Detecting Interests in Social Media

Joshua Fitzmaurice
Department of Computer Science
University of Warwick

Supervised by Arshad Jhumka
Year of Study: 3rd

1 May 2023

Acknowledgement

I would like to thank my supervisor, Dr Arshad Jhumka, for his guidance and support throughout this project. I would also like to thank Tomás Freitas Fernandes for his help with debugging errors on DCS batch compute nodes.

Keywords

Natural Language Processing, Social Media, Topic Modelling, Classification, Short Text Classification, Context, Long Short Term Memory, Recurrent Neural Networks, Bidirectional Encoder Representations from Transformers, Robustly Optimized BERT Approach.

Abstract

Thanks to recommender systems it is common that a user's social media feed will be filled with posts relating to others they have previously interacted with. In this project it is viewed that this can lead to a user being shown posts that are biased towards their own opinions. This project aims to identify topics that users are interested in based off of their social media feed.

First, a comparison of different Natural Language Processing models is completed to identify what model is best suited for the task of classifying social media posts into topics (sports, politics, etc.). It was found that RoBERTa outperformed RNNs, LSTMs, and BERT. RoBERTa was then fine-tuned on Reddit and Wikipedia data for this classification problem. This resulted in a model with a 71% accuracy.

Next, the project attempts to improve this accuracy by including context to aid with the classification process. The context that was included was the media attached to the post as well as the comments/threads that the post was a part of. This resulted in around a 20% improvement in accuracy.

Finally, the project works on creating a User Interface that allows a user to view the difference in topics they see on social media compared to what is commonly available across the whole of the social media site. This allows users to gauge what topics they are interested in. On top of this users are able to search for posts relating to any topics of their choice.

Contents

1 Introduction **6**

1.1 Motivation 6

1.1.1 What got me interested 6

1.1.2 Where the problem lies 8

1.1.3 Why is this a problem? 9

1.2 Problem Statement 10

1.3 Topics 10

1.4 Related work 11

1.4.1 Latent Dirichlet Allocation - (7) 11

1.4.2 Pythia - (8) 11

1.4.3 Deep Short Text Classification with Knowledge Powered Attention - (9) 14

1.4.4 Topic tracking of student-generated posts - (12) 16

1.4.5 Topic classification of blogs - (13) 16

1.5 Objectives 16

2 Background **18**

2.1 Topic Modelling 18

2.1.1 Unsupervised Learning 18

2.1.2 Supervised Learning 19
2.2 Text Classification 21
2.2.1 Recurrent Neural Networks (RNN) 21
2.2.2 Long Short-Term Memory (LSTM) - (23) 23
2.2.3 Transformers 24
2.2.4 Bidirectional Encoder Representation of Transformers (BERT) - (25) 27
2.2.5 Robustly Optimised BERT Approach (RoBERTa) - (26) . . 29

3 Design **30**
3.1 Topic Classification 30
3.2 Adding Context 31
3.2.1 What is context? and why is it important? 31
3.2.2 Methods for adding context 31
3.2.3 Context Aware Model 32
3.3 Python Application 33
3.3.1 Requirements Analysis 33
3.3.2 Frontend 37
3.3.3 Backend 38

4 Implementation **42**
4.1 Data Collection 42
4.1.1 Preprocessing 43
4.1.2 Wikipedia Data 43
4.1.3 Reddit Data 46
4.2 RNN and LSTM 47
4.3 BERT 48

4.4 RoBERTa 49
4.5 Context Aware Model 50
4.5.1 Setup tweepy 51
4.5.2 Get Tweets 51
4.5.3 Feeding into RoBERTa 52
4.5.4 Problems with Context Model 54
4.6 Python Application 55
4.6.1 Frontend 55
4.6.2 Backend 58

5 Evaluation **63**
5.1 RNN and LSTM 63
5.2 BERT vs RoBERTa 65
5.3 Principal Component Analysis (PCA) of data 66
5.3.1 Findings from PCA 67
5.4 Context 72
5.4.1 Media - Images and Videos 72
5.4.2 Retweets and Threads 73
5.4.3 Context Aware Conclusion 75
5.5 Python Application 75

6 Project Management **76**
6.1 Risk Management 76
6.2 Implementation Strategy 76
6.2.1 Model Creation 76
6.2.2 Python Application 77
6.3 Time Management 79

6.4 Resource Management 79
6.5 Testing 80
6.5.1 Model Testing 80
6.5.2 Unit Testing 81
6.5.3 Integration Testing 81
6.5.4 System Testing 81

7 Conclusions **82**
7.1 Future work 83
7.1.1 Advanced Context Input 83
7.1.2 Chrome Extension 84
7.1.3 Bias Analysis 84

A Code listings **92**

B Specification **95**

Chapter 1

Introduction

1.1 Motivation

1.1.1 What got me interested

When deciding what to do for my dissertation, I wanted to work on something that affects me on a daily basis. Most days I spend some time on social media so I asked myself the question, "What can I do with social media?" The answer hit me when looking at my instagram for you page (fig. 1.1).

The content being displayed is all very similar; there is no diversity in the content. This made me ask 3 questions:

- What is causing the non-diversity in the content?
- Is there any information I can establish about myself from the content?
- Is there a way I can force the platform to show me different content?

Questions 1.1: Main questions for the project

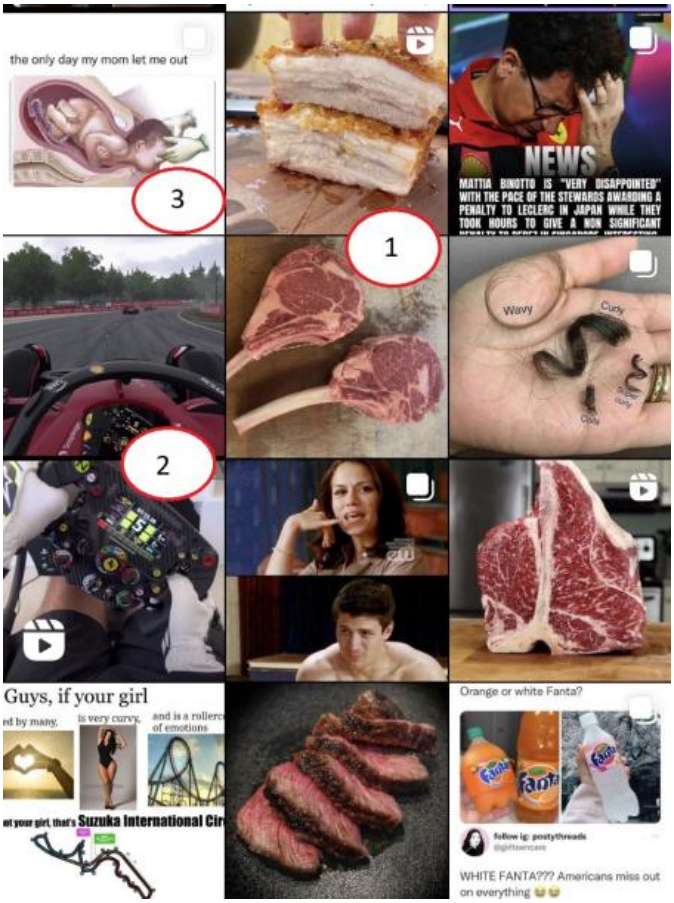

Figure 1.1: Example tweet

1.1.2 Where the problem lies

Answering the first question (Questions 1.1) was rather simple after some research. Social media wites use recommender systems to show users content they are likely to be "interested" in (1). Recently Twitter made their recommender system public.

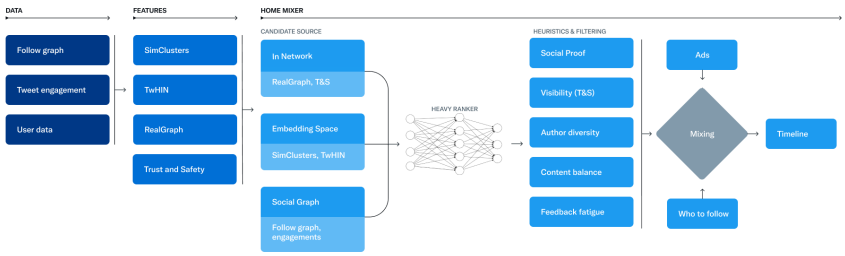

Figure 1.2: Twitter's recommender system

fig. 1.2 briefly shows how Twitter's recommender system works. The main system is split into three parts; the first part is 'Candidate Sourcing' which selects around 1500 tweets (50% 'in-network' and 50% 'out-of-network'). Secondly, these posts are ranked using a large neural network. Finally, some heuristics and filters are applied to the ranking.
There is already areas here that could cause users to constantly see similar content. This is due to the fact candidate sourcing is 50% 'in-network'; users are likely to see content created by people they follow. Although this seems mild (due to the other 50% being 'out-of-network') this doesn't necessarily balance out someones feed as the method used to identify candidate 'out-of-network' tweets relies on answering the question "Who likes similar Tweets to me, and what else have they recently liked?" (1). Diving deeper into the heuristics and filters, it is possible to identify another area which would cause users to see similar content - 'Feedback-based Fatigue'. This is where users are not shown posts similar to posts they have negatively interacted with prior.
If a user views a certain type of post they are likely to view similar posts in the future (1). With this in mind, it should be possible to identify a users interests through their social media posts. As a regular user of social media platforms,

I have noticed that the content I see on my feed is always very similar (although may change over larger periods of time). Noticing this, I decided to play around with how I can force the social media platform to show me differing content. I noticed that (on Instagram) if I simply chose specific posts to look at on my 'for you' page, these posts would be shown to me more often. This shows promise that it is possible to answer the 3rd question (Questions 1.1). This experimentation was a large motivating factor for the original aims of this project set out in my specification (appendix B) - to analyse how differing 'strategies' (ways in which someone uses social media) for using social media affect the content shown to the user. However, due to Terms of Service restrictions, this was not possible. Instead, this project will focus on identifying a users interests through their social media posts. This was an easy transition as the original project aims required the same prerequisites as this project; both projects need to be able to classify social media posts into topics. Another benefit of this project is that given permission from social media platforms in the future, it will be possible to extend this project to analyse 'strategies' as set out in the original project aims.

1.1.3 Why is this a problem?

Because recommender systems tend to show users content they have (positively) interacted with in the past echo chambers can be created. Echo chambers are where the ideas and beliefs of a group of people are reinforced through confirmation bias (2). When a user is scrolling through their social media feed they are likely to pick out posts that they agree with and like/share them (3)(4). This interaction with the post will cause the recommender system to show more of this type of content to the user. Another, although similar, problem is the spread of misinformation. This is where content that is false is spread to a large number of people (4)(5). These people then like/share the content which causes more of the post to be shown to more people. Which can very quickly spread false information to a large number of people.
If it is possible for users to identify how their social media content leans (e.g. political leaning) it allows them to identify whether they are in an echo cham-

ber and then take steps to resolve this issue.

1.2 Problem Statement

The main crux of this project is attempting to answer the 2nd question (Questions 1.1). Using the idea that recommender systems will show users content that they believe the user will be interested in (1), it should be possible to identify a users interests through their social media feed. This project works on classifying social media posts. This will allow us to identify interests through comparing similarities between posts shown to the user. This will be followed by quantifying a set of posts to compare the similarity between a users posts and posts from the entire social media site. Finally, this project aims to create a user interface that allows users to discover what their social media feed says about their interests and how they compare to the rest of the social media site. The user interface will also allow users to discover posts that are dissimilar to the posts they are shown.

1.3 Topics

This project will use the notion of topics to identify what the posts are about.

a subject that is discussed, written about, or studied

Figure 1.3: Topic Definition - (6)

This definition of a topic is a good starting point for this project. This definition is a bit too specific. If this definition were to be used there would be no meaningful output from this project - due to the fact you could classify each post into its own unique topic. To overcome this, we will use a more general definition of a topic. Essentially, a topic comprises of a set of words that are related to each other and reflect a common theme/subject. For example, the topic of ‘food’ would contain words like ‘steak’, ‘chicken’, ‘pizza’ etc.

1.4 Related work

1.4.1 Latent Dirichlet Allocation - (7)

Latent Dirichlet Allocation (LDA) is a generative probabilistic model that is used to classify documents into topics (7). LDA aims to find a set of topics that are representative of the documents in a corpus. It does this by assigning each word in a document to a topic and then assessing the following probabilities:

- $p(\text{topic t}|\text{document d})$ - The probability of a topic being assigned to a document
- $p(\text{word w}|\text{topic t})$ - The probability of a word being assigned to a topic

LDA then uses Gibbs sampling to assign each word in a document to a topic based on the above probabilities. Below is a simplified version of the LDA algorithm to give some intuition on how it works

$$p(\text{word w with topic t}) = p(\text{topic t}|\text{document d}) * p(\text{word w}|\text{topic t}) \quad (1.1)$$

LDA is a very powerful unsupervised learning technique that can be used to classify documents into topics. However, it has some limitations (due to it being an unsupervised technique) that will be discussed in section 2.1.

1.4.2 Pythia - (8)

Pythia is an automated system for short text classification. It makes use of Wikipedia structure and articles to identify topics of posts. Essentially, "Wikipedia contains articles organized in various taxonomies, called categories". Pythia then goes on to use this information as their training data as well as handling sparseness in posts on social media.

Pythia also demonstrates a method to overcome the lack of context in short texts - This is a large problem in identifying smaller social media posts like

tweets, and will be further worked on in this project. They use a method called "Post Enrichment", which performs i) Named Entity Recognition then ii) Lemmatization and stop word removal. We then use the named entities to query wikipedia for similar articles that are then appended to the post.

Although this method works well in cases where keywords are used, there are cases where no keywords are used, and more context is needed. Take for example the following tweet:

Figure 1.4: Example tweet

The text gives us very little context; What is this tweet about? the best guess could be is it is a message to other users. If these users had wikipedia articles, we could use them. But in reality this post is about something else. Lets add some other form of context; add the media the tweet contains.

Figure 1.5: Example tweet - media

This gives us a lot more context; the tweet is discussing the earthquake that hit Turkey and Syria, if we use this for our query we get more relevant articles:

Figure 1.6: Wikipedia articles relating to the Syria Turkey earthquake

We now have a lot of relevant information to append to the post. We could use other information for context as well: Tweet author, images/media in tweets, retweet information, like information, etc.

1.4.3 Deep Short Text Classification with Knowledge Powered Attention - (9)

Following on from the work of adding context in section 1.4.2, this paper proposes a more sophisticated method of adding conceptual information via Knowledge Bases (KBs). Knowledge Bases are a store of information/rules that an AI/ML model can use to make decisions (10). A knowledge base system isn't programmed to solve a specific problem, but rather it is given a set of declarative rules that it can use to solve a problem (10). They make use of a method called "Knowledge Powered Attention" (KPA) to incorporate KBs into their model. What is Knowledge Powered Attention?

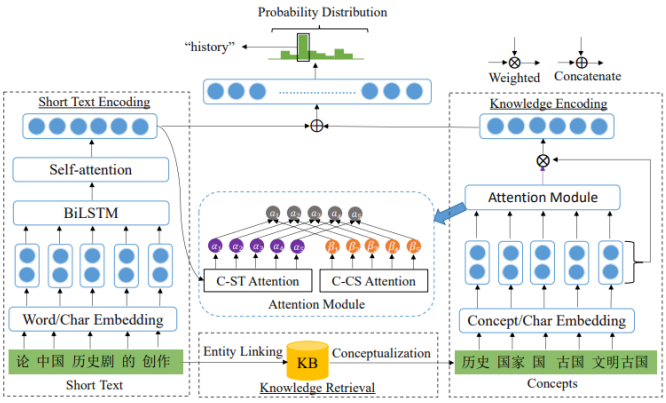

Figure 1.7: Knowledge Powered Attention architecture

fig. 1.7 show the architecture proposed by the paper. There are several separated modules, but to understand how Knowledge is used, the focus will be on:

- Knowledge Retrieval
- Attention Model

Knowledge Retrieval

The premise of this module is to take a short text s and retrieve a 'concept set' C. There are two steps to this:

1. Entity Linking
2. Concept Set Retrieval

Entity Linking is used to identify the entities mentioned in the short text (9). Concept Set Retrieval uses the entities found in the previous step and queries a KB to find the concepts related to the entities.

Attention Model

The attention model comprises of 2 separate attention mechanisms called: 'Concept towards Short Text (C-ST) Attention' and 'Concept towards Concept Set (C-CS) Attention'.
C-ST attention is put in place to attempt to reduce the bad influence from improper concepts in the concept set. It does this by measuring the semantic similarity between the concepts and the short text. The paper uses 'vanilla' attention and does not elaborate what is meant by this. Nonetheless, the idea of measuring similarity of a set based on information from another set could perform better using cross-attention mechanisms. Cross-attention could work well due to the fact it uses the set being compared against to fill the 'Key' and 'Value' vectors. Then use the set being compared as the 'Query' (11). This in-built separation shows how cross-attention could work well in this case.
C-CS attention is put in place to measure how important each concept is relative to the other concepts. It uses basic self-attention to do this.
The two attention mechanisms are then combined to form the final attention vector used for the knowledge embeddings using the equation in eq. (1.2).

$$a_i = softmax(\gamma\alpha_i + (1-\gamma)\beta_i) \tag{1.2}$$

where γ is a hyperparameter to control the importance of each attention mechanism.

1.4.4 Topic tracking of student-generated posts - (12)

This paper proposes a solution for determining valuable information/topics discussed in student forums on online courses. It uses a model called "Time Information-Emotion Behaviour Model" or otherwise called "TI-EBTM" to detect key topics discussions , keeping in mind the progress of time throughout the forum.
The concept that time is important in determining the topic of a post is a good one. It could be possible to incorporate a time-sensitive model into the project to determine the topic of a post. This is not done in this project but is a good idea for future work.

1.4.5 Topic classification of blogs - (13)

This paper uses Distant Supervision - 'an extension of the paradigm used by (14) for exploiting WordNet to extract hypernym (is-a) relations between entities' - to get training data via Wikipedia articles. Then trains their own designed model on this data to be able to classify topics via a multi-class recognition model (69% accuracy) and via a binary classification model (90% accuracy).

1.5 Objectives

To achieve the problem statement, the following objectives must be met:

- Generate a list of topics for classification
- Implement methods for identifying the topics in social media posts
- Compare and contrast the results of the different methods
- Create a user interface
  - Allow users to compare their interests to the rest of the social media site

- Allow users to discover posts that are dissimilar to the posts they are shown

The largest problem with this project is the second objective of finding and creating methods for identifying topics.

Chapter 2

Background

2.1 Topic Modelling

As discussed in Chapter 1, this project uses topics to identify what the posts are about. This section will discuss how topics are created and how they are used in this project.
There are 2 methods of topic modelling that were considered for this project. The first method is Unsupervised Learning, and the second method is Supervised Learning.

2.1.1 Unsupervised Learning

Unsupervised learning is a method of machine learning that does not require labelled data. This method is used to identify patterns in data (15). For this project, the patterns are topics.
Using gensim LDA (16) with pyLDAvis (17), and BERTTOPIC (18), topics were identified from a set of unlabelled documents.

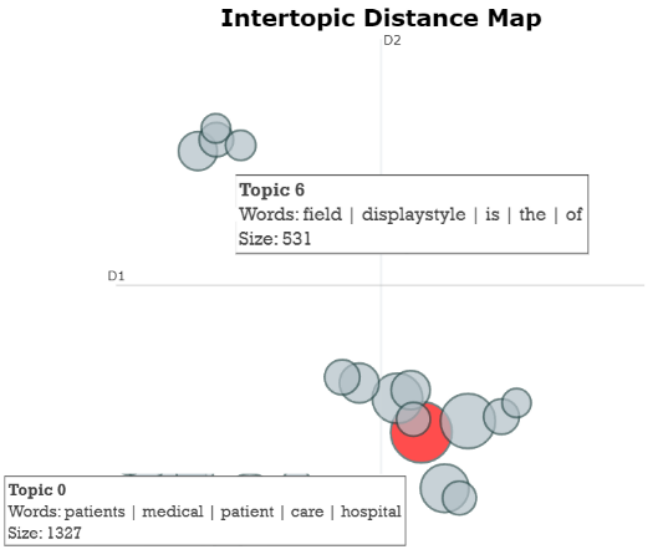

Figure 2.1: LDATopic creating topics from a set of unlabelled documents

The benefits of using unsupervised learning are that it does not require labelled data (15), it is easy to implement, and the classification of posts with the generated topics should be accurate - as the topics are generated to be separable. The downsides of using unsupervised learning are that the topics generated may not be meaningful; Take for example Topic 6 from Figure 2.1. This topic is made up of the words 'displaystyle', 'is', 'the', and 'of'. It is quite hard for a human to understand what this topic is about.

2.1.2 Supervised Learning

Supervised learning is a method of machine learning that requires labelled data. The labelled data acts as a teacher to the model, and the model learns from the labelled data (15).
For this project, a set of topics can be manually created and then used to label the posts. This method is much easier to implement than unsupervised learning, as the topics are already created. The downside of this method is that it requires labelled data, which can be time consuming to create.

For this project, supervised learning was used to classify the posts into the topics that were created. The original set of topics were:

- **Culture**
- **Entertainment**
- **News**
- **Philosophy**
- **Religion**
- **Science**
- **Sports**
- **Technology**
- **Law**
- **History**
- **Geography**
- **Video Games**
- **Music**
- **Medicine**
- **Business**
- **Foods**
- **Disasters**
- **Nature**
- **Education**
- **Politics**
- **Economics**
- **Computer Science**
- **Mathematics**

Pythia (8) gave inspiration for most of the topics. Some extra topics were included to deal with uncovered topics, such as 'Video Games' and 'Foods'.

Once the topics were created, labelled data was created using a distant supervision method. Using Subreddits and Wikipedia categories as ground truth labels, and posts within those Subreddits and Wikipedia categories as the data. The labelled data was then used to train a supervised learning model. Although distant supervision allows us to quickly create labelled data, it is not perfect. The ground truth labels may not be accurate, and the posts may not be relevant to the topic. This is due to the fact that the posts can be made by anyone and it is possible that someone may post something that is not relevant

to the Subreddit or Wikipedia category they are posting in.

2.2 Text Classification

As mentioned in Chapter 1 there exists research on topic identification in short texts such as social media posts. Topic analysis on short posts is harder than on longer texts because of the lack of context (9); Twitter posts are limited to 280 characters (19), so users tend to attach images, or reference other tweets (via retweeting) to add context that would not be obvious from the text alone.

2.2.1 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) are Neural Networks that are used to model sequential data (20). They are good at modelling sequential data because they can take into account the previous inputs in the sequence when making a prediction. This makes them good for text as the text input can be seen as a sequence of words and we can leverage the previous words to make a prediction about the whole sentence.

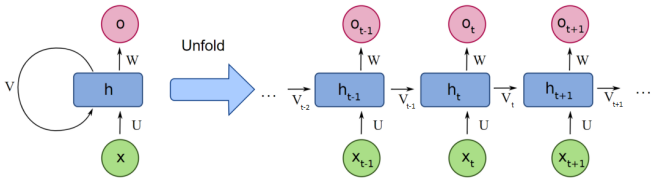

Figure 2.2: RNN Architecture

The RNN architecture in Figure 2.2 is a simple RNN. It takes in a sequence of words (X_i), and outputs a prediction. The RNN takes input words one at a time, and passes a hidden state (V_i) to the next word. The hidden state contains information about the previous words in the sequence. At the end of the sequence, the final hidden state of the RNN is fed through a softmax layer to get a probability distribution over the possible classes.

The main problem with RNNs is the affect of short-term memory (long-term dependencies are hard to learn) (21). For every new word in the sequence, the RNN 'forgets' parts of the previous words in the sequence. This is because the hidden state is updated by each new word.

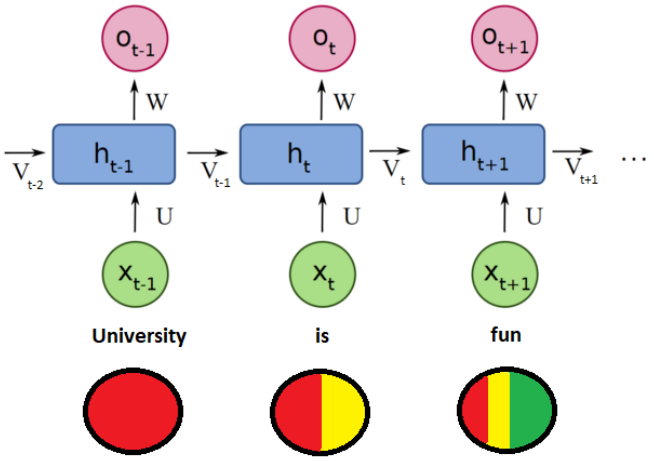

Figure 2.3: Problem with RNNs

In fig. 2.3, we can see how much each 'word' in the sequence affects the hidden state over time. For long sequences, the hidden state is predominantly affected by the last words in the sequence. This means we lose information about the earlier words in the sequence. This is caused by vanishing/exploding gradients (21). Where earlier elements in the sequence have extremely small (or large) gradients making backpropagation hard to perform (22). This could be a problem with our text classification problem as the key information in the text may be in the beginning of the sentence. For example: "Manchester United lost the match 2-1, it was a poor performance but the atmosphere in the theatre of dreams was astounding." The key information in this sentence is that Manchester United lost the match (meaning the text is about sport). However, this information could be lost due to the short-term memory problem. In the case above it is likely we classify the text as entertainment due to the later references of 'performance' and 'theatre'

2.2.2 Long Short-Term Memory (LSTM) - (23)

The Long Short-Term Memory (LSTM) model was developed in 1997 to solve the short-term memory problem in RNNs (23). It does this by using 3 memory gates: the input gate, the forget gate and the output gate. These gates perform different functions:

- **Input Gate:** The input gate decides which values from the current input should be added to the cell state.
- **Forget Gate:** The forget gate decides which values from the cell state should be kept.
- **Output Gate:** The output gate decides which values from the cell state should be used to make a prediction.

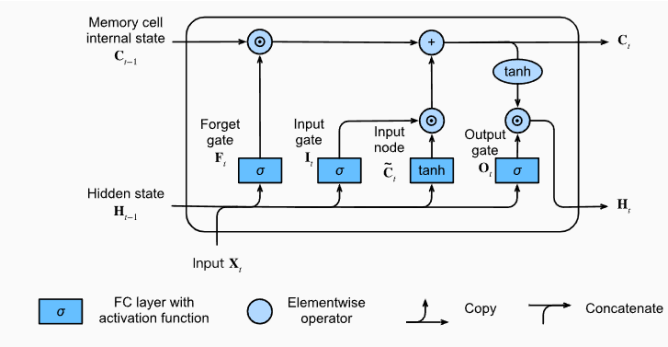

Figure 2.4: LSTM Cell Architecture

The basic intuition behind how LSTM's solve the vanishing gradient problem is that they use gates to pass on information. These gates assign a value between 0 and 1 that corresponds to how important the information is. Although this can still cause a vanishing gradient, it will be the case that if a gradient is vanished then it's input in the forward direction was not important and therefore the gradient will not be important in the backward direction.

LSTM's (as well as RNNs) have another problem: they can only process information in one direction at a time. This means that they can only use previous

words to understand the current word, or the next words to understand the current word. Bidirectional versions of these models aim to solve this problem by processing the sequence in both directions at the same time. However, this is not a perfect solution as it only 'learns' the context from both directions and not as a whole. In section 2.2.3 we will discuss a model that solves this problem.

2.2.3 Transformers

Architecture

In 2017 Google released a paper called *Attention is all you need* (11). This paper introduced the Transformer architecture.

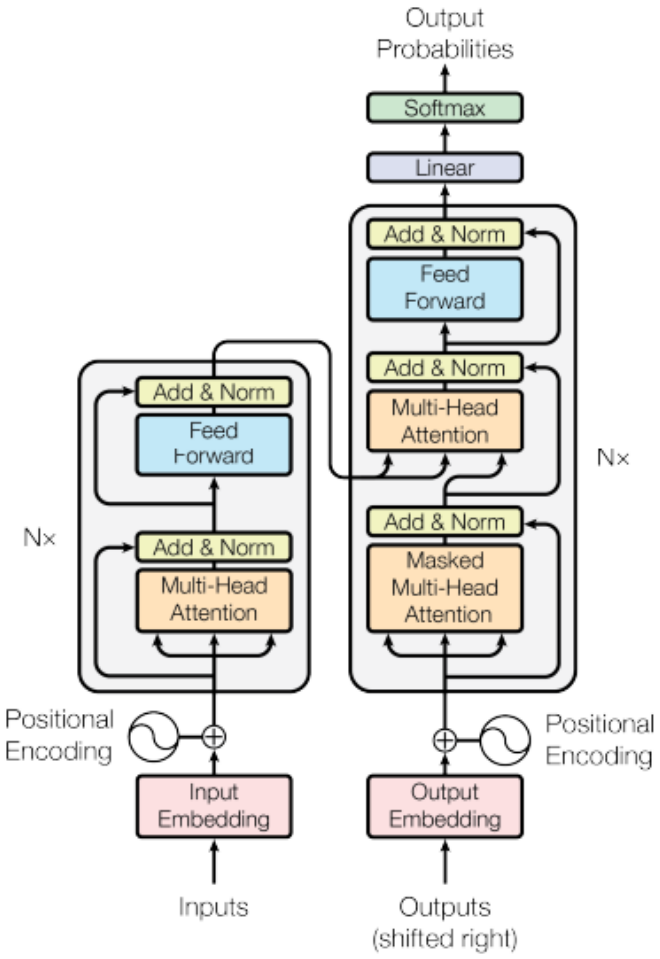

Figure 2.5: Transformer Architecture - (11)

Figure 2.5 shows the architecture of a Transformer model. Focussing on the encoder, the input is passed through a 'Multi-Head Attention' layer. This layer takes in the input and creates a matrix of attention weights. These attention weights essentially say how much of an impact each word has on each other.

Self-Attention

Before looking into Multi-Head Attention, let's discuss Self-Attention.
Self-Attention is a Sequence-to-Sequence model that calculates the attention weights between each word in the sequence (11). The attention weights are

how important each word is to the other words in the sequence. The attention weights are calculated using 3 vectors: Query, Key, and Value (11). All of these vectors are made from passing the input through a linear layer. Each vector is made using a different linear layer.

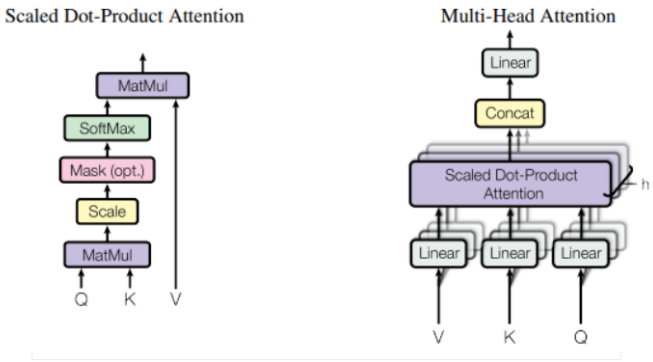

Figure 2.6: Self-Attention - (11)

Figure 2.6 shows how the attention weights are calculated. First, the Query and Key vectors are multiplied together. This is then passed through a softmax layer to convert all weights into probabilities. These probabilities act as the attention weights. The attention weights are then multiplied by the Value vector to get the final output.

Multi-Head Attention

The paper *Attention is all you need* (11) also introduces the Multi-Head Attention layer. This layer uses self-attention but splits the Query, Key and Value vectors into multiple heads. Each head calculate the attention weights independently, and their outputs are concatenated together.

Multi-head attention provides a more stable model throughout training (24). Although deep single-head attention models have been shown to be able to outperform shallow multi-head attention models, they require very specific initialisation before training (24). This is where multi-head attention models shine. They are able to converge to a good solution without the need for specific initialisation of weights (24).

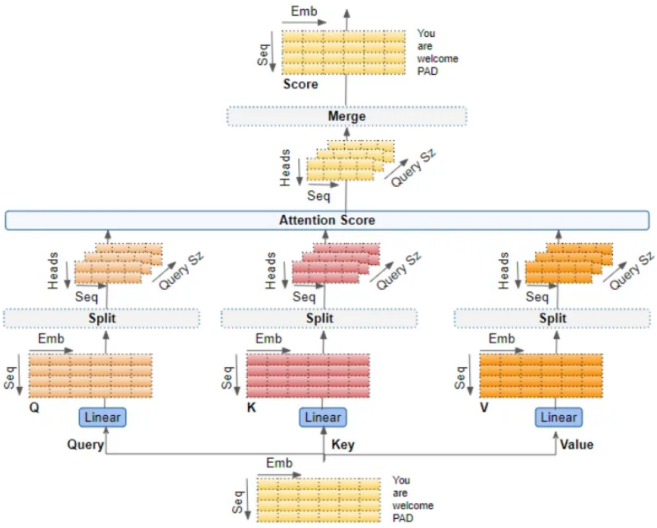

Figure 2.7: Multi-Head Attention

2.2.4 Bidirectional Encoder Representation of Transformers (BERT) - (25)

BERT is an acronym for Bidirectional Encoder Representations from Transformers. It's architecture uses several Transformer encoders put together. Transformers make use of self-attention to learn contextual representation of words. This solves the problem discussed in Section 2.2.1 where the RNNs only look at the previous words in the sentence, or both directions independently.

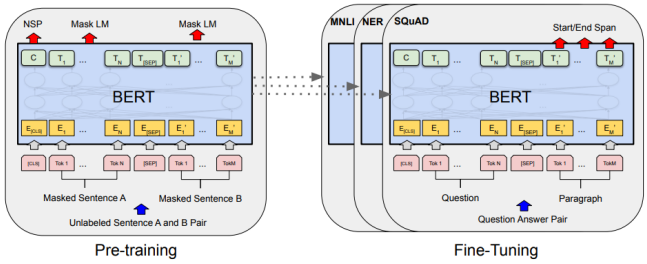

Figure 2.8: BERT Architecture

A BERT classification model is trained in 2 steps: pre-training and fine-tuning. The pre-training step is done on a large corpus by Google. The fine-tuning step is done on a smaller dataset specific to the task at hand.

pre-training

The pre-training step of the BERT model consists of learning 2 objectives: masked language modelling (MLM) and next sentence prediction (NSP).
Masked Language Modelling (MLM) is a task where the model is given a sentence with some words masked out. The model is then expected to predict the masked words.
Next Sentence Prediction (NSP) is a task where the model is given 2 sentences, a and b. The model is then expected to predict whether sentence b is the next sentence following a.

The pre-training step is done on a large corpus of unlabelled data. This makes it easy to train the model, as it does not require collecting a large amount of labelled data. Using Transformers makes the model more efficient, as it can process the entire sentence at once instead of processing the sentence word by word.

fine-tuning

To use BERT for our task of topic classification, we need to fine-tune the model using a labelled dataset - posts labelled with the corresponding topic they belong to (as figured out in Section 2.1.2).
Before fine-tuning, the model needs a final output layer to be added. This output layer will be a linear layer with x neurons, where x is the number of topics (so currently 23). The fine-tuning step is trained using categorical cross-entropy loss, and the Adam optimizer. The model is trained for 6 epochs, with a batch size of 32. The model is then evaluated on the test set, achieving an accuracy of $\approx$ 42%. Please refer to Chapter 5 for a more detailed analysis of the model's performance.

2.2.5 Robustly Optimised BERT Approach (RoBERTa) - (26)

RoBERTa is a variant of BERT that is more efficient and robust. It uses the same architecture as BERT, but with some modifications to the pre-training step. RoBERTa only performs the masked language modelling task, and does not perform the next sentence prediction task. On top of this, RoBERTa uses dynamic masking, where during runtime the model randomly masks out words in the sentence, instead of masking out words statically (masking the same words for a given sentence). The paper introducing RoBERTa also analysed the effect of the pre-training corpus size, batch size, and number of training steps on the model's performance. They found "t performance can be substantially improved by training the model longer, with bigger batches over more data" (26)
Hence the pre-training of RoBERTa was done on a larger corpus of text than BERT. This improved the model's performance.

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
Single-task single models on dev										
$\text{BERT}_{\text{LARGE}}$	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
$\text{XLNet}_{\text{LARGE}}$	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	**90.2/90.2**	**94.7**	**92.2**	**86.6**	**96.4**	**90.9**	**68.0**	**92.4**	**91.3**	-

Figure 2.9: Comparing RoBERTa and BERT - (26)

Figure 2.9 shows how RoBERTa performs better than BERT on most NLP benchmark tests. This led to RoBERTa being tested for this project. RoBERTa was fine-tuned using the same method as BERT, and achieved an accuracy of $\approx 71\%$ on the test set. Evaluation of the model is discussed in Chapter 5.

Chapter 3

Design

3.1 Topic Classification

Chapter 2 discussed 4 different models for classifying text into topics: RNN, LSTM, BERT, and RoBERTa. The chapter also used intuition to determine which model would be expected to perform best - RoBERTa. The reason being, RNNs and LSTMs suffer from being unable to understand context of a whole sentence; they are limited to understanding context in a single direction (left to right or right to left). BERT and RoBERTa both use a technique called self-attention to overcome this limitation. From observing results in (26) RoBERTa outperforms BERT in most cases.
Building off of the RoBERTa model described in Chapter 2, we use the model to classify posts into topics. The design of this model is relatively simple due to the fact we are performing transfer learning on a pre-trained model.

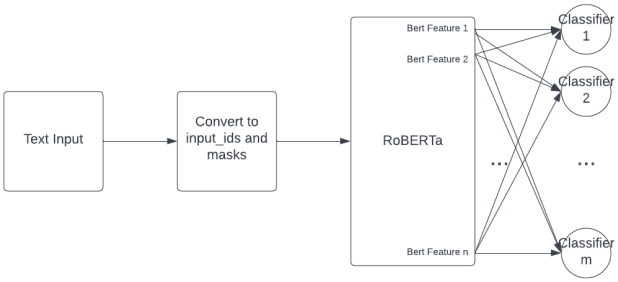

Figure 3.1: RoBERTa model

As seen in the diagram in fig. 3.1, the RoBERTa model has been altered with a new classification layer.

3.2 Adding Context

3.2.1 What is context? and why is it important?

As discussed in section 1.4.2, context is an important factor in classifying posts. In fig. 1.4 there is a tweet that would be hard to classify with the text alone. The section then goes onto show how adding the image that the tweet was posted with (fig. 1.5) adds more information to the post and makes it easier to classify.

3.2.2 Methods for adding context

There are many ways to add context to a post. Pythia uses Named Entity Recognition (NER) for adding context. In this project the use of Optical Character Recognition, Audio Transcription, and Threads/Retweets are explored.

Optical Character Recognition - OCR

Images in posts may contain text. This text adds context to the post and can be helpful in classifying the post. Using OCR we can extract any text from an

image and use it in addition to the text of the post. This should improve the accuracy of the model when infographics/text based images are used.

Audio Transcription - Wav2Vec

Some posts may contain videos. The audio in the videos may contain useful context. Using Wav2Vec we can extract the audio from the video in the form of text. This text can be used in addition to the text of the post. This should improve the accuracy of the model when audio descriptions/explanations are used in a post.

Retweets and Threads

Although a post alone may not contain enough context to classify it, there may be a conversation around the post. This conversation would be found in the retweets and threads of the post. If the retweets and threads are discussing the same topic as the post, then the extra context given by the retweets and threads can be used to help classify the post.

3.2.3 Context Aware Model

The context aware model created in this project will use OCR, Wav2Vec, and retweets/threads to add context to posts. The extra text extracted from these methods will be added to the text of the post. This will be done before the post is classified. The model will then classified with all the additional text.

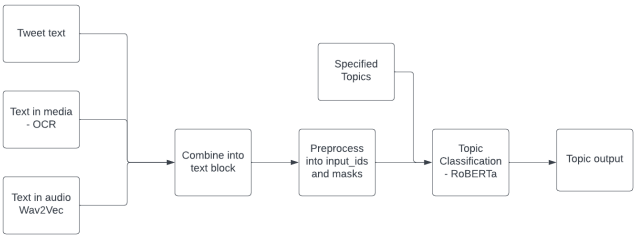

Figure 3.2: Context Aware Model

The RoBERTa classification model will be the same as the one designed fig. 3.1. The only difference is the input to the model.

3.3 Python Application

The Python application is made to show users the topics they are interested in and allows them to compare their interests to the social media site. For this project, this application will be made as a prototype to show such a system is feasible.
The first step of design for the application was to decide what features the application should have. This led to the process of requirements analysis.

3.3.1 Requirements Analysis

User Stories

Before building the application, user stories were created to help guide what features are required. The user stories created were:

1. As a user, I want to be able to see what topics I see the most on social media/see what topics I am interested in.

2. As a user, I want to be able to compare what I see on social media to what other people see on social media.

3. As a user, I want to be able to reach out and find posts on topics I am not interested in.

4. As a user, I want to be able to see what topics are trending on social media.

These stories help set up the requirements for the application.

Business Cases

The next step is to outline the business cases (methods of solving the problem) for the application. For this project, All business cases will be made by the author of this dissertation. The business cases created were:

1. **Do Nothing** - This does not improve the problems users face. It is a baseline case.
2. **Chrome Extension** - Allow users to see what percentage of their social media feed is made up of each topic, as well as compare this to live data from the social media platform.
3. **Python Application** - Allow users to see what percentage of their social media feed is made up of each topic, as well as compare this to live data from the social media platform.

The difference between the chrome extension and the python application is the framework they are built in as well as how they are interacted with. The chrome extension would be built in JavaScript, whereas the python application would be built in Python. The chrome extension would be accessible from a chrome browser (via the extensions store), whereas the python application would be run from a python script.
Although, a chrome extension would be more accessible to users and easier to distribute, it would be more difficult to implement as I would have to learn JavaScript and the chrome extension API. I am already familiar with Python and the python libraries used in this project.

Requirements

Using the user stories and chosen business case, the requirements for the application can be determined. **C** - User Requirement **D** - System Requirements

- Functional Requirements
    1. C) The user should be able to see what topics they see the most on social media/see what topics they are interested in.

1.1 D) The application should be able to get access to the users social media feed via the social media API.

1.2 D) The application should be able to classify the posts in the users social media feed.

1.3 D) The application should be able to calculate the percentage representation of each topic in a users social media feed.

1.4 D) The application should display the top 5 topics the user is interested in as well as their percentage impact on the users social media feed.

2. C) The user should be able to compare what they see on social media to what other people see on social media.

2.1 D) The application should be able to get access to live posts from the social media API.

2.2 D) The application should be able to classify the live posts from the social media API.

2.3 D) The application should be able to calculate the percentage representation of each topic from live social media data.

2.4 D) The application should display the top 5 topics that are trending on social media as well as their percentage impact on the social media platform.

2.5 D) The application should display a similarity metric between the users social media feed and the live posts.

3. C) The user should be able to reach out and find posts on topics they are not interested in.

3.1 D) The application should store all posts that are classified as a topic to be able to search through them.

3.2 D) The application should store alongside the post the top topic it was classified as.

3.3 D) The application should allow the user to search for posts by topic.

3.4 D) The application should display a random selection of posts that are classified as the topic the user searched for.

- Non-Functional Requirements

    1. C) The User Interface should be easy to use within 5 minutes of use.
    2. C) The User Interface should not be unresponsive for more than 5 seconds.
    3. C) The User Interface should be suitable for users with no technical experience.

3.3.2 Frontend

User Interface Design

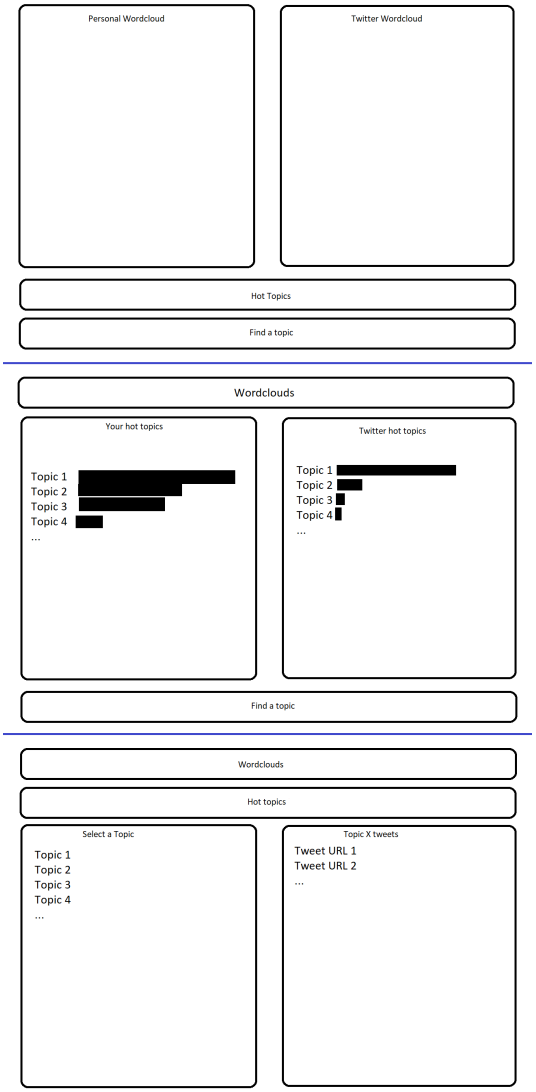

Figure 3.3: Prototype

Figure 3.3 shows the drawn out design of the user interface for the prototype. All 3 customer requirements are met with this design.

The user should be able to see what topics they see the most on social media/see what topics they are interested in.
In the middle display layout there are 10 percentage bars that show the percentage of the users top 5 topics, as well as the percentage of the top 5 topics on social media. This identifies what topics the user is interested in.

The user should be able to compare what they see on social media to what other people see on social media.
As mentioned above, the user is also able to see the top 5 topics on social media. This allows the user to compare what they see compared to what is available. On top of this, the left hand side display is created to show Wordclouds of both the users social media feed and the live posts. This adds a visual representation to the differences between the users social media feed and the live posts.

The user should be able to reach out and find posts on topics they are not interested in.
The right hand side display allows users to select a topic by clicking on the button labelled with the topic. This will then display the text of the 4 posts that are classified as the topic. On top of this, links to the post are provided to allow the user to view the post on the social media platform.

3.3.3 Backend

The backend of the application is responsible for making calls to the social media API to get required posts. It is also responsible for classifying the posts and storing them in a database. When data is required by the frontend, the backend will retrieve the data from the database.

API Design

To hit requirement 1.1 and 2.1, the application needs to be able to access the social medias API. For this project, the social media platform that will be used is Twitter.
The module 'Tweepy' will be used to access the Twitter API. This module acts as a wrapper for the twitter API and allows for easy access to twitter data. The module can be found at `https://www.tweepy.org/`. One downside to the twitter API is that the default access level does not allow for access to any media that is attached to a tweet. This means that the application will not be able to make use of images or videos to add context prior to classification. This was thought of as not being a major issue as the time taken to run OCR and Wav2Vec on the media would be too long to be practical.

For requirements 1.2 and 2.2, the application needs to be able to classify the posts. As described in section 3.1, the application will use a fine-tuned RoBERTa model for classification. For each individual post, we will store the 'tweetid' and the 'top topic', it was classified as, in the database. This information is also useful to fulfill requirement 3.1 and 3.2. The chosen database for this project is sqlite3. This is a lightweight database that is easy. Due to the nature of this project being a prototype, the database will be stored locally on the users machine.

Requirements 1.3 and 2.3 require the application to calculate the percentage of posts that are about each topic. To do this, the application will take the confidence score (probabilistic weight of output nodes in the classification layer) of each post in a set of posts. Using the confidence scores, the application will sum them together then together then convert them to a percentage. This will give the percentage representation of each topic. The application will store these values alongside an ID to identify the set of posts.

Requirement 3.2 will be achieved by taking the topic input given from the frontend request and searching the database for all posts that are primarily classified as that topic. Then, 4 posts will be randomly selected to be sent to

the frontend for display.

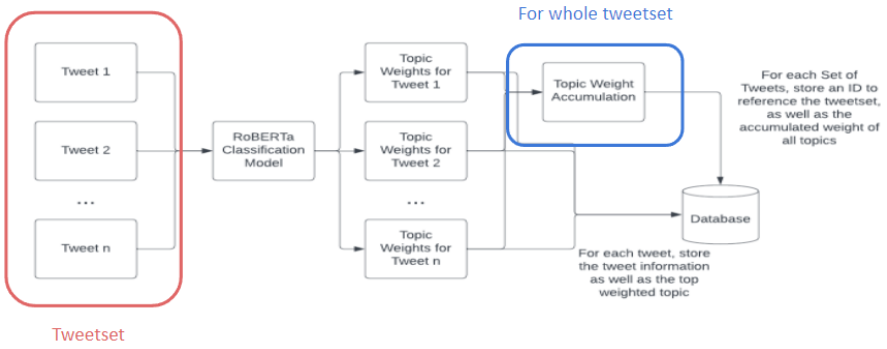

Figure 3.4: Backend Architecture

Figure 3.4 shows the backend architecture. The points raised above can be seen in this diagram along with how the frontend interacts with the backend.

Database Design

The database needs to be able to store the following information:

- The tweetid of a post.
- The tweet text.
- The tweet link.
- any hashtags in the tweet.
- The top topic a post was classified as.
- The ID of a set of posts.
- The percentage representation of each topic in a set of posts.
- The conversation ID that links posts to its parent post.

The database will be a sqlite3 database. The schema can be seen in fig. 3.5 The information required can be accessed using a set of queries. The implementation specifics of these queries will be discussed in chapter 4.

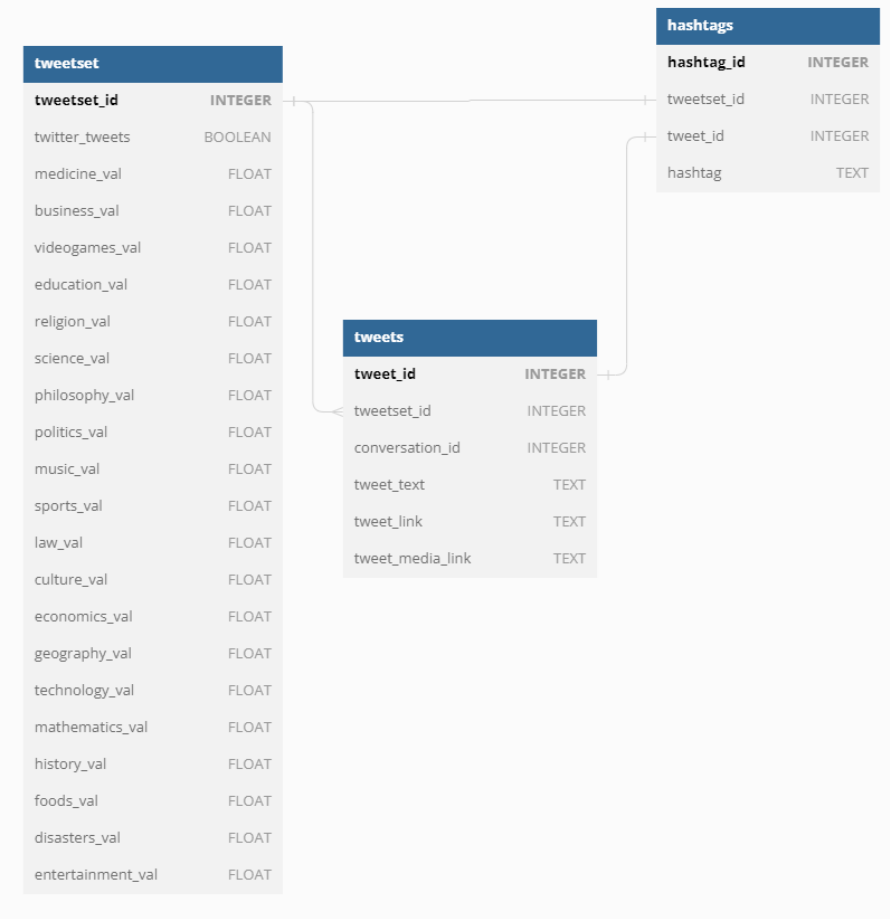

Figure 3.5: Database Schema

Chapter 4

Implementation

4.1 Data Collection

To facilitate with the fine-tuning of the BERT/RoBERTa models and the training of the LSTM/RNN models, it was necessary to collect a large amount of labelled data. As discussed in chapter 2, data was collected from Wikipedia and Reddit. Wikipedia was first chosen due to the fact it has a large amount of data and all are labelled into categories. The one downside to Wikipedia is the style of writing is very formal and factual which does not represent how social media posts are written. This is why Reddit was also used. Reddit is a social media platform, whose Subreddits give us a large amount of labelled data.
The input data went through a preprocessing step before being used to train the models.
A post/article corresponds to the input data, and the Subreddit/Wikipedia category corresponds to that inputs label. The data was collected using the Reddit API and Wikipedia API.
After collecting all the data, there was roughly 1500 data points for each label (topic).

4.1.1 Preprocessing

Some preprocessing was necessary to clean the data. The preprocessing step would remove any punctuation and remove stop words. This was done to reduce the size of the input while keeping the most important words.
In retrospect, it could be possible that the self-attention mechanism of BERT/RoBERTa would be able to learn the context of some stop words and punctuation to improve the accuracy. This has been left as future work.

4.1.2 Wikipedia Data

Wikipedia data was collected using the Wikipedia API. There exists a python library called 'Wikipedia-API' (27) that acts as a wrapper for the Wikipedia API. The library supplies useful functionalities such as:

- **WikiAPI.page** - Takes in a string that acts as a query for the API. In this project the query string is "Category:{topic}" which returns a list of articles/subcategories (28).
- **categorymembers.values()** - given the results of the page query, this function is used to return the articles/subcategories (28).

The way WikiAPI.page works means that we may be returned another category. Because of this, a recursive function is used to take the subcategories found and get the articles from those subcategories. This could recurse on indefinitely, so a maximum depth of 1 was used to prevent this. A depth of 1 means that we get articles from the main category and the subcategories of the main category. The recursive function fetches the titles of articles so they can be queried afterwards.

Algorithm 1 *get_category_members*

INPUT: category, level, max_level
category_members ← list of articles in category
−**WikiAPI.page("Category:{category}").categorymembers.values()**
titles ← empty list
for each member in category_members **do**
  if member is a category AND level<max_level **then**
    $titles.append(get\_category\_members(member, level + 1, max\_level))$
  else
    $titles.append(\text{title of member})$
  end if
end for
return titles

Using this function (algorithm 1) the labelled data can be collected as follows:

Algorithm 2 Algorithm to Retrieve Wikipedia Data

```
topics ← list of topics to collect data for
data ← empty list
for each topic in topics do
  titles ← get_category_members(topic, 0, 1)
  for each title in titles do
    page ← WikiAPI.page(title)
    text ← page.text
    if text is empty then
      text ← page.summary
    end if
    if text is empty then
      CONTINUE
    end if
    text ← preprocess(text)
    data.append((page.content, topic))
  end for
end for
Write fata to csv file
```

The algorithm loops through each topic and uses 'get_category_member' to get the article titles relating to it. It then loops through each title, attempts to find text in the page, and if it does, it appends the preprocessed text and the topic to the data list. The data list is then written to a csv file.

4.1.3 Reddit Data

Reddit data was collected using the Reddit API. Their exists a python library called 'PRAW' (29) that acts as a wrapper for the Reddit API. The library provides useful functionalities. The ones used in this project are:

- **PRAW.reddit** - Takes in API key and secret to authenticate with Reddit API. Returns a Reddit object that can be used to query the API (29).
- **reddit.subreddit** - Takes in a string that acts as a query for the API. In this project the query string is "{topic}" (29).
- **subreddit.hot()** - Returns a list of hot posts in the subreddit. Optionally a limit parameter can be set (29).

These functions are used to get the data as follows: Fetching Reddit data is

Algorithm 3 Algorithm to Retrieve Reddit Data

```
topics ← list of topics to collect data for
data ← empty list
for each topic in topics do
    subreddit ← reddit.subreddit(topic)
    hot ← subreddit.hot(limit=100)
    for each post in hot do
        text ← preprocess(post.title + text)
        data.append((text, topic))
    end for
end for
Write fata to csv file
```

much simpler than fetching Wikipedia data due to the fact that when querying a subreddit, only posts are returned and no other subreddits.

4.2 RNN and LSTM

For creating the RNN and LSTM models, the 'Tensorflow' and 'Keras' libraries were used (30)(31). This was to take advantage of the 'Tensorflow' GPU support (30) on the Department of Computer Science's (DCSs) batch compute system. Also, previous experience with these libraries made it easier to use them.
For both RNN and LSTM models the data was preprocessed in the same way. The data was tokenized using the 'Keras Tokenizer' (32). A tokenizer is used to take text and convert it into a sequence of meaningful integers. The Keras tokenizer has a 'fit_on_texts' function that takes in a list of texts and generates a dictionary of words and their corresponding integer values (33). The output values were one-hot encoded using Pandas 'get_dummies' (34). The data was split into training and testing sets using the 'train_test_split' function from 'sklearn.model_selection' (35).
The RNN model was created using the 'Sequential' model from 'Keras'. A sequential model allows us to build up our model by adding layers. The architecture of the model is as follows:

- **Embedding** - input_dim=1000, output_dim=64, input_length=1000
- **Bidirectional(SimpleRNN)** - RNN: units=64
- **Dense** - units=64, activation='relu'
- **Dense** - units=20, activation='softmax'

The model was compiled using the 'categorical_crossentropy' loss function and the 'adam' optimizer. The model was trained for 10 epochs with a batch size of 64.
A bidirectional RNN was used to attempt to overcome the problem of unidirectional context.

The LSTM model was created similarly to the RNN model. However, using a bidirectional LSTM instead of a bidirectional RNN. The architecture of the model is as follows:

- **Embedding** - input_dim=1000, output_dim=64, input_length=1000
- **Bidirectional(LSTM)** - LSTM: units=64
- **Dense** - units=64, activation='relu'
- **Dense** - units=20, activation='softmax'

The same loss function and optimizer were used. The model was trained for 10 epochs with a batch size of 64.

4.3 BERT

To get the BERT model, the 'tensorflow_hub' library was used (36). This library gives developers access to pre-trained models. The BERT model and preprocessing functions can be imported as follows:

```
preprocess = tensorflow_hub.KerasLayer(
    "https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3")
bert = tensorflow_hub.KerasLayer(
    "https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-
512_A-8/1")
```

When creating the model, these layers can be included in the model architecture. The architecture of the model is as follows:

- **Input** - data type: tf.string
- **preprocess** -
  Found at: https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/3
- **bert** -
  Found at: "https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/1"
- **Dropout** - rate=0.1

- **Dense** - units=20, activation='sigmoid'

The model was compiled using the 'categorical_crossentropy' loss function and the 'adam' optimizer. The model was trained for 6 epochs with a batch size of 32.

4.4 RoBERTa

While Tensorflow and Keras were used for all previous models, the 'Huggingface' library was used for the RoBERTa model (37). Huggingface was only discovered during development of RoBERTa after struggling to get RoBERTa to work using tensorflow and Keras. Huggingface is a company based in New York that aims to advance artificial intelligence through open source projects (37). They have a library called 'transformers' (38) that provides access to pre-trained models. A major benefit to the transformers library is it supports interoperability between different frameworks (PyTorch, TensorFlow, and JAX) (38). Currently, there are 190 supported models in the transformers library including RoBERTa (38).
The transformers library provides a model and a tokenizer for RoBERTa (39). the Tokenizer is important as RoBERTa does not directly accept text as input. Instead, the text must be 'tokenized' and converted into a list of integers (40). The tokenizer and model can be imported as follows:

```
tokenizer = transformers.RobertaTokenizerFast.from_pretrained(
    "roberta-base")
model = transformers.RobertaForSequenceClassification.from_pretrained(
    "roberta-base", max_length=512)
```

Following this, the data must be tokenized using this tokenizer.

```
data = tokenizer(data, padding=True, truncation=True)
```

The data was then split into training and testing sets by randomly shuffling the data and splitting it into 90% training and 10% testing. Then the data was

converted into a Pandas Dataframe and converted into a PyTorch dataset - This can be used with the transformers library.
The next step with Huggingface Transformers is to construct a 'TrainingArguments' object. This object specifies training parameters. The parameters used are as follows:

- **output_dir** - roberta-finetuned-topic
- **evaluation_strategy** - epoch
- **save_strategy** - epoch
- **num_train_epochs** - 5
- **learning_rate** - 2e-5
- **per_gpu_train_batch_size** - 4
- **per_gpu_eval_batch_size** - 4
- **weight_decay** - 0.01
- **load_best_model_at_end** - True
- **metric_for_best_model** - accuracy
- **push_to_hub** - True

The next step is to create a 'Trainer' object. This object is used to train the model. This object takes the base RoBERTa model, the training arguments, the training dataset, the testing dataset, and a method for calculating metrics (37). The metrics used are accuracy.

4.5 Context Aware Model

The next step for this project was establishing a method of getting context for tweets. As discussed in chapter 3, the context model uses media sources and comments to get context for tweets. Focussing on twitter as the social

media platform, the context model uses the 'Tweepy' library as a wrapper for the twitter API. The 'Basic access' API was not a suitable option for adding context. This was due to the fact the API does not procide access to media sources. Because of this the 'Elevated access' API was used. This allows us to get access to media sources (41).

4.5.1 Setup tweepy

To setup connection to the twitter API via Tweepy, the use of the 'API' and 'Client' objects were used. The API object gives access to v1.1 of the twitter API (42). v1.1 is required to get access to media sources. This function call requires passing in authentication credentials. These are created via twitter developer portal and saved in a '.secret' file that is not included in the repository. This file is parsed and the credentials are passed into the 'API' function. The credentials required are (42):

- **consumer_key**
- **consumer_secret**
- **access_token**
- **access_token_secret**

The Client object is initiated in a similar way with the same credentials. This object gives us access to the newer v2 of the twitter API (42). However, as stated previously, this does not give access to media sources. But does give access to 'conversation_id' which allows us to find comments/threads/retweets (42).

4.5.2 Get Tweets

Given a tweet id, the 'get_tweet' function can be used to get the tweet data from the API (42). This function also takes in optional parameters to specify any additional fields that should be returned. This is done via the 'tweet_fields' and 'expansions' parameters (42). Because we want to get media sources, the

tweet field 'entities' is required. Then the 'extended_entities' expansion is used to get the media sources. The tweet data returned (in JSON format) can be queried for URLs to the media sources.

```
tweet_data = api.get_tweet(tweet_id,
    tweet_fields='entities',
    expansions='extended_entities'
)
media_sources = tweet_data.extended_entities["media"][0]
```

Following this, the URL can be used to get the media source using the "Requests" library (43). This library is used to make HTTP requests. The media is saved locally and the path to the media is returned. This path can then be used by the model to get the media to add context.
The 'get_tweet' function can also be used to get the 'conversation_id' of a tweet (42). To do this, the 'tweet_fields' parameter includes 'conversation_id'. Then the 'conversation_id' can be used to get the comments/threads/retweets of a tweet via the 'search_recent_tweets' function. This function takes in a query and returns a list of tweets. In this case, the query would be "conversation_id:{conversation_id}" (42). The downside to using this function is that it can only return tweets if they have been posted in the last 7 days. There does exist a function to search all tweets but this requires a higher level of access to the API (44). All posts with the same 'conversation_id' are returned and are then appended onto the original tweet text.

4.5.3 Feeding into RoBERTa

When all information is gathered (including media) then the tweet can be processed. First the media is processed. This is done either by using 'Wav2Vec', if the media is a video, or 'Optical Character Recognition' (OCR), if the media is an image. This turns the media into text that can be prepended to the tweet text. This text block is then passed through the RoBERTa model to get the topic. The next 2 sections will discuss how media is stored and processed in

more detail.

Photos and Images

The JSON data returned from the twitter API contains a field that displays the type of the media source (if there is one). This field can be accessed via the 'tweet.extended_entities["media"][0]["type"]' field. If the type is 'photo' then the media is an image. If the type is 'video' then the media is a video (42).
For photos, the media url is fetched from the JSON data and the image is downloaded using the 'requests' library and stored locally in the 'media store'. The media store is a collection of 3 folders: 'jpg', 'mp4', and 'wav'. jpg will store all images, mp4 will store all videos, and wav will store all converted audio.
Before processing a tweet that contains an image. OCR is used to convert the image into text. The library used for OCR was 'Keras_ocr' (45). This library allows for easy conversion of images into text; The library is pre-trained and can be used with only a few lines of code.

```
pipeline = keras_ocr.pipeline.Pipeline(detector=keras_ocr.detection.Detector(),
recognizer=keras_ocr.recognition.Recognizer())
image = keras_ocr.tools.read(image_path)
prediction = pipeline.recognize(image)
```

Videos

Processing videos is a little more complicated. The goal is to convert any audible words in the video into text. To do this the 'Wav2Vec' model is used. For this the 'transformers' library from HuggingFace is used (38). This library contains a pre-trained Wav2Vec model. This requires the import of the 'Wav2Vec2ForCTC' model and the 'Wav2Vec2Tokenizer' tokenizer. The model is then loaded and the tokenizer is used to tokenize the audio. The audio is then passed through the model to get the text.

Using a similar method to the one used for images, the video is downloaded and stored locally. However, this will be stored in 'mp4' format. The audio needs to be extracted from the video. This is done using the 'Moviepy' library (46). This is simply done by loading the video and then extracting and writing the audio to a 'wav' file.

```
video = moviepy.editor.VideoFileClip(video_path)
audio = video.audio
audio.write_audiofile(audio_path)
```

Now that the audio is extracted, it can be used with the Wav2Vec tokenizer and model. This is implemented as such:

```
tokenizer = Wav2Vec2Tokenizer.from_pretrained("facebook/wav2vec2-base-960h")
model = Wav2Vec2ForCTC.from_pretrained("facebook/wav2vec2-base-960h")
input_audio = librosa.load(audio_path, sr=16000)[0]
input_values = tokenizer(input_audio, return_tensor="pt").input_values
logits = model(input_values).logits
predicted_ids = torch.argmax(logits, dim=-1)
transcription = tokenizer.batch_decode(predicted_ids)
```

The 'Librosa' library is used to convert the audio into a format that can be used by the tokenizer (Numpy array).

4.5.4 Problems with Context Model

The context model increased the running time of the application by a significant amount. This was due to the extra computation required to process the media. Because of this, only the conversation context will be included for the Python Application.

4.6 Python Application

4.6.1 Frontend

PyQT was the chosen library used to create the UI for the application (47). This was largely influenced by the 'PyQT Designer' tool (47). This tool gives developers the ability to design a UI without writing the code for it; it gives a drag and drop tool to create the UI. This tool is very useful for creating a UI quickly and easily. The UI design can then be imported into the python application and the code to link the UI to the backend can be written.
The development of the UI was heavily influenced by the original design set out in chapter 3. The UI was designed to be simple and easy to use with the aim to be easy to use within 5 minutes of use (Section 3.3.1). See Figures 4.1 to 4.3 for final UI designs:

Figure 4.1: Final UI design - Wordcloud

Figure 4.2: Final UI design - Top Topics

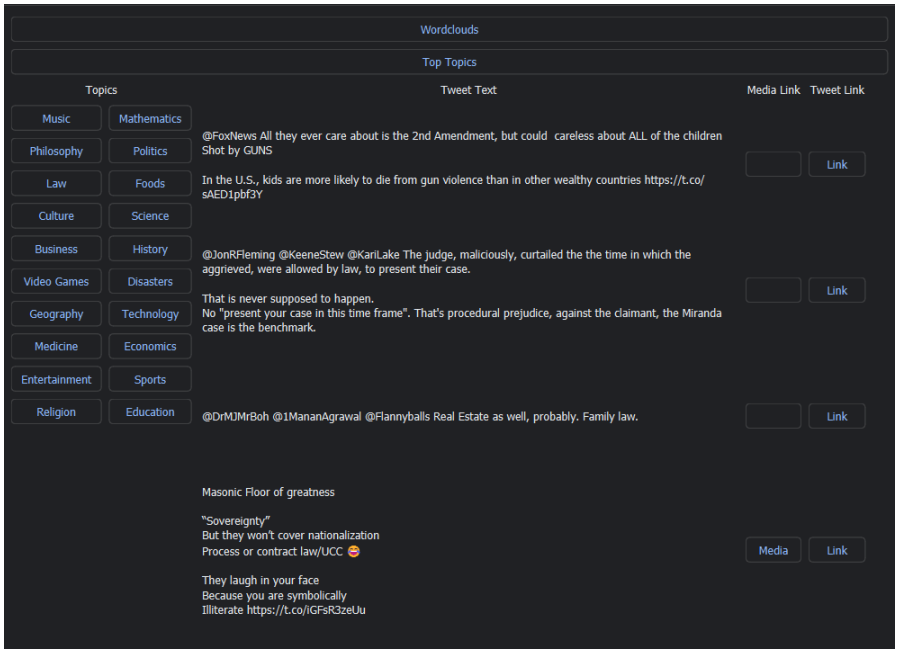

Figure 4.3: Final UI design - Find Topics

Analysis of UI

When creating an application it is important to consider the usability of the application. This can be done by assessing the application against Nielsen's usability principles (48).

- Use simple and natural dialogue
  - The UI minimizes the use of text and the test used is simple and easy to understand.
- Speak the user's language
  - The UI aims to use language that is familiar to twitter. For example, the use of 'tweet'.
- Minimize user memory load
  - The UI is all self-explanatory and does not require the user to remember anything.
- Be consistent
  - The layout of each page is consistent. This is achieved by placing the buttons for each page as an expandable menu on top/below the page.
- Provide feedback
  - The buttons on the UI provide feedback. The only negative is when opening a topics hashtags page, if there are noe hashtags, there is no feedback to announce this.
- Provide clearly marked exits
  - Every page has a clearly marked exit button.
- Provide shortcuts
  - The UI does not provide shortcuts due to the simplicity of the UI.

- Provide good error messages
  - The UI does not provide error messages. This is something that can be improved upon.
- Prevent errors
  - The UI does aim to prevent errors by minimizing user input to only buttons.

4.6.2 Backend

The backend was also developed using Python. This allowed for easy integration with the frontend. In fact, the backend was developed as a standalone class that could be imported into the frontend and used directly. The design shown in fig. 3.4 shows how the backend works. The design ignores any implementation specific details. For example, the design just shows that there is tweets and not how the tweets are retrieved. While working with the 'Tweepy' library, multiple methods for retrieving tweets were used. The first method was to use the 'get_home_timeline' method (42). This returns the most recent tweets from the user's home timeline (42). The second method was using a 'StreamingClient' to listen for live tweets (42). Finally, the use of the 'get_tweets' method was used to retrieve specific tweets via ID (42). On top of this, as discussed in section 4.5 the use of the 'conversation_id' tweet_field was used so that the retweets/replies to a tweet could be retrieved. For retrieving live tweets check fig. A.2 in appendix A for the code used.
For retrieving users tweets check fig. A.3 in appendix A for the code used.

The backend database schema laid out in chapter 3 required slight adjustments to include the ability to store tweets with the same 'conversation_id'. This was overlooked originally as the design was based on the assumption of using the base RoBERTa model and not the context aware model. Figure 4.4 shows the updated database schema

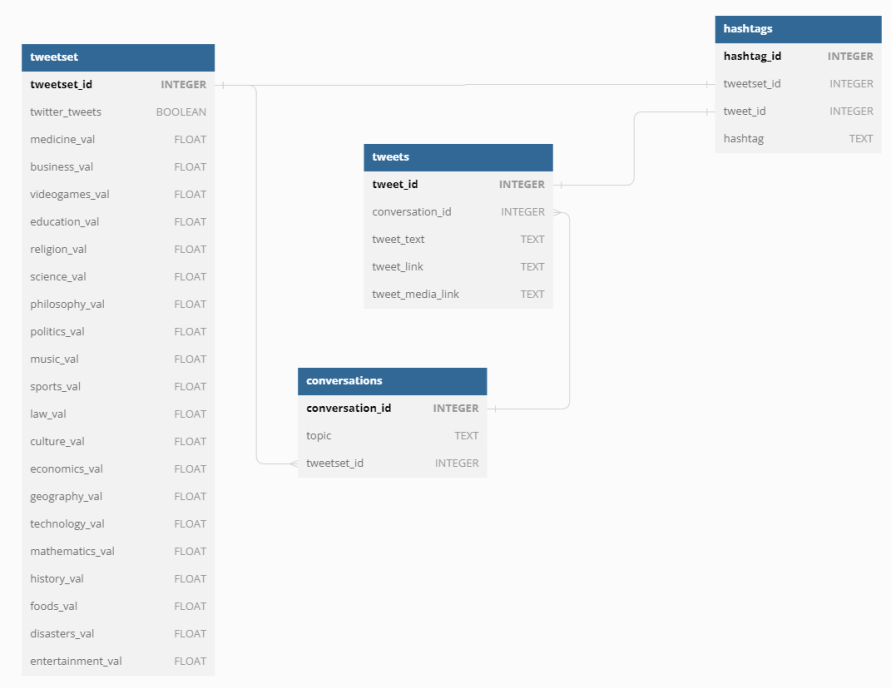

Figure 4.4: Updated database schema

The next step is to create the queries to create/update/retrieve data from the database in-line with the schema and backend design. Queries required:

- Create tweet

INSERT INTO tweets VALUES (NULL, conversation_id, text, tweet_url, media_url)

- Create conversation

INSERT INTO conversations VALUES (conversation_id, NULL, tweetset_id)

- Create tweetset

INSERT INTO tweetset VALUES (NULL, is_live, NULL ($\times$20))

- Set conversation topic

UPDATE conversations SET topic=topic WHERE conversation_id=conversation_id

- Set tweetset topic scores

UPDATE tweetset SET medicine_val=medicine_val, ... WHERE tweetset_id=tweetset_id

- Get hashtags for topic

```
SELECT hashtag FROM hashtags WHERE tweetset_id=tweetset_id AND tweet_id IN (SELECT tweet_id FROM tweets INNER JOIN conversations ON tweets.conversation_id=conversations.conversation_id WHERE conversation.topic=topic)
```

- Get top topics for tweetset

```
SELECT * FROM tweetset WHERE tweetset_id=tweetset_id
```

- Get tweets for a given tweetset

```
SELECT tweets.text, tweets.tweet_url, tweets.media_url, conversations.topic FROM tweets INNER JOIN conversations ON tweets.conversation_id=conversations.conversation_id WHERE conversations.tweetset_id=tweetset_id
```

- Get tweets for a given topic

```
SELECT text, tweet_url, media_url FROM tweetset INNER JOIN conversations ON tweets.tweetset_id=conversations.tweetset_id INNER JOIN tweets ON conversations.conversation_id=tweets.conversation_id WHERE conversations.topic=topic
```

Chapter 5

Evaluation

5.1 RNN and LSTM

To analyse the performance of BERT/RoBERTa, the performance of RNN and LSTM models were analysed for comparison. All models were trained on the same data discussed in section 4.1. There were 2 hypotheses made about RNNs and LSTMs.

1. RNNs and LSTMs will perform worse than BERT/RoBERTa

2. LSTMs will perform better than RNNs

The reasoning behind these hypotheses was discussed in chapter 3. This test was performed to test this hypothesis. Below are the results of The RNN and LSTM models.

Figure 5.1 and Figure 5.2 show the accuracy of the RNN and LSTM models respectively. The results show that the LSTM model outperforms the RNN model (26% accuracy compared to 48% accuracy). This helps validate the second hypothesis that LSTMs will outperform RNNs. The other hypothesis will be discussed in section 5.2.

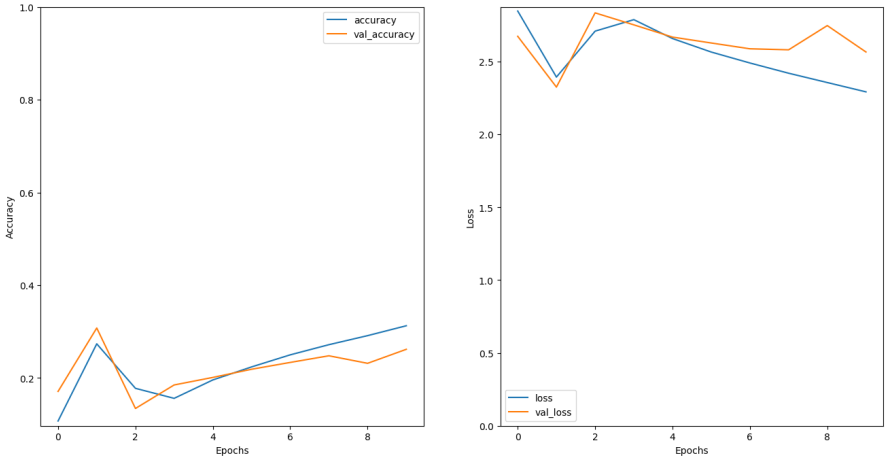

Figure 5.1: RNN results

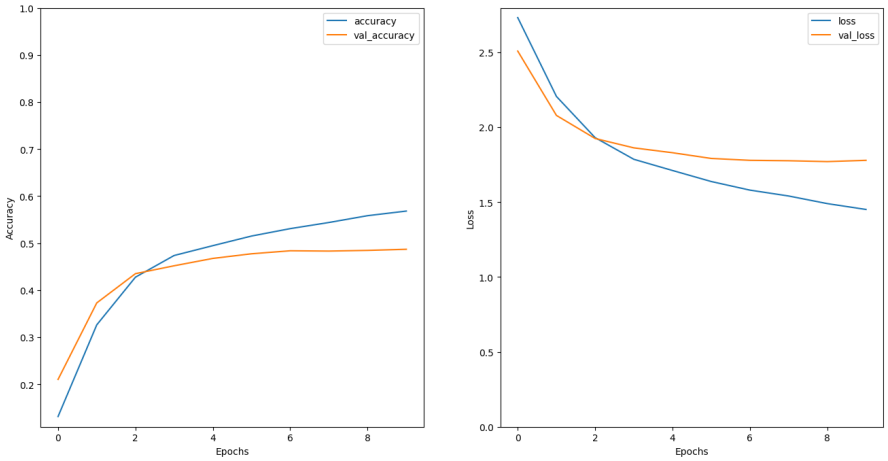

Figure 5.2: LSTM results

5.2 BERT vs RoBERTa

During development, both BERT and RoBERTa were used to classify the posts into topics. BERT and RoBERTa have been compared before in (26). Figure 5.3 shows the results of the paper.

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
Single-task single models on dev										
$\text{BERT}_{\text{LARGE}}$	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
$\text{XLNet}_{\text{LARGE}}$	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	**90.2/90.2**	**94.7**	**92.2**	**86.6**	**96.4**	**90.9**	**68.0**	**92.4**	**91.3**	-

Figure 5.3: RoBERTa vs BERT - (26)

The results of the paper show RoBERTa beats BERT in all given Natural Language Processing tasks. In this project similar tests were carried out on the specific topic classification task. The results of the classification were compared to see which model performed better.

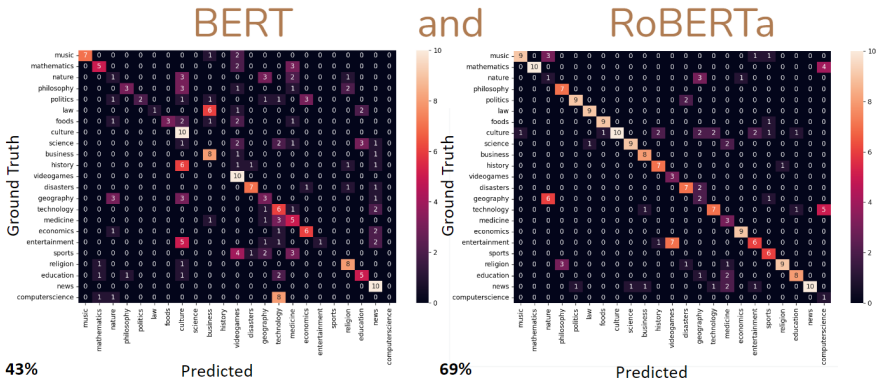

Figure 5.4: BERT vs RoBERTa on our problem

Figure 5.4 shows the confusion matrices of BERT and RoBERTa respectively for the test set - the test set was a random 10% sample of the training data. The columns of the confusion matrix represent the predicted labels and the rows represent the ground truth labels. It is obvious from the confusion matrices that BERT did not perform as well as RoBERTa; there are a lot of misclassifications. A misclassification can be identified as a cell in the confusion matrix that is not on the diagonal. Comparing this to the confusion matrix of RoBERTa, it

is clear that RoBERTa performed much better. In fact, RoBERTa is 60% more accurate than BERT (43% compared to 69%).
With respect to the hypotheses made in section 5.1, the first hypothesis holds form RoBERTa but not BERT. Recall the LSTM model achieved 48% accuracy. BERT could not beat this accuracy but RoBERTa did. This was a surprising result as it was believed that BERT would outperform LSTM models. The reason behind this could be that BERT was trained using only 6 epochs, whereas the LSTM model was trained using 10 epochs.

5.3 Principal Component Analysis (PCA) of data

One thing to note in the RoBERTa confusion matrix in Figure 5.4 is that the model has high concentration of misclassifications; the model makes the same misclassification often. After noticing this trend, it was believed that the topics that were commonly misclassified were similar to each other - could be grouped together.
Principal Component Analysis (PCA) was used to reduce the dimensionality of the data and to allow for visualisation of the data. PCA was performed on the output features of RoBERTa.

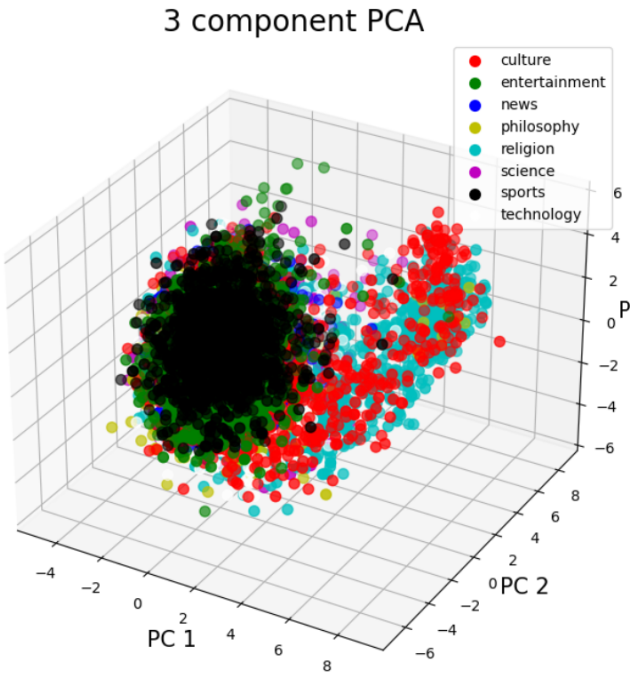

Figure 5.5: PCA of data

Figure 5.5 shows the PCA of the data. The data was reduced to 3 dimensions using PCA. The data was then plotted on a 3D graph. The colours of the points represent the topics (of which some are identified in the legend). The graph is not easy to interpret; the only visible trend is that 'Religion' and 'Culture' seem to be very dissimilar to the other topics.
To help interpret the data, the analysis was performed on 2 topics at a time. This time the data was reduced to 2 dimensions using PCA. The data was then plotted on a 2D graph.

5.3.1 Findings from PCA

In this section, the notable findings from the PCA are discussed.

Mathematics vs Music

This comparison was made to act as a baseline for the rest of the comparisons. The hypothesis was that these topics would be dissimilar to each other.

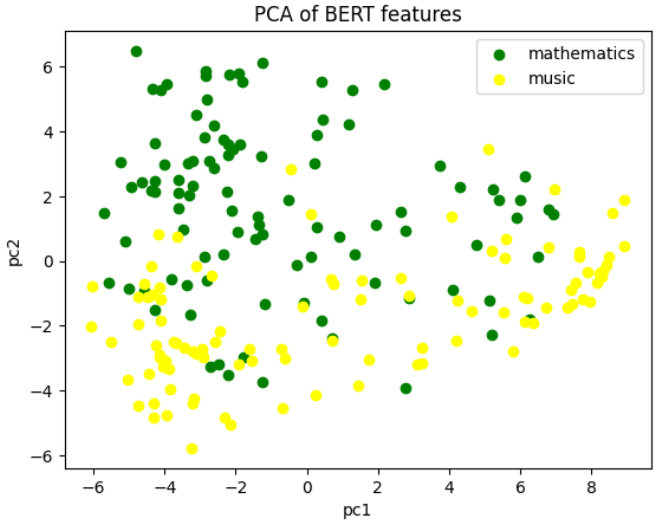

Figure 5.6: Mathematics vs Music

Figure 5.6 shows the PCA of the data for the 'Mathematics' and 'Music' topics. There is a clear separation between the two topics. Although the data is not linearly separable, there is a strong trend that the data points of the 'Mathematics' topic are on the bottom of the graph and the data points of the 'Music' topic are on the top of the graph. Using a Perceptron classifier on the 2 principal components, led to an accuracy of 73%. This is a strong indicator that the 2 topics are dissimilar to each other.

Geography vs Nature

Looking back at the confusion matrix of RoBERTa in Figure 5.4, the model predicts 'Geography' instead of 'Nature' 6 times out of 10. This is a very high concentration of misclassifications. The hypothesis was that the topics are very similar.

Figure 5.7 shows the PCA of the data for the 'Geography' and 'Nature' topics. In comparison to Figure 5.6, there is no clear separation between the two topics. In fact, visually, the two topics seem to be almost identical. Using a Perceptron classifier on the 2 principal components, led to an accuracy of 52%. This is a strong indicator that the 2 topics are similar; the fact the classifier is

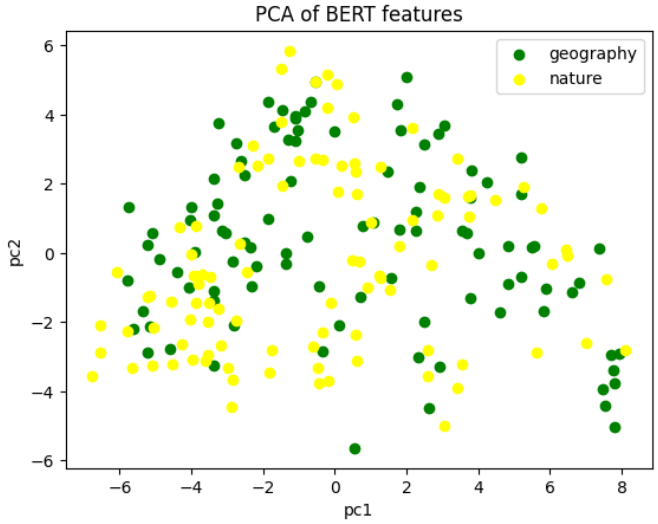

Figure 5.7: Geography vs Nature

only a minor improvement on random guessing shows that the 2 topics are similar.
From this finding it was concluded that the model was not able to distinguish between the two topics. This led to the merging of the 'Geography' and 'Nature' topics into a single topic. It was decided this topic would be called 'Geography'.

Technology vs Computer Science

Similarly to the previous comparison, the confusion matrix in fig. 5.4 shows that the model predicts 'Technology' instead of 'Computer Science' 5 times out of 10. The hypothesis was that the topics are very similar. However, the results from the Perceptron classifier on the 2 principal components showed a very high accuracy of 82.5%. This is not the result expected and caused confusion as to why the model was unable to distinguish between the two topics.
To further investigate this, the data was manually inspected to attempt to reach a new hypothesis. This led to the discovery of the unbalanced nature of the data. The 'Technology' topic had 10 times more data points than the 'Computer Science' topic. This answers why the P-erceptron classifier performed so well but the model was unable to distinguish between the two topics.

The imbalance of the data was a surprise as the script used to scrape the data was designed to scrape an equal number of data points. It was found that the search query for the 'Computer Science' topic yielded no results on wikipedia leading to a small number of data points.
It was decided to remove the computer science topic from the dataset due to the limited dataset.

News

In the confusion matrix for BERT in fig. 5.4, the model predicts 'News' often for 'economics', 'sports' and 'entertainment'. PCA analysis was performed on these topics to test their separability.

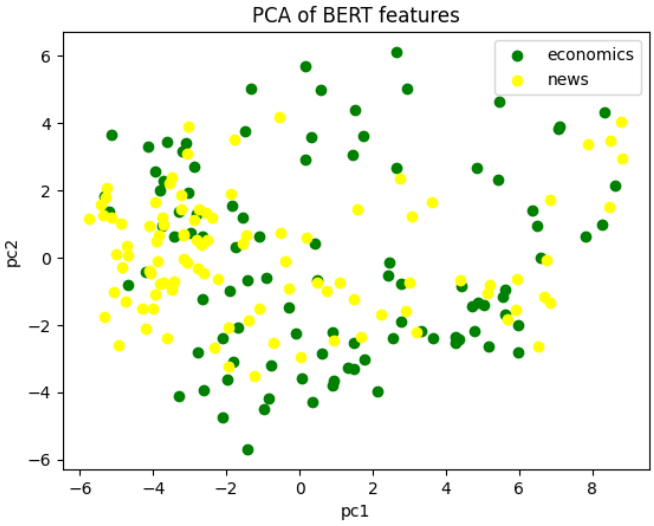

Figure 5.8: Economics vs News

Figures 5.8 to 5.10 show the PCA of the data for the 'Economics', 'Sports' and 'Entertainment' topics against the 'News' topic. The results for each of these tests gave a separability rate of 57.5%, 57.5% and 50% respectively. These results show that the data is not easy to separate. This is not surprising as the 'News' topic is a very broad topic and is likely to cover a lot of the other topics. Because of these findings, it was decided that the News topics should be removed.

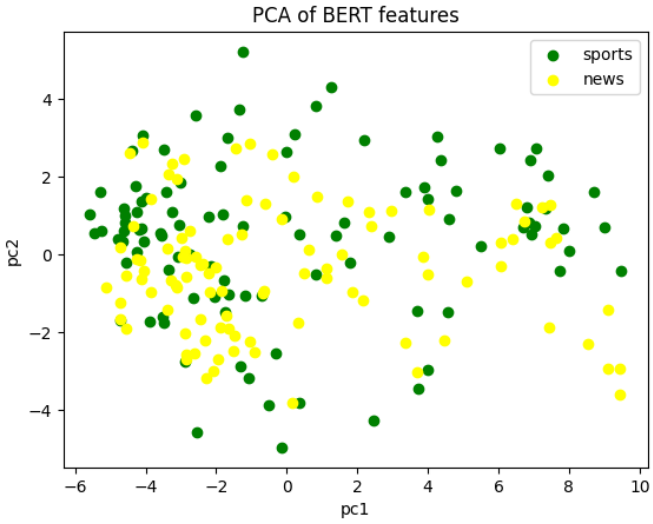

Figure 5.9: Sports vs News

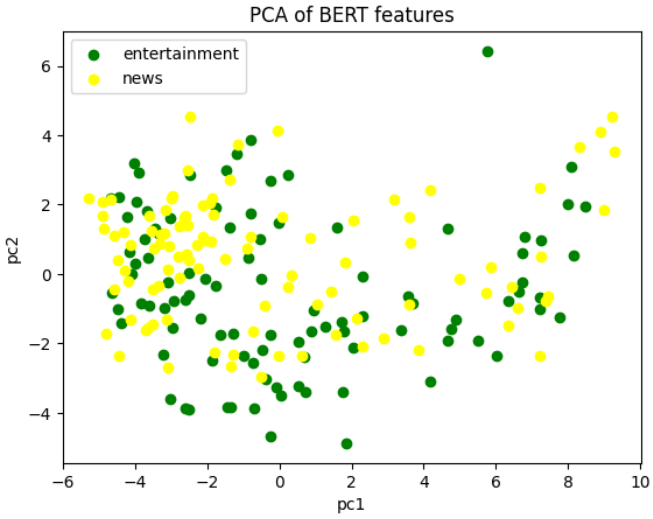

Figure 5.10: Entertainment vs News

5.4 Context

5.4.1 Media - Images and Videos

After implementing OCR and Wav2Vec, the model was tested on a set of 60 hand picked posts (3 for each topic). The updated context aware model was compared to the original model.

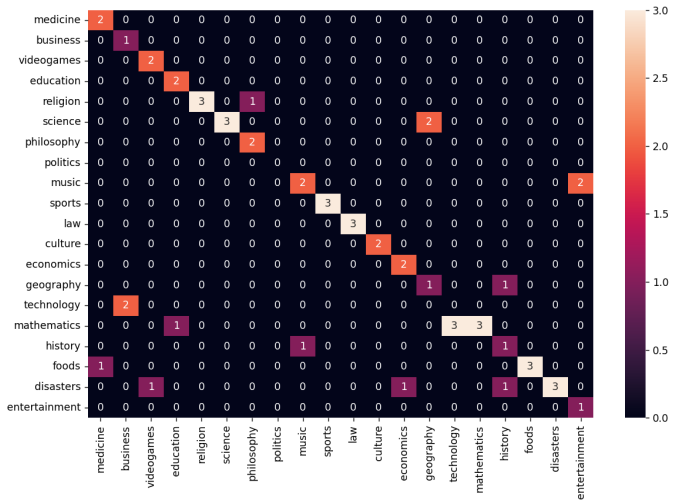

Figure 5.11: Confusion Matrix without Media Context

Figure 5.11 shows the confusion matrix of the original model. Figure 5.12 shows the confusion matrix of the updated context aware model. The addition of the media context improved the accuracy of the model by 10%. This is a notable increase in accuracy. However, due to the nature of gathering this test data, it must be noted some form of bias is present. The test data was hand picked, so some unconscious bias may have been introduced to provide posts that would be hard to classify without the media, but easy to classify with the media. Although, this could be the case this test does show that the context from media can still be useful in classifying posts.

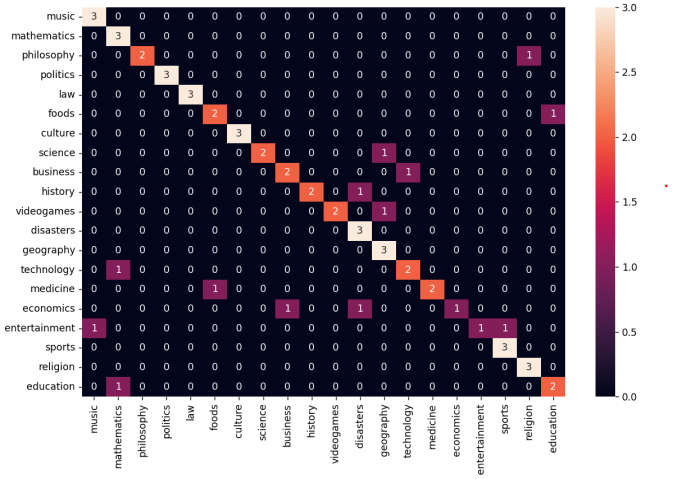

Figure 5.12: Confusion Matrix with Media Context

5.4.2 Retweets and Threads

Separately to implementing OCR and Wave2Vec, another model was created that added the context of retweets and threads. This model was made separately to allow us to see the performance impact of the retweet and thread context. If we had tested them together, we would have seen an improvement but would not have been able to identify the impact of each feature.
The model was tested on a set of 60 hand picked posts, that were different to the posts used in the previous test. The reason behind this is simply the API used to access the tweets was unable to access retweets and threads from posts older than 7 days - The posts used in the previous test were all older than 7 days when this test was performed.

Figure 5.13 shows the confusion matrix of the original model. Figure 5.14 shows the confusion matrix of the thread/retweet context aware model. Similarly to the previous test, the addition of the thread/retweet context improved the accuracy of the model by 10%. Again, this is a notable increase in accuracy although the same bias as the previous test is present.

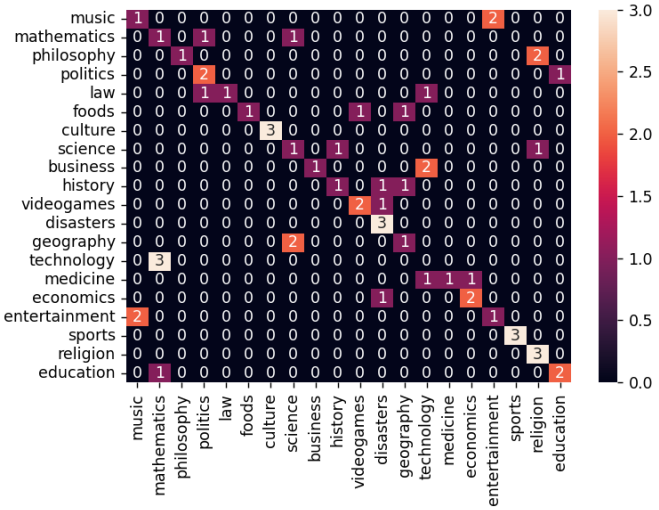

Figure 5.13: Confusion Matrix without Thread/Retweet Context

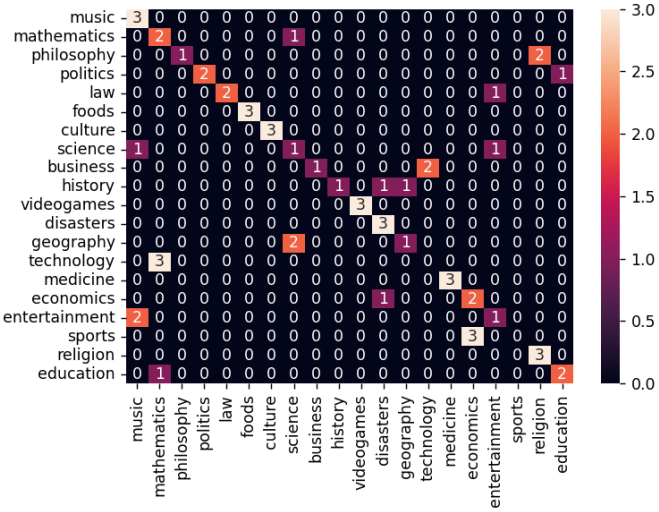

Figure 5.14: Confusion Matrix with Thread/Retweet Context

5.4.3 Context Aware Conclusion

The context aware models both improved the accuracy of the model by 10%. This shows how this extra context can be useful in classifying posts. However, there are cases where the extra context is not useful. For example, some videos are paired with songs for their audio. This audio is not related to the topic of the post. Which could cause the model to misclassify the post. In its current state the model is not able to identify whether the audio is related to the topic of the post or not.
This project shows that the context of the post can be useful in classifying posts. However, due to the naive approach taken in this project, the context is not always useful. In Chapter 7 we will discuss how this context can be used in a more sophisticated way to improve the accuracy of the model.

5.5 Python Application

The requirements set out in Chapter 3 were all met. The application was able to classify posts as well as show users their feeds topical distribution. Also, users are able to find posts on any given topic. Overall the Application was a success and met all the requirements.

Chapter 6

Project Management

6.1 Risk Management

Risks were identified in the project specification (appendix B). Almost all risks that were met during the project were identified prior to the project starting. However, the likelihood of some risks was apparently underestimated. For example, the risk of the twitter API becoming unavailable was deemed to be very unlikely. This risk was met during the project. Thankfully precautions were in place to mitigate this risk. The precaution set was to download the tweets in batches and store them locally. Then it would be possible to create a mock API that would act the same as the real Twitter API. Although this would not work in a commercial setting, it is valid for realising a prototype as well as setting up unit tests.

6.2 Implementation Strategy

6.2.1 Model Creation

While creating the different models for the topic detection an iterative approach was used. This allowed for the design, implementation and testing of each model to be done in a short period of time and then move on to the

next model. This waterfall-like approach works well for model creation, especially with the RoBERTa model; the first iteration of the model created the base topic classification model. This model was then used in the second iteration to create the context aware model.

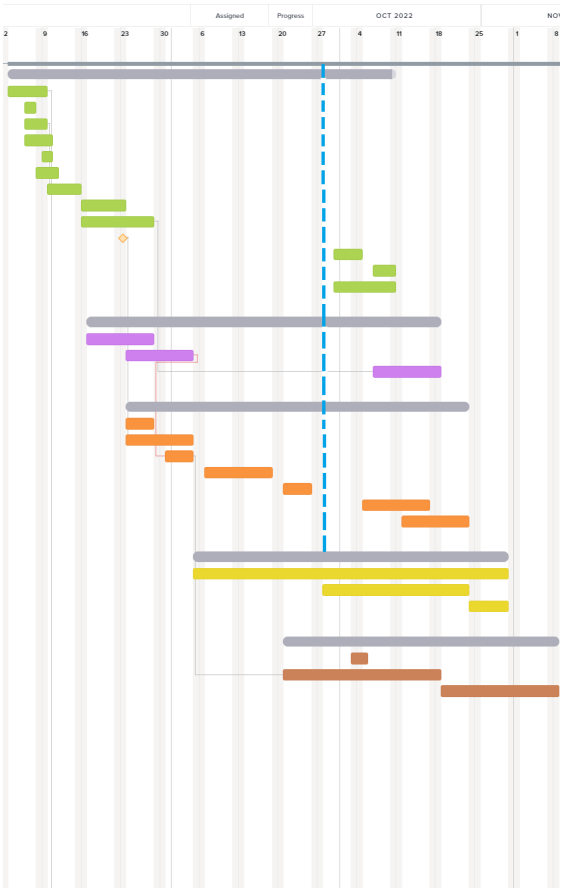

Figure 6.1: Iterative model creation

Figure 6.1 shows how the project timeline was split into iterations for the model development. The dashed blue line shows the separation between the 2 iterations performed in this project.

6.2.2 Python Application

For the Python application, an agile Kanban approach was used. This was partly due to the time constraints, but also because it allowed for any minor

changes required during development to be easily implemented. The Kanban board created consisted of the following columns:

- Ideas
- Designs
- In-Progress
- Testing
- Completed

Due to the fact this project was a solo project, the need for a more complex Kanban board was not required. On top of this, the solo project means less focus on how items move through the board is required. Usually, items would move using a given protocol (e.g. pull system). For this project the Kanban board was treated as a simple to-do list and allowed a visual representation of the status of the project.

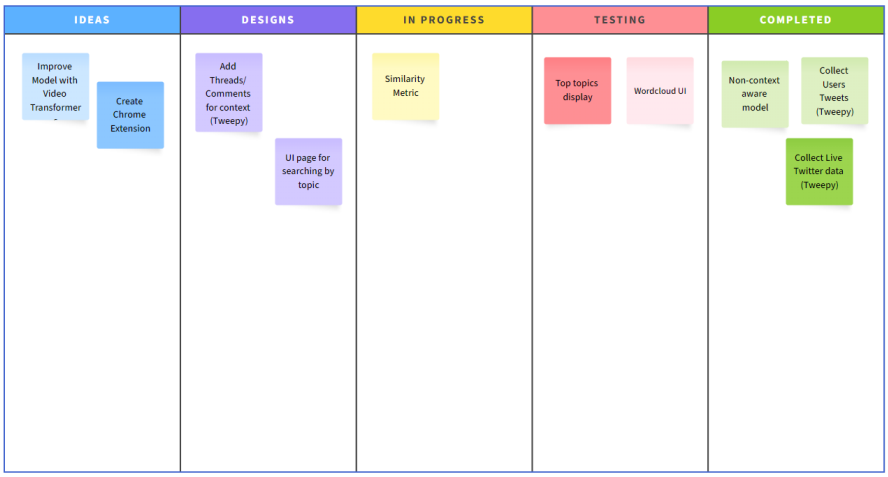

Figure 6.2: Kanban board

Figure 6.2 shows the Kanban board (as of 22/02/2023) used during development.

6.3 Time Management

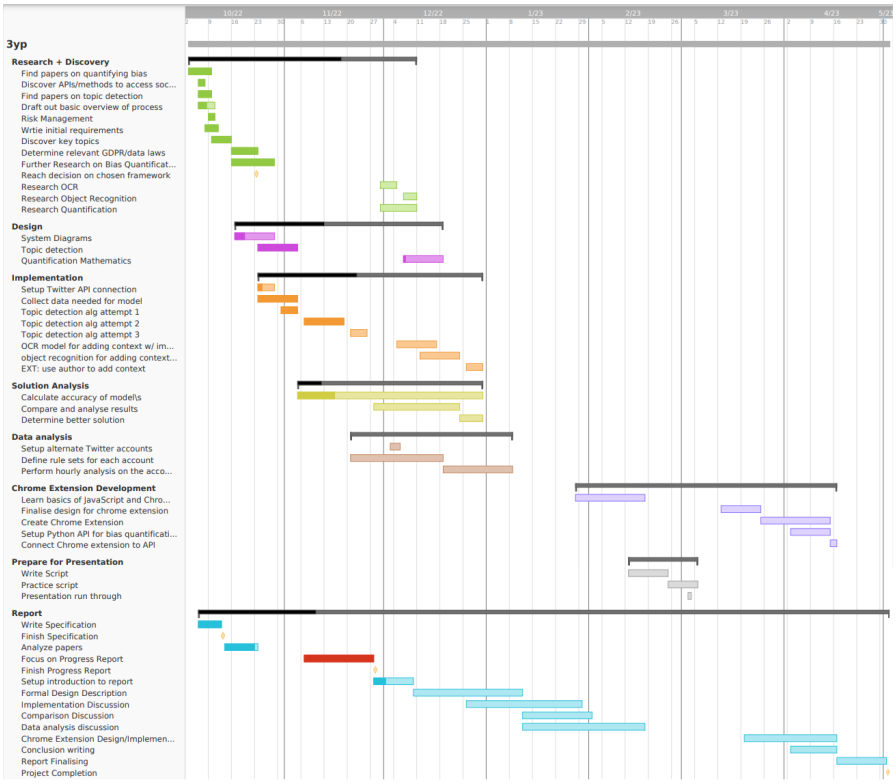

Figure 6.3: Complete Timetable

The method used to set out this timetable was to take the work packages set out in the projects Work Breakdown Structure (WBS) and assign each work package an amount of time. The idea was to be optimistic with the amount of time required. This is because it would be a strong motivator to get work done and would give a lot of 'slack' time to run into when needed. This was a good idea because when the progress of this project was halted over christmas due to technical difficulties with DCS batch compute nodes, there was already the built in slack that meant the project did not fall behind.

6.4 Resource Management

The main resources for this project were:

- Application code
- Jupyter notebooks for model creation and testing
- storage of model parameters for some large models
- Training Data for models

At the start of the project it was decided Git would be used for version control (49). This was an easy choice as it is the industry standard and allows for a structured history of the project to be kept - which we can revert back to if needed. Github was used as the remote repository for this project. The primary benefits for this were:

1. No extra cost
2. The ability to easily share the code across multiple machines
3. The ability to create a Continuous Integration (CI) pipeline

There were issues met with the resource management of this project. The main issue was the structure of the repository initially created. The original plan was to store the code, the reports, and the models in the same repository. Then uses different branches for updating each of these parts. Although this idea could work, it is far from ideal. It would be best to have a separate repository for each of these parts. This is something that has been noted during the project and action was taken to rectify this; the report was separated and moved into it's own repository.

6.5 Testing

6.5.1 Model Testing

Model testing was performed on all models and discussed primarily in chapter 5. The tests involved calculating the accuracy of the model on a test set.

6.5.2 Unit Testing

For the application unit testing was performed on the main functions of the application. This was done manually. In retrospect, it would have been better to use a testing framework such as 'Pytest' (50). This would have allowed for a more structured approach to testing and would make the testing process easier to repeat and report.

6.5.3 Integration Testing

Integration testing was performed to ensure data flowed between the database and UI components correctly. This was done manually. For example, when ensuring the topic scores were correctly showed on the UI after fetching data from the database, the database was populated manually and then the UI was checked to ensure the correct data was shown.

6.5.4 System Testing

System testing was performed as a form of acceptance testing. As the main users for this application is the author, the system testing was performed by the author. This was done by using the application as a user would and ensuring the results were as expected.

Chapter 7

Conclusions

The project aimed to:

- Classify social media posts
- Quantify and compare a users social media feed to the rest of the social media site to identify interests
- Build a user interface that allows users to discover what their social media feed says about their interests and how they compare to the rest of the social media site
- Build a user interface to allow users to find posts that are dissimilar to the posts they are shown

It is clear that all of these aims were met. RoBERTa was fine-tuned for the topic classification task. The fine-tuning was done using labelled data from Reddit and Wikipedia. On top of this, a method to add context via media and comments was attempted to improve accuracy. It was found that both of these context sources can increase accuracy of the mode by $\approx 5\%$. Quantification of a set of tweets was done using the mean of the probabilistic outputs from our model. A user interface was built that is able to show a comparison of topics between a users social media feed and the rest of the social media site. The user interface also allows users to discover posts that are dissimilar to the posts

they are shown.
From this, we can conclude that the project was a success.

7.1 Future work

7.1.1 Advanced Context Input

During this project the use of context was limited to media that was text based and comments/threads. However, there is a lot more information that can be used to improve accuracy of topic classification.
Key information such as author, location, and date can be used to add context to a post. For example, if a post is made by a politician, then the topic is likely to be about politics whereas if the post is made by a sports star, then the topic is likely to be about sports. This information in itself will not be 100% accurate, as politicians and sport stars can post about other topics as well. However, this information still serves as extra context to the post. Including this context, in a similar manner to how we included media and comments/threads, could improve the accuracy of the model.

Another method of improving the context aware part of the model is to create a more sophisticated method of incorporating the context data. Currently, the media text and comments/threads text is concatenated with the post text. This is a very naive method for adding context. For the media, a possible improvement could be to use transfer learning. This would allow us to extract more meaningful data from media and not just the text that is present in it. Transformers have been used for image/video classification. It could be possible to train a transformer model on a dataset of images/videos and then use the output (or possibly a feature vector) of this model as the context for the post. Transfer learning and concatenation of this model output with the RoBERTa model output could be used to create a new 'context aware' model. As seen in fig. 7.1, The post text is fed through RoBERTa and the media is fed through Video Transformers. The outputs of both models are then concatenated together. This is then fed through a dense layer and a softmax layer to get the

final output.

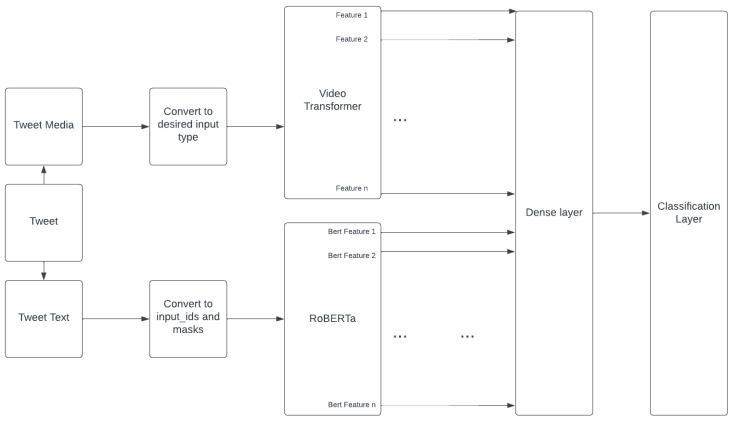

Figure 7.1: Transfer Learning with RoBERTa and Video Transformers

7.1.2 Chrome Extension

Originally, the project was going to create a chrome extension. However, due to the time constraints of the project and the fact that I have never worked with JavaScript or Chrome extensions before, this was not possible. An extension to this project would be to learn these frameworks and transfer the user interface to a chrome extension. This would give the application a better platform to be used on - chrome extension store.

7.1.3 Bias Analysis

When first developing the idea for this project the main aim was to be able to find bias in someones social media feed. The idea came from some reading on echo chambers: "an epistemic environment in which participants encounter beliefs and opinions that coincide with their own" (2). An Echo chamber can cause individuals to be subject to posts that agree with their ideologies, whether or not they are 'true' or 'representative' of all viewpoints. The notion of 'bias' comes from the fact the posts that are shown are not truly representative of all viewpoints; the user is only shown posts that agree with their own.

This project is a good baseline for analysis on bias in a users social media feed. This project allows us to identify what topics a user is interested in. However, this in itself does not gauge whether the feed is bias or not. A user may be interested in a topic and see posts that show all viewpoints on that topic. This is not bias. However, if a user is only shown posts about one side/part of the given topic, then this can be construed as bias. For example, take a user who is interested in sports. For this user to be non-bias they would need to see posts on a range of sports: Football, Rugby, Cricket, Athletics, and so on. However, if they only see posts on Football this can be seen as bias towards football.
This project can be extended to find the differences within topics to identify bias that is present in a users social media feed. Some early ideas for this include:

- Keyword analysis to find subtopics within a Topic.
- Building new models that given an overarching topic can identify subtopics.

Bibliography

[1] T. Developers, "_usTwitter's Recommendation Algorithm." [Online]. Available: https://blog.twitter.com/engineering/en_us/topics/open-source/2023/twitter-recommendation-algorithm

[2] M. Cinelli, G. De Francisci Morales, A. Galeazzi, W. Quattrociocchi, and M. Starnini, "The echo chamber effect on social media," *Proc Natl Acad Sci U S A*, vol. 118, no. 9, Mar. 2021.

[3] D. Mocanu, L. Rossi, Q. Zhang, M. Karsai, and W. Quattrociocchi, "Collective attention in the age of (mis)information," *Computers in Human Behavior*, vol. 51, pp. 1198–1204, 2015, computing for Human Learning, Behaviour and Collaboration in the Social and Mobile Networks Era. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0747563215000382

[4] A. Bessi, M. Coletto, G. A. Davidescu, A. Scala, G. Caldarelli, and W. Quattrociocchi, "Science vs conspiracy: Collective narratives in the age of misinformation," *PLOS ONE*, vol. 10, no. 2, pp. 1–17, 02 2015. [Online]. Available: https://doi.org/10.1371/journal.pone.0118093

[5] A. Zareie and R. Sakellariou, "Minimizing the spread of misinformation in online social networks: A survey," *Journal of Network and Computer Applications*, vol. 186, p. 103094, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804521001168

[6] Topic., *Cambridge Advanced Learner's Dictionary Thesaurus*. Cambridge University Press, n.d.

[7] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[8] I. Litou and V. Kalogeraki, "Pythia: A system for online topic discovery of social media posts," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 2497–2500.

[9] J. Chen, Y. Hu, J. Liu, Y. Xiao, and H. Jiang, "Deep short text classification with knowledge powered attention," *CoRR*, vol. abs/1902.08050, 2019. [Online]. Available: http://arxiv.org/abs/1902.08050

[10] H. J. Levesque and G. Lakemeyer, *The logic of knowledge bases*. MIT Press, 2001.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[12] X. Peng, C. Han, F. Ouyang, and Z. Liu, "Topic tracking model for analyzing student-generated posts in spoc discussion forums," *International Journal of Educational Technology in Higher Education*, vol. 17, no. 1, p. 35, Sep 2020. [Online]. Available: https://doi.org/10.1186/s41239-020-00211-4

[13] S. Husby and D. Barbosa, "Topic classification of blog posts using distant supervision," in *Proceedings of the Workshop on Semantic Analysis in Social Media*, 2012, pp. 28–36.

[14] R. Snow, D. Jurafsky, and A. Ng, "Learning syntactic patterns for automatic hypernym discovery," *Advances in neural information processing systems*, vol. 17, 2004.

[15] B. Mahesh, "Machine learning algorithms-a review," *International Journal of Science and Research (IJSR).[Internet]*, vol. 9, pp. 381–386, 2020.

[16] "Gensim: topic modelling for humans." [Online]. Available: https://radimrehurek.com/gensim/models/ldamulticore.html

[17] B. Mabey, "pyLDAvis: Interactive topic model visualization. Port of the R package." [Online]. Available: https://github.com/bmabey/pyLDAvis

[18] M. P. Grootendorst, "bertopic: BERTopic performs topic Modeling with state-of-the-art transformer models." [Online]. Available: https://github.com/MaartenGr/BERTopic

[19] T. Developers, "Counting characters." [Online]. Available: https://developer.twitter.com/en/docs/counting-characters

[20] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 369â376. [Online]. Available: https://doi.org/10.1145/1143844.1143891

[21] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[22] A. Graves, *Long Short-Term Memory*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 37–45. [Online]. Available: https://doi.org/10.1007/978-3-642-24797-2_4

[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[24] L. Liu, J. Liu, and J. Han, "Multi-head or single-head? an empirical comparison for transformer training," *CoRR*, vol. abs/2106.09650, 2021. [Online]. Available: https://arxiv.org/abs/2106.09650

[25] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[26] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019. [Online]. Available: http://arxiv.org/abs/1907.11692

[27] M. Majlis, "Wikipedia-API: Python Wrapper for Wikipedia." [Online]. Available: https://github.com/martin-majlis/Wikipedia-API

[28] "Wikipedia API â Wikipedia Python API 0.5.8 documentation." [Online]. Available: https://wikipedia-api.readthedocs.io/en/latest/README.html

[29] B. Boe, "PRAW: The Python Reddit API Wrapper â PRAW 7.7.0 documentation." [Online]. Available: https://praw.readthedocs.io/en/stable/%7D

[30] "TensorFlow." [Online]. Available: https://www.tensorflow.org/

[31] K. Team, "Keras documentation: Keras API reference." [Online]. Available: https://keras.io/api/

[32] "tf.keras.preprocessing.text.Tokenizer | TensorFlow v2.12.0." [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer

[33] D. Tensorflow, "tf.keras.preprocessing.text.Tokenizer | TensorFlow v2.12.0." [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer%7D

[34] "pandas.get_dummies â pandas 2.0.0 documentation." [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html

[35] "sklearn.model_selection.train_test_split." [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.model_selection.train_test_split.html

[36] "TensorFlow Hub." [Online]. Available: https://www.tensorflow.org/hub

[37] “huggingface (Hugging Face),” Mar. 2023. [Online]. Available: https://huggingface.co/huggingface

[38] D. Huggingface, “Transformers,” v4.27.2. [Online]. Available: https://huggingface.co/docs/transformers/index%7D

[39] “RoBERTa.” [Online]. Available: https://huggingface.co/docs/transformers/model_doc/roberta

[40] “Tokenizer.” [Online]. Available: https://huggingface.co/docs/transformers/main_classes/tokenizer

[41] D. Twitter, “Developer account support.” [Online]. Available: https://developer.twitter.com/en/support/twitter-api/developer-account%7D

[42] “Tweepy Documentation â tweepy 4.14.0 documentation.” [Online]. Available: https://docs.tweepy.org/en/stable/

[43] “Requests: HTTP for Humansâ¢ â Requests 2.29.0 documentation.” [Online]. Available: https://requests.readthedocs.io/en/latest/

[44] “GET /2/tweets/search/all.” [Online]. Available: https://developer.twitter.com/en/docs/twitter-api/tweets/search/api-reference/get-tweets-search-all

[45] “keras-ocr â keras_ocr documentation.” [Online]. Available: https://keras-ocr.readthedocs.io/en/latest/

[46] “User Guide â MoviePy 1.0.2 documentation.” [Online]. Available: https://zulko.github.io/moviepy/

[47] “PyQt5 Reference Guide â PyQt Documentation v5.15.4.” [Online]. Available: https://www.riverbankcomputing.com/static/Docs/PyQt5/

[48] J. Nielsen, “The usability engineering life cycle,” *Computer*, vol. 25, no. 3, pp. 12–22, 1992.

[49] J. Loeliger and M. McCullough, *Version Control with Git: Powerful tools and techniques for collaborative software development*. " O’Reilly Media, Inc.", 2012.

[50] "pytest | Read the Docs." [Online]. Available: https://readthedocs.org/projects/pytest/

Appendix A

Code listings

```
def get_client(self):
    # twitter API v1.1
    api = tweepy.API(auth=tweepy.OAuthHandler(self.tokens["consumer_key"],
                                              self.tokens["consumer_secret"],
                                              self.tokens["access_token"],
                                              self.tokens["access_token_secret"]))

    return api

def get_APIv2(self):
    # twitter API v2
    return tweepy.Client(self.tokens["bearer_token"],
                         self.tokens["consumer_key"],
                         self.tokens["consumer_secret"],
                         self.tokens["access_token"],
                         self.tokens["access_token_secret"])
```

Figure A.1: Code used to connect to twitter API

```python
class TestStreaming(tweepy.StreamingClient):
    def __init__(self, bearer_token, twitterfname):
        super().__init__(bearer_token=bearer_token) # set up StreamingClient
        self.received = 0 # received tweet count to limit number of tweets received
        self.twitterfname = twitterfname # filename to store tweets
        with open(self.twitterfname, 'w') as f:
            f.write("")

    def on_connect(self):
        print("Connected to streaming API")

    def on_tweet(self, tweet):
        tweettext = tweet.text.replace("\n", " ") # get tweet text with newlines removed
        tofile = tweettext + " " + str(tweet.id)+"\n" # add tweetid

        #check tweetid doesn't already exist
        with open(self.twitterfname, 'r', encoding="utf-8") as f:
            exists = (tofile in f.read())

        if not exists:
            # write tweet and tweetid to file
            with open(self.twitterfname, 'a', encoding="utf-8") as f:
                #write the tweet text, tweet id
                f.write(tofile)

            self.received += 1
            if self.received > 100:
                print("Disconnecting because found tweets")
                self.disconnect()

    def on_disconnect(self):
        print("Disconnected from streaming API")

    def on_error(self, status_code):
        print("Error: {}".format(status_code))
```

Figure A.2: Use of StreamingClient to collect live twitter data

```
get_tweets(self):
tweetset_id = self.db_create_tweetset(False)

# grab tweets from home timeline
tweets = self.APIv2.get_home_timeline(tweet_fields=["author_id",
                                                    "conversation_id",
                                                    "in_reply_to_user_id",
                                                    "referenced_tweets",
                                                    "entities"],
                                      expansions=["author_id",
                                                  "in_reply_to_user_id",
                                                  "referenced_tweets.id"],
                                                  max_results=100)

#sum together the prediction vectors for each tweet
predictions = np.zeros(20)
for tweet in tweets[0]:

    #grab conversation_id and create conversation
    convo_id = tweet["conversation_id"]
    self.db_create_conversation(convo_id, tweetset_id)
    self.db_create_tweet(tweetset_id, convo_id, tweet)

    # use conversation_id to grab all retweets/comments
    conversation = self.APIv2.search_recent_tweets(
        query=f"conversation_id:{tweet.id}", tweet_fields="entities"
    )
    text = tweet.text
    if conversation.data is not None and len(conversation.data) > 0:
        for convo in conversation[0]:
            text += " " + convo.text

    # filter input length to be below maximum allowed
    if len(text) > 512:
        text = text[:512]


    # make prediction and store individual tweets top conversation
    pred = self.processor.predict(text)
    self.db_set_conversation_topic(convo_id, self.labels[np.argmax(pred)])
    predictions += pred

predictions /= len(tweets[0])

# store normalised prediction valuesg cross all tweets in tweetset
self.db_set_tweetset_scores(tweetset_id, predictions)

return tweetset_id
```

Figure A.3: Use of API connections to get users home timeline

Appendix B

Specification

Detecting Interests in Social Media

Project Specification

Joshua Fitzmaurice
Department of Computer Science
University of Warwick

0.1 Glossary

Bias - within this paper, bias refers to the over-representation of specified labelled topics. e.g. feed consisting of a lot of sport posts, or news posts.

1 Introduction and Motivation

Bias in social media can easily be seen on anyone's social media. In fact, I can show this with ease by just taking a look at my Instagram's "Explore page".

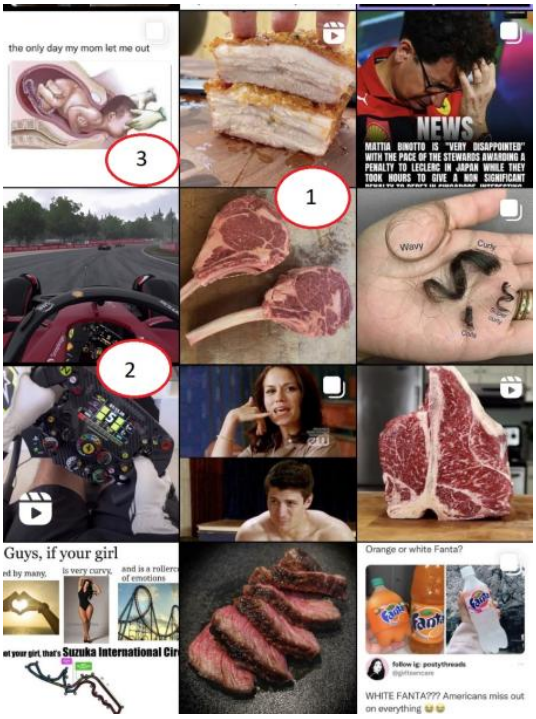

Figure 1: My Instagram for you page

Here we can notice a few common themes/biases: 1. Food, 2. Formula 1, 3. Memes. We want to be able to identify these biases for users so they can get an overview of the type of content they are receiving from social media.
With social media recommender systems programmed to entice users with content they will enjoy (Shin (2020)), it is common for similar groups of posts to be observed by a user if they have recently liked, commented, or viewed similar posts (Instagram).

2 Problem Statement

As discussed in Section 1, it is often the case that users are shown similar posts and not get a strong representation of posts from all aspects of social media. This in itself is not a major problem as users obviously want to see that content, hence why they like/view it. But it would be nice if users were able to see an analysis of the bias they observe in their social media feed (I know I want to see this information).

This project will involve creating a set of labels as well as training/identifying identifiers for said label. We can then scan through a users social media looking for the identifiers to analyse the type of posts prevalent. We can then further develop on this information by performing further data analysis (how, will be determined when patterns emerge in testing).
I have kept this description relatively concise and will develop further on it in my progress report/final report.

3 Objectives

Due to the late changes of project, the objectives/requirements are not fully complete. The goal of these objectives is to give a general understanding to any stakeholders in this project and not to enforce a rigid set of requirements for the implementation of this project.

3.1 Development of my own framework

This is the first, and primary, section of my project. It will involve designing and implementing a framework to detect and display over-representation of topics.

1. Be able to retrieve twitter homepage data from twitter API.
2. Generate a base list of topics we want to be able to detect.
3. Design a method of detecting different topics (here are some possible ideas)
    - Generate a list of keywords as "features"
    - Using the features we can train a ML model to predict the topic given a set of features.
4. Framework should be able to handles a set of posts (10-15) and for each post assign a topic it is representing
5. We can then use the results to perform further analysis.
    - This analysis will involve creating a set of rules for different social media accounts.

- Run the rules on different social media accounts and see how the bias changes over time.

6. EXTENSION - take into account images when analysing posts
    - First, use text detection to find text in the image, and add it to the post.
    - will require a method of object/item detection and labelling

3.2 Comparison of my framework to others

Once creating my framework, I need to analyse and compare it against others.

1. generate sample home twitter feeds.
    1.1 need to generate varying levels of bias within this dataset.
    1.2 generate erroneous test cases.
        - No matching keywords.
        - No posts given.
        - Post containing no text.
2. Determine accuracy of framework using sample twitter feeds.
3. Compare accuracy and other metrics with the given papers in Section 6.
4. Conclude the pros and cons of the different frameworks.

3.3 Data Analysis

This section describes what analysis could be done with the data as well as considerations needed to follow data privacy and data protection laws.

1. Determine a system for rule generation.
2. Setup social media accounts to test the rules on.
3. Analyse the data to determine the bias of the social media accounts.
4. Run the rules every hour on the top 100 posts of each account, analyising the new topical bias.

3.4 Chrome Extension development

After completing the analysis, I could, as an extension, create a chrome extension to display topical bias when on social media.
Functional Objectives:

- MUST
    1. Chrome extension must be visible when on Twitter
    2. Chrome extension must send post information as a request to API
        2.1 Scrape the first x posts from the homepage
        2.2 Using a GET/POST request, retrieve political alignment information from API
    3. API must be able to handle GET/POST requests giving post information
        3.1 receive a list of posts via a GET/POST request
        3.2 feed this list into the bias analysis framework
        3.3 return the corresponding results back to the chrome extension
    4. API must calculate the biases as per Section 3.1
    5. Chrome extension must display the bias information and further analytics via either numerical methods or visual methods.
- SHOULD
    1. Chrome extension/API should be able to handle search results as well as home page.
- COULD
    1. Chrome extension should be able to handle multiple social media sites

Non-Functional Objectives

- Chrome extension should update when social media site opened within 2 seconds
- Chrome extension should always appear to be updating/working
- Chrome extension should display when errors occur.
- Information displayed should be easily understood by the general public of the UK.

4 Testing

Different forms of testing will be used throughout the development of this project.
Black-/White-box unit tests will be created while designing/implementing the project. I plan on using test-driven development, so these tests will be necessary.
As well as this it will be important to test the framework as described in Section 3 to ensure we get useful information from the framework.
I will also include Integration/System testing for the software engineering part of the project (chrome extension).

5 Methods

5.1 Research

I will keep a written log of papers I read/use for this project as well as key areas of information found within the papers.

5.2 Technical Implementation

Python is the choice of language for the backend API as there are readily available libraries that provide the ability to create APIs, as well as access available social media APIs.
JavaScript is currently planned to be used for the Chrome extension.

The software methodology chosen for this project is a waterfall approach. I will not be sticking to a strict waterfall approach, however, as this could cause major disruptions in my time management if any changes to the requirements is needed. Any changes made to the specification during the development of the project must be added on in an agile-like manner to avoid missing deadlines.

6 Papers

In this section I will give a brief analysis of papers that attempt to achieve a similar goal to this project, and what useful information I have come across while reading these papers.

6.1 Pythia - Litou and Kalogeraki (2017)

Pythia is an automated system for short text classification. It makes use of Wikipedia structure and articles to identify topics of posts. Essentially, "Wikipedia contains articles organized in

various taxonomies, called categories". Pythia then goes on to use this information as their training data as well as handling sparseness in posts on social media.

6.2 Topic tracking of student-generated posts - Peng et al. (2020)

This paper proposes a solution for determining valuable information/topics discussed in student forums on online courses. It uses a model called "Time Information-Emotion Behaviour Model" or otherwise called "TI-EBTM" to detect key topics discussions , keeping in mind the progress of time throughout the forum.
Although this paper specializes in academic online forums, the approaches made could be relevant and useful for this project.

6.3 Topic classification of blogs - Husby and Barbosa (2012)

This paper uses Distant Supervision - 'an extension of the paradigm used by (Snow et al. (2004)) for exploiting WordNet to extract hypernym (is-a) relations between entitities' - to get training data via Wikipedia articles. Then trains their own designed model on this data to be able to classify topics via a multi-class recognition model (69% accuracy) and via a binary classification model (90% accuracy).

6.4 BERT - Glazkova (2021)

This paper analyses BERT (as well as modified BERT models such as RoBERTa) and how they can be used for text classification. The data used in this paper is a set of scientific papers that are classified into 7 different categories.

The paper shows that using a Feedforward Neural Network (FNN) on top of BERT can achieve a 91.76% accuracy on the dataset.

7 Timeline

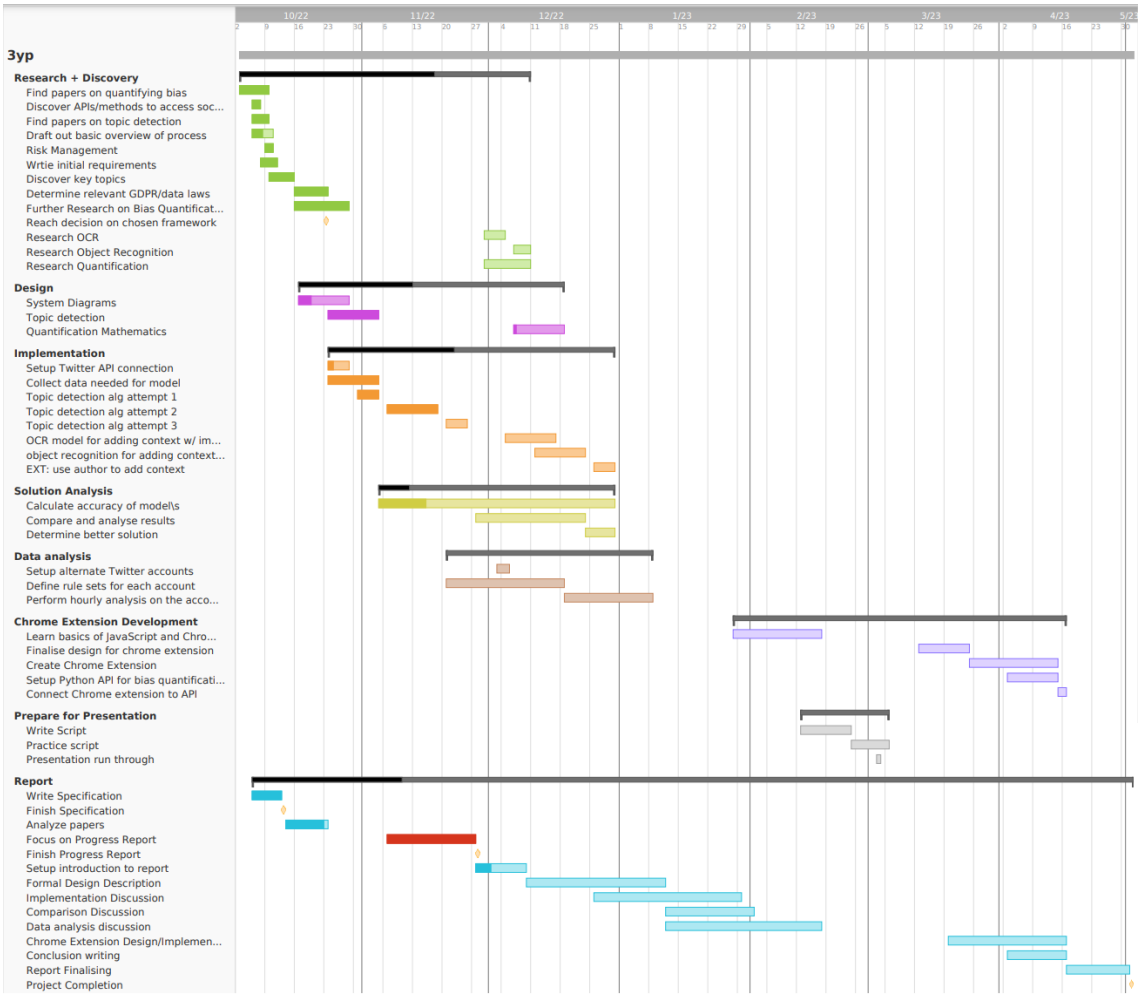

Figure 2: Project Timeline

8 Risk Management

There are several factors that pose a risk to this project. Table 1 is a table to illustrate what risks are prevalent, how big of an impact they will have, and how to counteract these risks. The scores are ranked from 1-5. 5=high, 1=low.

9 Ethical/Legal Considerations

As mentioned in Section 3.3 when storing information of users - such as demographics, identifiable information, and social media usage - It is important to ensure we follow relevant GDPR and data protection laws.
When gathering data, we will also required volunteers. The volunteers will need to have a thorough understanding of what information we are gathering and we need to ensure we use this data lawfully.

Risk	Likelihood of Occurring	Impact to Project	Impact Mitigation	New Impact
Personal issue causing delay in schedule	4	3	I have purposefully planned my project with room to	1
Chosen design philosophy fails to produce useful bias metrics	3	4	Firstly, this project is mainly research based, so no matter the outcome we will gain something from the comparison between approaches. For the Software side of the project we could instead use an already known method if ours does not work.	2
Changes to Twitter API	2	1	Any changes to the API should not dramatically affect this project. At worse it could involve some minor code refactoring to correct endpoints/REST queries.	1
Laptop dies	1	5	Using git+github for version control throughout this project (for code and report), no matter what happens to my own laptop, the codebase will be accessible from any machine	1
Unable to recreate other papers methods	4	4	Instead, I can take the papers results as true (after analysis of results credibility). I can also just create the chrome extension using my own method	1
inability to gather enough users to generate data for analysis	2	4	Can perform analysis based on other metrics than demographics (e.g. who they follow, what the like, etc.) This data can be generated artificially	2
Twitter website/API being taken off the internet	2	5	This is a very unlikely scenario, but if it were to happen, I would have to find another source of data to analyse.	1

Table 1: Possible risks

References

Glazkova, Anna. Identifying topics of scientific articles with bert-based approaches and topic modeling. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 98–105. Springer, 2021.

Husby, Stephanie & Barbosa, Denilson. Topic classification of blog posts using distant supervision. In *Proceedings of the Workshop on Semantic Analysis in Social Media*, pages 28–36, 2012.

Instagram, . How Instagram determines which posts appear as suggested posts | Instagram Help Centre. URL `https://help.instagram.com/381638392275939`.

Litou, Iouliana & Kalogeraki, Vana. Pythia: A system for online topic discovery of social media posts. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2497–2500, 2017. doi: 10.1109/ICDCS.2017.289.

Peng, Xian & Han, Chengyang & Ouyang, Fan & Liu, Zhi. Topic tracking model for analyzing student-generated posts in spoc discussion forums. *International Journal of Educational Technology in Higher Education*, 17(1):35, Sep 2020. ISSN 2365-9440. doi: 10.1186/s41239-020-00211-4. URL `https://doi.org/10.1186/s41239-020-00211-4`.

Shin, Donghee. How do users interact with algorithm recommender systems? the interaction of users, algorithms, and performance. *Computers in Human Behavior*, 109:106344, 2020. ISSN 0747-5632. doi: https://doi.org/10.1016/j.chb.2020.106344. URL `https://www.sciencedirect.com/science/article/pii/S0747563220300984`.

Snow, Rion & Jurafsky, Daniel & Ng, Andrew. Learning syntactic patterns for automatic hypernym discovery. *Advances in neural information processing systems*, 17, 2004.