Blagus, and M. Bajec. Target-level sentiment analysis for news articles. *Knowledge-Based Systems*, page 108939, 05 2022. doi: 10.1016/j.knosys.2022.108939.