Classifying Political Bias in News Articles

Rhys Mackenzie*, Gabriel Iturriaga†, Courtney Pick‡, and Ben Kelly§
Dalhousie University, Halifax, NS, Canada
Email:*rhys.mackenzie@dal.ca, †gabriel.iturriaga@dal.ca, ‡cr693886@dal.ca, §ben.kelly@dal.ca

Abstract—Misinformation in the news is on the rise, particularly in the United States. The public should be aware of biases present in news articles that tend toward Republican (right) or Democratic (left) ideology. This paper presents a web application that attempts to classify news articles as being either left- or right-biased. Four machine learning models are presented to accomplish this task: a stochastic gradient descent model written with the Python library's Sci-Kit Learn, Google's BERT model, Facebook AI's RoBERTa model, and Hugging Face's DistilBERT model. A training dataset was assembled by scraping articles from Reddit. Articles were labelled as left-leaning or right-leaning based on the subreddit they were posted in. Experiments were also performed using the Hyperpartisan database available via Hugging Face. The Hyperpartisan dataset was deemed nonviable due to an excess of noise in the data. Models trained using the data gathered from Reddit performed well. The stochastic gradient descent achieved the highest test accuracy with a score of 0.977. We expect that this model performed best because it can accept entire articles as input, rather than the first 512 characters as is the case with the other models. All four models are incorporated into the web application.

1 Introduction

With the rise of misinformation in news and media, it is essential as consumer to be critical of the content you consume, as well as understand the biases at play. Among the most biased media is political news articles. Often times, news articles contain a political stance on the topic they are delivering to consumers. In the United States, the most common biases present are Republican (right) and Democratic (left). News and media literacy is extremely important to recognize whether or not the content you are consuming is credible. Gaining news and media literacy is not an easy task for readers as there are many factors to look out for when consuming content. The objective of this research is to deliver a proof-of-concept tool that employs a machine learning model that predicts the bias of news articles. The goal is that consumers can use this tool to understand the biases in the content they consume on a daily basis, improving their news and media literacy.

We provide a simple web application where users can submit a URL to a news article and receive a prediction of either "Left" or "Right", representing the bias of the article. For comparison, the web application displays predictions from 4 different models. Three of the four models are pre-trained transformer models: BERT, RoBERTa, and DistilBERT. The final model is a linear model.

Subsequent sections will be organized as follows. In Section 2, we describe related work. In Section 3, we define the research problem. In Section 4, we describe the methodology used to build a data set, create the machine learning models, and describe our application. In Section 5, we summarize the findings of our research.

2 Related Work

Detecting political bias in news is not an exhaustively researched topic, leaving plenty of room for new innovations. Previous work [1] has shown that linear models can perform well when using word-embeddings as the features when detecting extreme bias in news articles. The same work provided two data sets, the first of which had 1273 manually labeled articles, the second contained 754,000 articles that were automatically labeled based on the publisher they came from. The highest accuracy received on the smaller data set was 0.822, where a linear model was used with the pre-trained model BERT providing the word-embeddings for features. The larger data set proved less useful as it had too much noise, with the best accuracy being 0.706.

Previous work has analyzed using BERT-based models for detecting emotions in text [2]. The models anaylzed were BERT, RoBERTa, DistilBERT, and XLNET. RoBERTa was found to perform the best on the emotions data set with an accuracy of 0.7431. In terms of training time, DistilBERT took the shortest amount of time to train, with RoBERTa in second place.

3 Problem Definition

Due to the decrease of trust in the media and increasingly polarizing political views, it has become difficult to determine what you can trust. Our goal is to deliver an application that allows users to better determine the political biases that may be at play. When a user provides a URL to a news article, our application will classify the political bias present in the news article to be left- or right-leaning.

4 Problem Methodology

Our methodology for creating a solution to this problem contained three fundamental steps. The first step was to create or find a suitable dataset. To address our problem, the dataset needed to contain raw text from news articles with a label demonstrating the type of political bias present in that article. Moreover, the dataset needed to have thousands of examples so that the machine learning models could effectively learn how political bias presents in news articles.

The second step was building or implementing various machine learning models that were capable of language comprehension. We experimented with various forms of machine learning models in order to find the model that would make the most accurate predictions. The success of the model depended on the accuracy it achieved when predicting the political bias of unseen data.

The final step of the project was making the knowledge of the machine learning models accessible to users in the format of a web app. This web app would allow users to enter the URL of the news article they are reading and receive a prediction from the models as to what political bias is present in the article. Therefore, the final product would enable users to be aware of what political influences are occurring in the news they read.

4.1 Data

4.1.1 Political Bias Dataset

To create a dataset, we aimed to take articles posted to political subreddits on Reddit. Reddit offers a rich source of data that can be narrowed down to meet our needs. By considering only articles posted in political subreddits, we are partly ensuring that our dataset will be composed of politically relevant articles thus eliminating a lot of the noise found in the hyper partisan dataset.

First we identified biased political subreddits, namely communities of people that identify with a certain political movement of philosophy. Using the Reddit API, we considered the top posts of the year with a positive upvote ratio. By only considering posts from political subreddits with positive upvote ratios, we ensure that the articles are relevant to the community and politically relevant.

The articles are classified as either left- or right-leaning, based upon the subreddit in which they were posted. While this may introduce some bias from the authors in deciding where to classify each subreddit, in general the subreddits chosen are well defined as either left- or right-leaning. Namely, we do not consider subreddits that are neutral or subreddits that could be considered neutral. This allows for a clear line to be drawn when deciding the classification label of a subreddit.

By deciding the labels based on a classification of the subreddit, we are limiting the potential author bias to just this simple classification decision. The articles are effectively crowd sourced by members of each individual subreddit, with a positive upvote ratio indicating that an article is relevant or agreeable to those who participate in the community. The result is a curated set of articles that are a reflection of the views of that community. Our assumption is that in general these articles will be biased towards the views of the community.

All news articles are then scraped using the News Please web crawler [3]. A difficulty that was encountered was a higher error rate when scraping articles in bulk. Many websites would send a timeout response or even an error code. This impacted the size of our data set and due to time constraints there was no clear solution found. Any further work with this data set would likely start with attempting to reduce the error rate to increase the amount of articles successfully retrieved.

The final result is a data set containing 884 pre-scraped articles, all labelled as left- or right-leaning. While there is a slight class imbalance, with there being slightly more left-leaning articles than right-leaning articles, we believe that this balance is small enough that it can be handled easily or even ignored.

4.1.2 Hyperpartisan Dataset

In addition to creating our own dataset, we experimented with the Hyperpartisan dataset available via Hugging Face [1]. This dataset was created for SemEval 2019 and contains 1.2 million articles from various news sources. Articles in this dataset were labelled according to All Sides' Media Bias ratings [4], which assigns a bias label to new outlets. Each article in the dataset is assigned one of five possible labels: right, center-right, neutral, center-left, or left.

For the scope of our project, we needed only two labels: right and left. As such, we edited the Hyperpartisan dataset to fit these requirements. We categorized all articles labelled as center-right or center-left to simply be right or left, respectively. Additionally, we discarded all articles labelled as neutral (374,228 articles total). When working in Google Colab, the size of the dataset caused sessions to crash due to an insufficient amount of RAM. To circumvent this, we further reduced the size of the dataset, discarding random articles until we were left with 100,000. This was deemed to be a suitable number of articles for the classification task.

4.2 Models

4.2.1 Linear Model

Using the Python library Sci-kit Learn [5], we implemented a linear stochastic-gradient descent model to predict left or right bias in news articles. We used the term frequency-inverse document frequency as the features for the model. The data set was split such that training contained 85% and testing had 15%.

As an experiment to determine the best loss function to use, we used a constant learning rate of 1e-5 and varied the loss function to find the model that performed the best. The result, as seen in 1, was that the squared hinge loss function performed the best with a test accuracy of 0.970

To further improve accuracy, we experimented with different learning rates, using the hinged loss function. The result of this experiment, as seen in 2, was that a learning rate of 1-e7 performed the best with a test accuracy of 0.977.

4.2.2 BERT

Bidirectional Encoder Representation from Transformers (BERT) is a pre-trained model by Google that achieves state-of-the-art performance on many natural language processing tasks [6]. The model was trained on Wikipedia and BookCorpus. Using the Python Library Hugging Face [7], we are able to download BERT and build our own architecture around the existing structure. We can then fine tune the model by training it on our custom data set.

Using BERT as the base model, we added 4 fully connected layers onto the end. The number of nodes in each layer is 768, 64, 64, and 2 respectively. Every layer uses

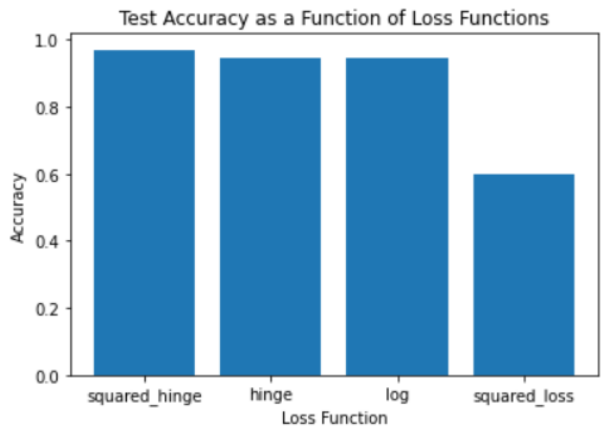


Fig. 1. Bar graph depicting the test accuracy of a linear model using different loss functions. The best test accuracy achieved was 0.970 with the squared hinge loss function.

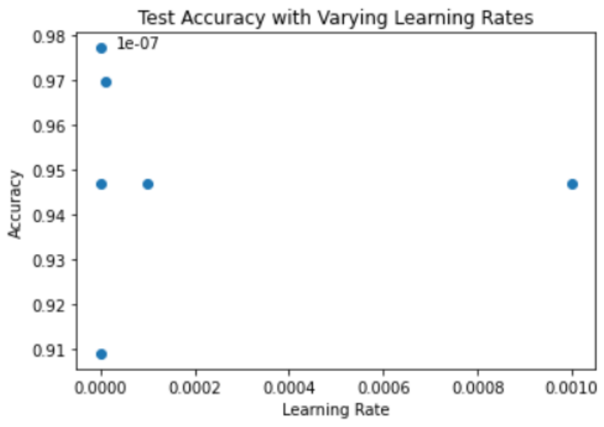


Fig. 2. Scatter plot depicting the test accuracy of a linear model varying the learning rate. The highest value was a learning rate of 1e-7 with a test accuracy of 0.977.

a ReLU activation function and the output layer applies a softmax function. For the input, we split the data set such that 70% was used for training and 15% for each of validation and testing. To account for class imbalance, we calculated the class weights and passed them as a parameter to our loss function.

BERT also provides a tokenizer that can be used on input text to transform it to a proper format that the model will understand. A downfall of BERT is that the input is restricted to a max character length of 512. The average length of each input in our data set is approximately 700. Therefore, we chose to cut off the end of articles that were too long, and pad inputs that were too short.

While training the model with our custom dataset, we experimented with different values for the hyperparameters learning rate and batch size. We found that the optimal settings for these parameters were 0.00001 and 8 respectively. BERT also allows you to unfreeze or freeze the layers in the original model. Freezing a layer means its weights will not be updated while training. We noticed a drastic increase in performance when unfreezing every layer. The difference in training the two models can be seen in Fig. 3 and Fig. 4. The test accuracy of the model with all layers frozen was 0.61, while the model with no freezing scored a 0.87 accuracy on the test set.

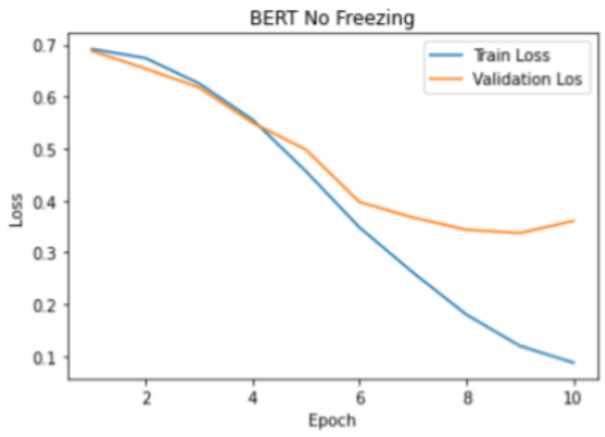


Fig. 3. Training and validation loss of the BERT model with all layers unfrozen over 10 epochs.

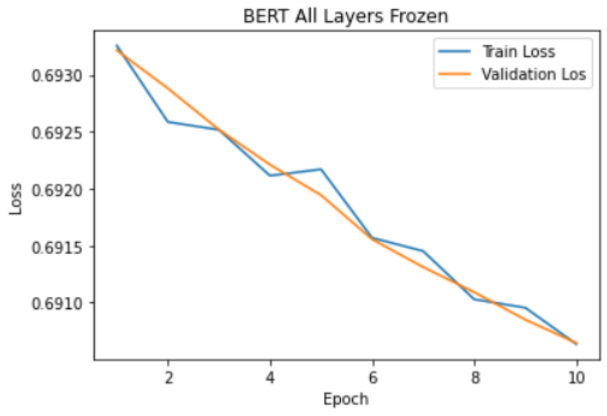


Fig. 4. Training and validation loss of the BERT model with all layers frozen over 10 epochs.

We proceeded by training the same BERT model with the Hyperpartisan dataset. We expected that this would improve model performance due simply to the size of the Hyperpartisan dataset. Surprisingly, doing this with the optimal model architecture from our experiments with the custom dataset yielded a far worse performance, with a maximum test accuracy of 0.51. Experimenting with different hyperparameters did not improve the test accuracy.

We expect that this performance is due in part to the number of epochs for which we trained the model. The size of the Hyperpartisan dataset presented a need for a lot of computing power. Each training epoch took a significant amount of time to complete, so it was simply unfeasible to train the model for more than 10 epochs.

An additional problem with the Hyperpartisan dataset is the nature of the articles contained within. While there is a substantially larger number of articles than in our custom dataset, not all of the articles are political in nature. Due to this, we feel that it may not be appropriate to train models using these articles for the purpose of classifying political bias. Because of the underwhelming performance of the BERT model when trained on the Hyperpartisan dataset, we deemed the dataset to be nonviable for the purposes of our classification task.

4.2.3 *RoBERTa*

Robustly optimized BERT approach, or RoBERTa, is a model developed by Facebook AI and aims to improve upon the BERT model [8]. RoBERTa is an attempt to improve the performance of BERT by increasing the training data and rethinking design decisions made when developing BERT. The model is trained on 160GB of data, including the original data utilized by BERT, CC-News, OpenWebText, and Stories datasets [8]. In particular, the additional training on the CC-News and OpenWebText datasets seem particularly relevant to this project. The CC-News dataset is composed of 63 million news articles and the OpenWebText dataset is made of scraped articles posted on Reddit. This is interesting because it appears to have a clear overlap with our problem of classifying political news articles, where our methodology for retrieving data is similar. It may be the case that this additional exposure to news articles will improve the performance of our classifier built using RoBERTa. With this potential overlap in training data, along with the improved performance yielded from design changes, RoBERTa appears to be a strong candidate solution.

Our implementation of RoBERTa makes use of the Hugging Face transformers in order to download the RoBERTa model and train it using our own data. Using the same architecture as described with our BERT implementation, but simply swapping out the BERT base with RoBERTa, we saw an instant improvement. By swapping RoBERTa for BERT, we saw an immediate increase in test accuracy of about 0.05. With further experimentation a test accuracy of 0.95 was achieved.

The best results with the RoBERTa implementation occurred with a learning rate of 0.00001, 15 epochs, and fully unfrozen layers, shown in Fig. 6. It was found that there was an improvement of about 0.03 when training with fully unfrozen layers rather than partially frozen layers. The original intent behind freezing layers was to save time on computation, but upon experimenting with fully unfrozen parameters it was found that the computational time saved was not as large as expected. Thus, given the improved performance when unfreezing layers, it was decided to continue with fully unfrozen layers.

When considering learning rate, it was found that anything larger than 0.00001 was too chaotic to be useful. As can be seen in Fig. 5, the training and validation loss are incredibly chaotic, constantly bouncing up and down. This effect completely goes away when choosing a smaller learning rate, something that yields a more gradual decrease in loss and better results.

An interesting observation is how few epochs are required to reach a near optimal state when using the RoBERTa model. When comparing two learning rates, 0.00001 and 0.000001, similar results were achieved, but in greatly differing numbers of epochs. Using the two respective learning rates, the RoBERTa implementation was able to achieve 0.95 test set accuracy. As can be seen in Fig. 6, the model achieves its best result after about 7 epochs, with a lowest validation loss of just under 0.4. Similarly, the model shown in Fig. 7 also reaches it's best point at just under 0.4.

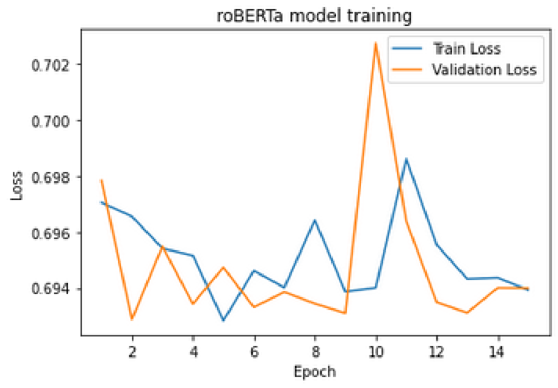


Fig. 5. Graph showing training and validation loss of the RoBERTa model over 15 epochs with a learning rate of 0.0001. The training and validation loss are extremely chaotic.

The key difference here is that the model in Fig. 7 achieves its best result after 60 epochs. This is a key result in showing that while it may be possible to extract extra performance with a lower learning rate and more epochs, the end result is similar enough to that which can be achieved with a higher learning rate and much fewer epochs.

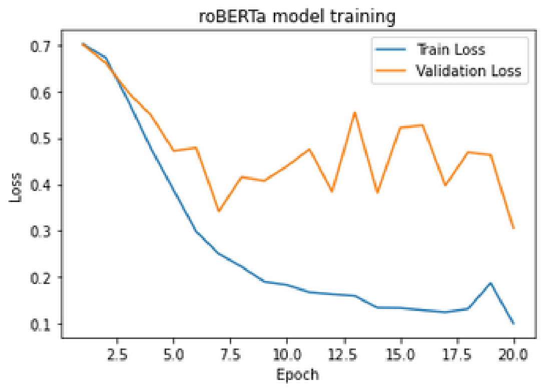


Fig. 6. Graph showing training and validation loss for a RoBERTa model with a learning rate of 0.00001.

Overall, our implementation of a RoBERTa based model out performed an identical BERT based model by 0.08. This result is expected due to the additional training performed to optimize RoBERTa over the base BERT implementation.

4.2.4 *DistilBERT*

The DistilBERT model shares a few commonalities with the BERT model. The two models are both transformer models that can perform very well for a variety of natural language processing tasks. However, DistilBERT differs from BERT in both its size and speed. DistilBERT gets its name through the distillation of language knowledge during its pre-training phase. This distillation allowed the model to become 40% smaller than the BERT model and also 60% faster. These advantages came at the cost of losing 3% of the language understanding capabilities when compared to BERT. However, the DistilBERT model still retains 97% of its language

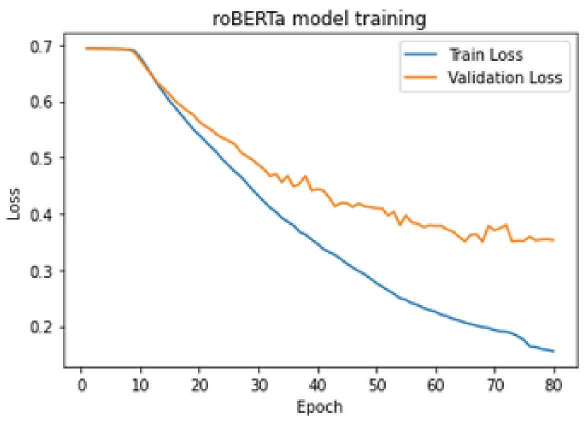


Fig. 7. Graph showing training and validation loss for a RoBERTa model with a learning rate of 0.000001.

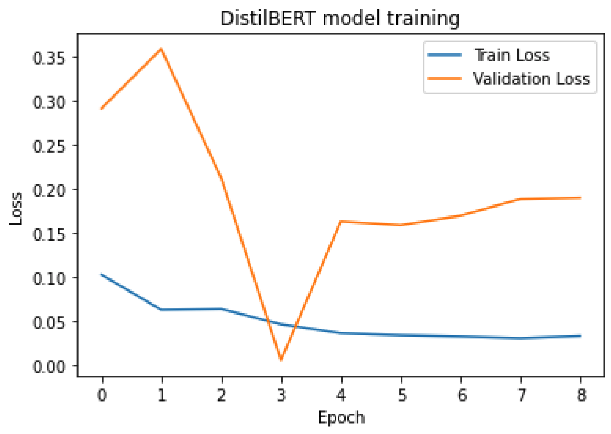


Fig. 8. Graph comparing the training and validation loss of the model over 9 epochs. The best validation accuracy occurs at 3 epochs. After this point, the training loss converges and the validation loss increases, showcasing how the model begins to over-fit the training data.

comprehension, which makes it a suitable model option for this implementation project [9].

The implementation of the DistilBERT model in this project was similar to the approach taken for both the BERT and RoBERTa models. The approach was similar in that we retained the original architecture of each model, but appended a few layers at the end. This approach would enable us to benefit from the model's general knowledge of language comprehension, but would also enable us to employ transfer learning. This transfer learning enabled the general model to understand our labeled dataset and therefore be fine-tuned for the current problem.

In the implementation of DistilBERT, we appended two layers to the end of the original architecture. These layers consisted of one Dropout Layer, with the probability of dropout being 0.3. The second layer was a fully connected dense layer, which employed a ReLU activation function. When implementing these layers, we decided to retain DistilBERT's original parameters. However, we unfroze the parameters in the final two layers we appended, which enabled the model to be fine-tuned to suit our dataset.

After designing the DistilBERT architecture for this project, we trained the model. Upon training this model, we utilized several different hyperparameter settings. These settings were altered based on the model's training and validation loss per epoch. After several iterations, we found that the optimal hyperparameter settings included a learning rate of 0.00005, a batch size of 8, and a dropout probability of 0.3. While training the model, we also noticed that the training loss converged after 3 epochs, as seen in Figure 6. If more epochs were used, the model began to over-fit the training data, which increased the validation loss. We also noticed that increasing the size of the training set created better accuracy, as the model had more examples to learn from. Therefore, the test set included only 8% of the total dataset. These decisions results in a test accuracy of 92% on unseen data.

4.3 Web Application

4.3.1 Implementation

The purpose of this project was to give people a tool that would allow them to realize the political bias in the information they consume. The models we implemented in the previous sections have the capability of identifying the political bias in news articles, however, these models are not accessible to the majority of users. Therefore, we created a simplistic website application, which would enable users to easily determine the political bias of the news articles they are reading.

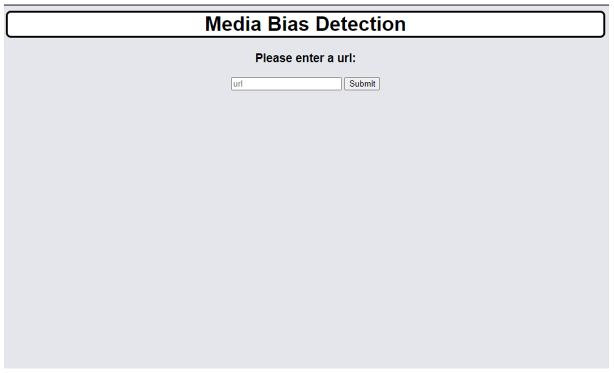
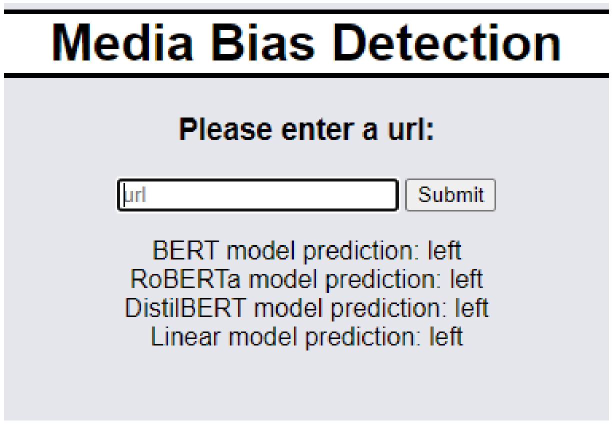


Fig. 9. Landing page for political bias web app.

Our Flask web app, Fig. 9, is setup such that a user enters the URL of the news article they are reading. The URL is then scraped with the news-please Python API, which scrapes the raw text from the website. This text is then ingested by the four models mentioned above (Linear, BERT, RoBERTa, and DistilBERT), each model makes a prediction as to the type of political bias present in the news article, and the predictions are shown to the user. These predictions enable the user to think more critically of the information they are about to consume. One interesting aspect of this natural language processing implementation is how the trained models are queried in the web app. When developing this project, the models were initially created in separate Google Colab notebooks. After architecting and training these models, we tested the models on unseen data. We then saved the weights of each model with the highest test accuracy and imported them into our web application. This allowed us to create the models on the backend with the same architecture and load the saved weights so we could query them for predictions. The raw text scraped from the user's URL was then treated similarly to test data in Google Colab. Each model simply ingested the data and made a prediction. This prediction was then presented to the user.

4.3.2 *Interesting Results*

In many instances, the various models seem to reach a consensus. For example, when predicting an article favourable to immigration, found in the progressive subreddit, all models successfully predicted the article as left-leaning as shown in Fig. 10. Across a diverse set of articles, both left- and right-leaning, there are many instances where there is a consensus classifications. This is a great result, as the goal of the web app is to provide a definitive clue into the political bias present in the article.


Media Bias Detection

Please enter a url:

url Submit

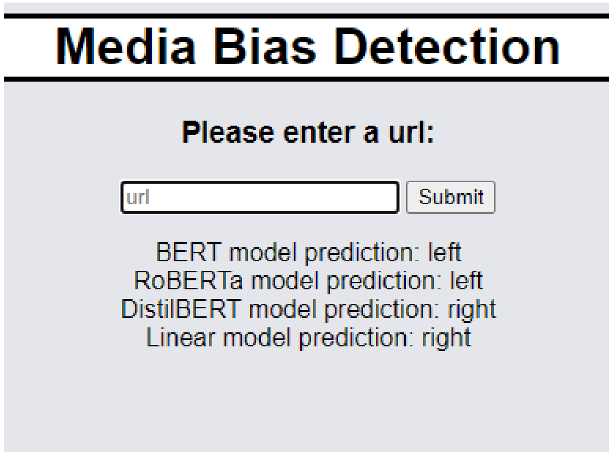
BERT model prediction: left
RoBERTa model prediction: left
DistilBERT model prediction: left
Linear model prediction: left

Fig. 10. An example of a correct consensus result for a left leaning article.

While a correct consensus result is ideal, there are several instances where the different models disagree on the classification of an article. This can be due to many factors such as poor representation of a topic in our training data, mislabeled articles, or unclear bias in the article. As a result of our strict binary classification, where there is no neutral label, there is the possibility that an article can not strongly fit either label and thus produces unclear results.

One instance of an unclear result occurs for an article criticizing republicans. The predictions, shown in Fig. 11, are split with two models predicting each respective label. The correct label in this instance is right-leaning. This article was found in the conservative subreddit, therefore we would consider it right-leaning. Additionally, upon reading the article it is definitely right-leaning in terms of bias. The content of the article is critical of Republicans not standing up to the agenda of the Democrats. Based upon the content of the article it potentially makes sense for two models to classifying it as left leaning. Based on the author being critical of republicans it potentially explains why this article could be classified as right-leaning. Typically it would be expected for articles critical of republicans to be left-leaning and this would likely often be the case in our dataset if there were similar instances of articles critical of republicans.

This is just one example of the nuance involved in politics. The views of both left- and right-leaning individuals


Media Bias Detection

Please enter a url:

url Submit

BERT model prediction: left
RoBERTa model prediction: left
DistilBERT model prediction: right
Linear model prediction: right

Fig. 11. An example of an unclear result for a right leaning article.

may change drastically depending on which party is in power or the relevant current events. The speed at which politics changes can also play a part. The political landscape has drastically changed within the last four years. While this is an extreme example, politics is constantly evolving and changing. The line between right and left is vastly different today than it was fifty or one hundred years ago. The fact that politics is constantly evolving adds to the difficulty in detecting political bias, it can be hard to pick a side when the line is constantly moving.

5 Conclusion

We successfully delivered a web application capable of predicting bias in news articles. The web application provides a simple interface, allowing users to input URLs to news articles and receive accurate predictions on the bias of said article. The web application employs one linear model and three BERT-based models to generate the predictions. All models performed well in classifying bias in news articles when trained on our custom dataset. Overall, the linear model resulted in the highest test accuracy with a score of 0.977. Training the BERT model with the Hyperpartisan dataset did not go as well as expected, achieving a test accuracy of only 0.51.

We also provide a political bias dataset with 884 labeled data points. This dataset was automatically created by scraping news articles from well-defined left- or right-leaning subreddits, and assigning a label to the article based on subreddit it came from. Based on the poor performance of our models trained with the Hyperpartisan data, we conclude that our dataset is better suited for the political bias classification problem. Further, we believe automatically labeling the bias of articles based on publisher (as the Hyperpartisan dataset does) introduces too much noise. Instead, labelling based on the community the article was posted in is a better method.

We expect that the linear model performed best because of its ability to accept an entire article as input. The other models used for this project were able to accept only the first 512 characters of an article as input. This limitation likely led to decreased performance from the pre-trained models, as they may have missed useful text in the article.

For future work in this area, we recommend exploring models that have less restrictions on the length of input, as this will ensure the model learn from all the text in the article. Additionally, expanding the size of the dataset would further improve classification accuracy.

The web app could also benefit from the creation of additional functionality. For example, we could allow users to supplement our dataset by classifying the political bias in the news articles they read. Perhaps, a user could enter a URL to a news article and classify the political bias themselves. This data could then be added to our current dataset and the models could periodically be retrained using the crowd sourced dataset. Perhaps in the case where models disagree on the political bias of an article, a user could read the article and mark which model was correct in its classification. This information could be used to create a larger and more diverse dataset. It would also make the application more effective over time.

The user experience of our web app is not a true depiction of how users consume news articles. Therefore, it is unlikely that our web app would be used frequently, as users need to disrupt their work flow to access the web app (i.e. users have to exit their news platform of choice, navigate to the URL of our web app, paste the URL of the news article into the form, and wait for the prediction). Instead, the user experience would benefit from being packaged into a browser extension. Thus allowing users to be notified automatically of the political bias present in the media they are actively consuming. This integrated capability will increase the accessibility of tools which enable users to think critically about the underlying bias present in the media they consume.

REFERENCES

[1] J. Kiesel, M. Mestre, R. Shukla, E. Vincent, P. Adineh, D. Corney, B. Stein, and M. Potthast, "SemEval-2019 task 4: Hyperpartisan news detection," in *Proceedings of the 13th International Workshop on Semantic Evaluation*. Minneapolis, Minnesota, USA: Association for Computational Linguistics, 06 2019, pp. 829–839. [Online]. Available: https://www.aclweb.org/anthology/S19-2145

[2] A. F. Adoma, N. M. Henry, and W. Chen, "Comparative analyses of bert, roberta, distilbert, and xlnet for text-based emotion recognition," in *2020 17th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, 2020, pp. 117–121.

[3] F. Hamborg, N. Meuschke, C. Breitinger, and B. Gipp, "news-please: A generic news crawler and extractor," in *Proceedings of the 15th International Symposium of Information Science*, 03 2017, pp. 218–223.

[4] AllSides, "Media bias ratings." [Online]. Available: https://www.allsides.com/media-bias/media-bias-ratings

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

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

[7] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

[8] 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

[9] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," 2020.