Political News Bias Detection using Machine Learning

Minh Vu
Department of Computer Science
Earlham College
801 National Road West
Richmond, Indiana 47374
mdvu15@earlham.edu

ABSTRACT

Over the past decade, political campaigns have been increasingly waged on not only news sites but also social media networks. However, there have been ongoing concerns regarding how one's political opinions affect the information they were fed online towards the 2016 presidential election result. Researches in sentiment analysis and political bias detection have been using Recurrent Neural Network (Long Short-Term Memory) to achieve results with high accuracy. Recently, Multilayer Perceptron (MLP) and Convolutional Neural Networks (CNN) have appeared as deep learning models with promising results in the field of natural language processing, deviating away from the traditional solutions of Recurrent Neural Networks. This study aims to take a different technical approach to the problem of political bias detection using Multilayer Perceptron model. The word embedding matrix for the MLP model is initialized with Facebook's fastText word vector representation model. The resulting classifier is implemented in a browser tracking system, specifically a Google Chrome extension, which can perform an extensive scan whenever users navigate to a news article. The extension then determines the ideological components of the article, in percentages of conservative, liberal or neutral sentences. The training dataset is the Ideological Books Corpus (IBC), which consists of 4,062 sentences annotated for political ideology at a sub-sentential level. Results show that: the MLP model achieved an F1 score of 81% on the test dataset, higher than that obtained by an RNN model (72%). Real-time political news classification using the MLP classifier did not yield promising results, however. This can be attributed to the use of metaphors and negative phrases in articles, along with the model's inability to consider the relation between adjacent words. Further research is necessary in order to achieve a complete, finalized political bias detection system.

KEYWORDS

Multilayer Perceptron (MLP), Convolution Neural Networks (CNN), Natural Language Processing (NLP), political bias, deep learning, machine learning, neural network

1 INTRODUCTION

The current political climate in the United States is sharply divided into two major ideologies: liberal and conservative, represented by the Democratic and Republican parties, respectively. These two political parties advance different policies reflecting opposite opinions on various social issues. In the era of massive social media usage, such ideologies are disseminated at an unprecedented rate. There have been a plethora of active and engaging political discussions on media feeds, backed by various sources of public debates, interviews, and speeches. Specifically, during the 2016 US presidential election, political campaigns were increasingly waged on news sites and social media networks, hence the name "the social media election" [7]. This phenomenon gave rise to a nontrivial problem and an ongoing concern about how one's political position affects the type of contents on their social media feeds. People with certain political opinions get fed news and information supporting the same political viewpoints, and thus creating an ideological bubble that hinders them from a subjective, neutral information environment. Over time, this process reinforces personal biases, given the fact that political parties' agendas are biased over different issues.

At the same time, Natural Language Processing (NLP) and opinion mining are becoming more and more popular in solving tasks of analyzing private states such as opinions, sentiment, and beliefs from texts. The sudden eruption of interest in the area of opinion mining and sentiment analysis has also encouraged studies in building applications that deal directly with opinion classification [22]. Detecting political bias, which was conceived to be nontrivial, has been the main topic of interest for many scholars in the NLP community over recent years. A variety of machine learning and deep learning techniques were applied to solve the task, including: Support Vector Machine (SVM) [2] [28], Recursive Neural Networks (RvNN) [13], Recurrent Neural Networks (RNN) [21], etc.

In recent years, Multilayer Perceptron (MLP) along with Convolutional Neural Networks (CNN) have gained popularity within the field of NLP for their promising results. MLP models are effective for various NLP problems and achieved excellent results in semantic parsing[11], sentence modeling[15], classification[18] [16], prediction [8] and other traditional NLP tasks[9]. Given these promising results, this study uses MLP to solve the current problem of political bias detection, contributing to the existing body of works that use neural networks to detect political ideologies. It also implements an application (a web browser extension) which can utilize such framework to perform real-time political ideologies classification.

The remaining of this paper is structured as follows: Section 2 discusses relevant literature in the subject of political ideologies classification, with three dominant models being Recursive Neural Network (RvNN), Recurrent Neural Network (RNN) and Hidden Markov Model (HMM). Section 3 outlines the designs of different components and modules in this study. Software implementations and experiment results are discussed in section 4. Finally, section 5 presents the conclusion of this study along with future directions.

2 RELATED WORKS

Much research in the NLP community has been conducted in recent years to build efficient machine learning models that can accurately

classify political opinions. For instance, Iyyer et al. [13] developed a Recursive Neural Network (RvNN) model to create an Ideological Book Corpus (IBC) [14], which labels sentences and phrases based on their political ideologies. Their neural network was proven to outperform contemporary methods (bag-of-words models and hand-designed lexica) at the task given. Other studies were able to produce state-of-the-art models with a robust performance [21], label and determine inter-relationships between political debates [22], and determine ideological proportions of speeches [26]. In this section, we examine some of the current approaches in political bias detection with the use of machine learning techniques. Table 1 is a summary of the technology and their abbreviations that are used in this section.

Table 1: Abbreviations and terms

Abbreviation	Full term
NLP	Natural Language Processing
RvNN	Recursive Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
HMM	Hidden Markov Model
MLP	Multilayer Perceptron
CNN	Convolutional Neural Network
IBC	Ideological Book Corpus

2.1 Recursive Neural Network (RvNN)

Iyyer et al. developed a recursive neural network (RvNN) [13] to identify the political position evinced by a sentence, a problem where previous work relies heavily on bag-of-words models and hand-designed lexica. Their approach was to break sentences into smaller semantic compositions, which RvNNs can model. The principle is that sentences are composed of phrases, and a phrase's meaning is a combination of the meaning of the words within that phrase and the syntax that connects those words. As a result, most ideological bias can only be detected at higher levels of a sentence tree (as shown in Figure 1).

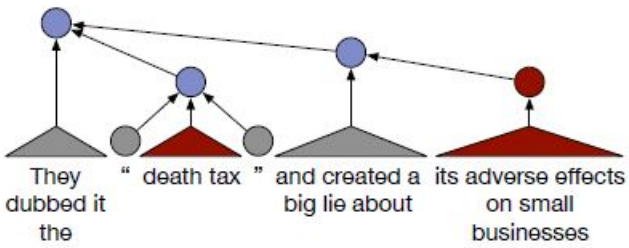

Figure 1: An example of compositionality in ideological bias detection (red → conservative, blue → liberal, gray → neutral) in which modifier phrases and punctuation cause polarity switches at higher levels of the parse tree. [13]

The basic mechanism behind their RvNN model is the use of word vectors. Each word w is represented by a vector $x_w \in \mathbb{R}^d$. Based on a parse tree, these words form phrases p. Each of these phrases also has an associated vector $x_p \in \mathbb{R}^d$ of the same dimension as the word vectors. These phrase vectors represent the meaning of the phrases composed of individual words. As phrases themselves merge into complete sentences, the underlying vector representation is trained to retain the sentence's meaning. They then divide the vector space into two smaller vector spaces, x_d and x_r, which represent liberal and conservative sentences, respectively. Given the dataset in the study is labeled, the authors employed supervised learning to classify sentence vectors by applying a regression. The discrepancy between categorical predictions and annotations is measured through the cross-entropy loss. They then optimized the model parameters to minimize the cross-entropy loss over all sentences in the corpus. Through experiments, they concluded that their RvNN model outperforms the bag-of-words baselines as well as the word2vec baseline on both datasets (Congressional Debate Transcripts and their own IBC).

2.2 Recurrent Neural Network (RNN)

Misra et al. used a different approach to solve the task of labeling political ideologies[21]. Instead of analyzing sentences recursively from the word level, they employed Recurrent Neural Network, a generative model which can predict the labels at the end of a sequence of words.

Labeling ideologies requires capturing long-range correlation between words in the text because the data loss does not get contributions from every word in the network and is calculated only at the very last time step and the gradient of the loss has to backpropagate to the beginning of the sentence. However, traditional RNNs were not able to propagate the gradient over many time steps because of the problems of vanishing or exploding gradients [4]. Long Short-Term Memory (LSTM) model, on the other hand, was capable of solving the problem and was employed for the study.

The basic unidirectional model did not perform as well on the IBC dataset [14] compared to the RvNN model [13]. However, Misra et al. argued that it was not a shortcoming of the model, but rather a combination of complexity and lack of volume of the data which led to the results. They tested their hypothesis by making an OTI dataset, upon which the single layer, unidirectional LSTM model was able to achieve better results with an F1 score of 0.718, at par with the best results on bias and opinion detection available in contemporary literature [12]. The LSTM model developed works only marginally better than a bag-of-words approach on the OTI dataset, because, deep learning based models need a substantially larger amount of data to outperform the traditional methods of the domain. However, the RNN model has the potential to be further developed and become robust enough so that it can be applied to a variety of different contexts without the need of any detailed manual labeling, which is always difficult to obtain, as in the RvNN model[13].

2.3 Hidden Markov Model (HMM)

Sim et al. approached the task of labeling ideological biases rather differently than the previous two approaches[26]. They used a Hidden Markov Model (HMM) with a low dimensional representation of political speeches: a speech is a sequence of cues interspersed with lags, where:

- Cues are terms that are strongly associated with an ideology.
- Lags correspond to the lengths of sequences of non-cue words, which are treated as irrelevant to the inference problem at hand.

The HMM employed was called **cue-lag ideological proportions** (CLIP). Each state in the model corresponds to an ideology or BACKGROUND. Emission from a state consists of (i) a cue from the ideological lexicon (L) and (ii) a lag value. In order to capture the intuition that a longer lag after a cue term should increase the entropy of the model over the next ideology state, the authors introduced a restart probability in the transition distribution in the HMM, which is conditioned on the length of the most recent lag.

Testing was based on a set of 14 pre-registered hypotheses proposed by the authors. CLIP correctly identified sixteen LEFT/RIGHT alignments of primary presidential candidates based on their speech transcripts, and only failed to determine one candidate's political orientation. There is one systematic issue with CLIP, however. That is, it associates candidates' names with political positions. For example: terms mentioning John McCain are associated with the RIGHT, which makes Obama's mentions of his opponent taken as evidence for rightward positioning; in total, mentions of McCain contributed 4% absolute to Obama's RIGHT ideological proportion. Similarly, barack_obama and president_obama are LEFT cues (though senator_obama is a RIGHT cue). The authors suggested that filtering candidates' names in the first stage will be beneficial.

3 DESIGN AND IMPLEMENTATION

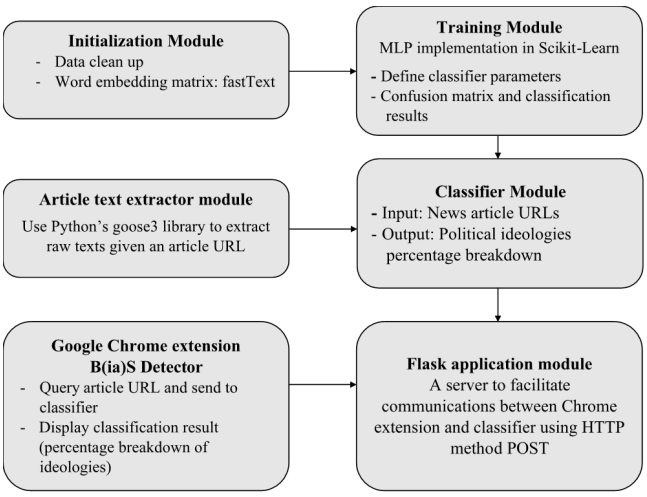

Figure 2: Framework of the Political News Bias Detection System

This study focuses on developing a solution to the problem of political bias detection, using Multilayer Perceptron. As shown in Figure 2, this project consists of six components:

- Initialization module
- Training module
- Classifier module
- Article text extractor module
- Flask application module
- Google Chrome extension - B(ia)S Detector

The initialization module initializes the word embedding matrix and passes it to the training module, which utilizes this matrix to vectorize words and sentences in the training dataset. The resulting trained classifier is invoked in the classifier module, which then takes articles' texts (extracted through the article extractor module) as its input and returns a percentage breakdown of the political ideologies. This output is then communicated with Flask application module and sent to the Chrome extension for display.

3.1 Initialization module

The word embedding matrix used for the MLP model was initialized using fastText, an unsupervised learning algorithm for obtaining vector representations for words developed by Facebook's AI Research (FAIR) lab [5]. The main difference between fastText and other word embedding models (word2vec [19], GloVe [25]) is that fastText treats each word as composed of character n grams, so the vector for a word is also made of the sum of this character n grams. Word2vec (and GloVe) treat words as the smallest unit to train on [19] [25]. This means that fastText can generate better word embeddings for rare words. Also fastText can generate word embeddings for out of vocabulary word while word2vec and GloVe can not do this.

This study employed the pre-trained word vector model with 1 million words on Wikipedia 2017[1]. These vectors in dimension 300 were obtained using the skip-gram model described in the work of Bojanowski et al. (2016) with default parameters[5]. Due to the large size of this pre-trained model (4.3GB), Pickle, a built-in Python module for serializing and de-serializing Python object structures[2] was implemented. The output word embedding matrix was 'pickled' to a .pkl file of 1.3GB in size.

Labeled sentences and phrases from the IBC were passed to a Python function that outputs a sentence matrix representing these sentences (or phrases). This sentence matrix was constructed as the average of all word matrices instead of the sum. Some sentences are longer than others and taking the average can normalize this difference in while keeping the sentences' main ideologies.

3.2 Training module and classifier module

A multilayer perceptron (MLP) is a feed-forward neural network. It has three or more layers and utilizes a nonlinear activation function (mainly hyperbolic tangent or logistic function) which could classify data that is not linearly separable. Every node in a layer connects (with a certain weight) to every node in the following layer making the network fully connected. Learning occurs in the perceptron by changing connection weights after each piece of data

[1] https://fasttext.cc/docs/en/english-vectors.html
[2] https://docs.python.org/3/library/pickle.html

is processed, based on the amount of error in the output compared to the expected result. Learning in MLP is supervised learning, and is carried out through back propagation, a generalization of the least mean squares algorithm in the linear perceptron. MLP applications in Natural Language Processing include speech recognition and machine translation [10].

In this project, the Multilayer Perceptron classifier takes matrix representations of words and sentences and outputs the predicted political ideology. The classifier was implemented with *scikit-learn* library (version 0.19.1), an open source machine learning library for Python (version 3.6.5) [24]. *Scikit-learn* is a Python module integrating a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems. This library uses a general-purpose high-level language for machine learning [24]. *Scikit-learn* implementation of an MLP model allows the following parameters[6]: *hidden_layer_size*, *activation*, *solver*, *alpha*, *batch_size*, *learning_rate*, *learning_rate_init*, *power_t*, *max_iter* , *shuffle* , *random_state*, *tol*, *verbose*, *warm_start* , *momentum*, *nesterovs_momentum* , *early_stopping* , *validation_fraction* , *beta_1* , *beta_2* , *epsilon*, *n_iter_no_change*.

3.2.1 Training module. For the purpose of training, 75% of the IBC (chosen randomly using *scikit-learn*'s *train_test_split()* function[6]) was employed. No dataset specific tuning was performed. Training was done through stochastic gradient descent over mini-batches[17]. The training module (*classifierTrain.py*) gets the preprocessed sentence embedding matrix from the initialization module, and splits training and testing data. The module also outputs testing results (precision, recall, F1 score)[3] for analysis and packages the resulting classifier (when the classifier meets the experiment requirements) into a .pkl file for further use.

3.2.2 Classifier module. The classifier module (*classifier.py*) imports the classifier file outputted by the training module along with the word embedding matrix. The module contains a function that takes a news article URL, extract its sentences through the article extractor module, vectorizes these sentences and pass them through the classifier. The classifier can then identify whether the sentence ideology is neutral, conservative or liberal, and outputs a percentage breakdown of the political ideologies.

3.3 Article extractor module

Python library *Goose3* was employed for the task of scraping plain texts of online news articles[4]. *Goose3* is an open source library that takes any news article or article-type URL and extracts not only what is the main body of the article but also all meta data and most probable image candidate.

For this study, the article extractor module was implemented to extract only the main text of an article. The module uses the basic syntaxes of *Goose3* and consists of one function: it takes a news article URL and outputs an array of sentences (in the article).

3.4 Flask application module

To facilitate communication between the classifier and the Google Chrome extension, a Flask application[5] was employed. Flask is a micro web framework in Python that allows developers to build a web server with basic HTTP methods.

Communications between the Chrome extension and the MLP classifier take place through a local Flask server using the HTTP POST method. The server is launched at the beginning of each session by calling the Flask application module (*app.py*) from the command line. The server has a specific URL path (https://localhost:5000/classify) that handles POST requests (containing article URLs) from the Chrome extension and responds with the classification results from the MLP classifier.

3.5 Google Chrome extension

A Google Chrome extension was developed and it is able to communicate with the classifier using the aforementioned Flask application module. The extension queries the news article URL and passes it to the Flask server, which in turn passes the address to the classifier module. The classifier acquires sentences from the article text (through calling the article extractor module), performs word vectorization and outputs political ideologies on a percentage basis. The extension then displays this percentage results of political biases, indicating how many percentages of the article are liberal, conservative, or neutral.

The Chrome extension uses the *tabs* API of Chrome API[6] to query the current active URL and triggers the extension whenever an user navigates to a news site. It contains a *connect()* function which sends the article URL as a JSON object to the Flask server, using HTTP POST request. If the POST request is received and responded successfully, the extension calls helper functions to displays the results in its HTML pop-up window, as shown in Figure 3:

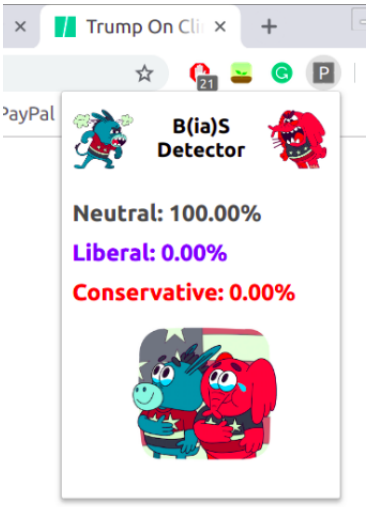

Figure 3: User interface of Chrome extension

[3]Please refer to the experiment subsection for definitions
[4]https://github.com/goose3/goose3
[5]http://flask.pocoo.org/
[6]https://developer.chrome.com/extensions/devguide

4 EXPERIMENTS AND RESULTS

4.1 Testbed and experiment design

The main dataset for this study is The Ideological Books Corpus (IBC) [14], which consists of 4,062 sentences annotated for political ideology at a sub-sentential level. Because the IBC contains labeled data, it is divided into two sets, one used for training and one used for testing, with ratio 3:1. That is, 25% of the IBC is the testing datasets for the classifier.

Training result was evaluated based on three measurements: precision, recall and F1 score (with F1 score being the main measurement):

- Precision is the ratio of correctly predicted positive observations to the total predicted positive observations[21]. If the classifier predicts 100 sentences are liberal (or conservative or neutral) and 80 of them are actually liberal based on the labeled data, then the precision is 80%.
- Recall is the ratio of correctly predicted positive observations to the all observations in the actual class [21]. If there are 100 liberal sentences in the testing dataset and the classifier successfully predict 70 of them, then the recall rate is 70%.
- F1 Score is the weighted average of Precision and Recall (as shown in Equation 1 [21]). This score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if there is an uneven class distribution, which is the case of the IBC dataset.

$$F1 = \frac{2 \times precision \times recall}{precision + recall} \qquad (1)$$

Experiments were carried out to find the optimal set of parameters for the MLP classifier. The relevant parameters to be adjusted are:

- *hidden_layer_size*: A tuple in which the i^{th} element represents the number of neurons in the i^{th} hidden layer.
- *max_iter*: Maximum number of iterations.
- *batch_size*: Size of mini-batches for stochastic optimizers.
- *warm_start*: When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution.
- *early_stopping*: When set to True, automatically set aside 10% of training data as validation and terminate when validation score is not improving.

Due to the nature of the classification problem, the rectifier activation function and Adam optimization algorithm (which works better with large datasets) were kept constant[3].

4.2 Experiments and results

4.2.1 Classifier experiments. In the first set of experiment, the classifier was trained using the default MLP parameters in *scikit-learn* with four hidden layers, each contains 10 neurons, 200 maximum iterations, no mini-batch was employed and neither was warm start nor early stopping. The measurements of this attempt are shown in Table 2:

Table 2: Experiment results 1

	Precision	Recall	F1 score	# of sentences
Conservative	59%	53%	56%	1572
Liberal	64%	66%	65%	1925
Neutral	77%	81%	79%	2159
Average	**68%**	**68%**	**68%**	**5656**

The output of the first experiment also showed that the maximum number of iterations (200) was reached but the optimization had not yet converged. To solve this problem, parameter *max_iter* was gradually increased. At *max_iter* = 1,000, optimization converged. However, the F1 score results did not improve as compared to the first experiment, and thus are not shown due to their triviality.

In the following experiment, the number of neurons in each layer was doubled to 20 each. Maximum number of iterations was set to 1,000, and *batch_size*, *warm_start*, *early_stopping* were kept the same as the first experiment. The results are shown in Table 3:

Table 3: Experiment results 2

	Precision	Recall	F1 score	# of sentences
Conservative	67%	54%	60%	1539
Liberal	67%	73%	70%	1948
Neutral	81%	84%	82%	2169
Average	**72%**	**72%**	**72%**	**5656**

The new set of results improved slightly compared to that obtained in the first experiment. However, several trials revealed that performance measurements improved only with increases in the size of the first hidden layer but not with subsequent hidden layers, as shown in Figure 4:

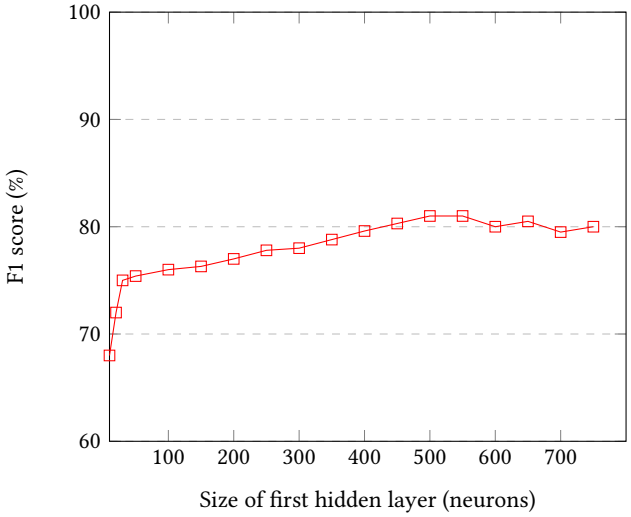

Figure 4: F1 scores relative to size of first hidden layer

As a result, the optimal number of neurons in the first hidden layer was set to be 500. Other parameters (*batch_size*, *warm_start*, *early_stopping*) were also adjusted to speed up the training process.

According to Bengio, mini-batch gradient descent is the recommended variant of gradient descent for most applications, especially in deep learning [3]. A few advantages of using mini-batches are: The model update frequency is higher than batch gradient descent which allows for a more robust convergence, avoiding local minima; the batched updates provide a computationally more efficient process than stochastic gradient descent; batching allows both the efficiency of not having all training data in memory and algorithm implementations. The recommended mini-batch size is 32, which shows to take advantage of the speedup of matrix-matrix products over matrix-vector products.

Besides, parameter *warm_start* was set to true, which reuses aspects of the model learned from the previous parameter value, saving time. *early_stopping* was also set to true. This consists in stopping an iterative optimization method before the convergence of the training loss, to avoid over-fitting.

The training results of the new set of parameters (*hidden_layer_sizes*=(500, 20, 20, 20), *max_iter*=500, *batch_size*=32, *warm_start*=True, *early_stopping*=True) are presented in Table 4:

Table 4: Experiment results 3

	Precision	Recall	F1 score	# of sentences
Conservative	77%	73%	75%	1502
Liberal	81%	79%	80%	1896
Neutral	84%	88%	86%	2258
Average	**81%**	**81%**	**81%**	**5656**

These results show that the MLP classifier has achieved an F1 score higher than that obtained by an RNN model on the same dataset [21]. A summary of the significant experiments along with the parameters used in each set of experiment are shown in Table 5:

Table 5: Experiments summary

	Experiment 1	Experiment 2	Experiment 3
hidden_layer_size	10,10,10,10	20,20,20,20	500,20,20,20
max_iter	200	1000	1000
batch_size	None	None	32
warm_start	False	False	True
early_stopping	False	False	True
F1 score	**68%**	**72%**	**81%**

4.2.2 Chrome extension experiments. Based on the positive results obtained by the classifier, the Chrome extension was implemented to classify real news articles. Experiments on 20 online news articles did not yield promising results, however. Most articles were classified as 100% neutral, although many are taken from conventionally opinionated news sites. Some were classified with the opposite political ideology from what was expected, as shown in Table 6:

Table 6: Online news articles classification

Article	Liberal	Neutral	Conservative
Huffington Post article #1	0%	99.54%	0.46%
Huffington Post article #2	0%	100%	0%
Bloomberg article	0%	100%	0%
CNN article #1	0%	100%	0%
CNN article #2	0%	100%	0%
Fox News article #1	0%	100%	0%
Fox News article #2	0%	100%	0%
Breitbart article #1	5.41%	94.59%	0%
Breitbart article #2	0%	100%	0%
The Economist article	0%	100%	0%
NYTimes article	0.34%	99.66%	0%
Wall Street Journal article	0%	100%	0%
The Blaze article	0%	100%	0%
Slate article #1	0%	100%	0%
Slate article #2	0%	99.91%	0.09%
NPR article #1	0%	100%	0%
NPR article #2	0%	99.69%	0.31%
BBC article #1	0%	100%	0%
BBC article #2	0%	100%	0%
Medium article	0%	100%	0%

These unexpected outcomes can be attributed to several reasons. First, news sites might be reporting with more structurally neutral sentences (while the overall article can still be biased), which neither a sentence-level nor word-level classifier could identify. Second, the use of metaphors and negative phrases were not captured by the classifier. For example, a phrase such as "the following statements are proved to be wrong" can revert the meaning of all the statements behind it. Finally, the MLP classifier did not take into consideration relations between different words and phrases, which led to inaccurate classification results of online news.

5 CONCLUSION

This study has explored a different technical approach to the problem of political bias detection using Multilayer Perceptron model, implemented by Python's machine learning library *scikit-learn*. The word embedding matrix for the MLP model is initialized with Facebook's fastText word vector representation model. The results of the classification task show that, with a proper set of parameters, the MLP model outperforms the recurrent neural network model on the IBC dataset of 4062 labeled sentences. While the RNN achieve an F1 score of 71.8%, the MLP model in this project was able to attain an F1 score of 81%. However, real-time political news classification (through Chrome extension) did not yield promising results. Sentence-level classification of 20 articles from different opinionated news sites yielded either "100% Neutral" results or results

opposite from what was expected. Future works on this project can explore several different directions. First, the MLP model can be designed to classify sentences based on their semantic structures in order to capture negation and metaphors. Second, the MLP model can utilize larger labeled datasets for better training results. Finally, other word embedding matrices (word2vec, GloVe, etc.) can be employed to provide different vector representations of words and sentences.

6 ACKNOWLEDGMENT

I would like to express my gratitude towards Dr. Xunfei Jiang, Dr. David Barbella, Dr. Ajit Chavan, and Dr. Charlie Peck for their support, guidance and feedback throughout the project. I would also like to thank Ahsan Khoja and Rei Rembeci, computer science students at Earlham College for their invaluable help in implementing the Chrome extension and writing up this study.

REFERENCES

[1] Amr Ahmed and Eric P Xing. 2010. Staying informed: supervised and semi-supervised multi-view topical analysis of ideological perspective. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 1140–1150.

[2] Alexandra Balahur, Zornitsa Kozareva, and Andrés Montoyo. 2009. Determining the polarity and source of opinions expressed in political debates. In *International Conference on Intelligent Text Processing and Computational Linguistics*. Springer, 468–480.

[3] Yoshua Bengio. 2012. Practical recommendations for gradient-based training of deep architectures. *CoRR* abs/1206.5533 (2012). arXiv:1206.5533 http://arxiv.org/abs/1206.5533

[4] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5, 2 (1994), 157–166.

[5] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching Word Vectors with Subword Information. *CoRR* abs/1607.04606 (2016). arXiv:1607.04606 http://arxiv.org/abs/1607.04606

[6] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. 2013. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 108–122.

[7] Matthew Carey. 2016. How Donald Trump and Hillary Clinton are changing the social media game. *Los Angeles Daily News* (2016). https://www.dailynews.com/2016/11/05/how-donald-trump-and-hillary-clinton-are-changing-the-social-media-game/

[8] Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*. ACM, 160–167.

[9] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. 2011. Natural Language Processing (almost) from Scratch. *CoRR* abs/1103.0398 (2011). arXiv:1103.0398 http://arxiv.org/abs/1103.0398

[10] Matt W Gardner and SR Dorling. 1998. Artificial neural networks (the multilayer perceptron) - a review of applications in the atmospheric sciences. *Atmospheric environment* 32, 14-15 (1998), 2627–2636.

[11] Edward Grefenstette, Phil Blunsom, Nando de Freitas, and Karl Moritz Hermann. 2014. A Deep Architecture for Semantic Parsing. *CoRR* abs/1404.7296 (2014). arXiv:1404.7296 http://arxiv.org/abs/1404.7296

[12] Ozan Irsoy and Claire Cardie. 2013. Bidirectional recursive neural networks for token-level labeling with structure. *arXiv preprint arXiv:1312.0493* (2013).

[13] Mohit Iyyer, Peter Enns, Jordan Boyd-Graber, and Philip Resnik. 2014. Political ideology detection using recursive neural networks. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 1113–1122.

[14] Mohit Iyyer, Peter Enns, Philip Resnik, and Jordan Boyd-Graber. 2014. The Ideological Books Corpus. https://www.cs.umd.edu/~miyyer/ibc/index.html. (2014).

[15] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A Convolutional Neural Network for Modelling Sentences. *CoRR* abs/1404.2188 (2014). arXiv:1404.2188 http://arxiv.org/abs/1404.2188

[16] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. *CoRR* abs/1408.5882 (2014). arXiv:1408.5882 http://arxiv.org/abs/1408.5882

[17] Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* (2014).

[18] K. Kowsari, D. E. Brown, M. Heidarysafa, K. Jafari Meimandi, M. S. Gerber, and L. E. Barnes. 2017. HDLTex: Hierarchical Deep Learning for Text Classification. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 364–371. https://doi.org/10.1109/ICMLA.2017.0-134

[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. *CoRR* abs/1310.4546 (2013). arXiv:1310.4546 http://arxiv.org/abs/1310.4546

[20] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[21] Arkajyoti Misra and Sanjib Basak. 2017. Political Bias Analysis. (2017).

[22] Bo Pang, Lillian Lee, et al. 2008. Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval* 2, 1–2 (2008), 1–135.

[23] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*. Association for Computational Linguistics, 79–86.

[24] 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. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[25] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543. http://www.aclweb.org/anthology/D14-1162

[26] Yanchuan Sim, Brice DL Acree, Justin H Gross, and Noah A Smith. 2013. Measuring ideological proportions in political speeches. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. 91–101.

[27] Matt Thomas, Lillian Lee, and Bo Pang. 2006. Congressional floor transcript debate. http://www.cs.cornell.edu/home/llee/data/convote.html. (2006).

[28] Matt Thomas, Bo Pang, and Lillian Lee. 2006. Get out the vote: Determining support or opposition from Congressional floor-debate transcripts. In *Proceedings of the 2006 conference on empirical methods in natural language processing*. Association for Computational Linguistics, 327–335.