Final Report

Hyperpartisan News Detection

Cardiff University

School of Computer Science and Informatics

CM3203 - One Semester Individual Project

Author: Patrick Noyau

Student Number: 1723822

Supervisor: Luis Espinosa-Anke

Moderator: Dr Daniel J. Finnegan

Abstract

Hyperpartisan news publishers are far more likely to produce news articles containing falsehoods than neutral mainstream publishers. The impact fake news can have on societies and democracies is huge, and since 2016 there has been much hype surrounding the issue, with companies and governments seeking to crack down on its emergence.

The aim of this project is to create a machine learning classification system, that can successfully predict if a news article is hyperpartisan or not, and to see which features of news articles are most successful at differentiating between hyperpartisan and neutral news.

Table of Contents

Table of Figures

Introduction

Since starting this project in January much in the world has changed; the way we all live our lives, for at least the apparent future will be highly unfamiliar. The way that many people accrue their news has changed, and although not a trend that started during the Coronavirus Pandemic (Ponsford 2019), in the UK at least, the restrictions on people's ability to buy traditional print newspapers in shops have driven rises in reading news online (Sweney 2020). However, consuming news online, especially through social media sites like Facebook, can cause issue due to the prevalence of 'fake news'. As stated in (Bakir and McStay 2018), 'fake news' produces uninformed citizens, that remain uninformed due to the echo-chamber (only encountering opinions that coincide with your own) effect of social media. Research shows that hyperpartisan (very left-wing or very right-wing) news publishers have far higher numbers of falsehoods in their articles compared to neutral mainstream publishers (Silverman et al. 2016).

This is the motivation behind this project's aims; to create a machine learning classification system to detect hyperpartisan news. The creation of such a system could be very beneficial to many. Currently, immobilising fake news can rely on organisations or charities such as FullFact[1] to manually fact-check claims in a news article to assess their trustworthiness. The process is lengthy, especially when compared to the rate at which articles will be published online, meaning that by the time a rebuttal can be produced, the damage of the 'fake news' has already been done. A system that can quickly predict if an article is hyperpartisan could be embedded into social media websites to provide users with a fast indication if the article they are reading is reliable or not.

[1] fullfact.org

Outcomes

- Creation of system that can accept plain text files and convert them into Python objects in order to be classified
- Creation of functionality that can transform a text piece into an array of the key features it contains
    - Features are custom made, and also come from libraries
- Identification of which features best discern between hyperpartisan news articles and neutral news articles from established mainstream publishers
- Creation of system that can accept text from a command line, analyses the features it contains and outputs if it is predicted to be hyperpartisan or not

Background

The prevalence and effects of Hyperpartisan News

As discussed in the introduction, 'fake news' proves a big problem for democracies and societies. (Bakir and McStay 2018) references Jürgen Habermas' "democratic ideal", where people in democracies rationally speak and listen to each others viewpoints before agreeing the best way forward. They find that if people are "indoctrinated to disbelieve truthful facts" and rather believe falsehoods, then discontent with the democratic process and outcome is likely. The alarming end result that (Bakir and McStay 2018) predict says that societies will become highly polarised, with people who did not vote with the majority's confidence in a governments legitimacy decreased.

Just before the 2016 United States Presidential Election, investigators at Buzzfeed News published their research into hyperpartisan Facebook pages and the role they were playing in spreading false information (Silverman et al. 2016). Selecting pages with large followings from left-wing, right-wing and neutral mainstream organisations, they manually fact checked every post by these pages over a period of seven days, marking their content as 'mostly true', 'mostly false' or a mix of both .

Publisher type (articles analysed)	Mostly True	Mostly False	Mix	Combined Scores – Mix and Mostly False
Neutral (826)	97.6%	0%	0.969%	0.969%
Left-wing (256)	71.1%	5.86%	19.9%	25.8%
Right-wing (545)	50.6%	13.2%	8.07%	21.3%

Figure 1- Summary of categorisation. Excludes articles with "no factual claim or content" (Silverman et al. 2016)

Figure 1 shows the results, with the content of neutral mainstream publishers being categorised overwhelmingly as 'mostly true', whereas left-wing having just under 6% of articles being classed 'mostly false' and around a quarter being either 'mostly false' or a mixture of false and true. Right-wing publishers performed even more unfavourably, with 13% of its articles being classed ‘mostly false’, and 21% either mostly false or mixed.

Left- wing and right- wing articles containing higher proportions of falsehoods is significant as research by (Vosoughi et al. 2018) into the circulation of false news online, found that falsehoods spread online much faster than truths, with a falsehood reaching 1500 people online six times faster than a truth would. The research also found that false- political news spread even faster than other types of falsehoods. The emotional content of the replies to falsehoods online/ social media were more likely to be ‘disgust’, ‘anger’ or ‘surprise’ compared to the responses for truths, which were ‘anticipation’, ‘joy’, ‘trust’ and ‘sadness’ (Vosoughi et al. 2018). Finally, the research also found that false news was also more likely to be spread by humans peer to peer, essentially going ‘viral’.

(Vargo et al. 2018) studied the online media landscape between 2014 and 2016, found that the subject matter of fake news spread online would then influence the agendas of traditional partisan news outlets - perhaps with the justification for an issue being reported in a traditional outlet originating from the high levels of discussion happening online, that caused by dissemination of fake news. The impact of fake news setting the agenda for other, more traditional, media settings is significant, as it shows that fake news can be used to make certain issues a talking point.This may also explain the left- wing and right- wing news websites having still having significant levels of content classified as ‘mostly true’ in Figure 1. These outlets may be able to utilise the rapid spread of fake news to generate interest in a certain topic, which then can be used as basis for increased truth based reporting on a particular issue.

The high levels of falsehoods contained in news articles from right- wing and left-wing publishers, combined with the effects that false news can cause, indicate the need for a tool that can identify articles that originate from these sources.

Associated Theory

(Potthast et al. 2020)'s research into hyperpartisan news sought to establish if hyperpartisan news articles can be distinguished from neutral mainstream articles using it's stylistic features. Using the technique 'Unmasking' proposed by (Koppel et al. 2007), where in order to predict the author of a given text the most distinguishing features are recursively removed. With each recursion, the text with less distinguishing features is classified and the cross-validation accuracy is measured; the faster the accuracy degrades, the more likely the text was written by that author. When plotted on a graph, the gradient of the curve produced provides an indication of the similarity (steeper curve = more similar).

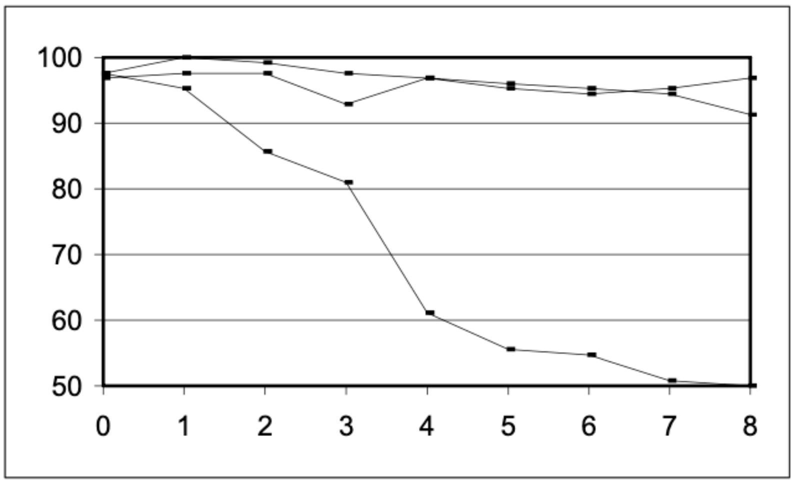


Figure 2- The steeper curve indicates the document is more similar to the previous ones by the author. (Koppel et al. 2007)

Potthast et al. applied the same principle to hyperpartisan and neutral mainstream news articles, unmasking left-wing and right-wing articles against neutral ones. Their results in Figure 2 show that when left-wing and right-wing articles are unmasked against each other their curve decreases considerably more rapidly than when compared unmasking against neutral mainstream ones. This means that left-wing articles are similar in style to right-wing ones (and vice versa), which is crucial as it means that we can simply classify between hyperpartisan and neutral articles rather than a three-way classification of left-wing, right-wing and neutral.

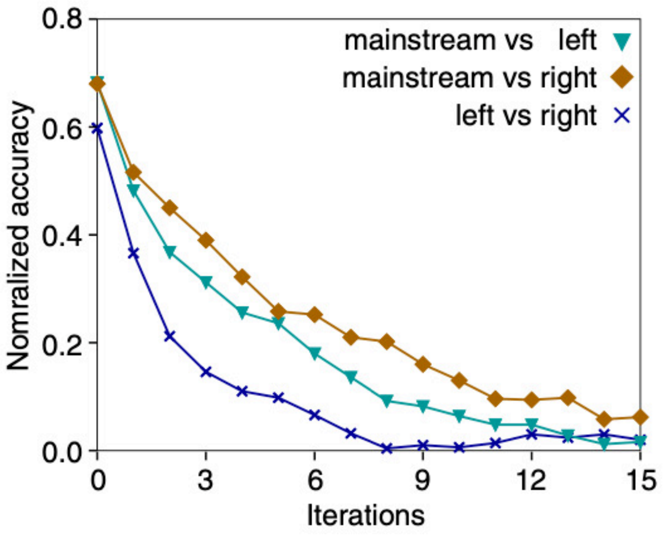
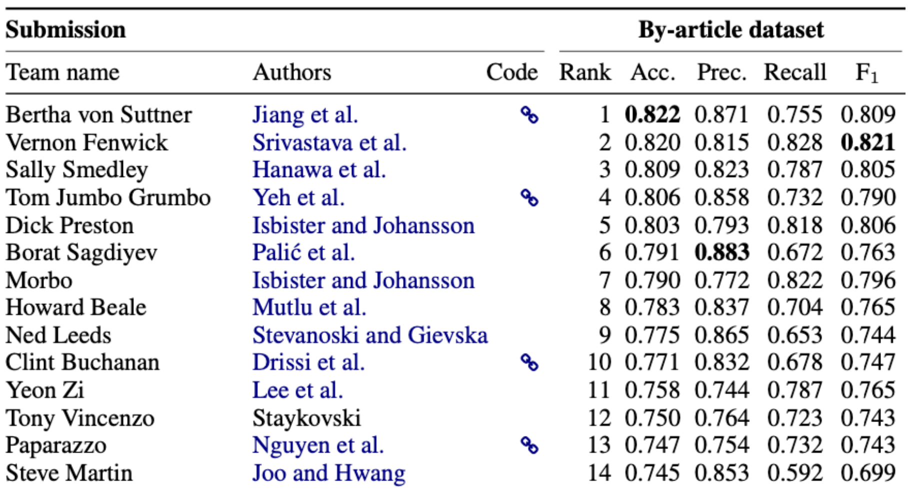


Figure 3- The steep blue curve shows that left-wing articles are similar in style to right-wing ones (Potthast et al.)

The 13th International Workshop on Semantic Evaluation in 2019 (SemEval – 2019), an "ongoing series of evaluations" of semantic evaluation systems to improve the state of semantic analysis (Association for Computational Linguistics 2019). Part of SemEval – 2019 included a task where teams were invited to create and submit a classification system to detect hyperpartisan news articles (Kiesel et al. 2019). All teams were given the same dataset of news articles to train their classification system with, which were manually classified as neutral or hyperpartisan. After the competition was completed, the entries were ranked by their accuracy and the results were published for the workshop.



Submission			**By-article dataset**				
Team name	Authors	Code	Rank	Acc.	Prec.	Recall	F_1
Bertha von Suttner	Jiang et al.	🔗	1	**0.822**	0.871	0.755	0.809
Vernon Fenwick	Srivastava et al.		2	0.820	0.815	0.828	**0.821**
Sally Smedley	Hanawa et al.		3	0.809	0.823	0.787	0.805
Tom Jumbo Grumbo	Yeh et al.	🔗	4	0.806	0.858	0.732	0.790
Dick Preston	Isbister and Johansson		5	0.803	0.793	0.818	0.806
Borat Sagdiyev	Palić et al.		6	0.791	**0.883**	0.672	0.763
Morbo	Isbister and Johansson		7	0.790	0.772	0.822	0.796
Howard Beale	Mutlu et al.		8	0.783	0.837	0.704	0.765
Ned Leeds	Stevanoski and Gievska		9	0.775	0.865	0.653	0.744
Clint Buchanan	Drissi et al.	🔗	10	0.771	0.832	0.678	0.747
Yeon Zi	Lee et al.		11	0.758	0.744	0.787	0.765
Tony Vincenzo	Staykovski		12	0.750	0.764	0.723	0.743
Paparazzo	Nguyen et al.	🔗	13	0.747	0.754	0.732	0.743
Steve Martin	Joo and Hwang		14	0.745	0.853	0.592	0.699

Figure 4 - Results of submissions to competition

The teams submissions were analysed and the competition organisers collated which methods worked effectively for the teams , which can be useful for this project, giving a head start on what makes an effective system.

Potential Stakeholders

Much of the hyperpartisan news is spread on social media websites, particularly in the 2016 presidential election, on Facebook (Potthast et al. 2020). Given this, the ability to be able to detect hyperpartisan news could be of potential interest to social media platforms looking to remove such content from their websites. This is especially credible following statements made by Facebook in the wake of the 2016 US Presidential election, where the company stated it’s aim to fight the spread of false news on their platforms (Mosseri 2017).

Other potential stakeholders could include national governments around the world seeking to limit the damaging effect false news can have on democracies (Bakir and McStay 2018). For example in 2019, the UK Government stated it’s aim to fight fake news and inroduced an £18 million package to that end (Foreign & Commonwealth Office 2019).

Potential tools to solve the problem

Scikit- learn (SK-Learn)[2], is Python library specifically designed for machine learning created by (Pedregosa et al. 2011). The library contains many tools for formatting and processing data in order to be classified, and also classifiers themselves that can learn from data inputted and make predictions.

The 'Sentiment Intensity Analyser' module from the Natural Language Toolkit[3] (Bird et al. 2009), provides a tool to analyse the sentiment intensity of a piece of text. Created by (Hutto and Gilbert 2014), the 'VADER' tool utilises a sentiment lexicon created from 'micro-blog' continent such as Twitter and Facebook and generic rules to return a prediction of the sentiment that the text is inputted. The tool was designed to be 'computationally economical', while also not sacrificing its accuracy.

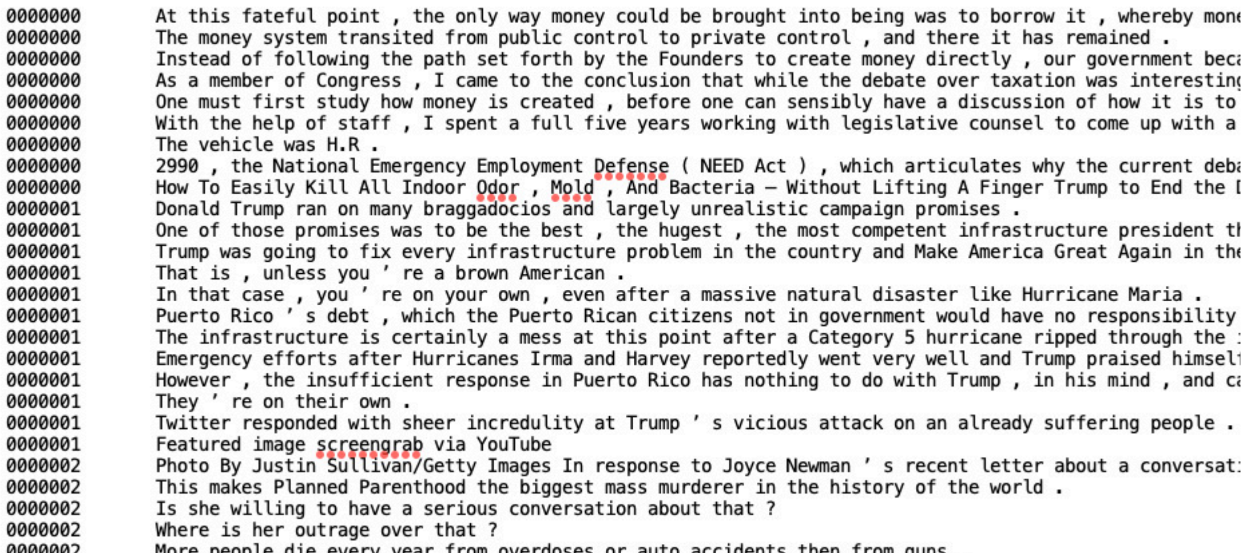
A 'Spelling Corrector' from the Python Package Index[4] , which is designed to be used as a tool to return predictions for the correct spelling of a word that is incorrectly spelt. The tool uses a dataset of the words in the English language, and the frequency of their common usage. The tool can return words that are not on this frequency list, so therefore are likely to be misspelled.

[2] scikit-learn.org
[3] www.nltk.org
[4] www.pypi.org/project/pyspellchecker/

Approach and Implementation

Loading files into the system

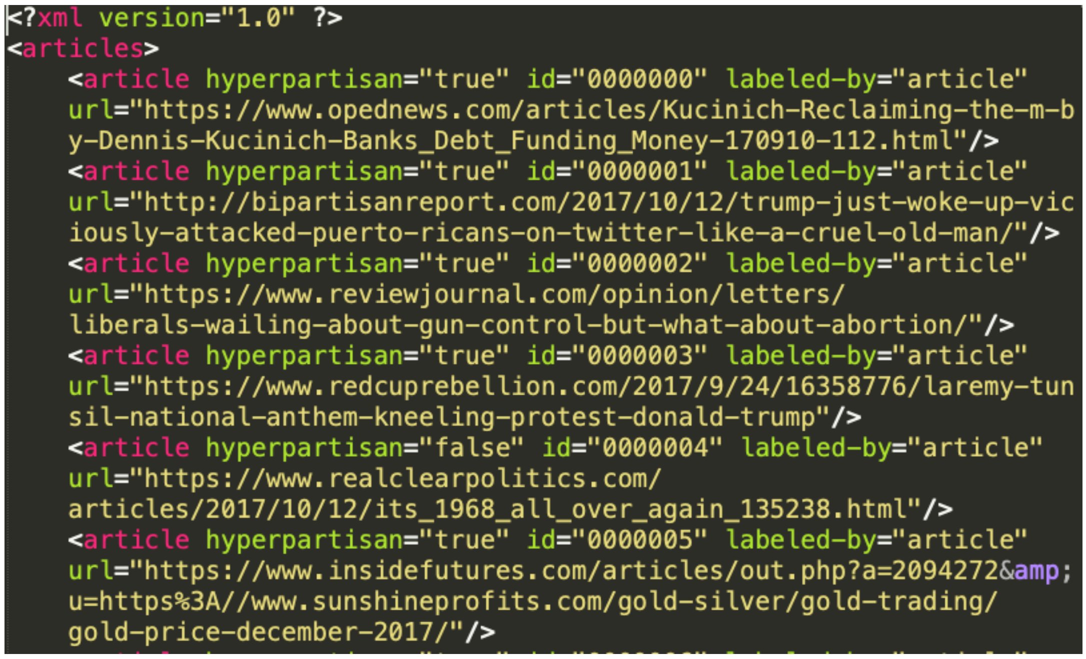
The system accepts its training and testing data from plaintext files that the Python program converts into python objects that the classifier can process. Each line from an article is stored as a line in a text file, where the first 6 characters are a unique article ID and the rest of the line is the text from the article.



```
0000000      At this fateful point , the only way money could be brought into being was to borrow it , whereby mone
0000000      The money system transited from public control to private control , and there it has remained .
0000000      Instead of following the path set forth by the Founders to create money directly , our government beca
0000000      As a member of Congress , I came to the conclusion that while the debate over taxation was interesting
0000000      One must first study how money is created , before one can sensibly have a discussion of how it is to
0000000      With the help of staff , I spent a full five years working with legislative counsel to come up with a
0000000      The vehicle was H.R .
0000000      2990 , the National Emergency Employment Defense ( NEED Act ) , which articulates why the current deba
0000000      How To Easily Kill All Indoor Odor , Mold , And Bacteria — Without Lifting A Finger Trump to End the I
0000001      Donald Trump ran on many braggadocios and largely unrealistic campaign promises .
0000001      One of those promises was to be the best , the hugest , the most competent infrastructure president th
0000001      Trump was going to fix every infrastructure problem in the country and Make America Great Again in the
0000001      That is , unless you ’ re a brown American .
0000001      In that case , you ’ re on your own , even after a massive natural disaster like Hurricane Maria .
0000001      Puerto Rico ’ s debt , which the Puerto Rican citizens not in government would have no responsibility
0000001      The infrastructure is certainly a mess at this point after a Category 5 hurricane ripped through the :
0000001      Emergency efforts after Hurricanes Irma and Harvey reportedly went very well and Trump praised himsel
0000001      However , the insufficient response in Puerto Rico has nothing to do with Trump , in his mind , and ca
0000001      They ’ re on their own .
0000001      Twitter responded with sheer incredulity at Trump ’ s vicious attack on an already suffering people .
0000001      Featured image screengrab via YouTube
0000002      Photo By Justin Sullivan/Getty Images In response to Joyce Newman ’ s recent letter about a conversat:
0000002      This makes Planned Parenthood the biggest mass murderer in the history of the world .
0000002      Is she willing to have a serious conversation about that ?
0000002      Where is her outrage over that ?
0000002      More people die every year from overdoses or auto accidents then from guns
```

Figure 5 - Articles from dataset. Lefthand side shows the unique article ID

The labels (if the article is hyperpartisan / neutral) for each article are stored in an XML file where the attributes for each XML element are the unique article ID, the URL to the original article, who performed the manual labelling and a Boolean value based on if it is hyperpartisan or not.



```
<?xml version="1.0" ?>
<articles>
    <article hyperpartisan="true" id="0000000" labeled-by="article"
    url="https://www.opednews.com/articles/Kucinich-Reclaiming-the-m-b
    y-Dennis-Kucinich-Banks_Debt_Funding_Money-170910-112.html"/>
    <article hyperpartisan="true" id="0000001" labeled-by="article"
    url="http://bipartisanreport.com/2017/10/12/trump-just-woke-up-vic
    iously-attacked-puerto-ricans-on-twitter-like-a-cruel-old-man/"/>
    <article hyperpartisan="true" id="0000002" labeled-by="article"
    url="https://www.reviewjournal.com/opinion/letters/
    liberals-wailing-about-gun-control-but-what-about-abortion/"/>
    <article hyperpartisan="true" id="0000003" labeled-by="article"
    url="https://www.redcuprebellion.com/2017/9/24/16358776/laremy-tun
    sil-national-anthem-kneeling-protest-donald-trump"/>
    <article hyperpartisan="false" id="0000004" labeled-by="article"
    url="https://www.realclearpolitics.com/
    articles/2017/10/12/its_1968_all_over_again_135238.html"/>
    <article hyperpartisan="true" id="0000005" labeled-by="article"
    url="https://www.insidefutures.com/articles/out.php?a=2094272&amp;
    u=https%3A//www.sunshineprofits.com/gold-silver/gold-trading/
    gold-price-december-2017/"/>
```

Figure 6- Extract from XML Label Document

The *readXML* function accepts the name of the XML file to process as string object, it then uses the XML Element Tree API [5]from the Python library to parse the XML document and create an Element Tree object. The program then iterates through the element tree, and creates two list objects- one containing the IDs of all the hyperpartisan articles and the other all the neutral articles.

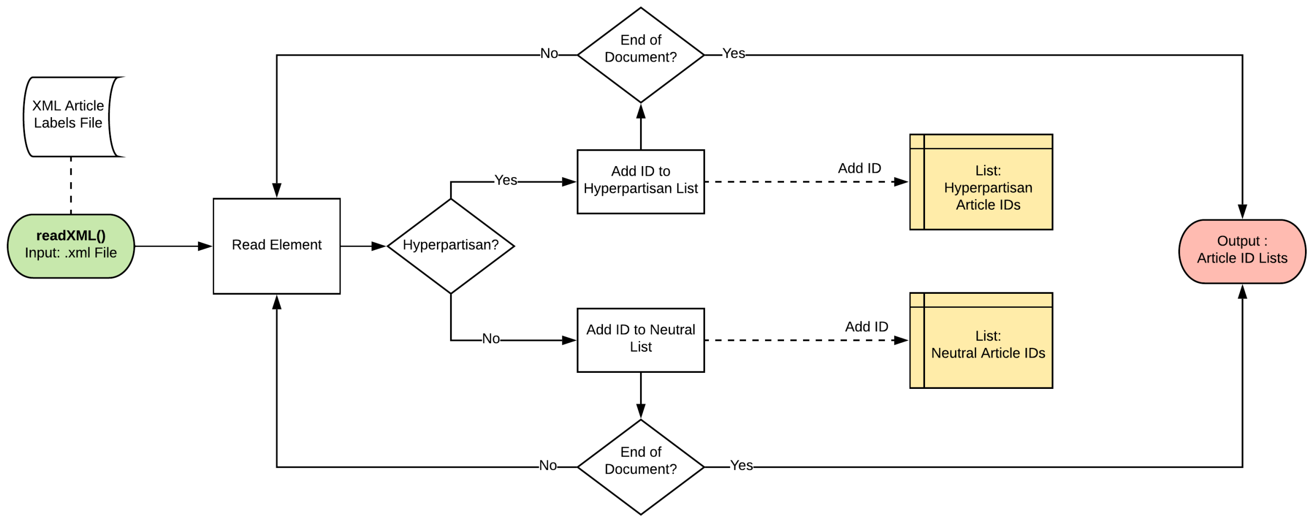
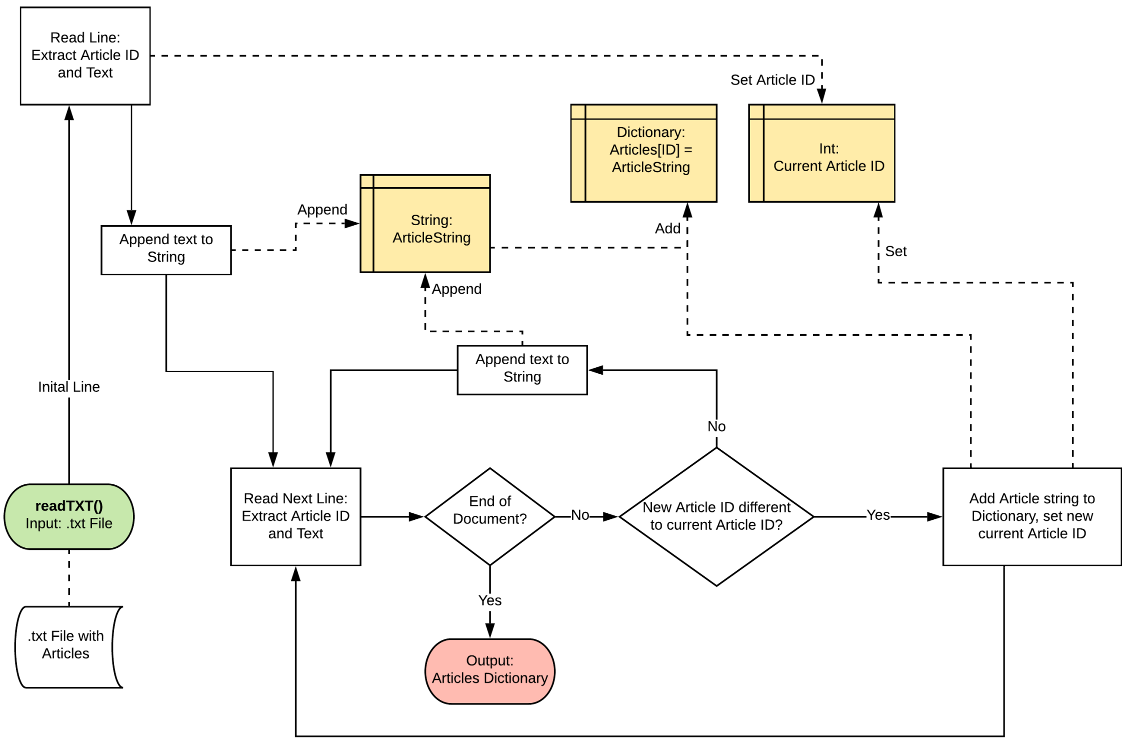

XML Article Labels File
readXML() Input: .xml File
Read Element
Hyperpartisan?
Yes
No
Add ID to Hyperpartisan List
Add ID to Neutral List
Add ID
Add ID
List: Hyperpartisan Article IDs
List: Neutral Article IDs
End of Document?
End of Document?
No
Yes
No
Yes
Output : Article ID Lists

Figure 7 - Diagram detailing readXML() function

[5] https://docs.python.org/3.8/library/xml.etree.elementtree.html

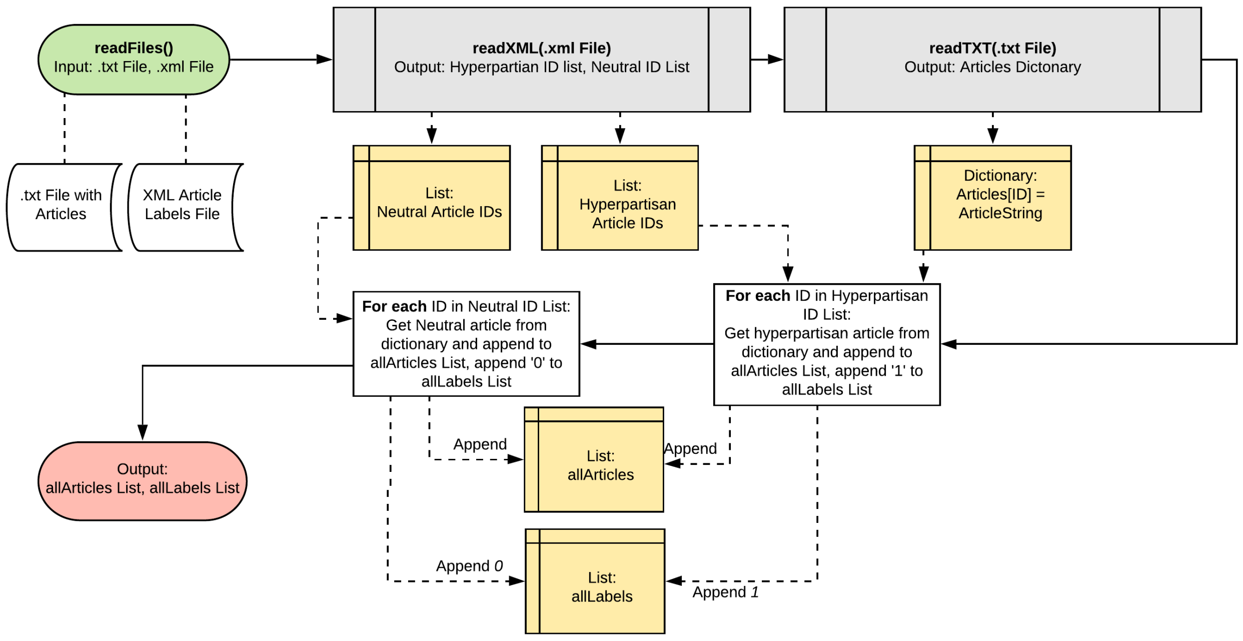
The *readTXT* function parses through the text file, from each line it reads the first 6 characters to identify the unique article ID. The line of text after the ID is then added to a string object. When it reads the next line, if the article ID is the same as the line before, it appends that line to the string object too. The process is repeated until it reaches a new article ID, meaning that the previous article has ended. The string object is added to the dictionary of articles, where the key is the article ID. The process then repeats itself until the end of the text file is reached.



Read Line: Extract Article ID and Text
Set Article ID
Dictionary: Articles[ID] = ArticleString
Int: Current Article ID
Append
String: ArticleString
Append text to String
Add
Set
Append
Append text to String
Inital Line
No
readTXT()
Input: .txt File
Read Next Line: Extract Article ID and Text
End of Document?
No
New Article ID different to current Article ID?
Yes
Add Article string to Dictionary, set new current Article ID
Yes
.txt File with Articles
Output: Articles Dictionary

Figure 8 - Diagram of readTXT() function

The *readFiles* function utilises both the *readXML* and *readTXT* methods, which return two lists containing the article IDs for all the hyperpartisan/ neutral articles and a dictionary with the articles referenced to it's article ID. Two further lists are then created, an *allArticles* list where the strings of every article can be contained and the other list for the labels of each article called *allLabels*. The two lists are ordered, so the first item of the *allLabels* list relates to the first string in the *allArticles* list. Starting with the list of hyperpartisan article IDs, the list is iterated through so for each hyperpartisan article added to the *allArticles* list, then the number '1' (indicating the article is hyperpartisan) is then added to the *allLabels* list. The same is then done with the neutral articles ID list, adding a '0' to the *allLabels* list to indicate that it is not hyperpartisan.



readFiles()
Input: .txt File, .xml File
.txt File with Articles
XML Article Labels File
readXML(.xml File)
Output: Hyperpartian ID list, Neutral ID List
readTXT(.txt File)
Output: Articles Dictonary
List: Neutral Article IDs
List: Hyperpartisan Article IDs
Dictionary: Articles[ID] = ArticleString
For each ID in Neutral ID List: Get Neutral article from dictionary and append to allArticles List, append '0' to allLabels List
For each ID in Hyperpartisan ID List: Get hyperpartisan article from dictionary and append to allArticles List, append '1' to allLabels List
Output: allArticles List, allLabels List
Append
List: allArticles
Append
Append 0
List: allLabels
Append 1

Figure 9 - Diagram showing how readFiles() function works

Transforming the dataset

The program has the textToFeatures function, which converts a list of news articles into an array of features that they contain. As input, the function takes the list of news articles, and Boolean arguments for each feature (essentially an on-off switch for that feature).

Example

Using as an example a sentence from a news article classed as hyperpartisan :

" The left's obsession with gun "control" is just that , control ."

The result of this sentence going through textToFeatures function with these features would be:

- **Feature 1**: Contains the word 'left' once or more
- **Feature 2:** Uses one or more exclamation mark (!)
- **Feature 3:** Number words that are 4 characters or longer

The output vector would be as follows:

Feature 1	Feature 2	Feature 3	Output
1	0	7	[1,0,7]

Figure 10 - Table showing textToFeature output

The vector for each news article from the textToFeatures function is then used by the classifier later on to predict the class of the article. The following features can be calculated using the textToFeatures function:

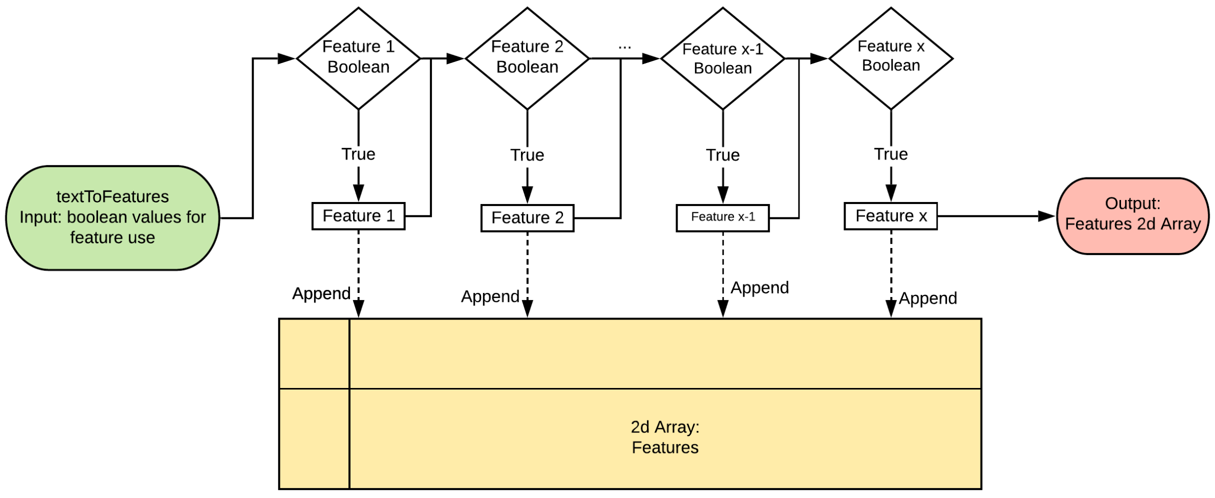


Figure 11 - textToFeatures function

Features

CountVectorizer (Word vectorizer): Utilising SK-Learn's Count Vectorizer feature, the vectorizer analyses the most common words across all the news articles, then for each article indicates if it contains each of the most common words or not.

TF-IDF Vectorizer: This vectorizer works in a similar way to the count vectorizer, but rather than just returning the most common words, it calculates the term frequency multiplied by the inverse document frequency.
Term Frequency (TF) – How many times a word appears in the news article
Inverse Document Frequency (IDF) – How common the word is across all the news articles. If a word is very common and therefore appears across multiple article the IDF will be close to 0, and conversely if it is very rare the IDF will be close to 1.

A higher TF-IDF score, means that the word is more relevant to the document.

Average Sentence Length: For each article, calculates the average number of characters per sentence in the article. (Sentence is defined as words between bullet points)

Word Lengths: For each article, calculates the percentage of words in the document that are: 10 characters or longer, 11 characters or longer and 12 characters or longer.

Sentiment Analysis: Using a tool from the Natural Language Tool Kit[6], which analyses the input news article for its sentiment (how emotional the words are). The tool outputs four scores for the input news articles positive, negative and neutral sentiment. It also releases a compound score, which summarises all three sentiment scores
For example, the sentence "Donald Trump has done a very bad job!", receives these scores:

Sentence:	"Donald Trump has done a very bad job!"		
Negative	Neutral	Positive	Compound
0.405	0.595	0.0	-0.623

[6] https://www.nltk.org/api/nltk.sentiment.html

Conversely, the sentence “Today, United States President visited Michigan for the first time.”, measures as follows:

Sentence:	“Today, United States President visited Michigan for the first time.”		
Negative	Neutral	Positive	Compound
0.0	0.763	0.237	0.4215

Punctuation: Measures the percentage of characters in the news articles that are exclamation points (!) and the percentage that are full stops (.).

Bias Words: This utilises a lexicon 654 of ‘bias-inducing’ words from researchers at Stanford University (Recasens et al. 2013). For each word in the lexicon, the number of times it appears in each news article is recorded.

Reduced Bias Words: When processing the occurrences of the above bias words in the news articles, the processing time was very high due the high computational cost of searching for 600 plus words across 600 news articles (the nested for loops have a big-O[7] notation of $O(n^2)$). In order to reduce the number of words in the list, the SKLearn Recursive Feature Elimination (RFE)[8] module was used. RFE ranked the features based on their importance in a decision tree classifier. K-fold validation was used to cross validate the classification, and the average importance of each feature across each fold was taken. The average importance of each feature was then ranked from best to worst, and the 100 most important bias words were extracted. Having a shorter list of bias words also may prevent overfitting of the data, where the classifier is too complex and starts trying to explain random errors in the dataset.

Total Bias Words: Using the same lexicon as above, for each article it returns the total number of bias words in the text.

Spelling Mistakes: Using a spell checker module[9] , which returns a list of words it doesn’t know (therefore may be spelled incorrectly). The feature returns the percentage of words in the news article that are unknown.

[7] https://en.wikipedia.org/wiki/Big_O_notation
[8] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html
[9] https://pypi.org/project/pyspellchecker/

Classification

The returned news articles, now represented as a list of the features they contain, are then entered into a classifier. The SK-Learn module offers many different classifiers in its package, so various options were tried in an attempt to find the most accurate results. The experiments carried out used three different classifiers from SKLearn;

- The first is a decision tree classifier. At each stage of the tree, a feature is picked which best splits the data so the Gini Impurity of the data either side is as low as possible (Zhou 2019). The Gini Impurity in this context is the probability of a random news article from the dataset being in the wrong category. Then at each split created, a new feature is chosen to best split the data again so that the Gini value is as low as possible. This process is repeated until the Gini value is zero or the tree has reached a maximum depth (Géron 2017).
- The next classifier is Logistic Regression, which works by estimating the probability that an news article is hyperpartisan or not (neutral), if the probability is greater than 50%, the article is classified as hyperpartisan.
- The last classifier tested was a random forest classifier, which trains lots of decision tree classifiers, each with a subset of features from the training data. When making a prediction, the data is processed through each decision tree, and the most common prediction from all the trees is returned as the prediction (Kirk 2017).

The data is split, so that some can be used for training the classifier, and the rest can be used to test the classifiers effectiveness. In order to ensure that any classifier is not effective purely by chance, K-Fold Cross validation was implemented, where the data is shuffled and split into k different folds (sets), where k can be any number (in this case it was 5). Each fold is then used once as a testing set, while the others are used as training data. The process is repeated, so each fold used as a testing set is changed with each iteration. When the classifier has been tested across all k groups, where an average can be taken from all the sets of results.

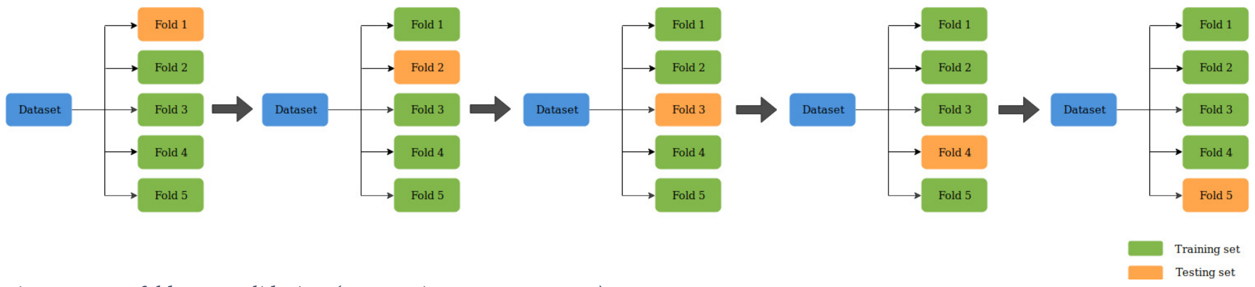


Figure 12 - K-fold cross validation (Data Driven Investor 2018)

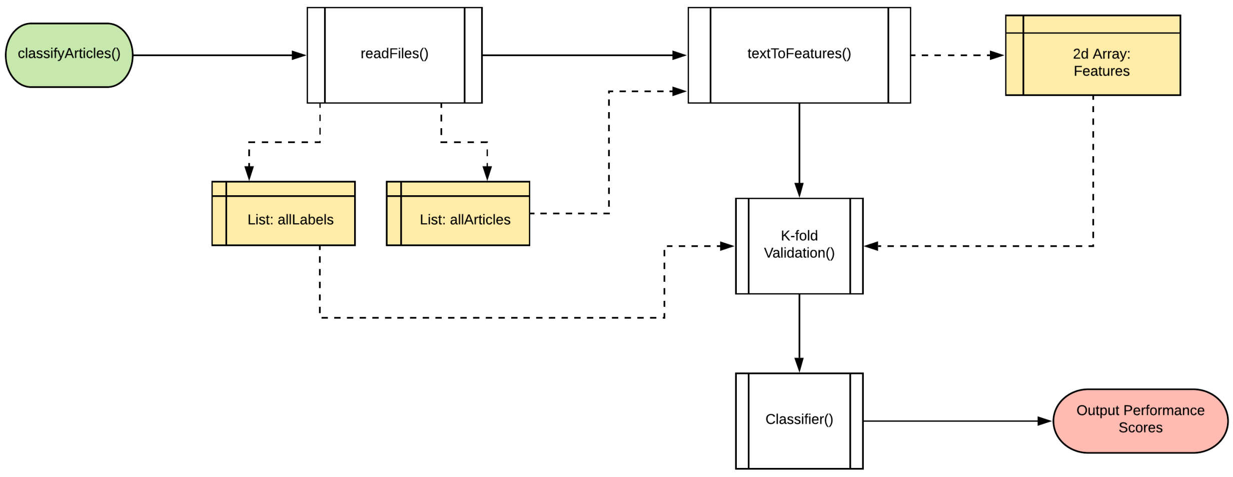


Figure 13 - Diagram of whole classification

Testing Features with Classifiers

In order to see which features were most effective with each classifier type, each classifier was tested with every possible combination of features. Using Python's 'itertools' module [10], a list was created of dictionaries, each containing a unique combination of feature booleans (True or False values), indicating if that feature should be used or not. Initially, the feature dictionary was passed straight to the *TextToFeatures* module to output a matrix with the feature information that could be classified by the classifier, however this method meant that the features in each article were being re-processed with each permutation, which was very time consuming considering that the articles used to test the classifier were kept the same each time so the *TextToFeatures* module was also outputting the same values every time. To remove this redundancy, at the start of the program the *TextToFeatures* module produces a 'master dictionary', where each key is the feature name and the value is the array of feature information for each article. For each dictionary in the feature

[10] https://docs.python.org/3/library/itertools.html

boolean list, the program makes a copy of the 'master dictonary', and deletes any features that have a false value. The remaining values in the copy of the 'master dictionary' are converted into a 2 -dimensional array, which can then be inputted into the classification module.

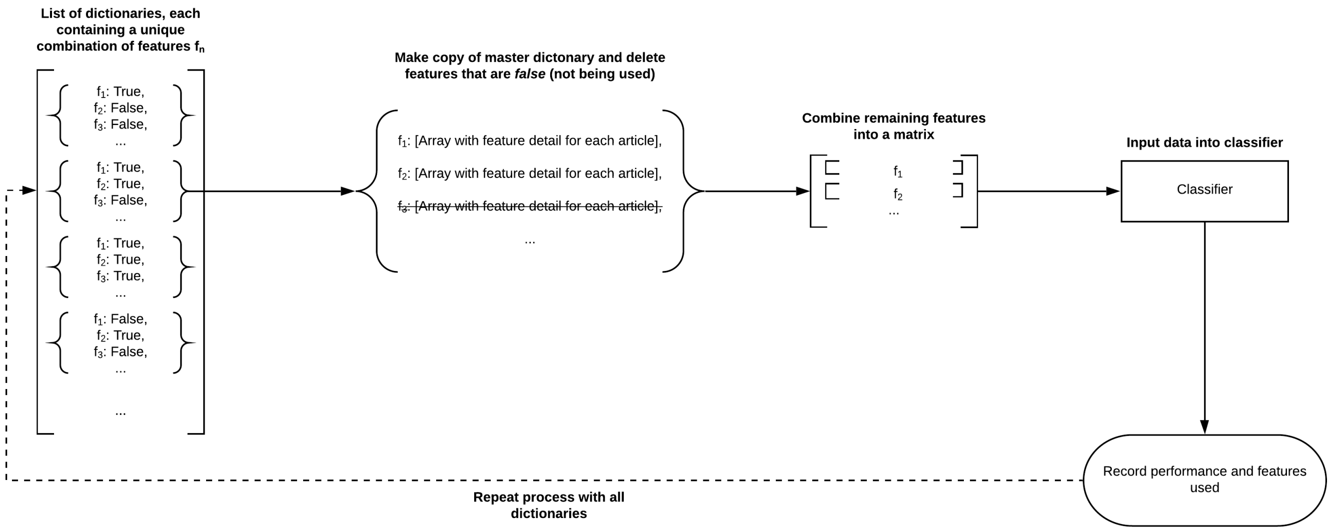

List of dictionaries, each containing a unique combination of features fn
f1: True, f2: False, f3: False, ...
f1: True, f2: True, f3: False, ...
f1: True, f2: True, f3: True, ...
f1: False, f2: True, f3: False, ...
...
Make copy of master dictonary and delete features that are false (not being used)
f1: [Array with feature detail for each article],
f2: [Array with feature detail for each article],
~~f3: [Array with feature detail for each article],~~
...
Combine remaining features into a matrix
f1
f2
...
Input data into classifier
Classifier
Record performance and features used
Repeat process with all dictionaries

Figure 14 - Diagram showing process of iterating through different feature combinations

The classification module splits the articles with k-fold cross validation, using the module from SKLearn[11]. In order to make sure each permutation of features is tested fairly, before the articles are split into folds, they are shuffled in the same way every time by fixing the random state, meaning that the same articles are submitted for training and testing for each permutation.

When testing different combinations of articles with the random forest classifier, the processing time to classify each combination was taking around 15-20 seconds, which repeated over 500 or so combinations would mean that the total time to test that classifier would total around 10,000 seconds or just under 3 hours. In order to try and improve the processing time for this classifier, multiprocessing was implemented. Multiprocessing makes use of the multiple processors built into most computers, splitting the processing operations are done in parallel on each processor rather than one by one (serial operation), meaning that overall the processing can be done quicker. Utilising Python's multiprocessing[12] module, the processing average processing time for each combination of features was reduced by 70% to around 6

[11] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html
[12] https://docs.python.org/3/library/multiprocessing.html

seconds. This meant the overall processing time was around 45 minutes. The results of each combination of features with each classifier was then saved to a CSV file, so that they could be analysed in Microsoft Excel.

Creating a command line tool

After the most effective classifier – feature combination was found, the classifer can be retrained with all the available data, without the cross validation. The Pickle[13] module from the Python library was used to save a copy of the most effective classifier. The other feature that required initial 'fitting' to the data was the TFIDF vectorizer, therefore a copy of the vectorizer was stored using Pickle too. This allowed for a program to be created that could accept a news article in the form of a string object as input from the command line terminal. The classifier and vectorizer that was previously fit could be loaded into the program; the vectorizer used to identify the presence of certain key words and the classifier used to make a prediction of the news article's partisanship. The program then outputs this information to the command line, along with the percentage certainty of the prediction.

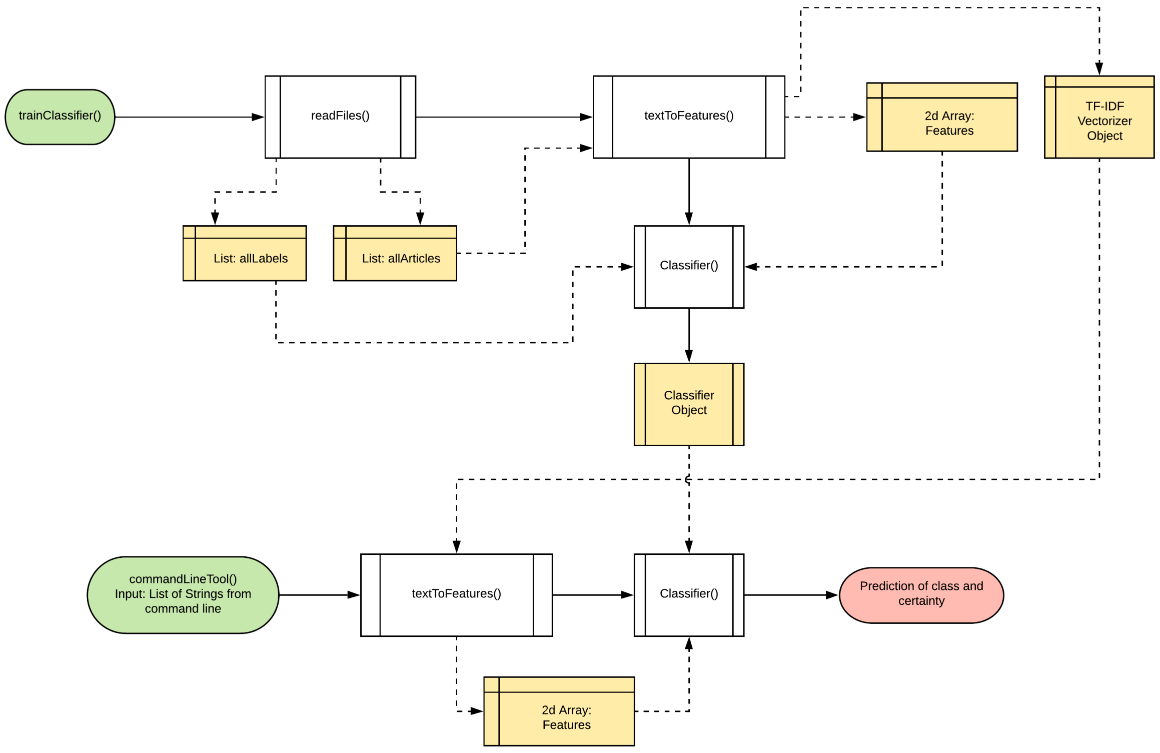
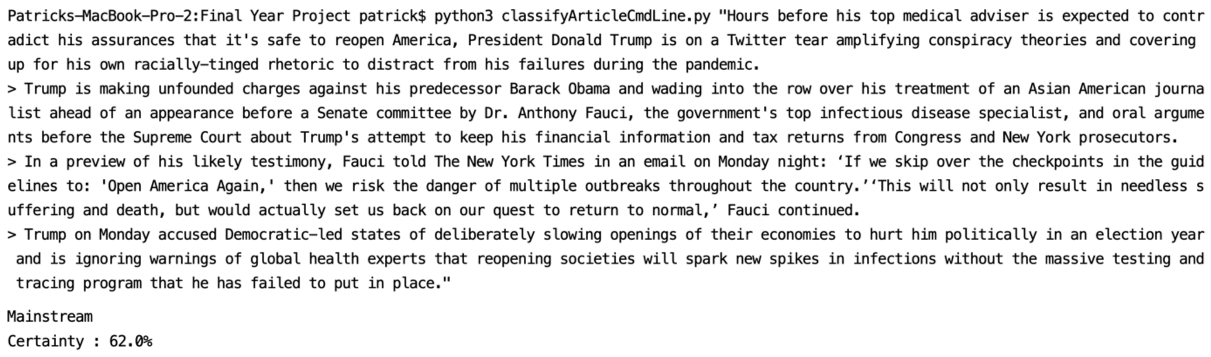


Figure 15 - Diagram of command line tool

[13] https://docs.python.org/3/library/pickle.html



```
Patricks-MacBook-Pro-2:Final Year Project patrick$ python3 classifyArticleCmdLine.py "Hours before his top medical adviser is expected to contr
adict his assurances that it's safe to reopen America, President Donald Trump is on a Twitter tear amplifying conspiracy theories and covering
up for his own racially-tinged rhetoric to distract from his failures during the pandemic.
> Trump is making unfounded charges against his predecessor Barack Obama and wading into the row over his treatment of an Asian American journa
list ahead of an appearance before a Senate committee by Dr. Anthony Fauci, the government's top infectious disease specialist, and oral argume
nts before the Supreme Court about Trump's attempt to keep his financial information and tax returns from Congress and New York prosecutors.
> In a preview of his likely testimony, Fauci told The New York Times in an email on Monday night: 'If we skip over the checkpoints in the guid
elines to: 'Open America Again,' then we risk the danger of multiple outbreaks throughout the country.''This will not only result in needless s
uffering and death, but would actually set us back on our quest to return to normal,' Fauci continued.
> Trump on Monday accused Democratic-led states of deliberately slowing openings of their economies to hurt him politically in an election year
 and is ignoring warnings of global health experts that reopening societies will spark new spikes in infections without the massive testing and
 tracing program that he has failed to put in place."
Mainstream
Certainty : 62.0%
```

Figure 16 - Screenshot showing ouput of command line terminal, when tested with article from CNN

Feature Significance

One of the benefits of using a decision tree classifier, is the ability to view the tree structure of the classifier to understand the decision process of the classifier. Using the GraphViz[14] module from SK-Learn, the decision tree can be automatically generated. In order to investigate which features were the best at dividing the data by partisanship, a tutorial[15] from SK-Learn's user guide , which was designed to output the decision tree structure in a text based way. The program was updated to calculate the difference in the Gini Impurity(the probability a random article in the set would be incorrectly classified) between each parent and child node in the decision tree. The change in Gini impurity is then divided by how many nodes are behind the child node in the tree, as a feature lower down in the decision tree will have less impact on the split of the data. The data is then exported in a CSV file format, so that the results could be easily viewed in another program (Microsoft Excel).

[14] https://scikit-learn.org/stable/modules/generated/sklearn.tree.export_graphviz.html
[15] https://scikit-learn.org/stable/auto_examples/tree/plot_unveil_tree_structure.html#sphx-glr-auto-examples-tree-plot-unveil-tree-structure-py

Results and Evaluation

Classifier Results

The initial focus of the experiments carried out was to produce a classifier with the highest possible accuracy score for differentiating between hyperpartisan and neutral news articles. Experiments were carried out using three different classifier types from SKLearn's machine learning library, a Decision Tree Classifier[16], Logistic Regression Classifier[17] and a Random Forest Classifier[18]. The data for the system's training/ testing was a collection of 645 news articles and their corresponding labels, given to participants of the 2019 International Workshop on Semantic Evaluation competition. Different features were inputted into each classifier type, in order to see which features were optimal for each type of classifier. Iteratively, all possible combinations of the nine features were tested, and if the classifier returned a higher accuracy score with those features than the features tested before, this was stored as the current best combination.

The most effective classifier was the Random Forest Classifier, which achieved a maximum accuracy score of **79.5%** using these features:

- Average Sentence Length of Article
- Average Word Length
- Sentiment Analysis
- TFIDF Vectorizer
- Reduced bias words list

An accuracy score of 79.5% signifies that 513 news articles out of the total 645 were classified correctly and 132 were classified incorrectly. The best accuracy score achieved by the decision tree classifier was 68.9%, using the features:

- Average Sentence Length of Article
- Average Word Length
- Sentiment Analysis
- TFIDF Vectorizer
- Percentage of all words in article classified as 'bias'
- Percentage of words with possible spelling mistakes

[16] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.htmlr
[17] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[18] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

The logistic regression classifier, positioned in the middle of the other two classifiers, reaching a best accuracy score of 75.0% using the following features:

- Sentiment Analysis
- TFIDF Vectorizer
- Full bias words list

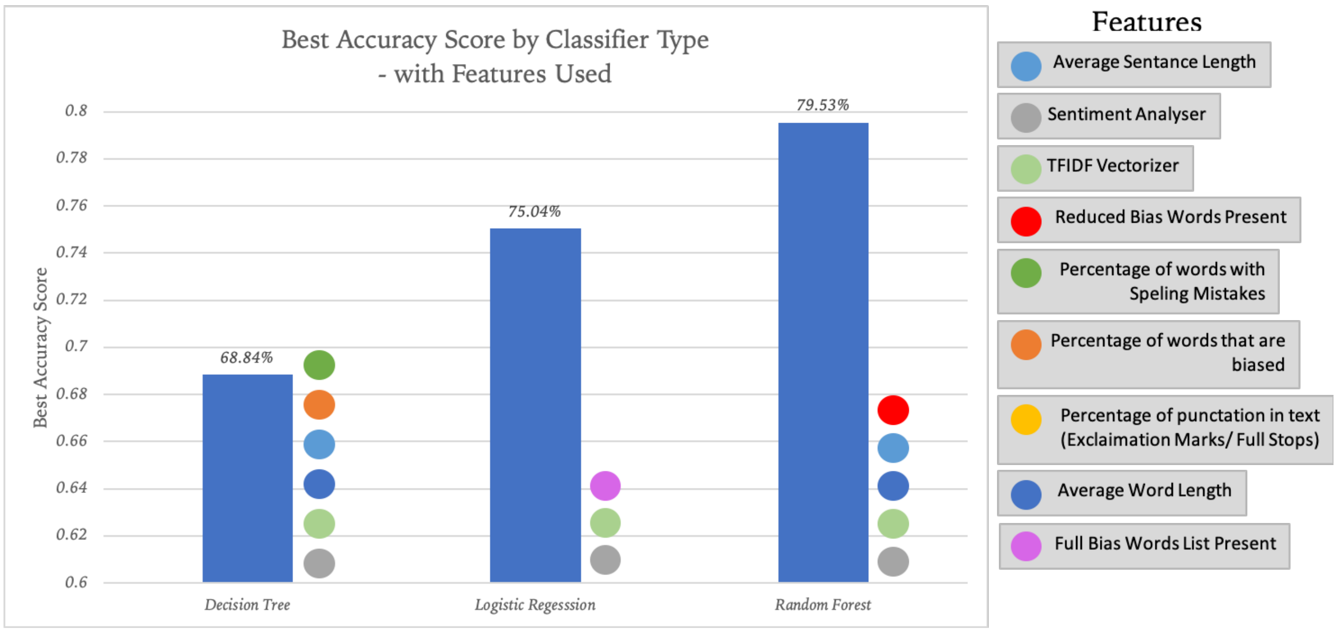


Figure 17 - Best accuracy score of each classifier type, with the features used to achieve the score

The precision of a classifier measures the percentage of positive results that were predicted correctly, compared to the amount of positives results that were predicated. A low precision score for the hyperpartisan class would mean that the classifier was predicating a high number of neutral articles as hyperpartisan incorrectly. The recall score of a classifier measures the percentage of positive results that were predicted correctly compared to the total of actual positive results. A low recall score for the hyperpartisan class would mean that the classifier was classing a lot of hyperpartisan articles as neutral ones.

The best random forest classifier 77.7% precise at classifying neutral articles, and 85.8% precise at classifying hyperpartisan ones, meaning the classifier was rarely classifying hyperpartisan articles as neutral ones, and vice versa. The recall for neutral news articles was 94.8%, and the recall for hyperpartisan articles was 53.4%; neutral articles were extremely unlikely to be predicted incorrectly, but hyperpartisan articles were predicted incorrectly around 47% of the time.

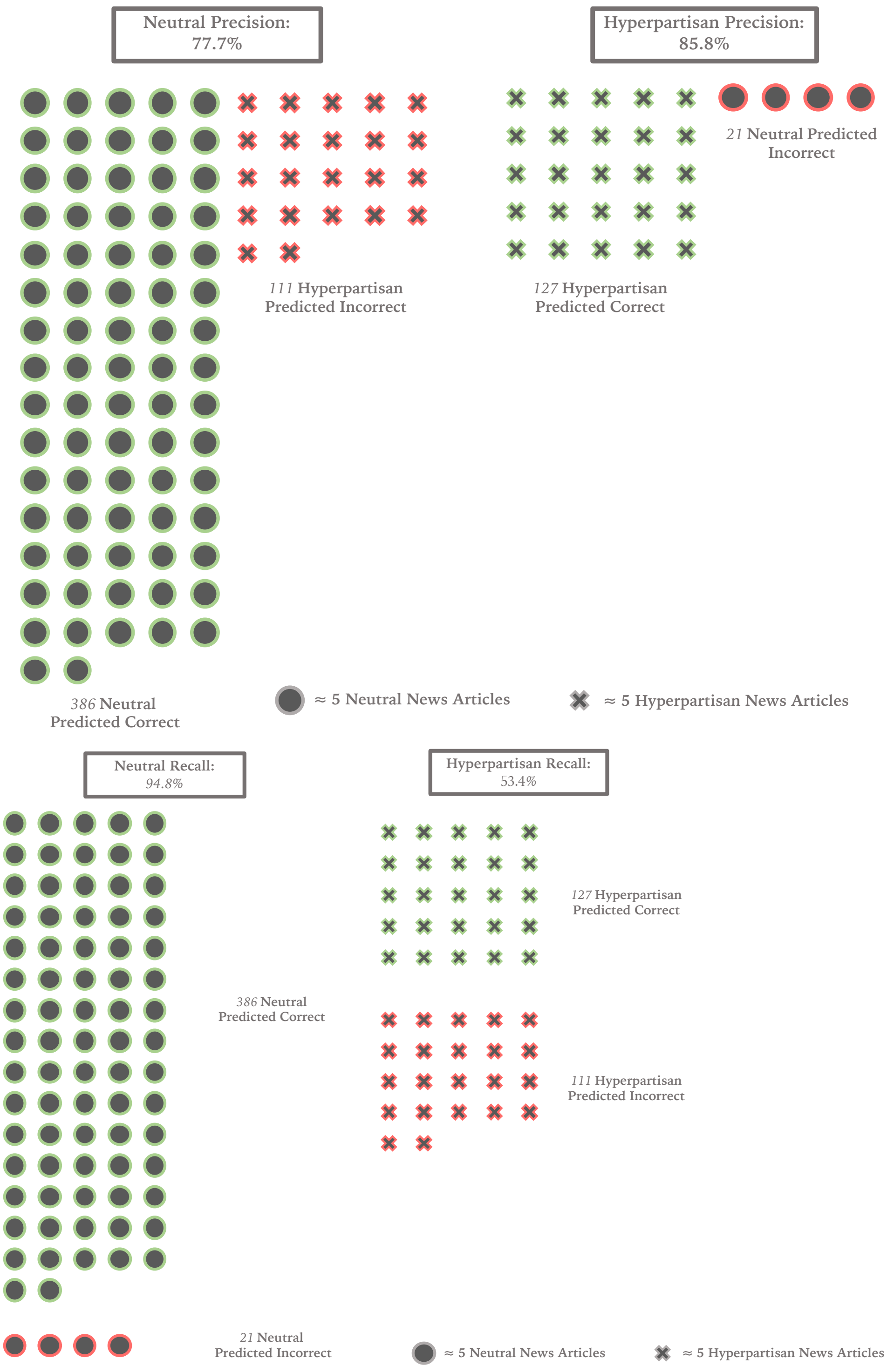


Figure 18 - Diagram showing precision and recall of the best random forest classifier

Feature Analysis

Next focus was on finding which features were the best for discerning hyperpartisan news articles. The first investigation was to test each feature individually on its own with each classifier type, to identify how well they were at categorising hyperpartisan news by themselves. The graphs below show each feature's accuracy score for each type of classifier.

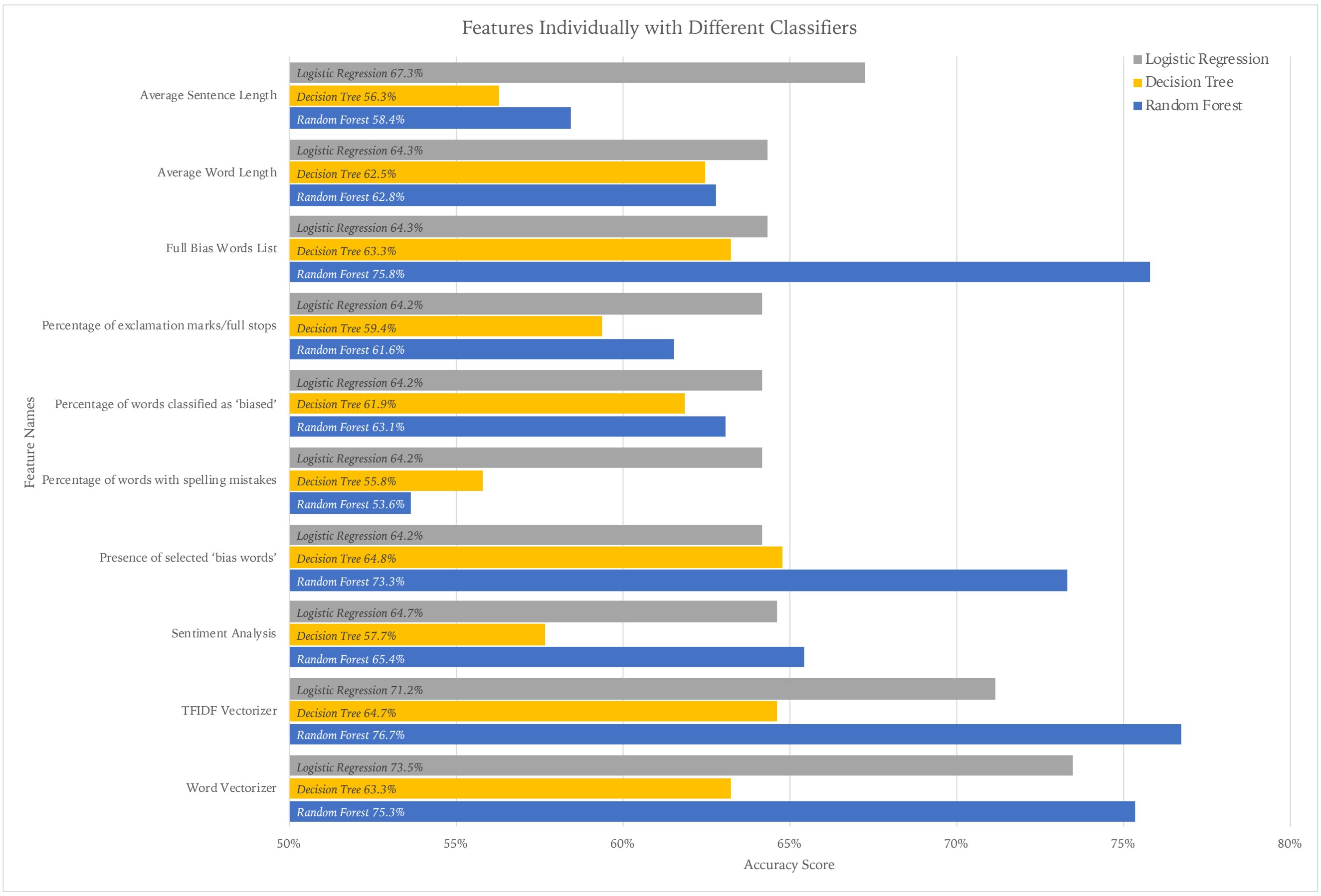


Figure 19 - Results of each feature tested on it's own with each classifier type

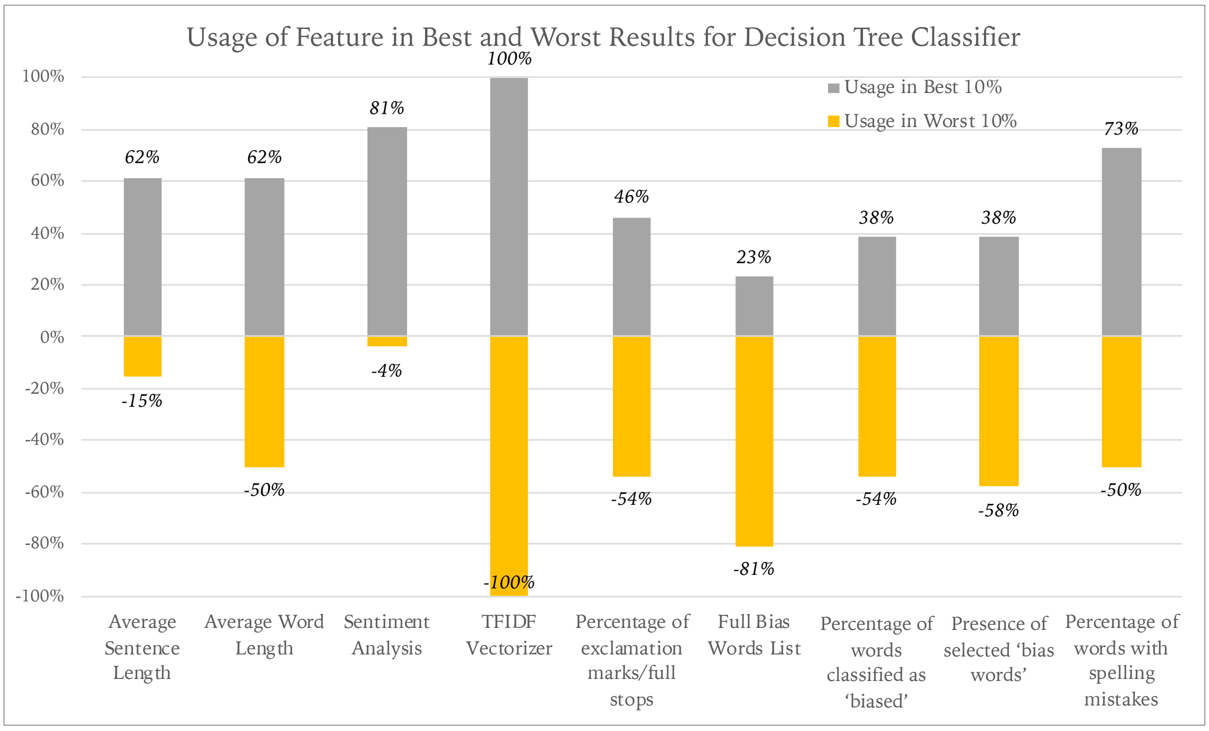
As the results show the TF-IDF vectorizer , on average, worked the best at splitting hyperpartisan news articles from neutral ones. The features which involved words from the bias lexicon performed particularly well with the random forest classifier, achieving close to 76% accuracy on its own. Another interesting performance is the feature that identifies the percentage of words with potential spelling mistakes, which performed around 10% better with the logistic regression classifier compared to the others. Given the nature of how random forest classifiers work (creating lots of trees from a subset of features), it is not unsurprising that features that only returned one data point (i.e average sentence length) did not perform as well for this classifier, as there was not enough data to divide into subsets.

As the TFIDF vectorizer had the most effective performance, and highest average accuracy score across all the features, to reduce the number of iterations required in the next experiment, the tests were carried out using the TFIDF vectorizer as a base level to identify which features could work with it to improve the accuracy score.

Different classifiers utilised different features for their optimal results. In order to see which features contributed the most to achieving a high accuracy score, the amount of times they were used in a classifiers best and worst results were calculated. If a feature is consistently used in the best results for a classifier and conversely never used in its worst, then the feature must be contributing to its high accuracy score.

The decision tree classifier used the sentiment analysis and the average sentence length features most commonly in its best performing accuracy scores, with sentiment analysis appearing in 81% of its best scores and average sentence length appearing in 62%. The same features only were used in 4% (sentiment analysis) and 15% (average sentence length) of the worst results for the feature.
When testing a logistic regression classifier, only the sentiment analysis feature worked extremely well to contribute to a high accuracy score along side the TFIDF vectorizer, being utilised in 100% of the best results for the classifier and 0% of the classifiers worst accuracy scores. The pared-back list of words from the bias lexicon also appeared in 100% of the random forest classifier's best scores and 38% of the classifiers worst. The average sentence length appeared in 69% of the classifiers best results and, similarly, appeared in 38% of the worst accuracy scores. Sentiment analysis generally also appeared to benefit the random forest classifier, with a net difference of 19% between the best and worst accuracy scores where it was used.

Decision Tree Classifier



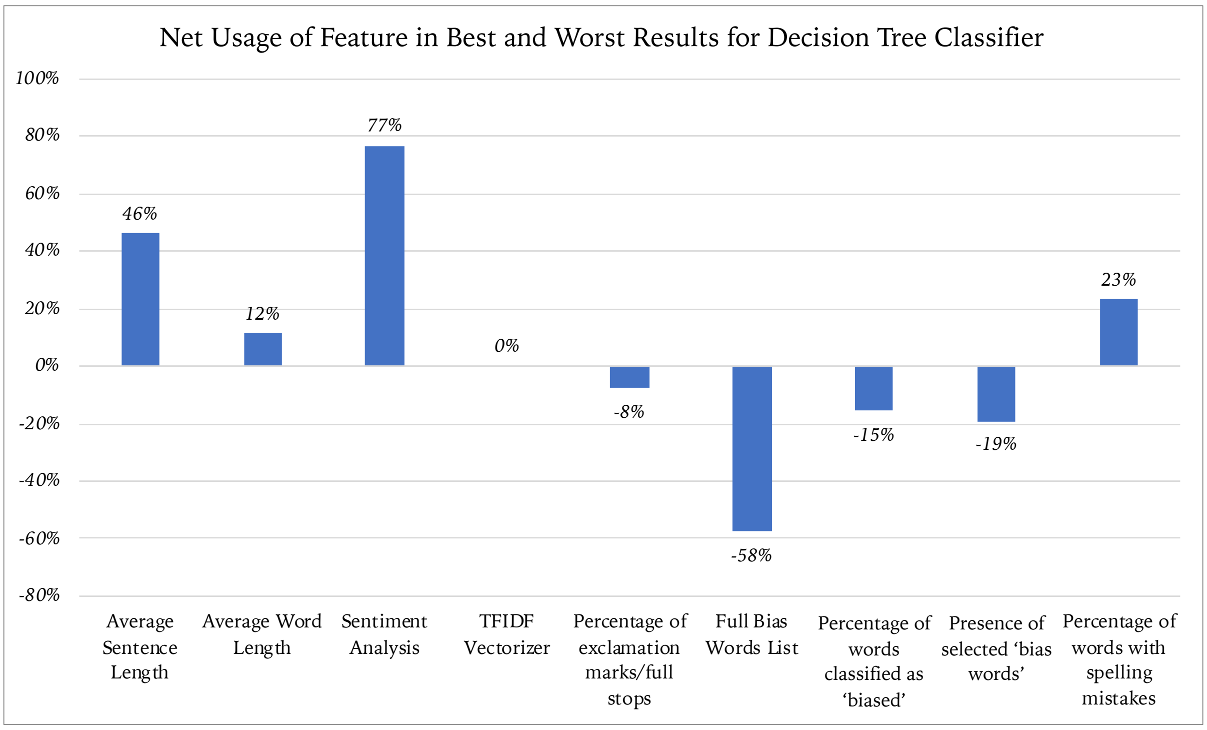
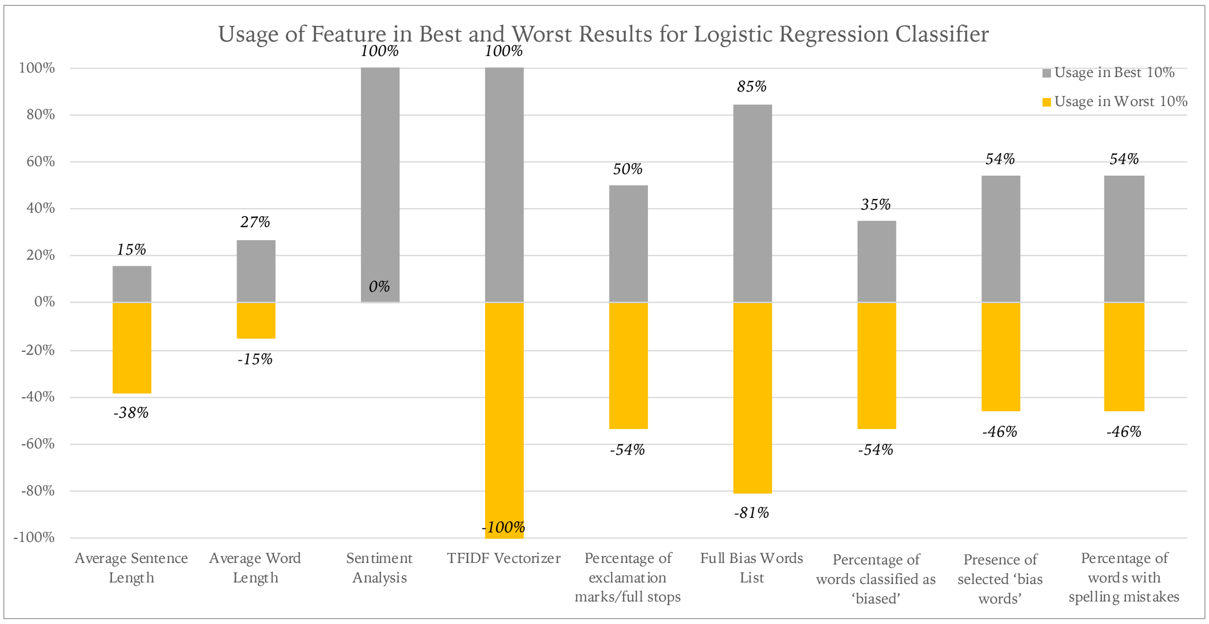


Figure 20 - Usage of feature in decision tree classifier's best and worst results. The TF-IDF vectorizer was used in every result, so is therefore in 100% of the worst and best results.

Logistic Regression Classifier



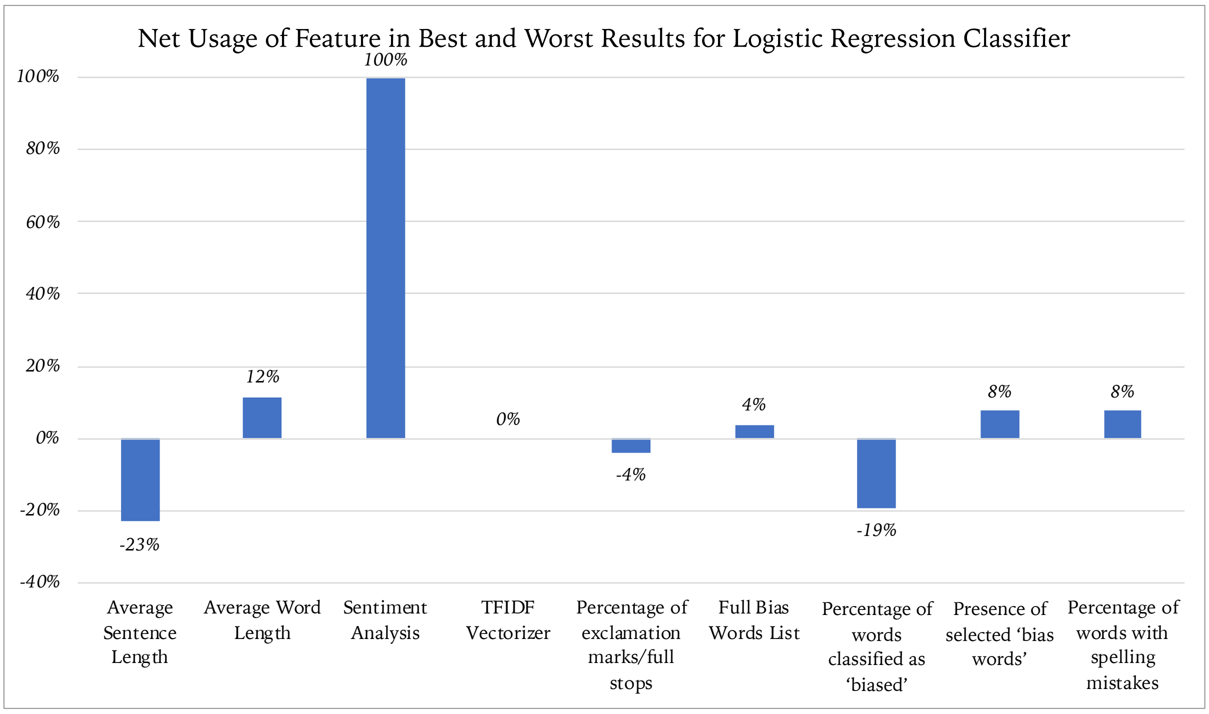
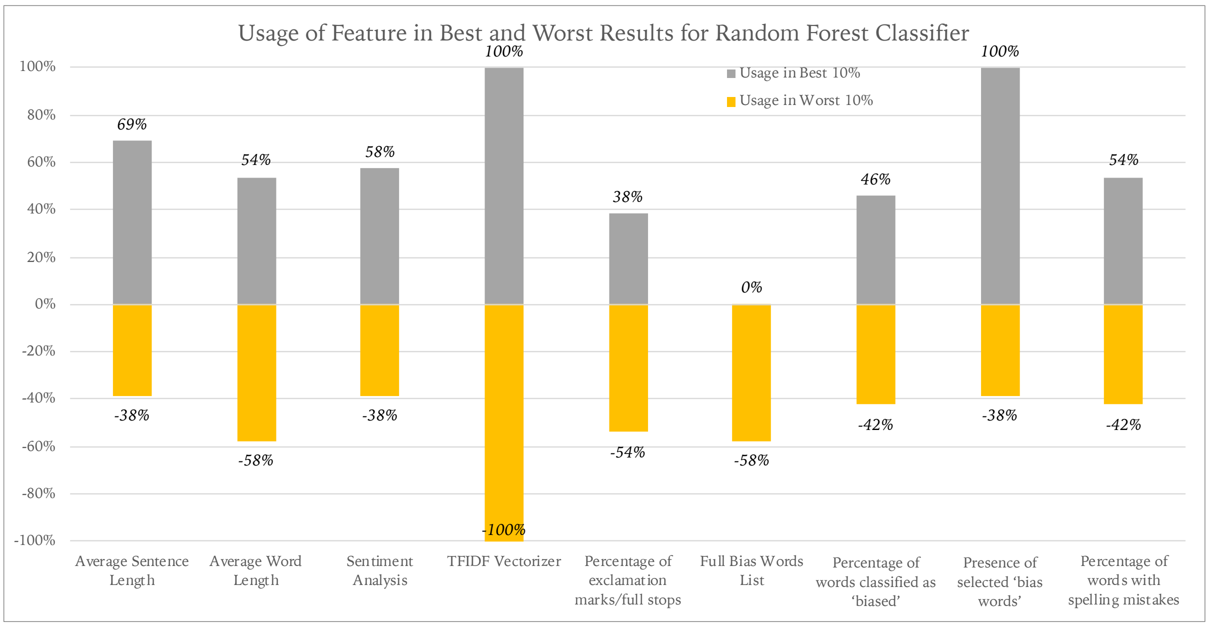


Figure 21 Usage of feature in logistic regression classifier's best and worst results. The TF-IDF vectorizer was used in every result, so is therefore in 100% of the worst and best results.

Random Forest Classifier



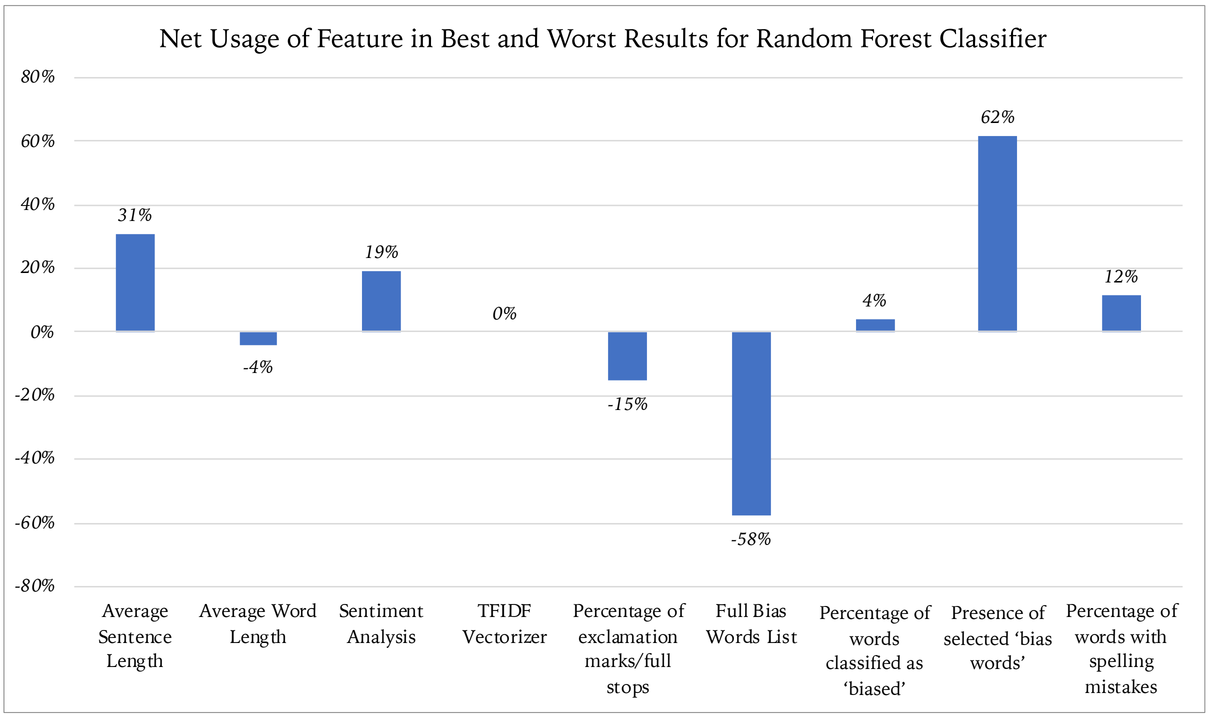


Figure 22 - Usage of feature in random forest classifier's best and worst results. The TF-IDF vectorizer was used in every result, so is therefore in 100% of the worst and best results.

Feature Importance in Random Forest

When generating a random forest classifier, the SK Learn library enables the data output of an importance score for each feature, signifying the usefulness of the feature to the overall classification. The importance scores were then ranked to see which features were the most useful to the random forest classifier.

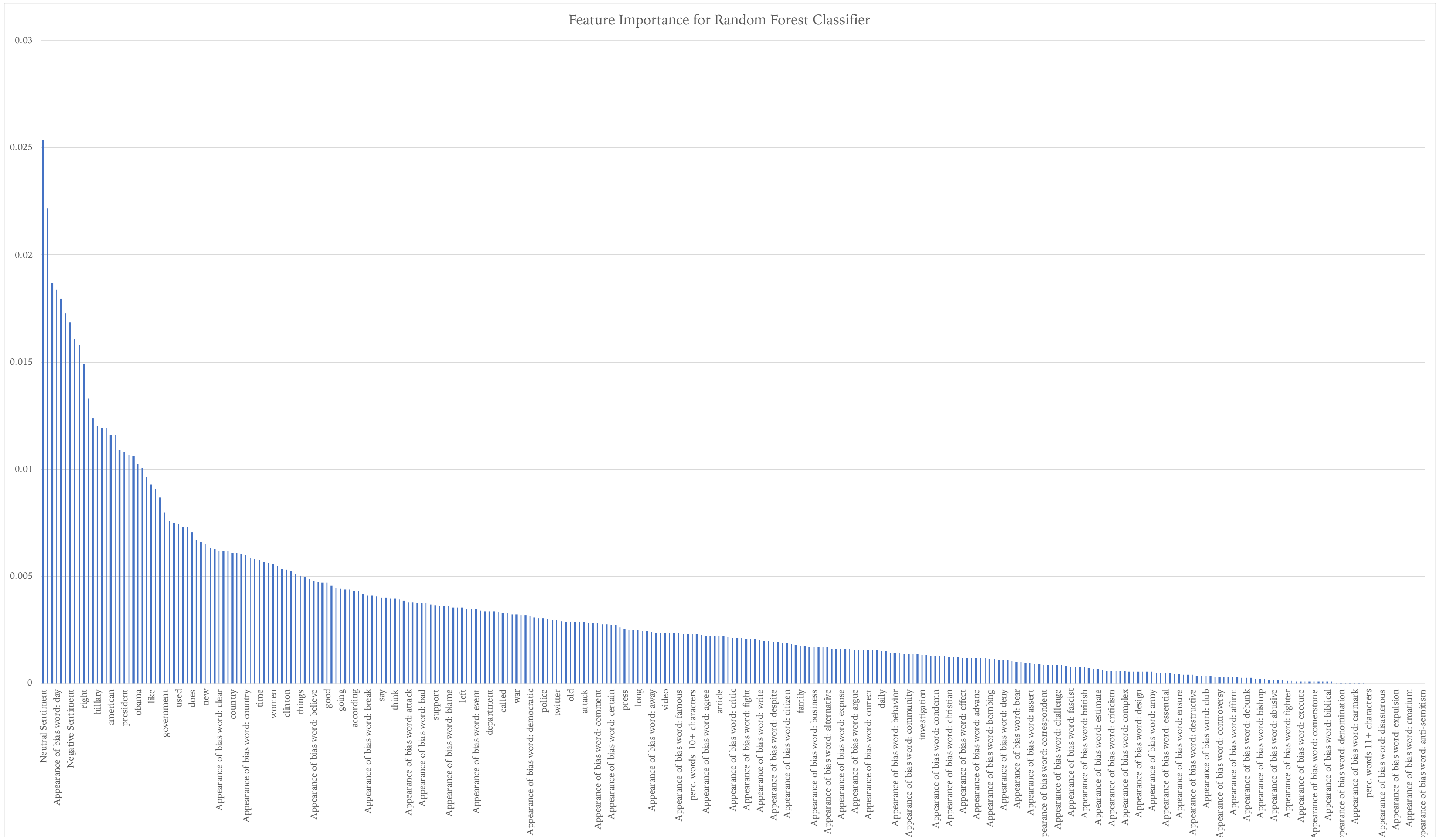


Figure 23- Feature importance for random forest classifier

Given there were around 300 individual features used in the best random forest classifier, there is some quite significant differences in the importance of all the features. On average, the top 30 features were 5 times more important than the next 278.

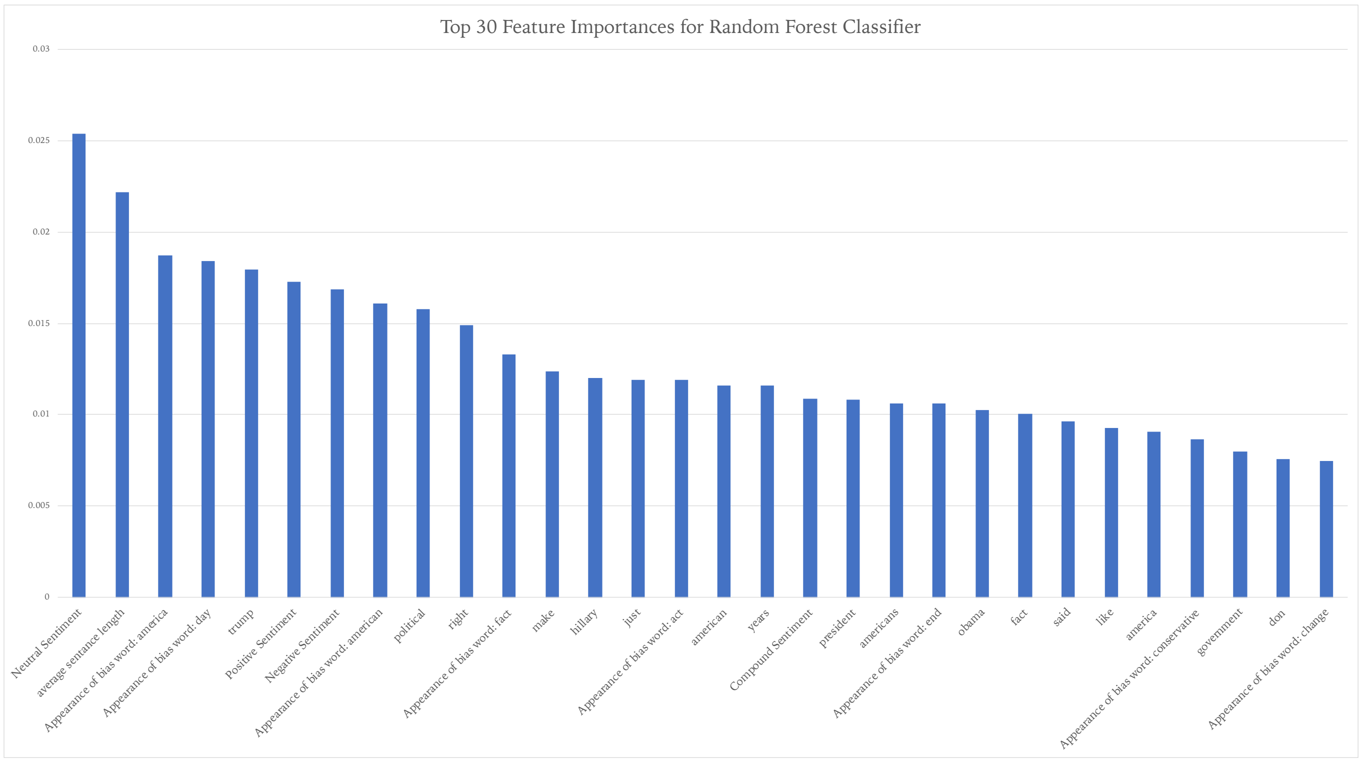


Figure 24 – Top 30 feature importancse for random forest classifier

The most important feature to the random forest classifier was the percentage of neutral sentiment the news article contained. All the sentiment analysis features (positive sentiment, negative sentiment, neutral sentiment and compound sentiment) ranked in the top 20 features for the classifier. The average sentence length of an article was the second most important feature to the classifier. When analysing features that looked at the pervasiveness of certain words, features with some variation on the word America ('America'/ 'Americans'/ 'American'), again all performed well, with the use of the word America being the third ranking feature in the classifier. This perhaps reflects the nature of the 2016 Presidential Election, with Republican candidate Donald Trump pushing for a more isolationist foreign policy to his predecessors (Schneider 2019). The name- checking of political figures relevant to the election also seemed to be an effective indicator for the classifier, with words such as 'Trump', 'Hillary', and 'Obama' all featuring near the top of the ranking.

Decision Tree Visualisation

The decision tree classifier brings the benefit that the classifier can be extracted and visualised as a flow chart showing how the decisions for a classification are made. The decision tree for classifier with the highest accuracy score is shown below.

Highest accuracy scoring decision tree

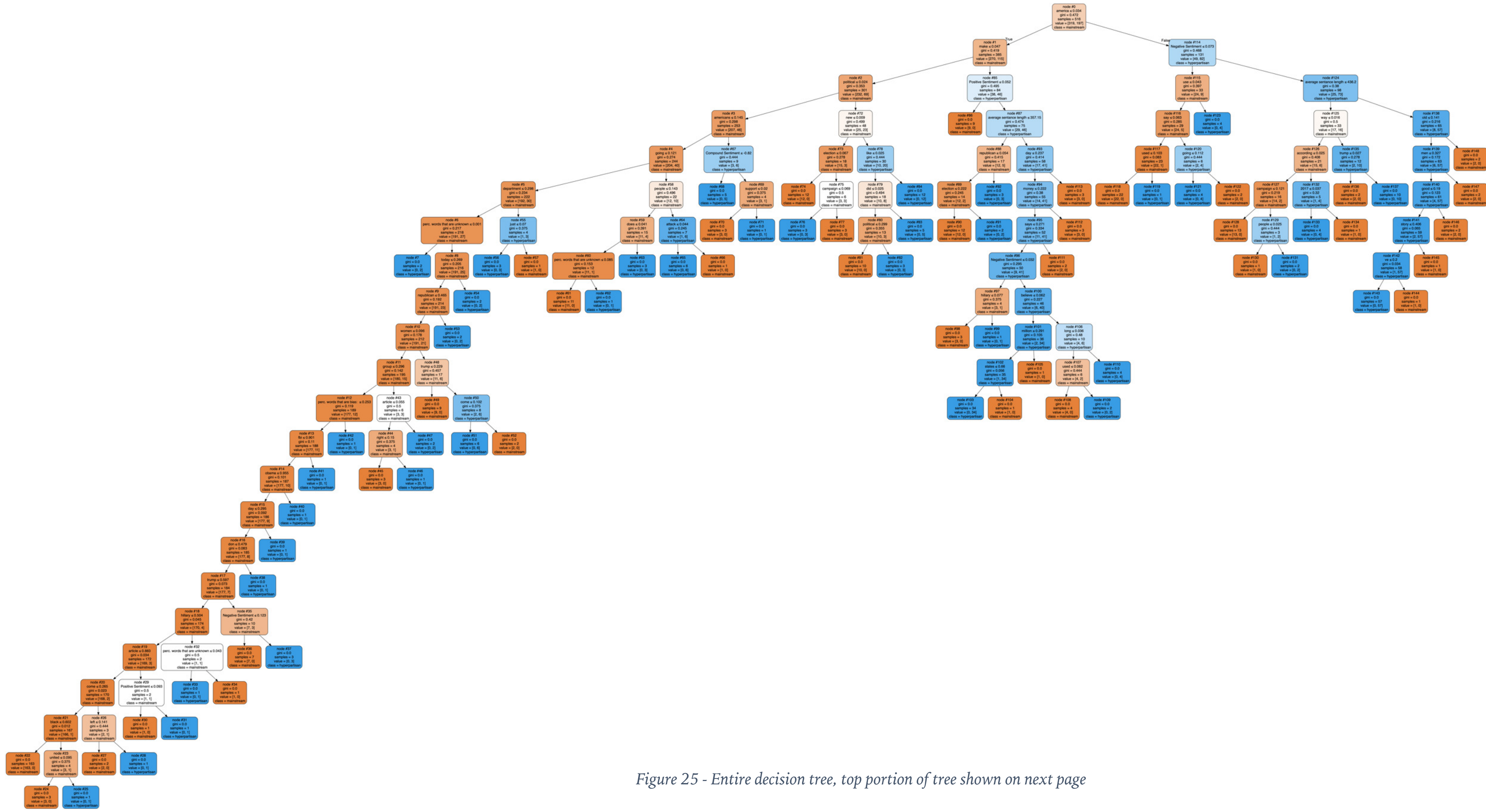


Figure 25 - Entire decision tree, top portion of tree shown on next page

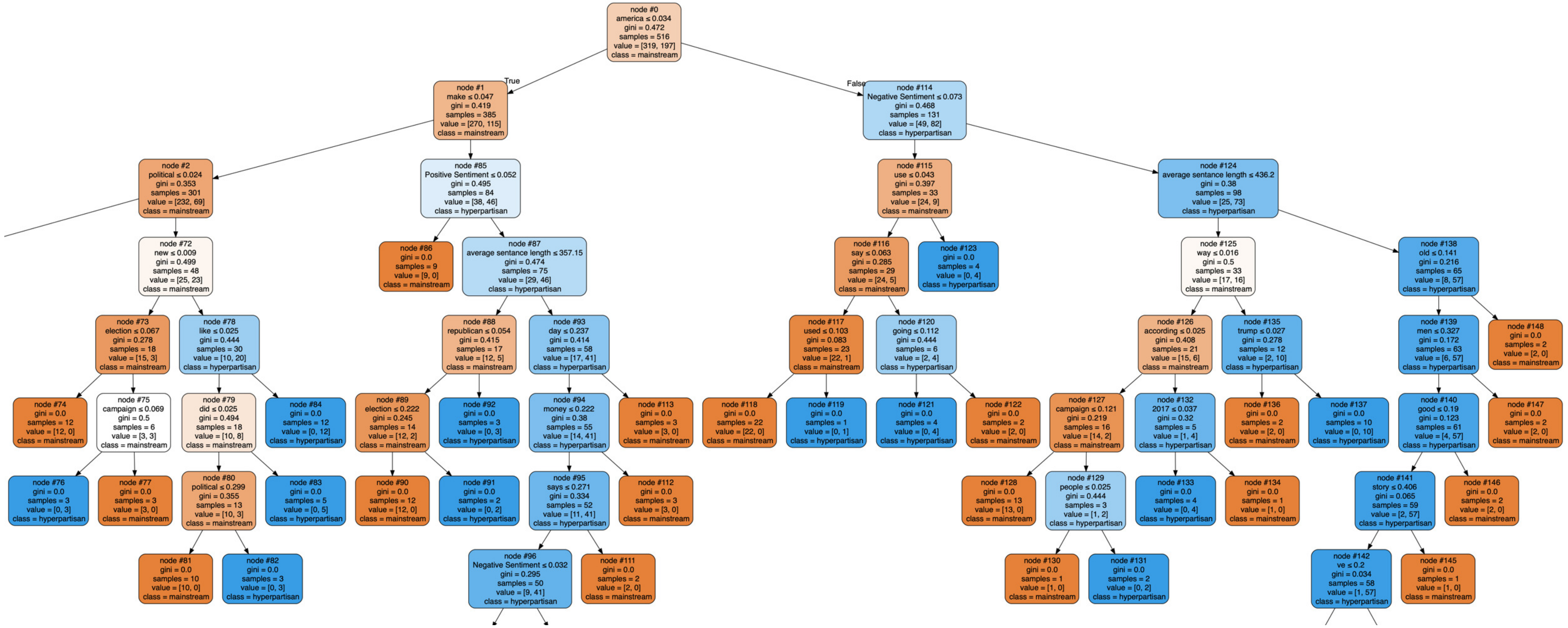

node #0 america ≤ 0.034 gini = 0.472 samples = 516 value = [319, 197] class = mainstream
True
False
node #1 make ≤ 0.047 gini = 0.419 samples = 385 value = [270, 115] class = mainstream
node #114 Negative Sentiment ≤ 0.073 gini = 0.468 samples = 131 value = [49, 82] class = hyperpartisan
node #2 political ≤ 0.024 gini = 0.353 samples = 301 value = [232, 69] class = mainstream
node #85 Positive Sentiment ≤ 0.052 gini = 0.495 samples = 84 value = [38, 46] class = hyperpartisan
node #115 use ≤ 0.043 gini = 0.397 samples = 33 value = [24, 9] class = mainstream
node #124 average sentance length ≤ 436.2 gini = 0.38 samples = 98 value = [25, 73] class = hyperpartisan
node #72 new ≤ 0.009 gini = 0.499 samples = 48 value = [25, 23] class = mainstream
node #86 gini = 0.0 samples = 9 value = [9, 0] class = mainstream
node #87 average sentance length ≤ 357.15 gini = 0.474 samples = 75 value = [29, 46] class = hyperpartisan
node #116 say ≤ 0.063 gini = 0.285 samples = 29 value = [24, 5] class = mainstream
node #123 gini = 0.0 samples = 4 value = [0, 4] class = hyperpartisan
node #125 way ≤ 0.016 gini = 0.5 samples = 33 value = [17, 16] class = mainstream
node #138 old ≤ 0.141 gini = 0.216 samples = 65 value = [8, 57] class = hyperpartisan
node #73 election ≤ 0.067 gini = 0.278 samples = 18 value = [15, 3] class = mainstream
node #78 like ≤ 0.025 gini = 0.444 samples = 30 value = [10, 20] class = hyperpartisan
node #88 republican ≤ 0.054 gini = 0.415 samples = 17 value = [12, 5] class = mainstream
node #93 day ≤ 0.237 gini = 0.414 samples = 58 value = [17, 41] class = hyperpartisan
node #117 used ≤ 0.103 gini = 0.083 samples = 23 value = [22, 1] class = mainstream
node #120 going ≤ 0.112 gini = 0.444 samples = 6 value = [2, 4] class = hyperpartisan
node #126 according ≤ 0.025 gini = 0.408 samples = 21 value = [15, 6] class = mainstream
node #135 trump ≤ 0.027 gini = 0.278 samples = 12 value = [2, 10] class = hyperpartisan
node #139 men ≤ 0.327 gini = 0.172 samples = 63 value = [6, 57] class = hyperpartisan
node #148 gini = 0.0 samples = 2 value = [2, 0] class = mainstream
node #74 gini = 0.0 samples = 12 value = [12, 0] class = mainstream
node #75 campaign ≤ 0.069 gini = 0.5 samples = 6 value = [3, 3] class = mainstream
node #79 did ≤ 0.025 gini = 0.494 samples = 18 value = [10, 8] class = mainstream
node #84 gini = 0.0 samples = 12 value = [0, 12] class = hyperpartisan
node #89 election ≤ 0.222 gini = 0.245 samples = 14 value = [12, 2] class = mainstream
node #92 gini = 0.0 samples = 3 value = [0, 3] class = hyperpartisan
node #94 money ≤ 0.222 gini = 0.38 samples = 55 value = [14, 41] class = hyperpartisan
node #113 gini = 0.0 samples = 3 value = [3, 0] class = mainstream
node #118 gini = 0.0 samples = 22 value = [22, 0] class = mainstream
node #119 gini = 0.0 samples = 1 value = [0, 1] class = hyperpartisan
node #121 gini = 0.0 samples = 4 value = [0, 4] class = hyperpartisan
node #122 gini = 0.0 samples = 2 value = [2, 0] class = mainstream
node #127 campaign ≤ 0.121 gini = 0.219 samples = 16 value = [14, 2] class = mainstream
node #132 2017 ≤ 0.037 gini = 0.32 samples = 5 value = [1, 4] class = hyperpartisan
node #136 gini = 0.0 samples = 2 value = [2, 0] class = mainstream
node #137 gini = 0.0 samples = 10 value = [0, 10] class = hyperpartisan
node #140 good ≤ 0.19 gini = 0.123 samples = 61 value = [4, 57] class = hyperpartisan
node #147 gini = 0.0 samples = 2 value = [2, 0] class = mainstream
node #76 gini = 0.0 samples = 3 value = [0, 3] class = hyperpartisan
node #77 gini = 0.0 samples = 3 value = [3, 0] class = mainstream
node #80 political ≤ 0.299 gini = 0.355 samples = 13 value = [10, 3] class = mainstream
node #83 gini = 0.0 samples = 5 value = [0, 5] class = hyperpartisan
node #90 gini = 0.0 samples = 12 value = [12, 0] class = mainstream
node #91 gini = 0.0 samples = 2 value = [0, 2] class = hyperpartisan
node #95 says ≤ 0.271 gini = 0.334 samples = 52 value = [11, 41] class = hyperpartisan
node #112 gini = 0.0 samples = 3 value = [3, 0] class = mainstream
node #128 gini = 0.0 samples = 13 value = [13, 0] class = mainstream
node #129 people ≤ 0.025 gini = 0.444 samples = 3 value = [1, 2] class = hyperpartisan
node #133 gini = 0.0 samples = 4 value = [0, 4] class = hyperpartisan
node #134 gini = 0.0 samples = 1 value = [1, 0] class = mainstream
node #141 story ≤ 0.406 gini = 0.065 samples = 59 value = [2, 57] class = hyperpartisan
node #146 gini = 0.0 samples = 2 value = [2, 0] class = mainstream
node #81 gini = 0.0 samples = 10 value = [10, 0] class = mainstream
node #82 gini = 0.0 samples = 3 value = [0, 3] class = hyperpartisan
node #96 Negative Sentiment ≤ 0.032 gini = 0.295 samples = 50 value = [9, 41] class = hyperpartisan
node #111 gini = 0.0 samples = 2 value = [2, 0] class = mainstream
node #130 gini = 0.0 samples = 1 value = [1, 0] class = mainstream
node #131 gini = 0.0 samples = 2 value = [0, 2] class = hyperpartisan
node #142 ve ≤ 0.2 gini = 0.034 samples = 58 value = [1, 57] class = hyperpartisan
node #145 gini = 0.0 samples = 1 value = [1, 0] class = mainstream

Figure 26 - Top portion of decision tree

The tree shows that the best feature at splitting the entire dataset of news articles was the commonality of the word 'America', with articles with a higher TF-IDF value for the word 'America' more likely to be hyperpartisan. Node #1 checks for the TF-IDF value for the word 'make', with articles with higher values more likely to be hyperpartisan. This appears to be checking if the article for the phrase 'Make America Great Again', especially due to the next node checking for positive sentiment, possibly as this would be boosted by the word 'great'.

Ranking change in Gini Impurity

A similar ranking to the random forest classifiers feature importance was extracted from the decision tree classifier, however extracting this information from the decision tree has the added benefit that you can see how each feature benefits the classification towards hyper-partisanship or not. In order to investigate this, the GINI impurity (probability a randomly picked article is wrongly classified) change in from a parent node to a child was measured, and then divided by how far down the tree the child node was. The metric calculated balanced the fact that nodes higher up the tree have less features splitting them, so therefore any change in impurity is more dependent on that specific feature.

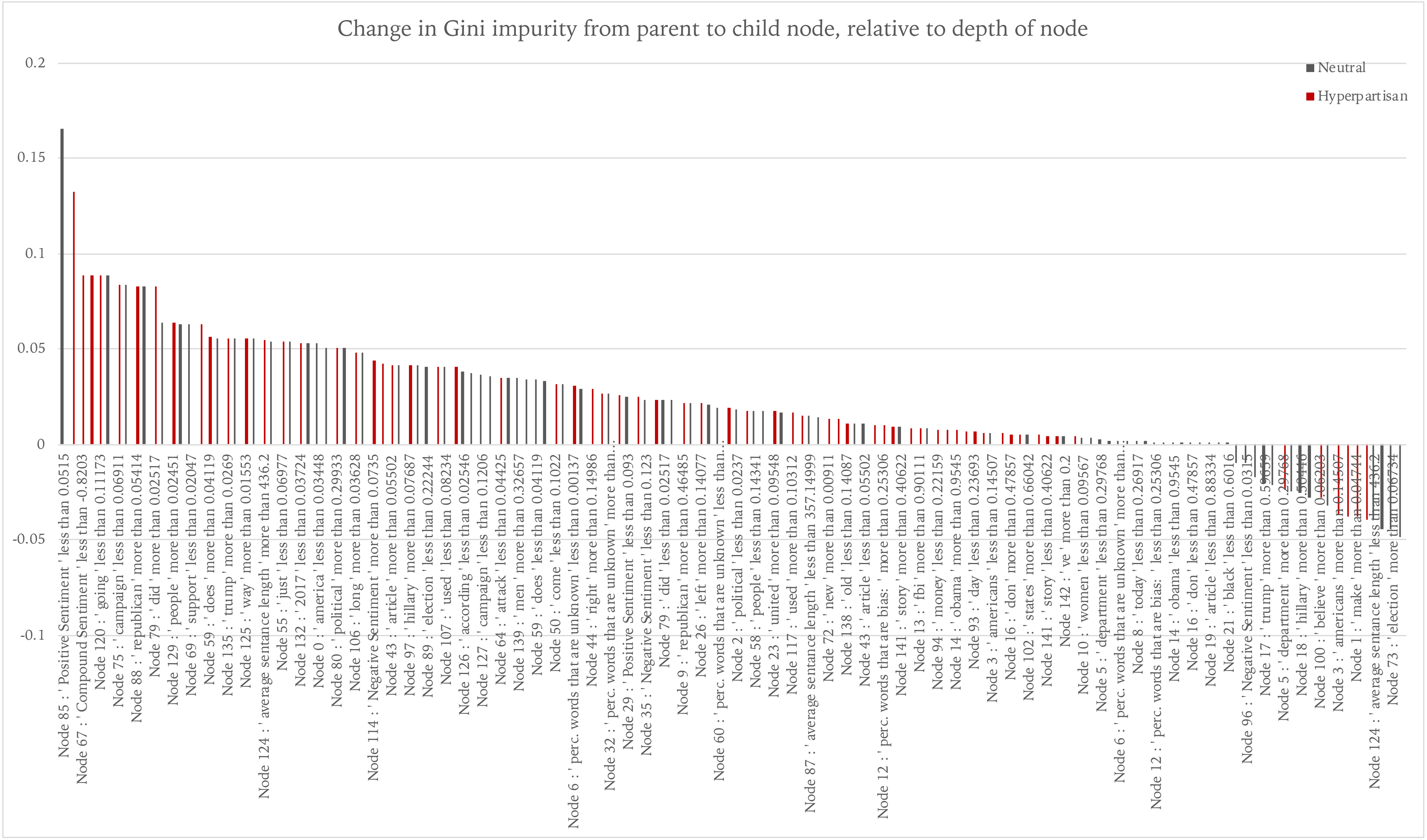


Figure 27 - Graph of change in Gini impurity, relative to the depth of the node

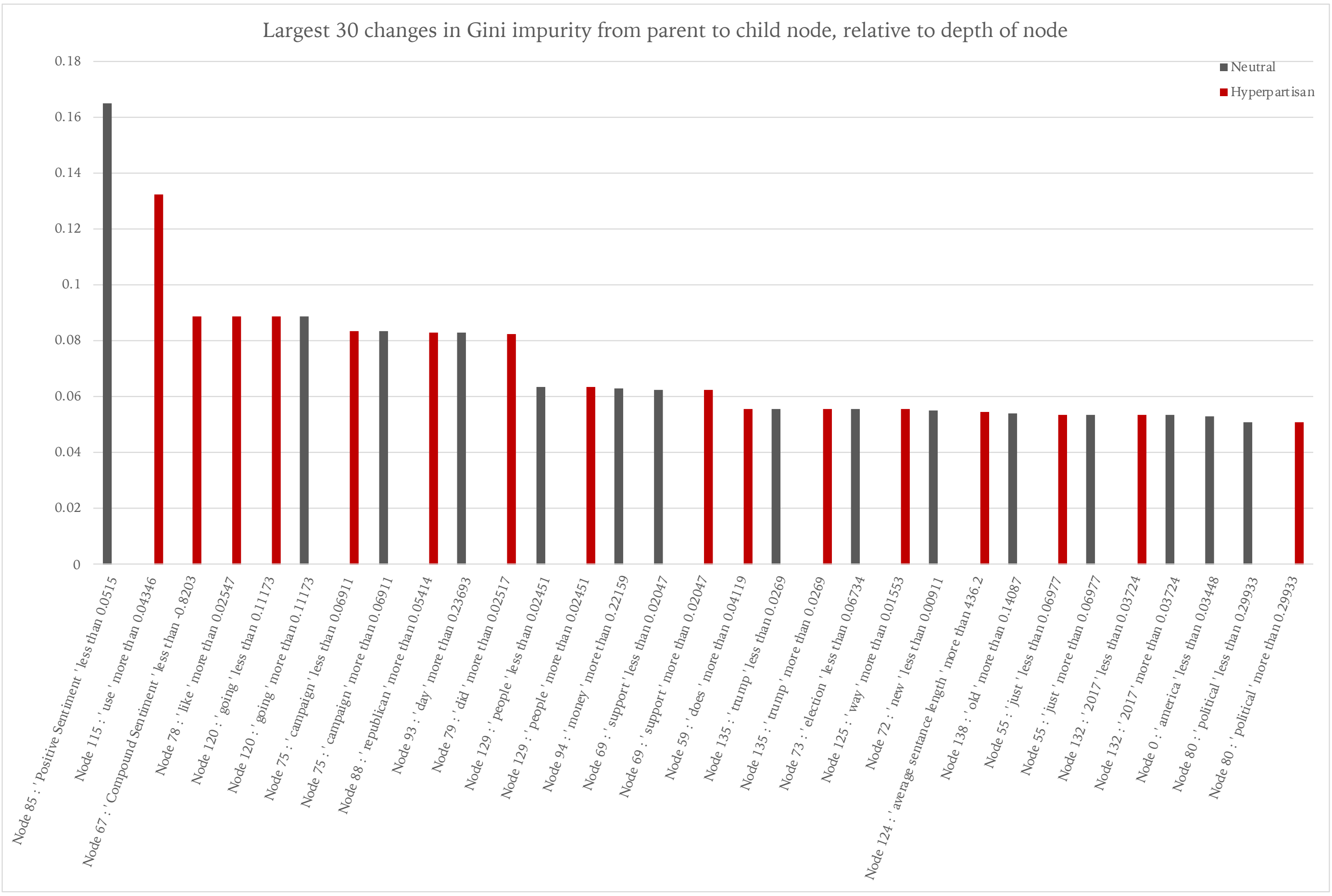


Figure 28 - Top 30 changes in Gini impurity, relative to the depth

The greatest change in Gini, relative to the depth of the node, is the amount of positive sentiment in node #85, again possible due to the phrase 'Make America Great Again' being a good indicator of the article being hyperpartisan. Generally, the sentiment performed well in splitting the articles, appearing twice in the top three changes. Many of the jumps appear twice on the graph, such as node #135 for the TF-IDF value of the word 'trump'. This is because the node splits into two leaf nodes, and therefore the difference in Gini change between is identical as the Gini impurity for leaf nodes is zero. Node #135 shows that a higher TF-IDF value for the word 'trump' causes a hyperpartisan prediction, and #75 shows a higher value for the word 'campaign' will cause a neutral prediction. The data in this chart is useful in identifying some interesting splits, however due to the relatively small amount of articles, some high changes in Gini impurity are just due to some chance characteristics in the data.

Summary

Overall, the system works moderately effectively for classifying news articles, although there are some caveats to it's performance. The best accuracy score the system was able to accomplish was 79.5%, so can effectively detect 4 out of every 5 articles correctly. When compared to teams who completed the Semantic Evaluation Workshop (SemEval) challenge in 2019, this system compares favourably- out of the 42 submissions this system would have ranked 6th for the accuracy score that it achieved. The highest accuracy score achieved in the competition was 82.2%, so this system is 2.7% less accurate than the best submission. Similarly, the precision of the classifier at detecting hyperpartisan news articles performs comfortably compared to the other submissions with a performance of 85.8%, with the best score from the competition being 88.3%. The classifier has a low recall score however, meaning that there were too many false negative results for hyperpartisan news articles. Again, comparing to other teams in the SemEval - 2019 challenge, it seemed a common among high accuracy scoring entries to struggle with lower recall values.

The original accuracy of the random forest classifier with just the TF-IDF vectorizer was 76.7%, so the custom features that were created for the classifier were effective in boosting the score by around 3%, which is satisfactory given the trickiness in increasing accuracy scores for classifier the closer to 100% accuracy they get. Furthermore, compared to the initial version of the system which used a decision tree classifier and word vectorizer its only feature (which was 63.3% accurate), the development has been successful.

Across all the classifiers, sentiment analysis worked the best in combination with a TF-IDF vectorizer. The net usage of the feature in the classifiers best scores were 77% for the decision tree classifier, 100% for the logistic regression classifier, and 19% for the random forest classifier, meaning that the feature consistently helped the classifiers reach the best accuracy scores. Furthermore, neutral sentiment ranked as the most important feature in the random forest classifier's feature importance, with all four sentiment analysis features ranking in the top 20 features for the classifier.

Future Work

If given more time to further develop the system, the initial step to take would be to source more news articles that can be included to further train and test the system in order to improve its performance. The current 645 news articles that were used to test/ train the current system were provided to competitors in the SemEval 2019, and were all manually labelled as ‘hyperpartisan’ or ‘mainstream’ (neutral) by factcheckers (Kiesel et al. 2019). To enhance the dataset for the classifier, a task could be undertaken to further manually label news articles using similar labelling criteria to those in the SemEval set. Another option would be to use a by-publisher dataset, where instead of manually checking each news article, you label an entire publisher as hyperpartisan or neutral by checking a subsection of their publications. Organizations such as Buzzfeed News have already compiled lists[19] of hyperpartisan news outlets that could be utilised. Once a publisher has been identified, you can then as suggested by (Kiesel et al. 2019) , use an automated program to trawl through the publishers output to gather news articles to train the classifier with. Manually labelling each news article would be a very time consuming task, however the data produced would be high quality and the labelling guaranteed to be accurate. In contrast, using by publisher data would be much faster, however there is no guarantee that every news article published by a hyperpartisan publisher is hyperpartisan, as shown in the table of Figure 1. Therefore there is some risk that the classifier will be confused by classifying incorrectly labelled articles. Increasing the number of hyperpartisan news articles to train with would be especially useful, as there were about 70% more articles from neutral mainstream publishers to train the classifier with than from hyperpartisan publishers.

Another component that could be added to improve the system would be to create an application programming interface (API), which would allow another service to send news articles to the system and receive in return a classification for the articles and the probability of the classification. Given that a tool for a command line input has already been created, the API could be an extension of this programming. Creating this API system would allow publishers of news articles, such as social media companies, almost instantaneously receive prediction for any new news article that was published to their website, meaning that audiences could receive a warning that the article may be hyperpartisan as soon as it is published, and before it was read and

[19] https://github.com/BuzzFeedNews/2017-08-partisan-sites-and-facebook-pages

shared to others. Another benefit of creating an API would be that the system would receive a lot of new articles that could be used to train the classifier. A procedure could be implemented where articles coming into the API could be manually labelled, perhaps by some crowdsourcing method where the people could submit their verdict on the articles partisanship and the majority vote would be applied. (Raykar and Yu 2012) suggests a method for collecting data labels by crowdsourcing, while avoiding spammers that could degrade the data quality.

Conclusion

The project successfully met the aim of creating a classification system for predicting the probability that a news article is hyperpartisan or neutral. The classifier created was just under 80% accurate, which was very competitive compared to other similar solutions created by teams at International Workshop on Semantic Evaluation 2019. The low recall score of the classifier is a slight obstruction to the total success of this aim, however given more time and more information to train with, the same system would likely improve in its effectiveness.

There was also the ambition to investigate which features performed most effectively in the classification of hyperpartisan news, which was accomplished with sentiment analysis showing as a key feature that worked across all the classifiers tested.

Furthermore, the creation of functionality to predict the classification of a string object from the command line, opens up the possibility for the system to be used in an API or as part of some other software package that would require hyperpartisan news detection.

Reflection on Learning

Doing this project has developed my understanding of machine learning and the theory behind. Coming into this project, I had little experience using a library like SK-Learn to program a machine learning system, so being able to actually implement a system has meant that my understanding of how the theory behind machine learning works has developed too. Having to deal with some of the convoluted tasks that are involved in creating real life systems, such as processing data from a text file, has given me more solid understanding of the time requirements for developing systems, often the most menial task can take the longest.

Analysing the performance of different combinations of classifiers and features has improved my ability to perform statistical analysis of data, as a fair amount of the project was spent exporting data from the machine learning models to Microsoft Excel to produce graphs and try and identify and understand trends.

Given the different variables in the classification system that can be experimented with, I spent a lot of time trying out different approaches trying to find more optimal results. However, the nature of the classifier is that there will not ever be one optimal solution and therefore it is possible to experiment with the system perpetually. With hindsight, I should have focused on only allowing experimenting within the time frames allowed in the initial Gantt chart created at the beginning of the project.

References

Association for Computational Linguistics 2019. SemEval 2020: Call For Participating In SemEval 2020 Tasks | ACL Member Portal. Available at: https://www.aclweb.org/portal/content/semeval-2020-call-participating-semeval-2020-tasks [Accessed: 14 May 2020].

Bakir, V. and McStay, A. [no date][a]. Fake News and the Economy of Emotion. Available at: https://research.bangor.ac.uk/portal/files/19296816/2017_Fake_news.pdf.

Bakir, V. and McStay, A. [no date][b]. Fake News and The Economy of Emotions., pp. 154–175. doi: DOI: 10.1080/21670811.2017.1345645.

Bird, S. et al. 2009. Natural Language Processing with Python. O'Reilly Media Inc.
Data Driven Investor 2018. K-Fold Cross Validation. Available at: https://medium.com/datadriveninvestor/k-fold-cross-validation-6b8518070833 [Accessed: 14 May 2020].

FAQs – Full Fact. [no date]. Full Fact . Available at: https://fullfact.org/about/frequently-asked-questions/.

Foreign & Commonwealth Office 2019. UK Steps Up Fight Against Fake News. Available at: https://www.gov.uk/government/news/uk-steps-up-fight-against-fake-news [Accessed: 14 May 2020].

Géron, A. 2017. Hands-On Machine Learning with Scikit-Learn and TensorFlow., p. 169.
Hutto, C. and Gilbert, E. 2014. VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text.

Kiesel, J. et al. 2019. SemEval-2019 Task 4: Hyperpartisan News Detection. Available at: 10.18653/v1/s19-2145 [Accessed: 14 May 2020].

Kirk, M. 2017. Thoughtful Machine Learning with Python., p. 75.

Koppel, M. et al. 2007. Measuring Differentiability: Unmasking Pseudonymous Authors. Journal of Machine Learning Research 8, pp. 1261–1276.

Mosseri, A. 2017. Working To Stop Misinformation And False News. Available at: https://www.facebook.com/facebookmedia/blog/working-to-stop-misinformation-and-false-news.

Pedregosa, F. et al. 2011. Scikit-learn: Machine Learning In Python. Available at: http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html [Accessed: 14 May 2020].

Ponsford, D. 2019. UK Newspaper And Website Readership 2018: Latest Pamco Figures. Press Gazette . Available at: https://www.pressgazette.co.uk/uk-newspaper-and-website-readership-2018-pamco/.

Potthast, M. et al. 2020. A Stylometric Inquiry into Hyperpartisan and Fake News. Available at: 10.18653/v1/p18-1022.

Proceedings, B. [no date]. of the 51st Annual Meeting. In: of the Association for Computational Linguistics (Volume 1: Long Papers)

Raykar, V.C. and Yu, S. 2012. Eliminating Spammers and Ranking Annotators for Crowdsourced Labeling Tasks. 13, pp. 491–51. Available at: http://jmlr.org/papers/v13/raykar12a.html [Accessed: 14 May 2020].

Recasens, M. et al. 2013. Linguistic Models for Analyzing and Detecting Biased Language. Available at: https://www.aclweb.org/anthology/P13-1162/.

Schneider, B. 2019. Isolationism Creeps Back Over America, As The President Looks Out For Himself. Available at: https://thehill.com/opinion/international/468690-isolationism-creeps-back-over-america [Accessed: 14 May 2020].

Silverman, C. et al. 2016. Hyperpartisan Facebook Pages Are Publishing False And Misleading Information At An Alarming Rate. BuzzFeed News . Available at: https://www.buzzfeednews.com/article/craigsilverman/partisan-fb-pages-analysis.

Silverman, C. and al., et [no date]. Hyperpartisan Facebook Pages Are Publishing False And Misleading Information At An Alarming Rate. fb-pages-analysis: Buzzfeed News. Available at: https://www.buzzfeednews.com/article/craigsilverman/partisan-.

Sweney, M. 2020. Newspapers To Lose £50m In Online Ads As Firms Use Coronavirus 'blacklist'. The Guardian . Available at: https://www.theguardian.com/media/2020/apr/01/newspapers-to-lose-50m-in-online-ads-as-firms-use-coronavirus-blacklist.

Vargo, C.J. et al. 2018. The agenda-setting power of fake news: A big data analysis of the online media landscape from 2014 to 2016. New Media & Society 20(5), pp. 2028–2049. Available at: 10.1177/1461444817712086 [Accessed: 14 May 2020].

Vosoughi, S. et al. 2018. The spread of true and false news online. Science 359(6380), pp. 1146–1151. Available at: 10.1126/science.aap9559 [Accessed: 14 May 2020].

Zhou, V. 2019. A Simple Explanation Of Gini Impurity - Victorzhou.com. Available at: https://victorzhou.com/blog/gini-impurity/ [Accessed: 14 May 2020].