Is the media influencing you?

Predicting political bias in news articles

Zara Siddique

Abstract

The fundamental aim of this project is to create a platform that can analyse any piece of news content online for political bias, and compare it to political bias within the same news outlet, and across others. The political bias is detected using a model trained on 480,000 news articles using DistiBERT. The approach involved developing an API using Python and Flask, hosted on pythonanywhere. The frontend is deployed using a React application, hosted with Netlify.

The web application for this project is available at: https://bias-radar.netlify.app/

Acknowledgements

I would like to thank Dr Liam Turner and Dr Roger Whitaker for their support throughout this project, and for their genuine interest in my work that pushed me to dive deeper and think critically. I would also like to thank my friends for keeping me fed and caffeinated throughout the process, as well as my daughter Lyra, for being my constant source of energy and inspiration.

Table of Contents

1 Introduction

> "The modern press itself is a new phenomenon… They can play up or down the news and its significance, foster and feed emotions, create complacent fictions and blind spots, misuse the great words and uphold empty slogans. Their scope and power are increasing every day as new instruments become available to them. These instruments can spread lies faster and farther than our forefathers dreamed…"

- "A Free and Responsible Press", 1947 Hutchins Commission

1.1 Background and Motivation

Stories of online content manipulating public opinion have become increasingly common over the past decade (Bradshaw & Howard, 2019). It is a worrying phenomenon, and one that can't be stopped without readers having methods to evaluate and contextualise their information. While political bias in itself is not inherently 'bad', the existence of echo chambers can lead to ideological polarisation, a loss of diversity of opinions and arguments, and a threat to the healthy functioning of democratic societies (Spohr, 2017). The core aim of this project is to make potential political bias behind news content online more visible and transparent for the general public.

The major existing attempts to classify political bias in news come from organisations such as AllSides, the Pew Research Center and Ad Fontes Media. All of these companies carry out large scale surveys or hire analysts to classify their articles for them. This is a time consuming and often expensive process, and a use-case that lends itself to being improved with machine learning.

Furthermore, these organisations rank political bias by news outlet, rather than by individual article. Authors can have individual opinions that may not be in line with the organisation that they work for, and it is important for readers to be able to contextualise this too. Context is key in all of this work - where an article sits compared to news from the same outlet, other outlets, or within a particular topic.

It would be impossible for analysts to classify every single political article on the internet in this way; however, using machine learning, I am able to generate a classification from any news article URL, as well as contextualise it amongst other news sources for readers.

1.2 Proposed Solution

This project will use machine learning and modern NLP techniques to determine political bias (left/right/center) in news articles, and compare this between news outlets. This will involve building a model, which can then be used to analyse any news article online. The model will be deployed via an API using Python and Flask. The front-end of this application will be built with React and hosted on Netlify. The public and participants will be able to give feedback on whether the political bias score is accurate. This data will be used to improve the results.

1.3 Aims and objectives

The aims of this project are as follows:

1. To accurately detect political bias with a machine learning model
2. To develop a web platform in which users can find a bias score for any article online
3. To create a clear, easy-to-use and attractive user interface
4. To develop bias scores for news outlets, in order to compare them to each other
5. To identify which news outlets are more/less biased, and in which direction

1.4 Project Scope

The project will cover the aims and objectives above, and will only look at political bias. There are various other measures of bias; gatekeeping bias which is the idea that media outlets can control which stories become prominent and which don't, or sensationalism which is bias in favour of the extraordinary over the ordinary.

While these can contribute to the context of the article, and would be excellent additions to the project, time and resource constraints mean this project will only be looking at political bias. Political bias, also known as hyperpartisanship or partisan bias, is a tendency to report to serve a particular political leaning.

I will look at ways in which to incorporate human feedback into the model, but this is not considered a key objective of the project.

1.5 What makes it different?

While the problems associated with a politically biased press are becoming more widely acknowledged, there is yet no definitive way to 'solve' problems caused by it. Some models have been created and published, but these are often not easy for the public to access without specialist knowledge.

The project achieves various things that are not usually possible with existing systems:

- Live feedback on a news article from a web platform
- Comparison to the average political bias score of its news outlet
- Comparison across news outlets

2 Literature Review

The initial research and literature review naturally splits into two sections. The first is political bias research, which looks into the nature of political bias, the impact it can have, and most importantly, previous attempts to quantify and classify it. The second section looks at NLP models - their history, advancements, and what the cutting edge is now.

2.1 Political Bias

The UKs National Union of Journalists wrote their code of conduct in 1936 (National Union of Journalists, 1936). In it, it states as one of its principles:

> "Freedom in the honest collection and publication of news facts, and the rights of fair comment and criticism, are principles which every journalist should defend."

Unfortunately, journalism today often strays from the honest collection and publication of news facts. In fact, less than half of Britons (44%) now say they trust the BBC to tell the truth despite its public charter to remain politically neutral (YouGov, 2019). Political bias is considered a form of media bias, involving the slanting or altering of information to make a political position or candidate seem more attractive. Bias isn't necessarily a bad thing, but hidden media biases can manipulate and mislead the public.

It is a difficult task to quantify the impact of political bias on society, as well as to quantify what political bias looks like. It has been found that the existence of echo chambers can lead to ideological polarisation, a loss of diversity of opinions and arguments, and a threat to the healthy functioning of democratic societies (Spohr, 2017). A report from the Pew Research Center in 2014 showed that there was little overlap in the news sources that liberals and conservatives turn to and trust, and that the partisans have moved to the left and the right, and as a result the share of Americans with mixed views has declined (Mitchell & Weisel, 2014).

A study from the University of Colorado (Keating et al., 2016) found that a short discussion with like-minded people further polarizes one's opinions, and more importantly, participants underestimated how much their attitudes had polarized, misremembering their pre-discussion attitudes as more polarized than they actually were. This cements the idea that being surrounded by political bias, whether left or right wing, can lead to more and more polarization without one being aware.

It is clear that there is a change happening in terms of political polarisation in society, and that the media contributes to this. The next step is to look at how others have tried to classify political bias in news outlets. Some early attempts to classify news outlets as on the left, right or in the center were from AllSides[1] in 2012 and Ad Fontes Media's Media Bias Chart[2] in 2016. These were created using expert analysts and/or large scale surveys - a time consuming and often expensive process.

[1] AllSides, https://www.allsides.com/unbiased-balanced-news
[2] Ad Fontes Media, https://adfontesmedia.com/

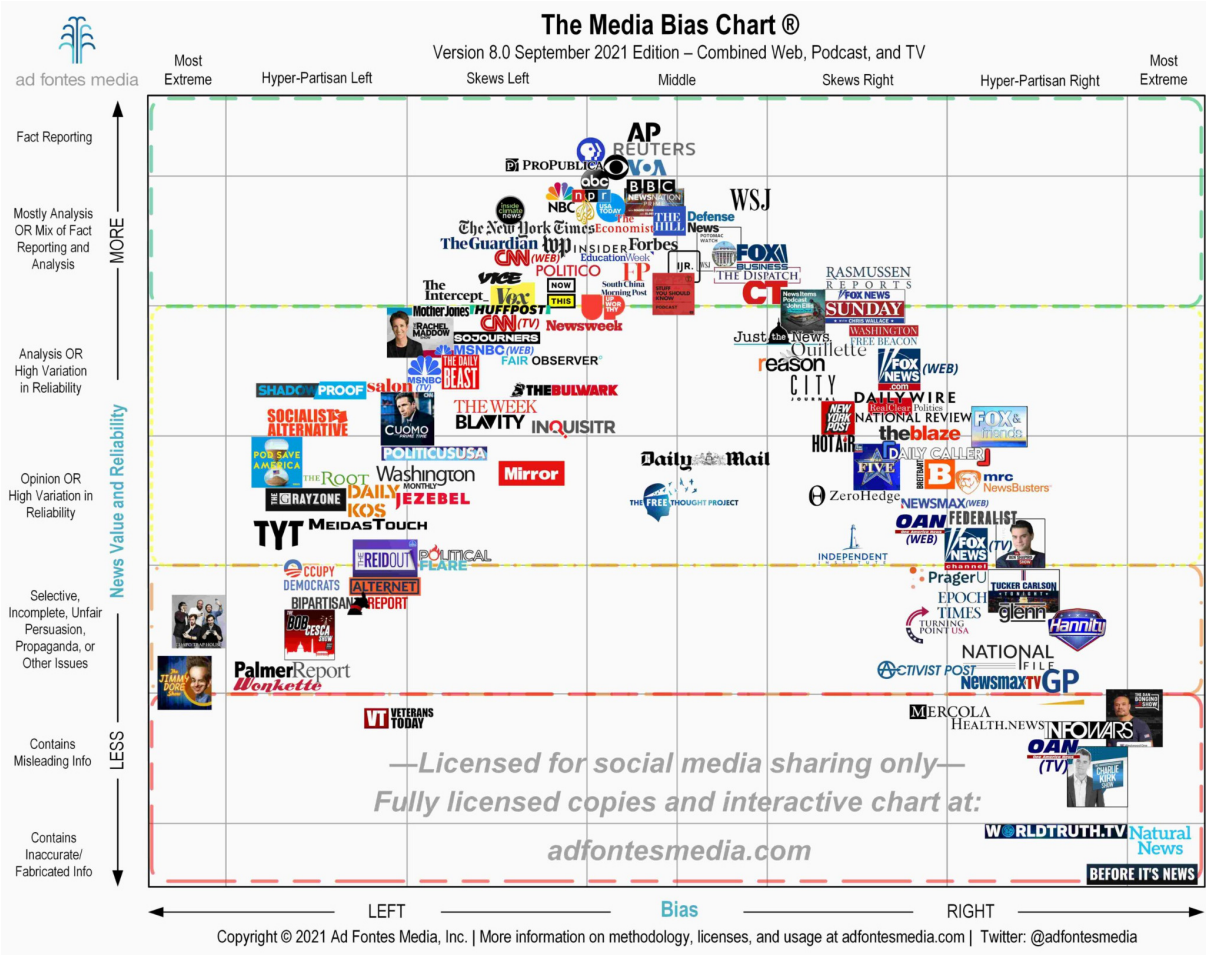

Fig 1: Media Bias Chart from Ad Fontes Media

Using machine learning for this task had not been widely attempted until a major competition in 2019 - SemEval's Hyperpartisan News Detection (PAN, 2019). This saw the release of a 750,000 article labelled dataset, and over 40 teams completing the task of predicting political bias on a 5 point scale, some of which went on to publish papers on their methods and findings (Kiesel et al., 2019).

Following this growth in papers on the subject came 'We Can Detect Your Bias: Predicting the Political Ideology of News Articles' from researchers at MIT and the Qatar Computing Research Institute in late 2020 (Baly et al., 2020). This was one of the first papers to look at a dataset which used individually annotated articles rather than projecting a bias label from a website onto all its articles, which is a form of distant supervision. This dataset came from AllSides, the media bias rating website mentioned earlier, and the project achieved an accuracy of 79.83% using BERT to classify whether an article was left, right or centered.

2.2 NLP Models

Natural language processing, or NLP, began as a field of research in the 1950s with symbolic NLP, which focused on rule-based parsing of text, with rules built from human-readable symbols and concepts, e.g. parts of speech, as well as mathematical logic, e.g. semantic relations between words (Russell & Norvig, 2009, p.19). Symbolic AI is built

around our known rules of language, leading it to be known as a ‘transparent box’ rather than a ‘black box’. All of these areas would be key to later development in the field.

After a period of low funding for artificial intelligence through the 1970s, statistical NLP broke through after advancements in computing power and mathematical tools (Russell & Norvig, 2009, pp. 25-26). This period formed the introduction of machine learning for natural language processing, and grew as more data was available throughout the 2000s.

The last 10 years have marked the largest breakthroughs in natural language processing, as neural NLP has developed. This began with Word2vec in 2013, which takes as an input a large corpus of words and creates a vector space of several hundred dimensions, where word vectors positioned in close proximity in the vector space share common contexts (Mikolov et al., 2013). This allows us to answer questions such as verbal analogies (‘king is to queen as prince is to princess’), or knowledge discovery, such as the uncovering of novel relationships between diseases and disease-genes associations in 2016 (Gligorijevic et al., 2016).

While Word2Vec was useful for semantic and syntactic relationships, it had several limitations such as not being able to handle out of vocabulary words and an inability to use parameter sharing for cross-lingual use. The next few paragraphs will also explain how having multiple **context-dependent** vectors rather than a single vector for each word can vastly improve upon Word2vec’s performance.

During 2014 and 2015, recurrent neural networks became popular for classification (e.g. political bias) and sequence-to-sequence tasks (e.g. language translation), using encoder-decoder models. RNNs did this better than Word2vec, however, a major downfall was it being difficult to process longer sentences and not being able to process sentences in parallel. This problem was solved with transformer models in Google’s groundbreaking paper: Attention Is All You Need (Vaswani et al., 2017).

A new type of architecture, known as a transformer model, was introduced in this paper. The key architecture of transformers is known as an encoder-decoder architecture. The diagram below illustrates how encoders are ‘stacked’ and the output of this is fed to a stack of decoders.

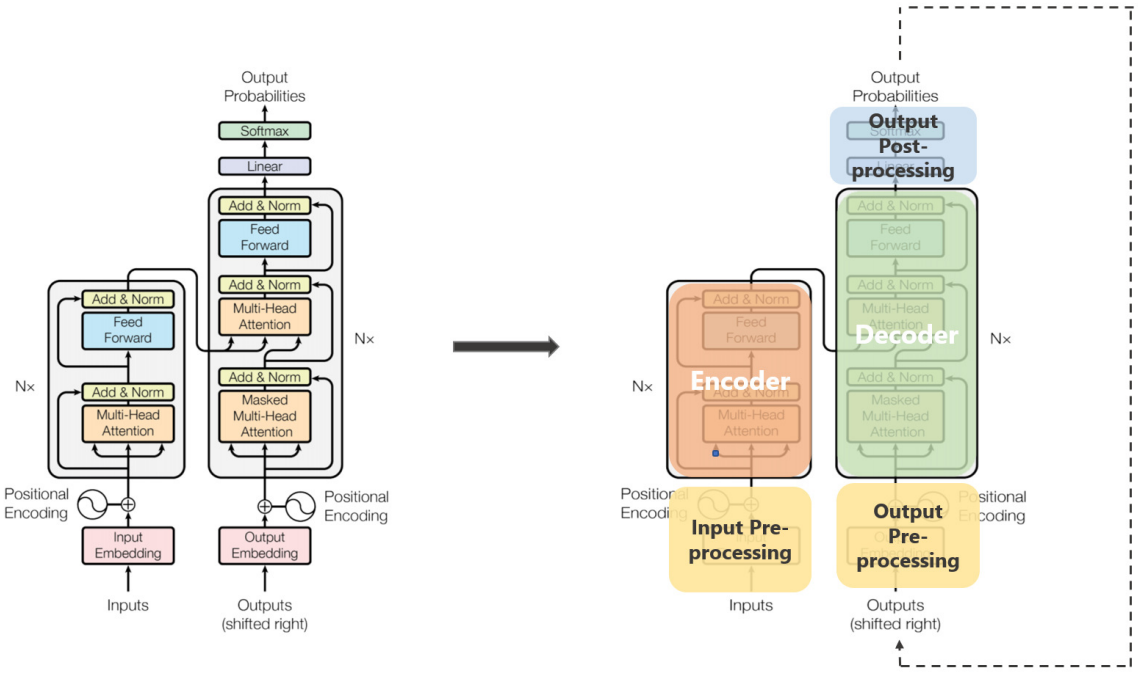

Fig 2: Transformers architecture from ‘Attention Is All You Need’

Similarly to Word2vec, the transformer begins with vector embeddings of words, but builds on this concept in many, many ways. The input embedding is a vector representation of the text, and uses a word embedding of our choice. This is followed by a positional encoding vector with a unique ID for each position in the sentence.

The stacked encoders capture raw properties such as parts of speech or dependencies and then more elaborate properties such as subtle relations between words. Each encoder begins with a self-attention layer and then a feed forward neural network.

Attention layers compute **importance** and look at the entire sequence of words at once; it is considered bi-directional, though it would be more accurate to say non-directional. Self-attention computes dependencies between different words and different parts of the sentence. There are often multiple attention ‘heads’ used at the same time to capture all the links within a sentence. This concept overcomes the limitations of RNNs being unable to cope with longer sentences.

The decoder also uses an attention layer and feed-forward neural network, however, it also uses a special encoder-decoder attention layer, which allows it to focus on the appropriate input part. The decoder is also composed of a linear layer followed by a softmax to solve the specific NLP task at hand.

There were a few major benefits of this new architecture. The first was faster training times due to parallelization; each word has its own path that can be computed in parallel to other words, so they do not have to wait for the previous word. The second was the non-use of recurrent neural networks, meaning transformers could deal with longer and more complex input data using just the attention mechanism.

Over the years following the transformers paper, there were many improvements upon the transformers architecture, one of the most notable being BERT - Bidirectional Encoder Representations from Transformers (Devlin et al., 2018). This built on the transformer architecture in two key ways - masked word prediction (MLM) and next sentence prediction (NSP).

Masked word prediction masks 15% of the words in a sequence and the transformer is tasked with predicting these. It uses this to improve the loss function, and improve the performance of the encoders. Next sentence prediction is implemented by half of the sentences in a sequence being the true next sentence, and half being random next sentences; BERT is then tasked with predicting which are true next sentences and which are not. Both of these methods led to improved accuracies on a range of NLP training tasks.

BERT was pre-trained on a huge amount of data (over 2,500 million words from Wikipedia and the BooksCorpus), and can then be fine-tuned on a specific dataset. This is known as **transfer learning** - it transfers the knowledge of a pre-trained model to a new model by initialising the second model with the first model's weights.

Fig 3: Parameter counts of several recently released pretrained language models

Several other papers were released with improvements upon the transformers architecture - RoBERTa , Ernie 2.0, XLNet, DistilBERT and more. These involved improvements such as fine-tuning of parameters, speed increases, and multi-task learning. The goal of many new models involves creating larger and larger models with minimal accuracy improvements, DistilBERT instead reduces the size of BERT by 40%, while retaining 97% of its performance capability and being 60% faster, making it a good choice for limited computing power (Sanh et al., 2019).

2.3 Conclusions

The study ‘We Can Detect Your Bias: Predicting the Political Ideology of News Articles’ (Baly et al., 2020) provides an excellent baseline of 79.83% accuracy to use as a measure against my own model. While many high accuracies have been achieved, including one of 82% for the PAN SemEval Hyperpartisan News Detection competition, there is a lack of ability for the general public to use these models to provide context for their own reading. This leaves a gap in the literature for a user-facing system which incorporates a state-of-the-art political bias model. Looking at the impact of political bias on the general public, there is a strong case for a platform like the one I intend to build.

Another issue that arose in the literature for political bias was the impact of models learning to classify by detecting the source of an article rather than its political ideology. The largest dataset for hyperpartisanship classification (PAN SemEval) is rated by publisher, rather than on an individual article basis. This is something to explore further and factor in when choosing datasets and evaluating the model.

It is clear that recent transformer models produce some of the highest accuracies for classification tasks, including tasks like mine - a multi-label classification of long form text data. The speed and accuracy increases from parallelization and transfer learning will provide major benefits to my own model. A factor to consider is the limited computing power available for this project, as well as time constraints when running the model. DistilBERT provides a solution for this, and could be a practical choice for my model’s framework.

3 Methodology

The methodology section outlines some of the method and design choices for the project. Some of the tools and methods used are decided here, and some are chosen later during the implementation section. This is due to the nature of an agile software development approach - the key decisions such as the model used and API framework are decided here, but other smaller decisions may need to be made based on evaluating the previous step and its success.

I began my approach by defining my requirements - what I needed my system to do. This naturally fell into five sections:

1. Choosing a dataset
2. Building a model
3. Using a model with an API in real-time
4. Finding static domain scores for each news outlet
5. Building a front end for users to interact with the model

3.1 Choosing a dataset

I began by searching for datasets that were labelled and pertained to political bias or hyperpartisanship. I found the Hyperpartisan News Detection Dataset, created for PAN @ SemEval 2019 Task 4 (PAN, 2019). There were not many other public datasets available for this use case and the Hyperpartisan News Dataset was perfectly aligned with the work I was doing. It was also important to me to find a dataset that was going to produce accurate results, produced by a clear and transparent methodology.

Given a news article, it decides whether it follows a hyperpartisan argumentation, i.e., whether it exhibits blind, prejudiced, or unreasoning allegiance to one party, faction, or cause. It has a bias classification label, with possible values including right (0), right-center (1), least (2), left-center (3), left (4).

The dataset is split into “bypublisher” which uses the overall bias of the publisher as provided by BuzzFeed journalists or MediaBiasFactCheck.com to label the articles. It contains 600,000 articles in the training set and 150,000 in the validation set, both of which have an even split of left and right wing articles. No publisher that occurs in the training set also occurs in the validation set, meaning that the model is being validated on new and unseen data. This is important to reduce overfitting and sample bias.

Due to the dataset only being rated by the publisher, there was the potential for bias within training. For example, if a news outlet wrote in a specific style, their articles were likely to be rated as left wing or right wing repeatedly, regardless of the content, due to the publisher rating. This could be overcome by finding a dataset rated by article; unfortunately, the AllSides dataset used in the ‘We Can Detect Your Bias’ paper (Baly et al., 2020) is not publicly available. There is a subset of the Hyperpartisan News Dataset that does this, however, it is significantly smaller (645 instead of 750,000), and uses a binary hyperpartisan classification rather than a 5 point scale.

Within my research, I found a data collection methodology from AllSides (AllSides, 2021) which may be of interest for future work. The organisation's rating system involves a blind bias survey, where people from all sides of the political spectrum rate articles without knowing their source. They factor in the political leaning of the participant, how different bias groups rate content differently and the average rating from all participants across the spectrum. They also factor in independent data and community feedback to give a confidence score on all of their ratings.

This methodology is far more rigorous and scientific than any other I could find, and would be great to implement on datasets collected by researchers, however as mentioned above, their dataset is not publicly available, so could not be used for this task.

3.2 Building a model

Now that I had a dataset, I began looking at models. Following my literature review, I elected to use the DistilBERT model as this is adapted from BERT, one of the best performing NLP transformers, but reduces the size of BERT by 40%, retains 97% of its performance capability and performs 60% faster (Sanh et al., 2019). Due to the limited computing capacities available to me, the speed and size benefits were a high priority.

After choosing the model, I researched the practicalities of how to implement it. The DistilBERT model was published by the team at Hugging Face, who provide an open source transformers library (Hugging Face, 2021). The library provides general purpose architectures for preprocessing data, fine-tuning and evaluating a transformer model, while also providing flexibility for adjusting parameters and evaluation measures.

Some parameters I am able to change are the length of the input, number of attention heads, dropout rate, batch sizes, and the learning rate. Parameters that were the most frequently tested in the literature review were:

- the learning rate - a tuning parameter in the optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function
- batch size - the number of samples from the training dataset that will be used to estimate the error gradient before the model weights are updated

I can test these for standard learning rates and batch sizes on a subset of the dataset to determine the optimal learning conditions for the model.

Evaluation of the model can be done against standard NLP benchmarks such as GLUE (Wang et al., 2018) or XTREME. These are useful for measuring accuracy against other popular models, especially when publishing a new type of model. Alternatively it can be evaluated against the validation set for standard measures such as precision, accuracy and F1 score. I have elected to use this method as it will give me key evaluation figures for the specific task at hand.

3.3 Using a model with an API in real-time

Following having built the model with Python and PyTorch, it would be natural to use a Python based API for real-time predictions. One of the most popular frameworks for this is Flask[3]. Using Flask, I will be able to load the trained model's weights into the API, and predict the political bias for a single article. The transformers library from Hugging Face can also be used within this Python framework for the tokenization of the article, prior to running the article through the model.

Another problem to tackle with the API is how to get article data from a URL. For this, I found the Newspaper3k library[4], which is an article scraping and extraction tool built for use with NLP tasks. It will allow me to extract an article's title, text and domain, in order to predict its bias and compare it to the domain average.

The API must also be able to receive feedback about the article scores and store these. Some popular options for databases are MongoDB and MySQL. The key consideration between these is whether to use a relational or non-relational database. I will use MongoDB as it offers the benefit of no predefined schema, allowing me flexibility in updating and adding to the database, making it easy to adapt and enhance the database over time with potential future development.

3.4 Getting domain scores

A key functionality of the system proposed is the ability to compare a news article's score to the average political bias score of its news outlet, and other news outlets. The Newspaper3k library mentioned above also allows me to gather articles from a domain, parse their text and then run all articles through my model.

Some domain scoring methods I could attempt are finding the average of all a domain's articles' bias predictions, finding the most frequent bias rating or to find some type of weighted average for all of the articles. Comparing the results of these will allow me to evaluate my domain scores critically. These can also be compared to current media bias ratings from AllSides and Ad Fontes Media for further contextualisation and evaluation.

3.5 Building a front end

The front-end will be built using ReactJS[5], a popular Javascript front-end framework built by Facebook. A key benefit of using React is the ability to create reusable components throughout the front-end and the http and axios libraries that will allow me to make API requests to the back-end easily. The community support and large collection of libraries built for React also make it a sensible choice for the project.

[3] Flask, https://flask.palletsprojects.com/en/2.0.x/
[4] Newspaper3k, https://newspaper.readthedocs.io/en/latest/index.html
[5] ReactJS, https://reactjs.org/

The design aims to be clear and minimalistic, and provide an intuitive user experience. For this, the key user flow is defined below, and can be used during the implementation stage to inform design decisions.

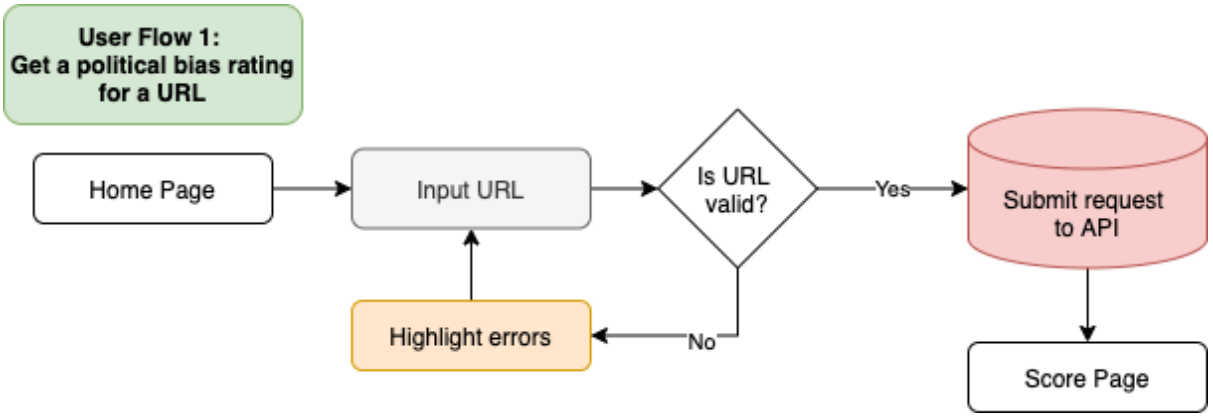

Figure 4: User flow for getting a political bias rating for an article

For version control for both the front-end and back-end, I will use GitHub[6]. This will allow me to revert to previous versions of the project and protect the code against any breaking changes during updates.

[6] GitHub, https://github.com/

4 Architecture

Below is a diagram and explanation of the full architecture of the project, including all tools used. This helps to give an overview of the different parts involved in building the machine learning pipeline - from the model and database at one end to the user at the other.

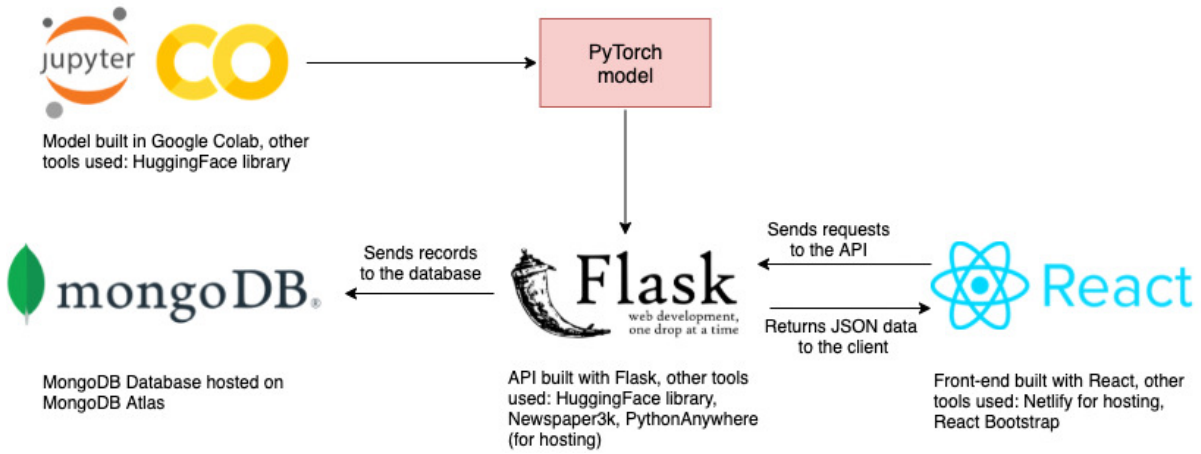

Figure 5: Project architecture

The user inputs a URL using the React user interface, which (if valid) is then sent to the Flask API. The Flask API uses the Newspaper3k library to download and parse the article, after validation checks, e.g. is it a news article, is the article in English. Once the article title and text have been downloaded, the text is tokenized using the HuggingFace library and the model weights are loaded. The article is processed by the model and a prediction is made, which is sent back to the front-end as a JSON document, along with the article's title, text and domain.

Once the political bias score has been displayed at the front-end, the user will be able to provide feedback on whether they believe the score to be an accurate representation of the bias of the article. This feedback rating is passed to the Flask API, which sends the feedback to a MongoDB database where it is stored. This did not get implemented for this project, but remains in the architecture diagram as an example of how the implementation would be done.

The model is built on Google Colab using the HuggingFace library. The decision to use Google Colab came from the ability to run code cell-by-cell which aids debugging, evaluating and reporting metrics. Furthermore, the option to connect to Google's GPUs and TPUs allowed me to run my models faster.

5 Implementation

The implementation section explains each section of the project in more detail - what I did, how I did it, any problems that arose and how I overcame them. Some of the code may be simplified for the purposes of explanation and conciseness.

5.1 Building the model

Implementation began by using HuggingFace's datasets library to import the PAN SemEval Hyperpartisan News Detection Dataset. As the dataset was already in their library, this made loading the train and validation sets a simple process.

```
train_dataset = load_dataset("hyperpartisan_news_detection", "bypublisher",
split="train")

validation_dataset = load_dataset("hyperpartisan_news_detection",
"bypublisher", split="validation")
```

Unfortunately, the dataset had been incorrectly loaded into the HuggingFace library, and after achieving a 98% accuracy on my model, I investigated and found that the train and validation set were the same dataset. To solve this, I used only the train set with an 80:20 train/validation split.

Tokenization

I used HuggingFace's AutoTokenizer function for tokenization of the datasets. This loads the tokenizer used for DistilBERT. I used padding and truncation to ensure all of my tokenized articles were the same size. Following this, I removed attributes that were not being used such as the url and published_at attributes, and renamed the bias column to labels. This step was necessary for the model to train properly.

```
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")

def tokenize_function(example):
  return tokenizer(example["text"], padding=True, truncation=True)

tokenized_train = train_dataset.map(tokenize_function, batched=True)
tokenized_validation = validation_dataset.map(tokenize_function,
batched=True)
```

TPU

Initially I used a GPU to run the model. This had a training time of 8 hours, which was unfeasible as Google Colab often times out before then, or restarts the hosted runtime. For the final model I used a Tensor Processing Unit, or TPU, which is an application specific circuit built specifically for neural network machine learning. It has a much higher performance per watt of energy than CPUs and GPUs. The core design benefit of the TPU is the use of their matrix multiply unit (MXU), or large systolic array - a large hard-wired matrix calculator that does not need memory access after every calculation. As the core of neural networks is a set of matrix multiplications and additions, this makes the TPU perfectly designed to speed up processing of them, while using less memory.

To use the TPU for my model, I used HuggingFace's Accelerate library, the initial code needed is shown below:

```
from accelerate import Accelerator, DistributedType

# Initialize accelerator
accelerator = Accelerator()

def collate_fn(examples):
   # Pad everything to the same length on TPU or training will be very slow
   if accelerator.distributed_type == DistributedType.TPU:
       return tokenizer.pad(examples, padding="max_length", max_length=128,
return_tensors="pt")
   return tokenizer.pad(examples, padding="longest", return_tensors="pt")

# Instantiate Dataloaders
train_dataloader = DataLoader(
   tokenized_train, shuffle=True, collate_fn=collate_fn,
batch_size=batch_size
)

eval_dataloader = DataLoader(
   tokenized_validation, shuffle=False, collate_fn=collate_fn,
batch_size=EVAL_BATCH_SIZE
)
```

The data loaders here create batches from our data. Batches are smaller groups of data, specifically groups of 16 articles in our dataset. Each batch is processed by the model in turn, meaning that only 16 articles are loaded into the memory at each time, and only 16 error and loss values need to be stored by the memory at any one time, instead of the entire dataset of 600,000 articles. The model also updates the weight and hyperparameters after each batch, so the performance of the model is affected by batch size.

There are a few more small steps to use the accelerator. These are shown overleaf. First, we instantiate the DistilBERT model with its pretrained weights. We then convert the model to a TPU optimized model and prepare all our data using the accelerator's prepare function.

```
# Instantiate the model
model = AutoModelForSequenceClassification.from_pretrained(
              "distilbert-base-uncased", return_dict=True, num_labels=5)

# Runs model on TPU
model = model.to(accelerator.device)

# Prepare everything
model, optimizer, train_dataloader, eval_dataloader = accelerator.prepare(
   model, optimizer, train_dataloader, eval_dataloader
)
```

Optimizer

The optimizer is the mechanism by which the model's loss is calculated, and model weights are incrementally improved. One of the most common optimizing methods is stochastic gradient descent. The gradient descent part is an iterative algorithm for finding the local minimum of a differentiable function. The stochastic part refers to stochastic approximation, which involves taking a subset of the data to calculate the gradient rather than the entire dataset, alleviating the computational burden. For batched data, there also exists batch gradient descent and mini-batch gradient descent.

For each iteration of the algorithm, the losses are calculated, and the optimizer adjusts the weights to find a minimum, where the loss is as close to zero as possible. The optimizer chosen for this project is AdamW (Loshchilov & Hutter, 2019), which is an adaptive gradient algorithm that builds on stochastic gradient descent. While SGD uses one learning rate for the entire process, AdamW uses dynamic different learning rates for each parameter. It has been shown to produce better optimization results than SGD on tasks similar to mine.

AdamW uses a 'warm-up' which are steps with a low learning rate at the beginning of training to allow the optimizer to compute the correct statistic of the gradients without early over-fitting.

Training and evaluating the model

We are now ready to train the model. The data has been prepared and the optimizer and model initialized. Training is done in epochs - this is simply the number of passes that the model completes for the entire dataset. There is usually a point at which the model reaches an accuracy that will not be improved by further training. Time taken to train is also a factor in choosing how many epochs to run the model for.

```
for epoch in range(num_epochs):
    model.train()

    for step, batch in enumerate(train_dataloader):
        outputs = model(**batch)

        loss = outputs.loss
        loss = loss / gradient_accumulation_steps
        accelerator.backward(loss)
        if step % gradient_accumulation_steps == 0:
            optimizer.step()
            lr_scheduler.step()
            optimizer.zero_grad()

    model.eval()
    # Init empty arrays for predictions and golds
    predictions, true_vals = [], []

    for step, batch in enumerate(eval_dataloader):

        with torch.no_grad():
            outputs = model(**batch)
        predictions = np.append(predictions, (outputs.logits.argmax(dim=-1))
                                .flatten().cpu().numpy())
        true_vals = np.append(true_vals, batch['labels']
                                .flatten().cpu().numpy())

    val_f1 = round(f1_score(predictions, true_vals, average='macro'), 2)
    accelerator.print("F1 Score: {}".format(val_f1))
    accelerator.print("Classification report")
    accelerator.print("\n" + classification_report(true_vals,predictions))
```

For each epoch, the model calculates the predictions for each batch. We then compute the gradients using the loss, after which the optimizer makes a training step. We update the learning rate with our scheduler for the next iteration, and zero the gradients before the next loop.

Once we have completed the training for an epoch, we calculate the predictions for our validation dataset and print the F1 score and classification report for that epoch. This allows us to see the progress of our metrics after each epoch.

Finding optimal parameters

Before the full model is trained, tests were run on a subset of the data to find the optimal parameters. As decided in methodology section 3.2, the learning rate and batch size were tested, and can be set in our model's config. The tests were carried out on a training set of size 60,000 and validation set of size 15,000 (an 80%/20% split), with a training time of roughly 30 minutes per test.

The results of the tests can be found in the appendix. The optimal learning rate was 2e-5 and the optimal batch size was 8, however, this yielded a very small improvement over a batch size of 16, and took 30% longer. Due to time constraints with Google Colab, I elected to use a learning rate of 2e-5 and batch size of 16.

Final model results

The final model classification report can be seen below.

```
epoch 3:
F1 Score: 0.95
Classification report

              precision    recall  f1-score   support

         0.0       0.97      0.96      0.96     30046
         1.0       0.93      0.92      0.93      8641
         2.0       0.99      0.99      0.99     37525
         3.0       0.92      0.94      0.93     13856
         4.0       0.94      0.94      0.94     29932

    accuracy                           0.96    120000
   macro avg       0.95      0.95      0.95    120000
weighted avg       0.96      0.96      0.96    120000
```

Figure 6: Classification report for the fully trained DistilBERT model

The final step was to save the model's weights, so they could be used in the Flask API:

```
# Converts model back to cpu before saving, to run it with Flask locally
model.to('cpu')
model.save_pretrained("./drive/MyDrive/model")
```

5.2 Incorporating domain scores

Finding the domain scores for each news outlet functioned as its own mini project. I began by loading the model into a new Google Colab notebook. As mentioned in methodology section 3.4, the Newspaper3k library allows me to gather articles from a domain, download their text and then run all articles through my model. My initial method did not obtain the results I was looking for, hence the section below is split into 'Attempt 1' and 'Attempt 2'.

Attempt 1

For my first method, I used the Newspaper3k library to build a news source for each paper of the ten main UK news outlets I chose to incorporate into the project. Using newspaper.build, I was able to construct a list of all recent articles from a news source, which Newspaper3k extracts from their RSS feeds.

```
import newspaper

guardian = newspaper.build('https://www.theguardian.com')
sun = newspaper.build('https://www.thesun.co.uk/')
bbc = newspaper.build('https://www.bbc.co.uk')
....
```

This provided me with between 150 and 500 articles for each news source. To download the content of the articles, I used the library's threading resource to avoid rate limiting. Following this, I predicted the political bias of each article using my trained DistilBERT model, and stored the predictions, the average of all each domain's articles' predictions, and the most frequent bias rating for each article.

The results are shown below. The most frequent bias rating for all news sources was 3 (left-center), and the average rating for each source was within a 0.5 range, between 2.3 and 2.8. Having researched the sources in my literature review and tested various articles manually on the model, this prediction did not seem true to reality.

News Outlet	Mode	Mean
The Guardian	3	2.660516605166052
The Sun	3	2.303116147308782
BBC News	3	2.6444444444444444
The Independent	3	2.37875751503006
The Mirror	3	2.764
Wales Online	3	2.6058091286307055
The Times	3	2.52
The Telegraph	3	2.544
The Express	3	2.32064128256513
The Daily Mail	3	2.3166332665330662

Figure 7: Mode and mean scores for all articles from 10 news sources

I tried to find a resource online to help contextualise where these news sources should potentially be on the political bias scale. I came across a YouGov survey[7] in which participants rated how left or right wing they perceived these news sources to be, and as expected, they were not all center left.

Some potential reasons for this homogenising of political bias scores are:

1. Most articles do not have any political context, and so the average non-political article may be given a rating of 2 or 3.
2. The dataset included US news sources which could be more extreme than the UK, leading to all UK news sources falling in the center of the distribution.
3. The model is working incorrectly.

[7] YouGov, How Left or Right Wing are the UKs Newspapers, https://yougov.co.uk/topics/politics/articles-reports/2017/03/07/how-left-or-right-wing-are-uks-newspapers

The easiest one of these to test was option 3, by manually inputting articles into the system. These all appeared to give the correct political bias rating. Option 2 was a strong possibility, however, rectifying this would require me to redesign the entire project from the initial dataset choice. I decided to tackle option 1, and find articles specific to politics, to see if I could improve the results.

Attempt 2

The Newspaper3k library does not allow for more specific article scraping than the base domain. In order to retry the domain scores with only political articles, I had to manually scrape the political RSS feeds for political articles for each news source.

To do this, I used web scraping library BeautifulSoup[8] and the Python requests library. For each RSS feed, I collected its contents and found every instance of an <item> tag, which represents an article in the feed. I extracted the article links from here and stored them in a separate list. The code can be seen below.

```
import requests
from bs4 import BeautifulSoup

def get_feedlinks(feed):
 response = requests.get(feed)
 webpage = response.content
 soup = BeautifulSoup(webpage, features='xml')
 items = soup.find_all('item')
 articles = []
 for item in items:
     link = item.find('link').text
     articles.append(link)

 print(len(articles))
 return articles
```

A limitation of this method was the small number of articles available from each feed. This was between 13 and 100 articles. An ideal method might involve collecting the articles over multiple days to build this dataset, however, due to time constraints I proceeded with the small number to see if results improved.

I downloaded each article using Newspaper3k and predicted the political bias of each article using my trained DistilBERT model. I stored the predictions, the average of all each domain's articles' predictions, and the most frequent bias rating for each article. The results are overleaf.

[8] BeautifulSoup4, https://beautiful-soup-4.readthedocs.io/en/latest/

News Outlet	Mode	Mean	Number of Articles
The Guardian	3	2.84615384615	26
The Express	1	1.56	25
BBC News	2	2.42307692308	91
The Independent	3	2.73684210526	19
The Mirror	3	2.56	25
Wales Online	3	2.61538461538	13
The Times	3	2.32	25
The Telegraph	3	2.36	100
The Sun	1	1.42857142857	14
The Daily Mail	1	1.24	100

Figure 8: Mode and mean scores for political articles from 10 news sources

The results were more spread and offered a more realistic interpretation of the political bias of each news source, which roughly matched the YouGov survey mentioned in the previous section, and could be used for the project. None of the news sources had a rating of 0 (right) or 4 (left), which indicated that there may have been some merit to the hypothesis that UK news sources were more politically centered than the US sources the data was trained on.

5.3 Building the API

I began building the API by installing Flask and creating a virtual environment to manage my Python dependencies. A simple ‘Hello World’ app looks like this.

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

Only one initial route was required. This was the /score GET route.

```
from flask import Flask, request, jsonify
from newspaper import Article
from model.model import run_model
from urllib.parse import urlparse

@app.route('/score', methods=['GET'])
def get_score():
    request_json = request.args.to_dict()
    url = request_json["url"]

    # get domain
```

```
    domain = urlparse(url).netloc
    # get text
    article = Article(url, keep_article_html=True, fetch_images=False)
    article.download()
    article.parse()

    prediction = run_model(article.text)

    data = {
        'title': article.title,
        'text': article.article_html,
        'score': prediction,
        'domain': domain
    }
    return jsonify(data)
```

The code here is the basic code for the route to work; for simplicity, I have omitted error handling and validation, however, you can view this in the full version of the code attached. The route begins by processing the URL from the API request, and using Python's urllib to obtain the base domain from this. It then uses Newspaper3k to download the article, and runs this through the prediction model. Finally, the article's title, text, score and domain and returned to the front-end client as a JSON document.

The prediction happens in a separate controller: model.py.

```
from transformers import DistilBertForSequenceClassification,
AutoTokenizer
import torch
from pathlib import Path


def run_model(text):
    MODEL_PATH = "model/biasmodel"
    model_path = Path(MODEL_PATH).absolute()
    tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased",
                                              use_fast=True)
    model = DistilBertForSequenceClassification
                        .from_pretrained(model_path)

    predict_input = tokenizer(text,
                              truncation=True,
                              padding=True,
                              return_tensors='pt')
    tf_output = model(**predict_input)
```

```
    prediction = torch.argmax(tf_output.logits, dim=-1)
    prediction = (prediction.tolist())[0]

    return prediction
```

This takes the article text as an argument, loads the tokenizer and pre-trained DistilBERT model (contained within the project as a separate file) using HuggingFace, then tokenizes the article and predicts the article's political bias score, which is returned to app.py.

5.4 Building the front-end

The front-end required both UI/UX design and engineering design before beginning coding. The key user flow was defined in the methodology section:

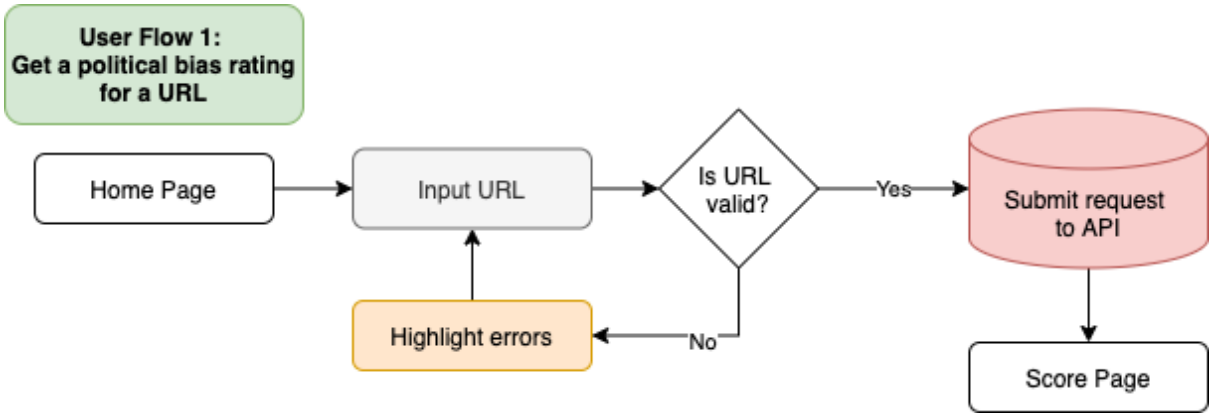

Figure 9: User flow for getting a political bias rating for an article

Wireframes

Two wireframes were drawn up based on this user flow: one for the home page and one for the score page. The homepage is shown below, and consists of a navigation bar, an introduction text, which will explain how to use the website and what it does, an input for the URL and a grid of some example articles and their political bias ratings.

Figure 10: Wireframe for the home page

The score page shown overleaf consists of the same navigation bar for consistency, the article title and text on the left, followed by various ‘cards’ on the right. The score card displays the domain, the average domain score and the score for the article. The feedback card provides a place for feedback on the score and the news sources context will show a list of the ten chosen news sources and their political bias scores, along with highlighting the source that the article is from.

Figure 11: Wireframe for the score page

The design aimed to be minimalistic, and only show the necessary information, so as not to overwhelm the user. Each of the grey boxes shown is built as an individual component, and can be seen in the code attached to this report.

Final UI

The final design can be seen in these screenshots. The API requests for the data were made using the axios library for React. The web platform was deployed through Netlify, using its GitHub repository.

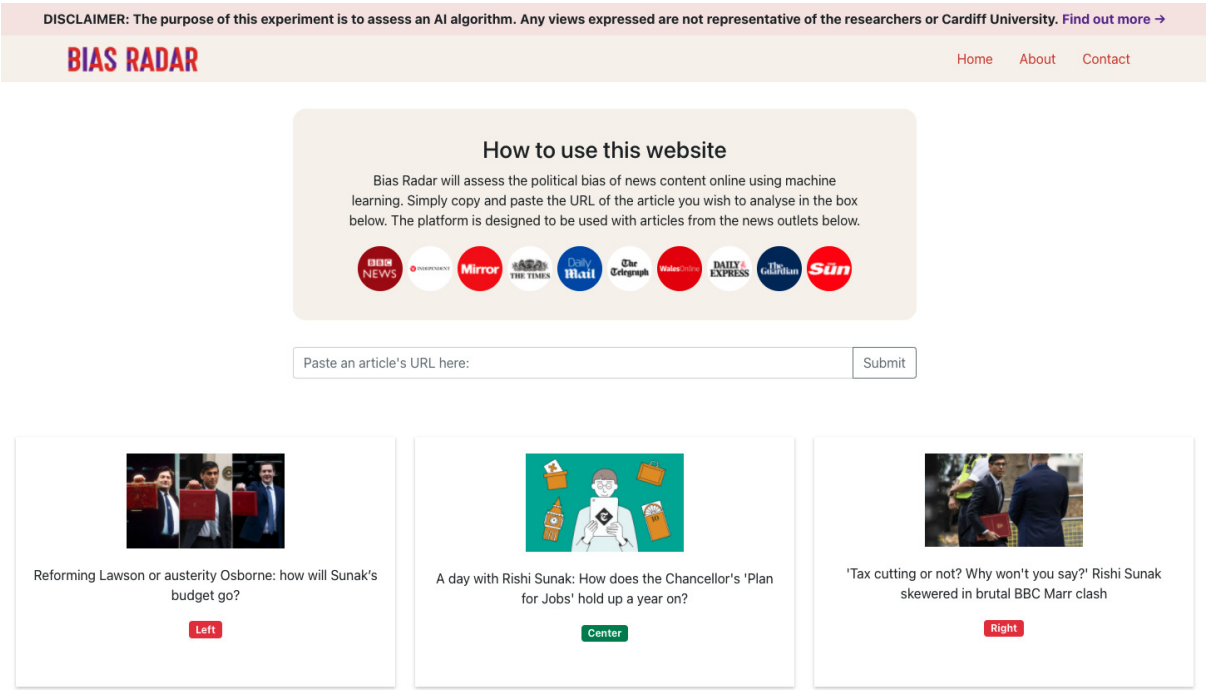

Figure 12: Screenshot of the home page

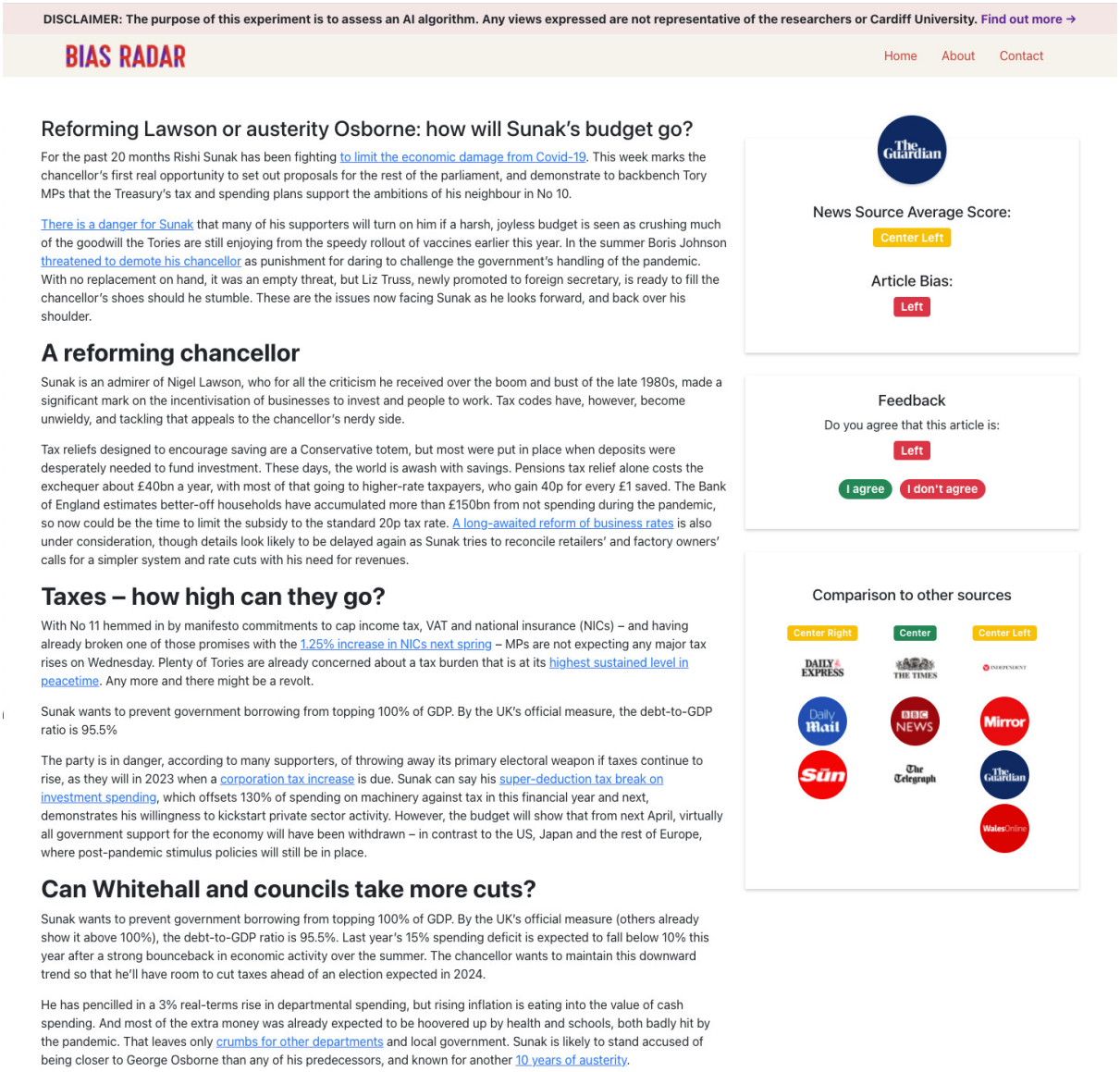

Figure 13: Screenshot of the score page

6 Results and Evaluation

This section details how the system was evaluated against the aims and objectives outlined in section 1.3 and the baseline accuracy of 79.83% for the model outlined in the literature review, section 2.3.

For reference from section 1.3:

> The aims of this project are as follows:
> 1. To accurately detect political bias with a machine learning model
> 2. To develop a web platform in which users can find a bias score for any article online
> 3. To create a clear, easy-to-use and attractive user interface
> 4. To develop bias scores for news outlets, in order to compare them to each other
> 5. To identify which news outlets are more/less biased, and in which direction

Model evaluation

The final model's classification report can be seen below. It achieved a 96% accuracy and 95% F1 score. It was training on 480,000 articles and evaluated using 120,000, in an 80% training, 20% validation split. This is a significant improvement on the baseline of 79% from the 'We Can Detect Your Bias: Predicting the Political Ideology of News Articles' (Baly et al., 2020) paper in 2020.

This paper used data from AllSides with individual rating scores for each article, whereas the Hyperpartican News Detection set I used had ratings based on domain scores. This left my model vulnerable to simply training by detecting the source of an article rather than its political ideology, and could explain the higher accuracy, which may not be representative of a high accuracy in a real-world application, but more a high accuracy at detecting which news source an article came from.This could be rectified by testing against an individual article dataset in future.

```
epoch 3:
F1 Score: 0.95
Classification report
```

	precision	recall	f1-score	support
0.0	0.97	0.96	0.96	30046
1.0	0.93	0.92	0.93	8641
2.0	0.99	0.99	0.99	37525
3.0	0.92	0.94	0.93	13856
4.0	0.94	0.94	0.94	29932
accuracy			0.96	120000
macro avg	0.95	0.95	0.95	120000
weighted avg	0.96	0.96	0.96	120000

Figure 14: Classification report for the fully trained DistilBERT model

Another reason for the high accuracy came from the parameter testing on a subset of the dataset, which allowed me to find the ideal training parameters for the model. Despite my concerns about the nature of the dataset, manual inspection showed that the model does recognise political bias, and is accurate in doing this. Therefore, the first aim of this project has been achieved.

Web platform evaluation

The web platform was a key differentiator of this project against other similar research projects. It carries out all key functionalities needed, this includes:

- Allowing a user to find a political bias score for any article online
- Allowing a user to see how this compares to the overall domain score
- Allowing a user to leave feedback on the political bias score (more below)
- Allowing a user to contextualise this score among other news outlets
- Providing a clear, easy-to-use and attractive user interface

It is available at: https://bias-radar.netlify.app/

As mentioned in the literature review, there is a distinct lack of ability for the general public to evaluate and contextualise their information. A user-facing system allows the project to be larger than a machine learning model created by researchers; it has the potential to become a tool for readers, where they have access to a state-of-the-art political bias model. It also has the potential to mitigate some of the negative effects of political polarization on society.

The tools chosen (React, Flask) allowed me to complete the project successfully, and were fit for purpose. Feedback was implemented on the front-end but not the back-end. This was considered an additional feature, and would be an excellent addition for adding human-in-the-loop interaction to the AI model. With no time constraints, I would have liked to implement this fully with a MongoDB database as was discussed in the architecture section, however, this was not possible for this project, and was not considered one of the core aims.

One way to build a feedback mechanism into the project would have been to collect user feedback into a database and create an internal annotation dashboard which asks an expert to verify user feedback. If more users give feedback on an article and are in agreement, it would receive a higher weighting, and the model would be automatically retrained and fine-tuned after a period of time, or after a specific number of new annotations was reached. This method would prevent single pieces of incorrect feedback skewing the data and would act as a form of crowdsourcing data, which is more reliable than a single annotator.

Domain score evaluation

The domain score section of the project was the most challenging in terms of methodology and achieving ‘good’ results. Some evaluation of this was discussed in the implementation section, the key difficulty being the homogenization of the average score for each domain. This was rectified somewhat by using only political articles instead of general articles, which were all likely to be classified in a similar way.

Although the results were more spread and offered a more realistic interpretation of the political bias of each news source, none of the news sources had a rating of 0 (right) or 4 (left), which indicated that there may have been some merit to the hypothesis that UK news sources were more politically centered than the US sources the data was trained on.

When compared to current media bias ratings from AllSides and Ad Fontes Media for further contextualisation and evaluation, the UK news sources featured were all close to the center of the distribution. Given more time, I would attempt to find or crowdsource a dataset based on only UK news outlets that gives a political bias score for each article rather than each domain. The adversity faced in this section taught me the importance of data exploration and analysis prior to running a model, more of which could have been done for this project.

The final aim of this project was to identify which news outlets are more/less biased, and in which direction. This was achieved in the front-end implementation, in the domain comparison section on the score page.

Conclusions

All aims and objectives set out in section 1.3 were achieved. There are some limitations due to the dataset and the domain scores that were outlined earlier, and could be rectified in future work on the project. The web platform was particularly successful in allowing the general public a way to evaluate and contextualise their news.

7 Future Work

There is a large scope for future additions and improvements to the project. Three interest me in particular, and are discussed below.

The first is the use of narrative clustering. This would be a web scraper and database that collates news articles based around the same topic, to allow users to compare them side-by-side, or to have suggestions available for articles that present a different view on the same topic. This would further the core aim of this project of contextualising news articles for readers.

The second is interpretability. Explainable AI (XAI) is a growing field that attempts to remove some of the ethical questions around black box models such as DistilBERT. Implementations such as LIME and SHAP allow developers to highlight sentences and words that are viewed as particularly right or left wing, to allow us to see in more detail why an article has been classified the way it has.

The final element of future work is implementing the feedback loop discussed in the evaluation section. The use of feedback would add another level to the model, meaning it will not just be trained on publicly available data, but also shaped by the users of the platform.

8 Reflections on learning

In a 2010 commencement address at the University of Michigan, President Obama issued a warning about the state of public discourse in America:

> If we choose only to expose ourselves to opinions and viewpoints that are in line with our own, studies suggest that we become more polarized, more set in our ways. That will only reinforce and even deepen the political divides in this country.

This project has allowed me to take an idea, to facilitate people to be able to distinguish biased information in a world full of ads and hidden agendas, and create it into something tangible and meaningful. The project has played an important role in preparing me for my career professionally and academically. Being able to select a topic I am deeply passionate about has motivated me to look beyond simply creating a model, looking at how I can expand on that to create a useful end-to-end machine learning pipeline.

The project taught me how to understand the problem and break it down into smaller pieces. Initially, there were certain areas that I knew were gaps in my knowledge, e.g. how to take a model from a Python notebook, and use that on a web platform. I chose to stretch myself by including various areas that I had no prior knowledge on, and I believe it paid off not only by expanding my skill set hugely, but also by improving my confidence to tackle completely new types of tasks with minimal support.

The literature review was a section I enjoyed more than expected, and I learned in more detail about the mechanics of different NLP models, and understood them far more deeply than before. Knowing the details of how a transformer works allowed me to make more informed decisions about how I used them. I was not anticipating the issues I had with the domain scores, but there was a valuable lesson there on how to overcome adversity, seek support, and rethink my approach to a problem.

Overall, the project has fuelled my desire to look further into the field of detecting bias and other measures in content online, and I would also like to look further into interpretability and explainable AI in the future. I have gained a myriad of new skills and improved existing ones, both technical and soft skills such as problem-solving and decision making.

Table of Figures

References

AllSides. (2021, October 18). *AllSides | Balanced news via media bias ratings for an unbiased news perspective*. AllSides. Retrieved October 18, 2021, from https://www.allsides.com/unbiased-balanced-news

Baly, R., Da San Martino, G., Glass, J., & Nakov, P. (2020). We Can Detect Your Bias: Predicting the Political Ideology of News Articles. *EMNLP-2020*. arXiv. 2010.05338

Bradshaw, S., & Howard, P. N. (2019). The Global Disinformation Order: 2019 Global Inventory of Organised Social Media Manipulation. *Oxford, UK: Project on Computational Propaganda*. https://demtech.oii.ox.ac.uk/wp-content/uploads/sites/93/2019/09/CyberTroop-Report19.pdf

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018, October 11). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv*. 1810.04805

Gligorijevic, D., Stojanovic, J., Djuric, N., Radosavljevic, V., Grbovic, M., Kulathinal, R. J., & Obradovic, Z. (2016, August). Large-Scale Discovery of Disease-Disease and Disease-Gene Associations. *Scientific Reports*, *6*(1), 32404. ResearchGate. 10.1038/srep32404

Hugging Face. (2021, October 6). *Transformers - transformers 4.11.3 documentation*. Hugging Face. Retrieved October 18, 2021, from https://huggingface.co/transformers/index.html

Keating, J., Van Boven, L., & Judd, C. M. (2016). Partisan underestimation of the polarizing influence of group discussion. *Journal of Experimental Social Psychology*, *65*, 52-58. Science Direct. https://doi.org/10.1016/j.jesp.2016.03.002.

Kiesel, J., Mestre, M., Shukla, R., Vincent, E., Adineh, P., Corney, D., Stein, B., & Potthast, M. (2019). SemEval-2019 Task 4: Hyperpartisan News Detection. In *Proceedings of the 13th International Workshop on Semantic Evaluation* (pp. 829-839). Association for Computational Linguistics. 10.18653/v1/S19-2145

Loshchilov, I., & Hutter, F. (2019, January 4). Decoupled Weight Decay Regularization. *ICLR 2019*. arXiv. 1711.05101v3

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013, January 16). Efficient Estimation of Word Representations in Vector Space. *arXiv*. 1301.3781v3

Mitchell, A., & Weisel, R. (2014, June 12). *Political Polarization in the American Public*. Pew Research Center. Retrieved October 18, 2021, from https://www.pewresearch.org/politics/2014/06/12/political-polarization-in-the-american-public/

National Union of Journalists. (1936). *NUJ code of professional conduct*. National Union of Journalists. Retrieved October 18, 2021, from https://www.nuj.org.uk/about-us/rules-and-guidance/code-of-conduct/first-nuj-code-of-conduct-1936.html

PAN. (2019). *Hyperpartisan News Detection*. PAN. Retrieved October 18, 2021, from https://pan.webis.de/semeval19/semeval19-web/#data

Russell, S. J., & Norvig, P. (2009). *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall Press. ISBN:0136042597

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019, October 2). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv*. 1910.01108

Spohr, D. (2017, August 23). Fake news and ideological polarization: Filter bubbles and selective exposure on social media. *Business Information Review*, *34*(3), 150-160. 10.1177/0266382117722446

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017, June 12). Attention Is All You Need. *Advances in neural information processing systems*, 5998-6008. arXiv. 1706.03762v5

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2018, April 20). GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. *ICLR 2019*. arXiv. 1804.07461

YouGov. (2019, December 16). *Do people in the UK trust the media?* YouGov. Retrieved October 18, 2021, from https://yougov.co.uk/topics/politics/articles-reports/2019/12/16/do-britons-trust-press

Appendix

Results of optimal parameters tests

Learning rate 2e-4, batch size 16:

```
epoch 3:
F1 Score: 0.86
Classification report

              precision    recall  f1-score   support

         0.0       0.90      0.89      0.90      3033
         1.0       0.81      0.79      0.80       847
         2.0       0.97      0.97      0.97      3779
         3.0       0.80      0.79      0.79      1357
         4.0       0.83      0.84      0.84      2984

    accuracy                           0.89     12000
   macro avg       0.86      0.86      0.86     12000
weighted avg       0.89      0.89      0.89     12000
```

Learning rate 2e-5, batch size 16:

```
epoch 3:
F1 Score: 0.9
Classification report

              precision    recall  f1-score   support

         0.0       0.93      0.92      0.93      3033
         1.0       0.88      0.86      0.87       847
         2.0       0.98      0.97      0.98      3779
         3.0       0.82      0.91      0.86      1357
         4.0       0.89      0.87      0.88      2984

    accuracy                           0.92     12000
   macro avg       0.90      0.91      0.90     12000
weighted avg       0.92      0.92      0.92     12000
```

Learning rate: 2e-6, batch size 16:

```
epoch 3:
F1 Score: 0.84
Classification report

              precision    recall  f1-score   support

         0.0       0.89      0.86      0.87      3033
         1.0       0.84      0.75      0.79       847
         2.0       0.98      0.93      0.96      3779
         3.0       0.69      0.85      0.76      1357
         4.0       0.80      0.82      0.81      2984

    accuracy                           0.86     12000
   macro avg       0.84      0.84      0.84     12000
weighted avg       0.87      0.86      0.87     12000
```

Learning rate 2e-5 most successful in the above three tests. This is then tested against two other batch sizes.

Learning rate 2e-5, batch size 8:

```
epoch 3:
F1 Score: 0.91
Classification report

              precision    recall  f1-score   support

         0.0       0.93      0.93      0.93      3033
         1.0       0.88      0.87      0.88       847
         2.0       0.98      0.98      0.98      3779
         3.0       0.84      0.90      0.86      1357
         4.0       0.90      0.88      0.89      2984

    accuracy                           0.92     12000
   macro avg       0.90      0.91      0.91     12000
weighted avg       0.92      0.92      0.92     12000
```

Learning rate 2e-5, batch size 32:

```
epoch 3:
F1 Score: 0.9
Classification report

              precision    recall  f1-score   support

         0.0       0.92      0.92      0.92      3033
         1.0       0.87      0.84      0.86       847
         2.0       0.98      0.97      0.98      3779
         3.0       0.82      0.90      0.86      1357
         4.0       0.88      0.86      0.87      2984

    accuracy                           0.91     12000
   macro avg       0.89      0.90      0.90     12000
weighted avg       0.91      0.91      0.91     12000
```