CS410 Course Project Final Report

US News Political Bias Detector

December 13th, 2020

Team Name: Bias Detectives

Names: Anh Nguyen, Muhammad Rafay, Nicholas Bachman

NetID's: anhn4@illinois.edu, mrafay2@illinois.edu, bachman5@illinois.edu

Team Captain: Nicholas Bachman

Free Topic: Text Classification and Sentiment Analysis

Public Repository: https://github.com/bachman5/CS410-BiasDetector

Project Timeline:

Milestone	Due Date
Submit Proposal	Oct 25th - Complete
Working Prototypes	Nov 22th – Complete
Code Video Demo	Nov 28th - Complete
Project Progress Report Submission	Nov 29th - Complete
2 x Initial Peer Reviews	Dec 2nd - Complete
Project Completion and Submission	Dec 13th - Complete
2 x Final Peer Reviews	Dec 16th

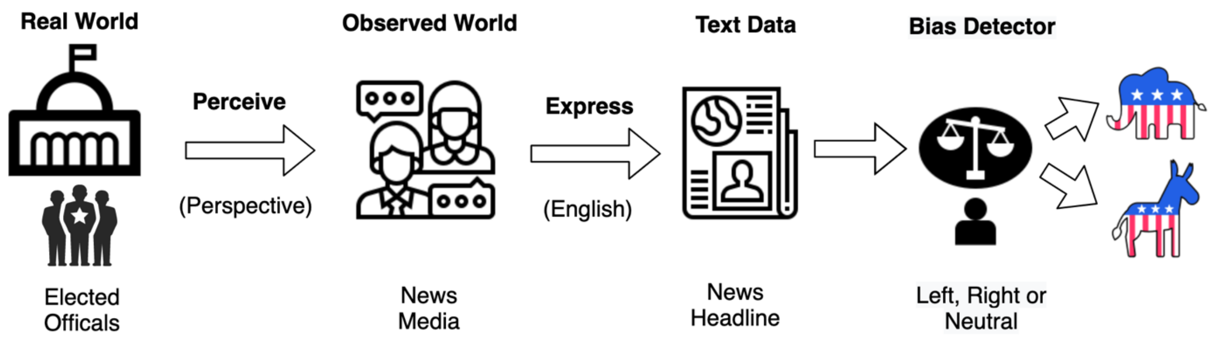
Workload Justification:

N = 3 team members

3 * 20 = 60 hours

Task	Estimated Hours
Build Text Mining / Clean News Headlines	4 hours - Complete
Collect and Label Test / Training Datasets (Corpus)	10 hours - Complete
Build / Tune Sentiment Analysis Technique	12 hours - Complete
Build / Train / Tune Logistic & SVM Model	12 hours - Complete
Build / Train / Tune Deep Learning Technique	12 hours – Complete
Develop Software Documentation	2 hours - Complete
Team Meetings	8 hours - Complete
Create Video Demonstrations	5 hours - Complete
Total	**65 hours**

Project Motivation



The political tension between the Democrat and Republican parties in the US has never seemed higher. Mainstream news media reports on events with a palpable bias that slants heavily toward one party or the other. Many Americans want unbiased, factual news reports to avoid being manipulated. Our free topic was to design a system that can examine mainstream news headlines and determine if they contain any political bias. The bias was determined using various sentiment analysis, shallow NLP and Machine Learning techniques. We then compared the results of three specific approaches by using all the knowledge gained from the CS 410 course.

Collect and Label Datasets (Corpus)

Raw Dataset: https://www.kaggle.com/snapcrack/all-the-news

Our Corpus was taken from ~200,000 News Articles published in the New York Times, Breitbart, CNN, Business Insider, the Atlantic, Fox News, Talking Points Memo, Buzzfeed News, National Review, New York Post, the Guardian, NPR, Reuters, Vox, and the Washington Post. The bulk of headlines were taken from 2015-2017 during a heated US presidential election.

Data Model

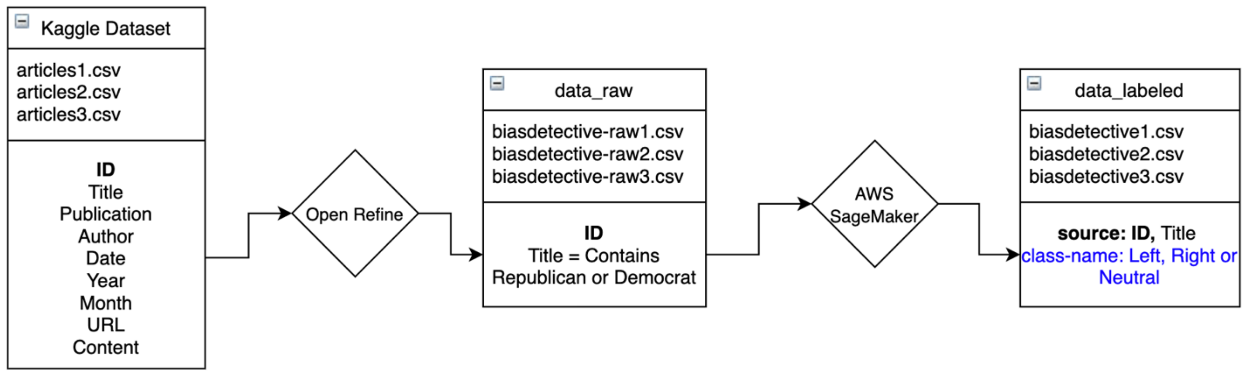
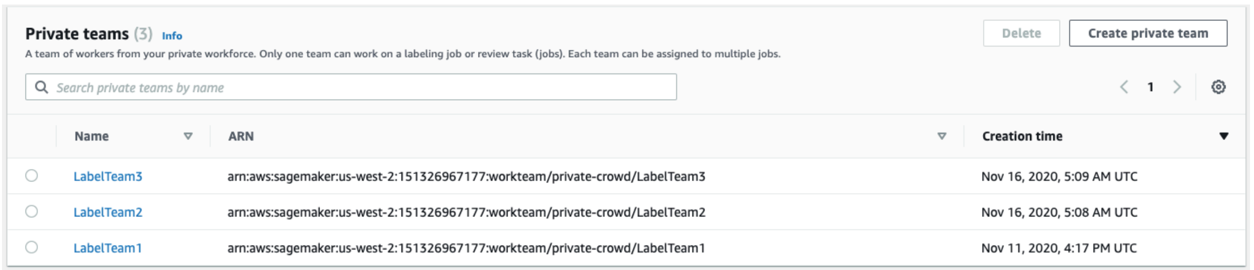


Figure 1: Data collection, filtering and labeling workflow

AWS SageMaker

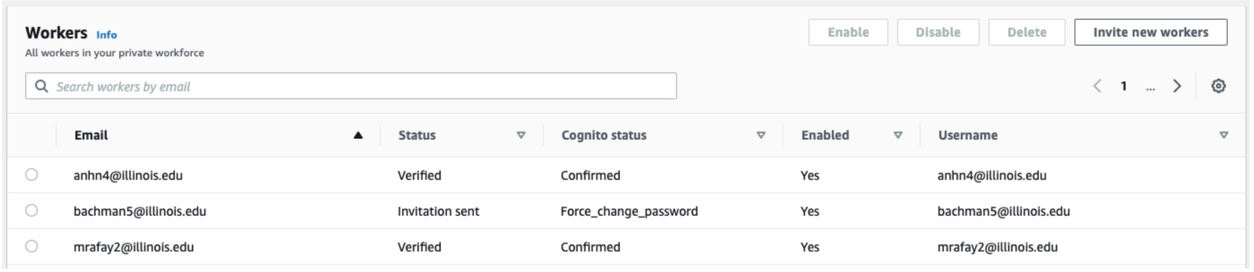
With AWS SageMaker, large labeling jobs can be broken up and assigned to public or private workforces. SageMaker increased the speed and accuracy of our labeling process. The jobs were broken up and assigned across all three team members. Each team member then recruited politically independent 3rd party members to assist with the large labeling task. We specifically tried to get labeling assistance from friends and family members that don't have strong political ties to either the Republican or Democratic ideologies. If the labeler has a strong political bias, then the resulting models could also reflect that bias.



Private teams (3) Info

A team of workers from your private workforce. Only one team can work on a labeling job or review task (jobs). Each team can be assigned to multiple jobs.

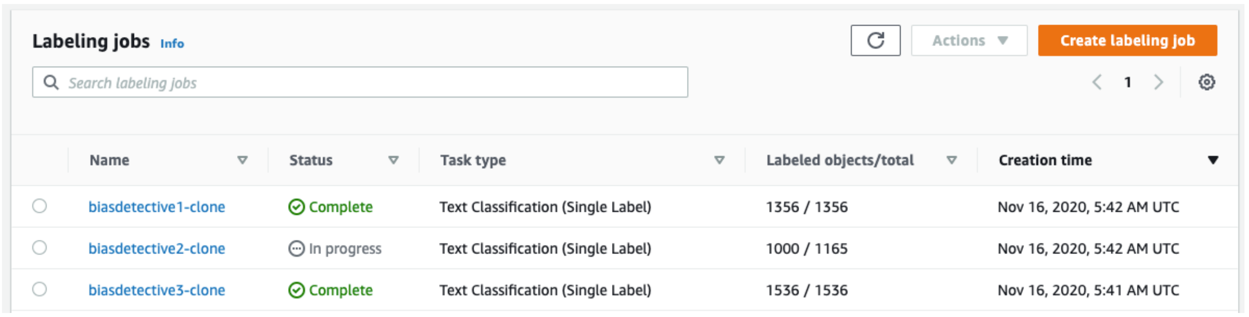
Name	ARN	Creation time
LabelTeam3	arn:aws:sagemaker:us-west-2:151326967177:workteam/private-crowd/LabelTeam3	Nov 16, 2020, 5:09 AM UTC
LabelTeam2	arn:aws:sagemaker:us-west-2:151326967177:workteam/private-crowd/LabelTeam2	Nov 16, 2020, 5:08 AM UTC
LabelTeam1	arn:aws:sagemaker:us-west-2:151326967177:workteam/private-crowd/LabelTeam1	Nov 11, 2020, 4:17 PM UTC



Workers Info

All workers in your private workforce

Email	Status	Cognito status	Enabled	Username
anhn4@illinois.edu	Verified	Confirmed	Yes	anhn4@illinois.edu
bachman5@illinois.edu	Invitation sent	Force_change_password	Yes	bachman5@illinois.edu
mrafay2@illinois.edu	Verified	Confirmed	Yes	mrafay2@illinois.edu



Labeling jobs Info

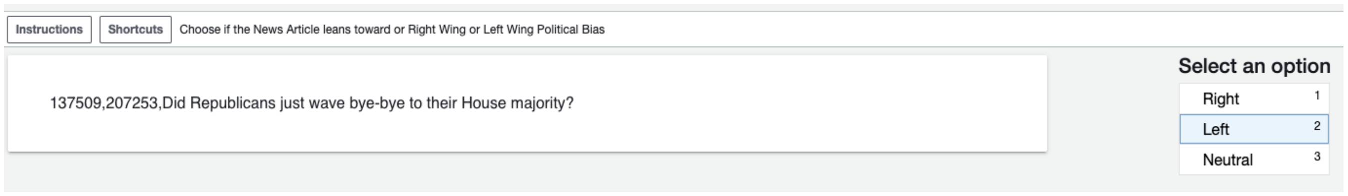
Name	Status	Task type	Labeled objects/total	Creation time
biasdetective1-clone	Complete	Text Classification (Single Label)	1356 / 1356	Nov 16, 2020, 5:42 AM UTC
biasdetective2-clone	In progress	Text Classification (Single Label)	1000 / 1165	Nov 16, 2020, 5:42 AM UTC
biasdetective3-clone	Complete	Text Classification (Single Label)	1536 / 1536	Nov 16, 2020, 5:41 AM UTC

AWS GroundTruth

AWS GroundTruth is the portal that private team members used to label our data.   A simple dashboard was created for the labeler to view a single news headline at a time with no additional information like publisher that could introduce additional bias. The GroundTruth user had options to label the headline as either having Right Bias, Left Bias or Neutral.

Final Labeled Test / Training Dataset: https://github.com/bachman5/CS410-BiasDetector/tree/main/data_labeled

Our completed training data contained over 4,000 filtered, cleaned and labeled headlines from the workflow. News headlines were labeled as either Right Wing bias, Left Wing Political bias or Neutral using AWS Sagemaker and GroundTruth. Here is an example record correctly labeled with Left Wing bias:

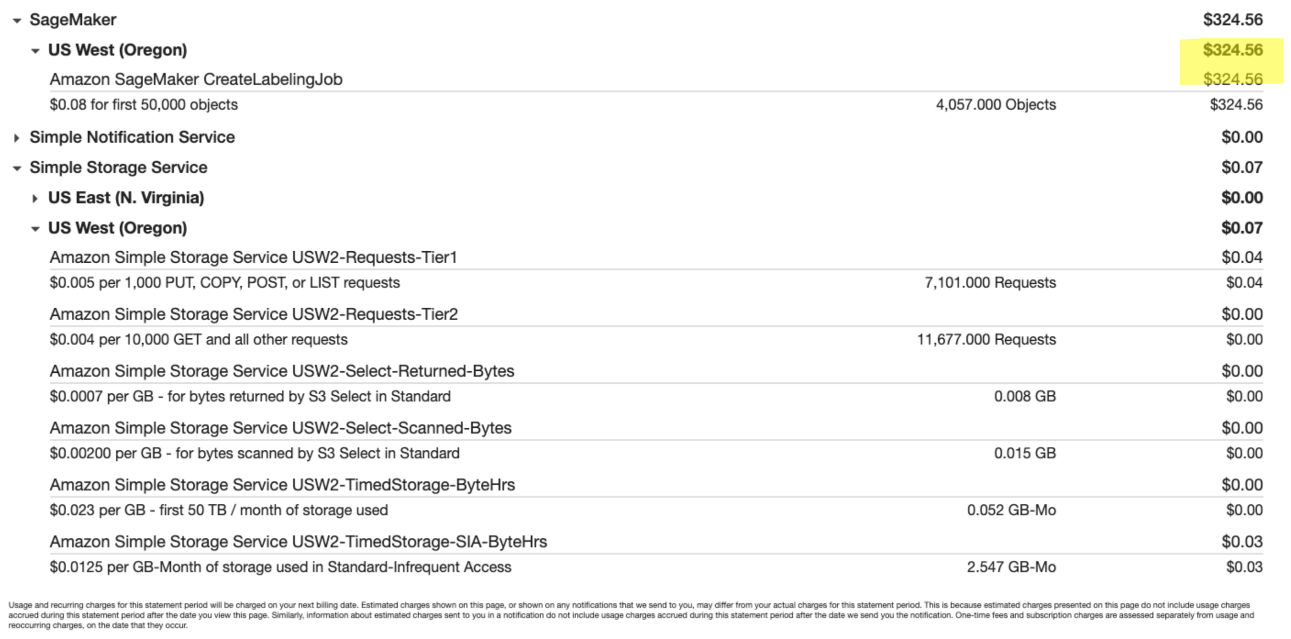


Choose if the News Article leans toward or Right Wing or Left Wing Political Bias

137509,207253,Did Republicans just wave bye-bye to their House majority?

Lessons Learned

AWS Ground Truth provides a great way to explain to the labeler exactly what you are looking for.   You can even provide them examples of correct and incorrect label.   These are great features, but they come at a high cost.   Our team was unaware that AWS charges .08 cents per label.   There is no warning or advertised cost until you get your monthly bill.   4,057 objects x .08 = $323.56.



SageMaker		$324.56
US West (Oregon)		$324.56
Amazon SageMaker CreateLabelingJob		$324.56
$0.08 for first 50,000 objects	4,057.000 Objects	$324.56
Simple Notification Service		$0.00
Simple Storage Service		$0.07
US East (N. Virginia)		$0.00
US West (Oregon)		$0.07
Amazon Simple Storage Service USW2-Requests-Tier1		$0.04
$0.005 per 1,000 PUT, COPY, POST, or LIST requests	7,101.000 Requests	$0.04
Amazon Simple Storage Service USW2-Requests-Tier2		$0.00
$0.004 per 10,000 GET and all other requests	11,677.000 Requests	$0.00
Amazon Simple Storage Service USW2-Select-Returned-Bytes		$0.00
$0.0007 per GB - for bytes returned by S3 Select in Standard	0.008 GB	$0.00
Amazon Simple Storage Service USW2-Select-Scanned-Bytes		$0.00
$0.00200 per GB - for bytes scanned by S3 Select in Standard	0.015 GB	$0.00
Amazon Simple Storage Service USW2-TimedStorage-ByteHrs		$0.00
$0.023 per GB - first 50 TB / month of storage used	0.052 GB-Mo	$0.00
Amazon Simple Storage Service USW2-TimedStorage-SIA-ByteHrs		$0.03
$0.0125 per GB-Month of storage used in Standard-Infrequent Access	2.547 GB-Mo	$0.03

Usage and recurring charges for this statement period will be charged on your next billing date. Estimated charges shown on this page, or shown on any notifications that we send to you, may differ from your actual charges for this statement period. This is because estimated charges presented on this page do not include usage charges accrued during this statement period after the date you view this page. Similarly, information about estimated charges sent to you in a notification do not include usage charges accrued during this statement period after the date we send you the notification. One-time fees and subscription charges are assessed separately from usage and reoccurring charges, on the date that they occur.

Category Classification + Sentiment Analysis Approach

Language: Python

Dependencies: nltk, nltk.vader, numpy, pandas

Public Repository: https://github.com/bachman5/CS410-BiasDetector/blob/main/sentiment_test.py

Primary Team Member: Nick Bachman

Overview: VADER (Valence Aware Dictionary for Sentiment Reasoning) is a model used for text sentiment analysis that is sensitive to both polarity (positive/negative) and intensity (strength) of emotion. It is included in the NLTK package and can be applied directly to unlabeled text data. It can also be customized for specific domains and use cases. Our first approach was to see if we could design a system that combines topic category and sentiment to accurately predict political bias.

Technical Approach:

The goal of this model is to correctly determine if a text headline includes Left or Right political bias. In the headline "Daily News mourns the death of the Republican Party, killed by epidemic of Trump", the subject of the news article is the Daily News. The Reuters writer Lucas Jackson used the terms mourns, death, killed and epidemic to communicate significantly negative sentiment about the Republican Party and Donald Trump. Most people would agree that this headline supports a Left-Wing political position and thus includes a Left bias.

{"source":"69446,"**Daily News** *mourns* the *death* of the Republican Party, *killed* by *epidemic* of Trump'","**class-name":"Left**"}

As a group, we spent time thinking about the definition of political bias and how to design a labeling and detection system. The chart below summarizes our design decision. Sentiment alone is not enough to determine political bias. You also need a way to determine the category that sentiment is being directed toward.

Topic Category	**Sentiment**	**Bias**
Republican	Negative :(	Left
Republican	Positive :)	Right
Republican	Neutral :|	Neutral
Democrat	Negative :(	Right
Democrat	Positive :)	Left
Democrat	Neutral :|	Neutral

Table 1: Combining Category and Sentiment to determine Bias

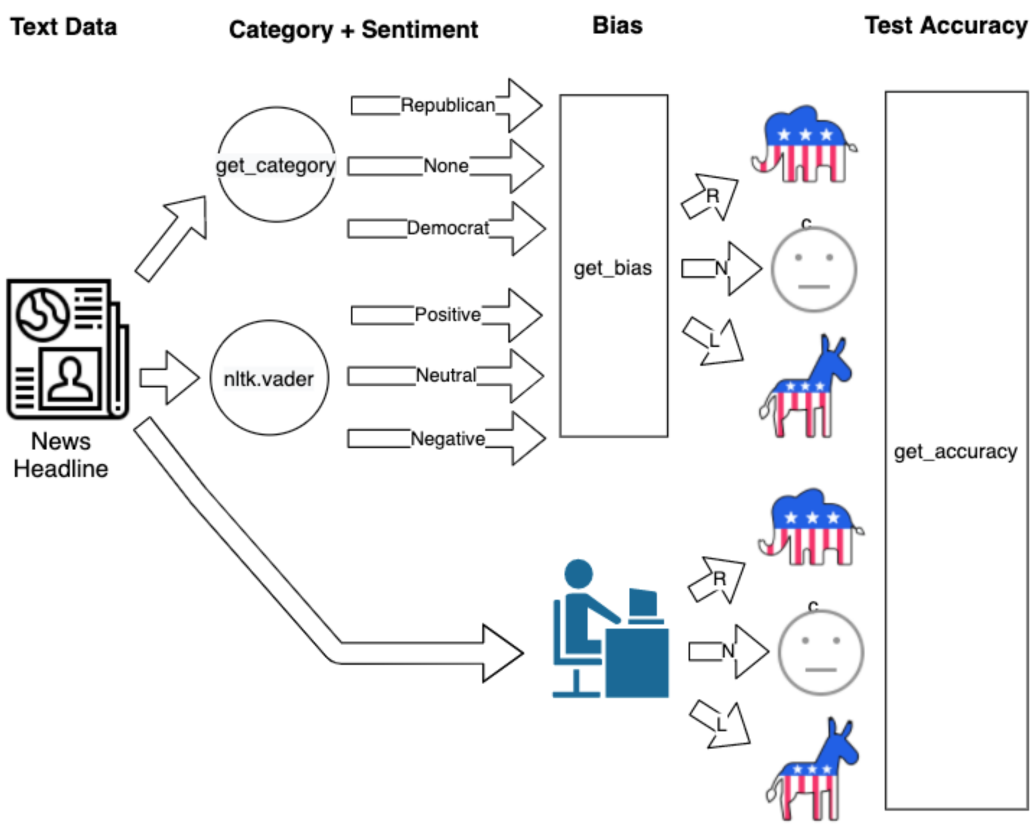


Figure 2: Combining Category and Sentiment to determine Bias

Technical Challenges:

Updating the category lexicon with descriptive terms and updating the VADER sentiment lexicon with popular political terms increased the accuracy by about 8-10%. Tunning the values for how much negative or positive sentiment yields a neutral response also helped because "Neutral" was the label with the most entropy. Example:

"Game of Thrones: Republicans hate it, Democrats love it".

GoT is the subject in this example. If the reader likes the HBO show Game of Thrones, it changes how they will label it. Both Republicans and Democrats are mentioned so there are multiple categories to attach the sentiment. Both Negative (hate) and Positive (Love) sentiment are expressed. The labeler may choose Neutral because they are unsure or do not know what Game of Thrones is in relation to this political topic. Complex sentiment and ambiguity make it very difficult to accurately label bias and accurately predict bias.

Lessons Learned

One additional feature that I added was putting a heavier weight on terms that occurred first in a headline. In English, most subjects occur to the left of the predicate. When both Democratic and Republican terms are mentioned, I subtracted the weights from each other. If the difference was minor, I added a category for "Both" that covers when Democrats and Republicans are both the subject (left of the predicate) and close together. This does not cover the minority cases when the subject is to the right of the predicate. This happens with questions, sentences that begin with "Here" or "There" and sentences beginning with one or more prepositional phrases. Additional rules could be added to check for these three exceptions to change the weighing when the subject occurs after the predicate. I tried using NLP external libraries such as Spacy but they also struggled with identifying the subject of many headlines. I think this is because news headlines are not using natural language. In other words, humans don't usually talk or write in the same way that news headlines are written to capture your attention. Example:

"Hillary Clinton: Republicans dishonor constitution by vowing to block Scalia replacement"

Hillary Clinton (Dem) is saying that Republicans are dishonoring the constitution. The colon is acting like the verb "says". My code correctly identifies Hillary Clinton (Left) as the subject and attaches the negative sentiment because of the word dishonor and determines that it's right bias. However, the reader sees that the negative sentiment is attached to Republicans and labels it as left bias. In this case the sentiment was not attached to the subject but rather what the subject said about another subject. This is just one example of the challenges faced by trying to attach sentiment to a subject.

Results:

	biasdetective1	**biasdetective2**	**biasdetective3**
Right Precision	.580	.487	.414
Right Recall	.593	.633	.708
Right F1	.586	.550	.523
Left Precision	.450	.575	.557
Left Recall	.565	.542	.578
Left F1	.503	.558	.568
Overall Accuracy	.521	.528	.483

Support Vector Machine Approach

Language: Python

Public Repository: https://github.com/bachman5/CS410-BiasDetector/tree/main/SVM_Model

Dependencies: numpy, panda, glove, spacy, sklearn

Primary Team Member: Anh Nguyen

Overview:

Support Vector Machine (SVM), a Discriminative Classification introduced in one of our lectures in Text Categorization, is another method that we decided to implement in our model. The method is mainly included in sklearn package. To improve the dataset and support model's accuracy, couple of NLP methods and Vector Embedding are also utilized and can be explain further in the technical approach

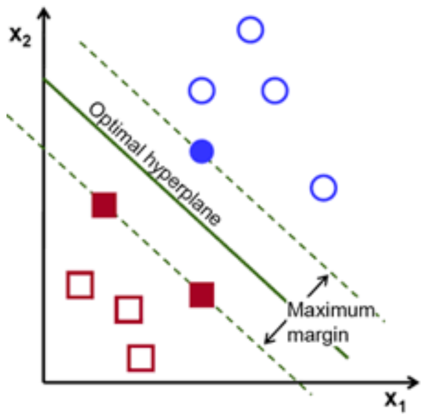


Figure 3: SVM explanation

Technical Approach:

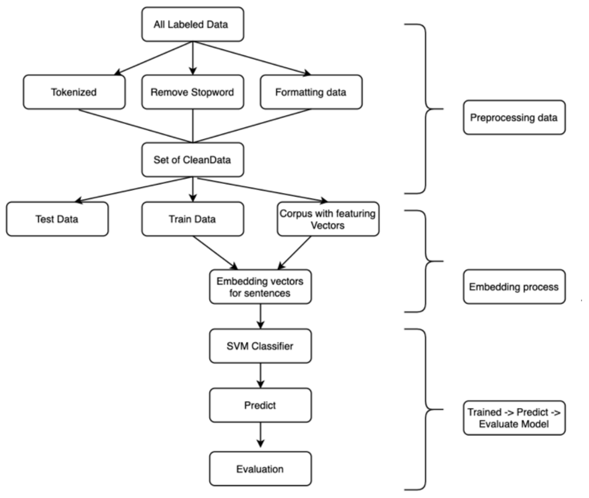
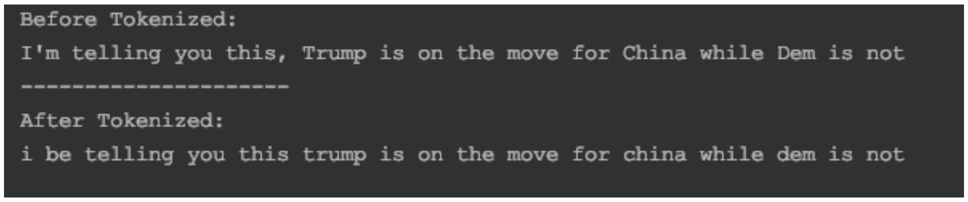


Figure 4: Overall Diagram for SVM Model

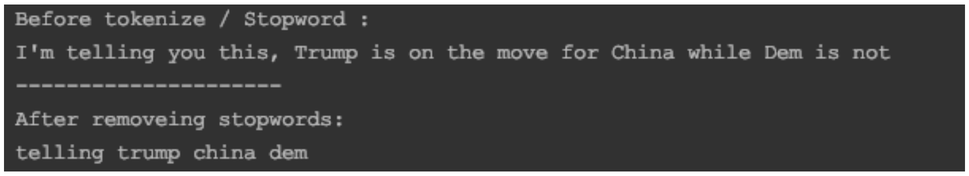
1. **Preprocessing Data (NLP): (Using Spacy)**

The main goal of this step is using some NLP techniques to conduct meaningful patterns and themes for the text data using **spacy** package: tokenization (breaking news titles into token), stopword (removing common stop works in sentence), punctuation (clean sentence better by ignoring punctuation like marks and spaces)



```
Before Tokenized:
I'm telling you this, Trump is on the move for China while Dem is not
---------------------
After Tokenized:
i be telling you this trump is on the move for china while dem is not
```

Before and after Word Tokenized



```
Before tokenize / Stopword :
I'm telling you this, Trump is on the move for China while Dem is not
---------------------
After removeing stopwords:
telling trump china dem
```

Before and after Removing stopwords

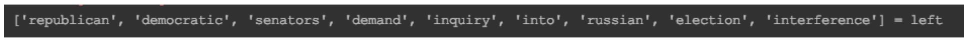
Word Tokenization appeared to perform accurately with the purpose of data cleaning. However, removing stopwords tend to remove all the context of the sentence.

2. **Word embedding & Magnitude (Using Glove)**

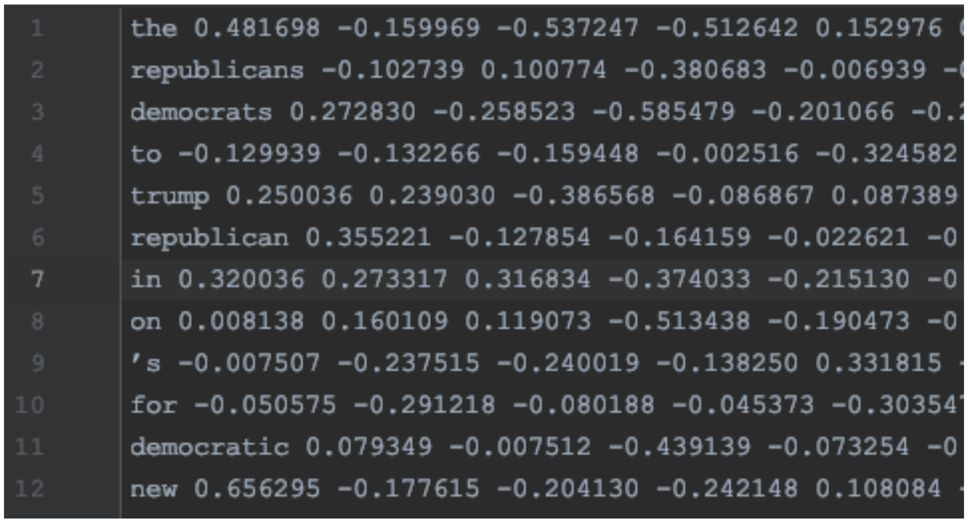
Building Corpus

After preprocessing data, we then store available data into three different text files: train.txt, text.txt and corpus.txt. Ideally Corpus should be a dictionary including all political words in bigger spectrum than only words in train & text data, however due to the limitation of the data our **Corpus** only built upon our available scope of data.

Words in Corpus then mapped to corresponding vectors (Word Embedding/word Vectorization) with the hope of capturing the meaning of potential relationship of word in term of similar contexts/syntax/spelling/co-occurrence.



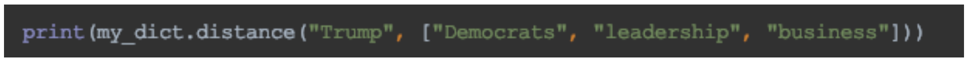
```
['republican', 'democratic', 'senators', 'demand', 'inquiry', 'into', 'russian', 'election', 'interference'] = left
```



```
the 0.481698 -0.159969 -0.537247 -0.512642 0.152976
republicans -0.102739 0.100774 -0.380683 -0.006939 -
democrats 0.272830 -0.258523 -0.585479 -0.201066 -0.
to -0.129939 -0.132266 -0.159448 -0.002516 -0.324582
trump 0.250036 0.239030 -0.386568 -0.086867 0.087389
republican 0.355221 -0.127854 -0.164159 -0.022621 -0
in 0.320036 0.273317 0.316834 -0.374033 -0.215130 -0
on 0.008138 0.160109 0.119073 -0.513438 -0.190473 -0
's -0.007507 -0.237515 -0.240019 -0.138250 0.331815
for -0.050575 -0.291218 -0.080188 -0.045373 -0.30354
democratic 0.079349 -0.007512 -0.439139 -0.073254 -0
new 0.656295 -0.177615 -0.204130 -0.242148 0.108084
```

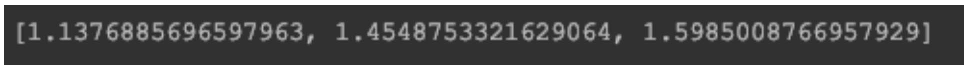
Corpus (Dictionary) with vectorized words

Testing word's distance:



```
print(my_dict.distance("Trump", ["Democrats", "leadership", "business"]))
```

Result:

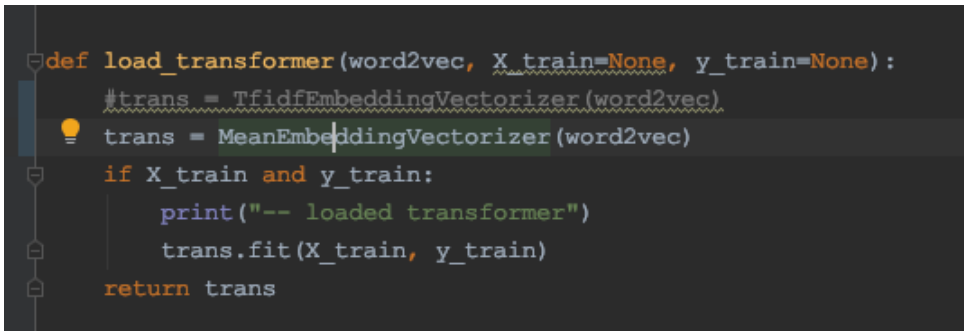


```
[1.1376885696597963, 1.4548753321629064, 1.5985008766957929]
```

Another method that we were using (using **Glove** package) was store our corpus to a magnitude file. Normally a corpus can be text-formatted but storing dictionary in magnitude file helps improving the processing time.

<u>Vectorizing titles</u>

Using MeanEmbeddingVectorizer & TF-IDF Embedding



```
def load_transformer(word2vec, X_train=None, y_train=None):
    #trans = TfidfEmbeddingVectorizer(word2vec)
    trans = MeanEmbeddingVectorizer(word2vec)
    if X_train and y_train:
        print("-- loaded transformer")
        trans.fit(X_train, y_train)
    return trans
```

3. Train/Tune Model (Using sklearn)

Current Result

	TF_IDF (Preprocessing)	**Mean Embedding Vector (preprocessing)**
Accuracy	0.514379622021364	0.5308134757600658
Precision	0.5306949522525308	0.4542249353200501
Recall	0.514379622021364	0.5308134757600658
F1 Score	0.4751926328128259	0.48172315509176467

Technical Challenge/ Lesson Learn:

The current accuracy is still in range 50% with TF-IDF embedding preprocessing as well as Vector preprocessing. While there is not much success in improving the accuracy levels. We started to doubt the truthfulness of our labelled data.

To assert the validity of our model pipeline, we switched to benchmarking using a different dataset while keeping everything else the same. The dataset is ATIS Airline Spoken Language Intent (train / test) which classifies a passenger's inquiry into 1 of the possible 18 intents: flight, flight time, meal, etc. The data size is about 5000 records and 18 classes (intents). The performance came out significantly better than what observed from our dataset:

```
Accuracy : 0.9113495200451722
Precision: 0.9107883118388134
Recall   : 0.9113495200451722
F1 score : 0.9011684778567066
```

One more example of our doubt in data integrity is that that we discovered the differences labels in one title. However, we do not complete blame on this issue as in the real world many opinions can appear for the same information.

```
100300," Republican, Democratic Senators Demand Inquiry Into Russian Election
Interference", Neutral
113763," Republican, Democratic Senators Demand Inquiry Into Russian Election
Interference", Left
100300," Republican, Democratic Senators Demand Inquiry Into Russian Election
Interference", Neutral
113763," Republican, Democratic Senators Demand Inquiry Into Russian Election
Interference", Neutral
```

Deep Learning: Recurrent Neural Network Classification Approach

Language: Python

Dependencies: TensorFlow

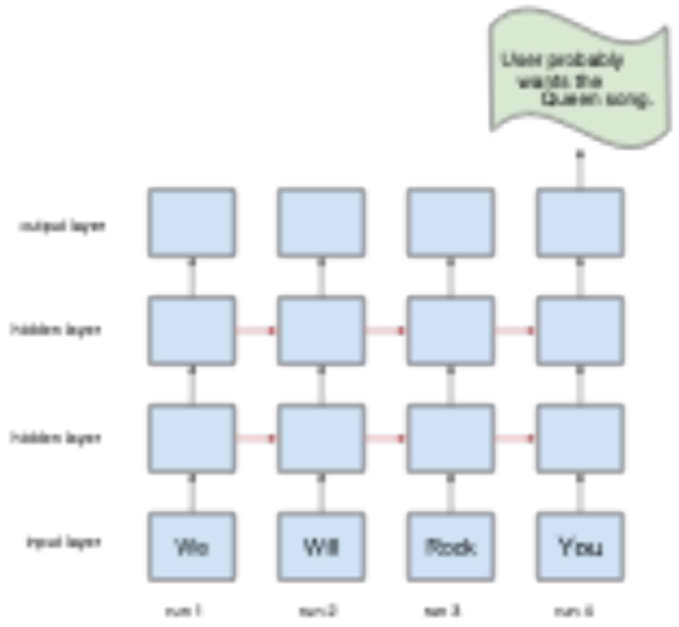
Primary Team Member: Muhammad Rafay

Overview:

The approach uses a recurrent neural network (RNN) to classify news headlines as Left or Right biased.

Technical Approach of RNN:

Unlike the feed forward neural network we used an RNN for classifying news headlines. The difference in RNN verses feed forward is that output of an RNN is not just influenced by the input we just fed but it is influenced by the history of inputs we have fed earlier. An example is given below:



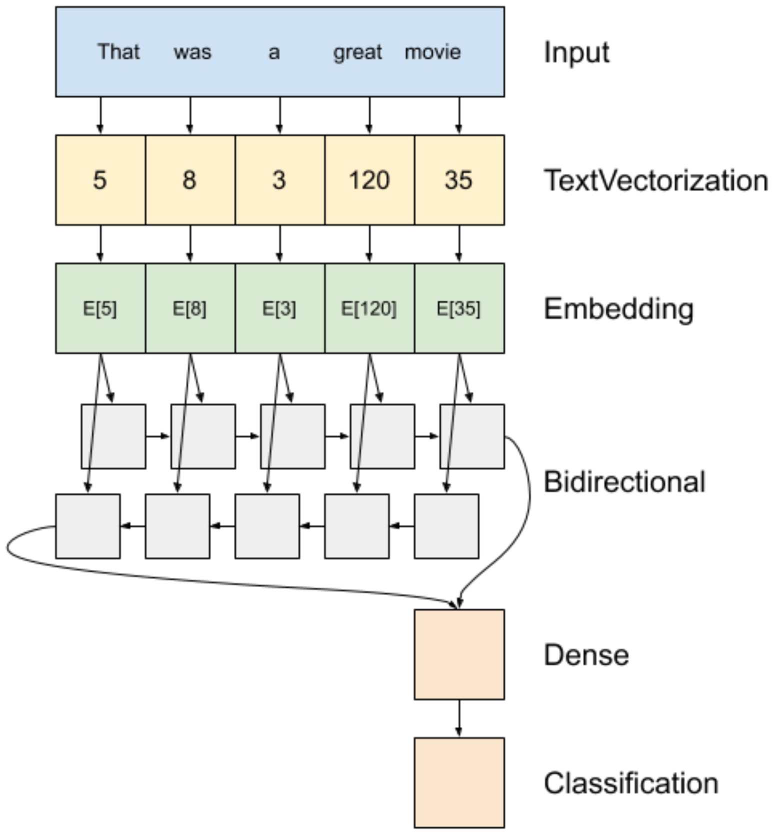
So, it means RNN is suitable for classifying sequence data and news headline is a sequence.

Since, neural network processes numbers we had to convert text to a sequence of token indices using an encoder. After the encoding is the embedding layer, this layer stores one vector per word after training words with similar meanings have similar vectors.

Next comes the RNN with a stack of hidden layers, it processes the sequence of input by iterating through the elements. At each step output from the previous step is passed with the input of the next step.

The sequence is converted to a single vector which is further converted to a single logit as the classification output.

The architecture of our RNN is like the one given below:



Lesson Learned:

Our current classification accuracy is 53%. Not much different from other models.

Since we are classifying based on political bias and not sentiments, the two classes: Left and Right has a lot of common vocabulary. Both text of both classes contains hate speech and words "democrats" and "republican" so in short, the vocabulary is not a very distinguishing feature for the two classes. The sentiment of bias is sometimes hidden behind the meaning of the sentence and it is not obvious e.g.

"Are Republicans more likely to prefer pulp in their orange juice?"

It is only obvious when something positive or negative is being said about one of the parties, and that involves recognizing the sentiment of the entity but not every headline is like that e.g.

"Republicans' patience with Trump may be running out"

We also suspected that the poor results are due to erroneous labeling of dataset. Hence, we explored and found another similar dataset Ideological Books Corpus (IBC). We trained/tested RNN on IBC dataset and found that might not me the case as the performance metrics for this dataset were like ours rather a bit poor

So, we need a model that would learn the hidden meaning behind the headlines.

On our own dataset

Accuracy: 0.528

Precision: 0.507

Recall: 0.558

F1 score: 0.531

On our own dataset

Accuracy: 0.504

Precision: 0.454

Recall: 0.432

F1 score: 0.442

Deep Learning: Classification using BERT

Language: Python

Dependencies: TensorFlow

Primary Team Member: Muhammad Rafay

Overview:

BERT is recent breakthrough in NLP. It makes use of a transformer mechanism that learns contextual relations between words. It has achieved state-of-the-art performance on several NLP tasks such as GLUE. As we looked forward to improving performance our next choice of model was BERT

Technical Approach:

As compared to an RNN based LSTMS which we implemented earlier BERT incorporates a non-directional encoder unlike LSTMS which read the test in sequence (left-to-right or right-to-left) BERT reads the entire sequence at once. This allows BERT to learn the context of a word based on all its surroundings. BERT models are pre-trained on a large corpus of text. They then fine-tuned for specific tasks

Like the last model BERT also uses vector space representations of natural language to learn and predict.

Lessons Learned:

This approach validates our hypothesis for this kind of problem we need a model that better grasps the context of the sentence, rather than just learning from the word sentiment. BERT proved to be a significantly better classifier than the other three, due to its bidirectional training. But still, not good enough to be used in a real application.

Current Result:

On our own dataset

Accuracy: 0.594

Precision: 0.568

Recall: 0.630

F1 score: 0.599

On IBC Dataset:

Accuracy: 0.598

Precision: 0.558

Recall: 0.568

F1 score: 0.563

Video Demonstration

Category + Sentiment Demo

https://drive.google.com/file/d/1ycSjyZAlT915zT_RqlEKu-lhGaig8hyr/view?usp=sharing

SVM Model Demo

https://drive.google.com/file/d/1rGzV26i5q7GmMNR9MwLthTDqPrfe5H-4/view?usp=sharing

Conclusion:

We learned that getting a correctly labeled dataset is extremely important when training any ML model. Inaccuracy, duplication of records, ambiguity and bias can all negatively affect the outcomes. We also learned that how you create the labels are important. We chose three labels to categorize headlines but there were five or more separate clusters of word distributions that overlapped with each other. Discriminately classifying them and defining similarity was a significant challenge.

The category/sentiment and SVM approach produced slightly lower accuracy than LSTM and BERT. Correctly identify the subject of the headline was difficult with shallow NLP which caused the sentiment to attach with the wrong subject and decreased the accuracy.

	Category + Sentiment	**SVM**	**LSTM**	**BERT**
Overall Precision	0.512	0.530	0.507	0.568
Overall Recall	0.601	0.514	0.558	0.630
Overall F1	0.550	0.475	0.531	0.599
Overall Accuracy	0.521	0.514	0.528	0.594

One approach that we could have done is doing text categorization and sentimental analysis as two different steps of this project. This might be a future improvement that we can look into.
Even though we didn't get to achieve the accuracy that we were hoping for, this project has brought much knowledge. It helps tie up all materials in class together from the way we do NLP, using TF-IDF to generative and classifier models like Naïve Bayes, Logistic Regression, SVM, etc. It's like a perfect summary that we need to put everything we learnt in materials into practice.

To conclude, we learned that bias detection is a harder text categorization problem than sentiment analysis because bias categories are weakly corelated to the surface features of the text such the sentiments of words in a sentence are not enough to predict bias. This hypothesis is validated by the result from BERT model. Since BERT learns contextual relationship between words it has performed better, and the accuracy has increased. Probably more fine-tuning can help it improve more.

The other bottleneck includes amount of training data. Since we couldn't find the dataset, we were looking for we had to hand label it our self. Which bring us to another problem, we later found instances where our labeling was not accurate. So better training data set in terms of amount of data and better labeling would help improve bias detection as well.

References

Hutto, C.J. & Gilbert, E.E. (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014.

https://www.tensorflow.org/tutorials/text/classify_text_with_bert

https://www.tensorflow.org/tutorials/text/text_classification_rnn

https://www.dataquest.io/blog/tutorial-text-classification-in-python-using-spacy/

https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/

https://aws.amazon.com/sagemaker/groundtruth/

https://spacy.io/usage/spacy-101