CS410 Course Project Progress Report

US News Political Bias Detector

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
Project Completion and Submission	Dec 13th
2 x Final Peer Reviews	Dec 16th

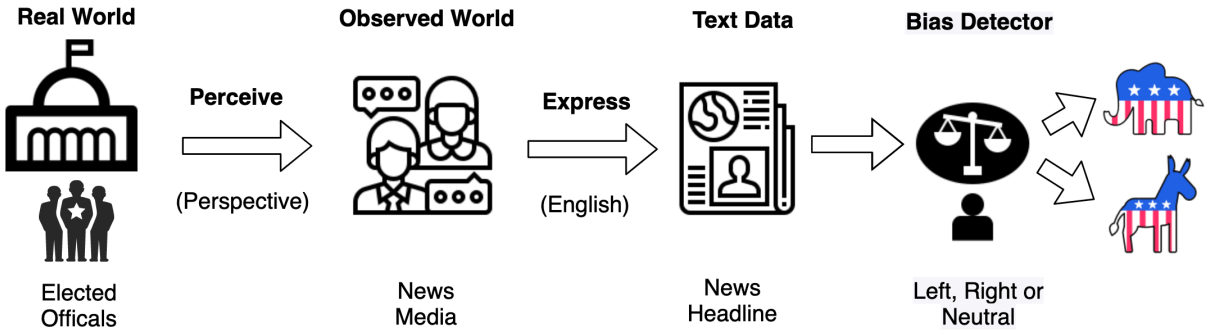
Workload Justification:

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	14 hours – In progress
Visualization to display results (Tableau or Website)	?
Develop Software Documentation	2
Team Meetings	5 hours
Create Video Demonstrations	4 hours
Total	

Project Motivation



Collect and Label Datasets (Corpus)

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

Description: 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

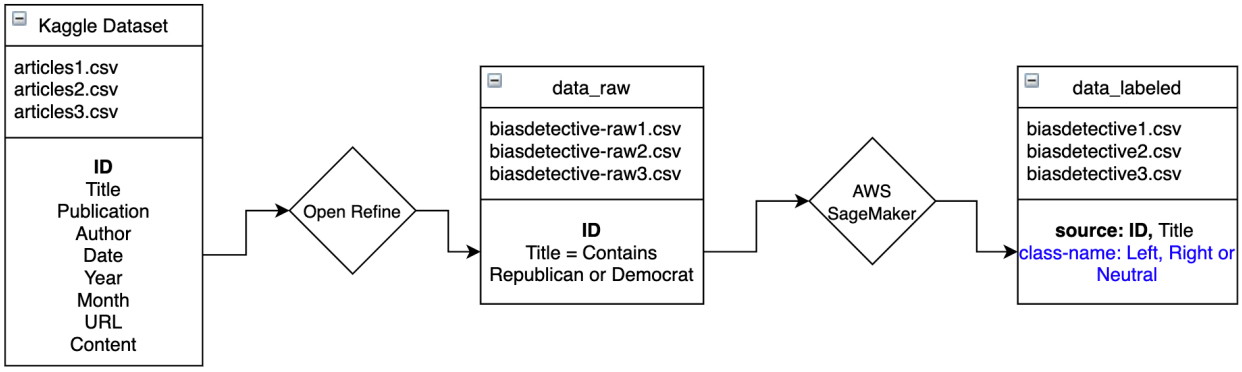
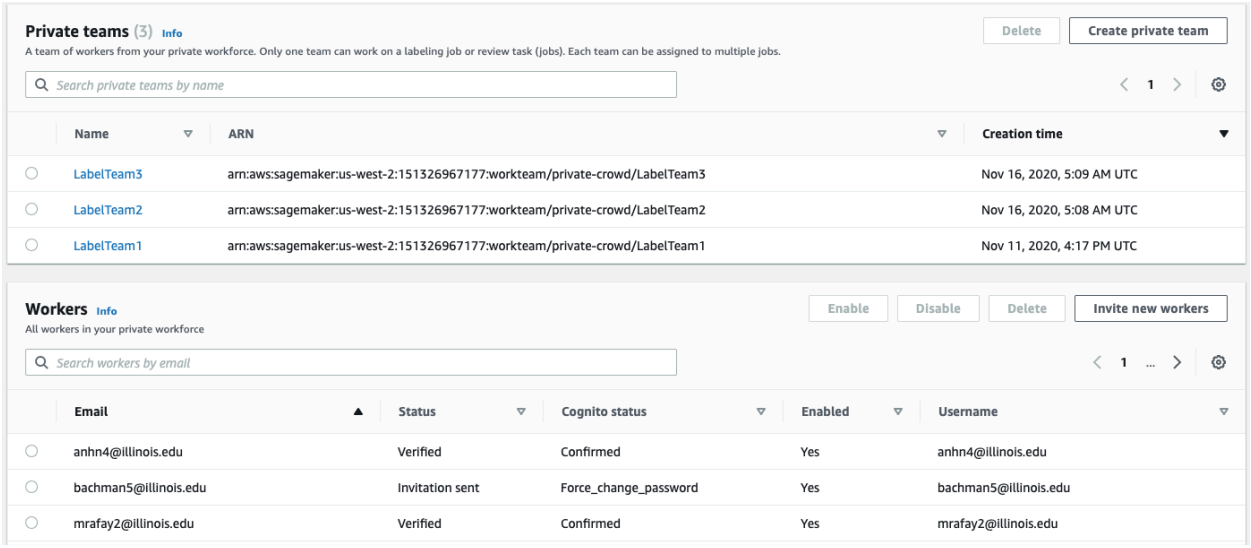
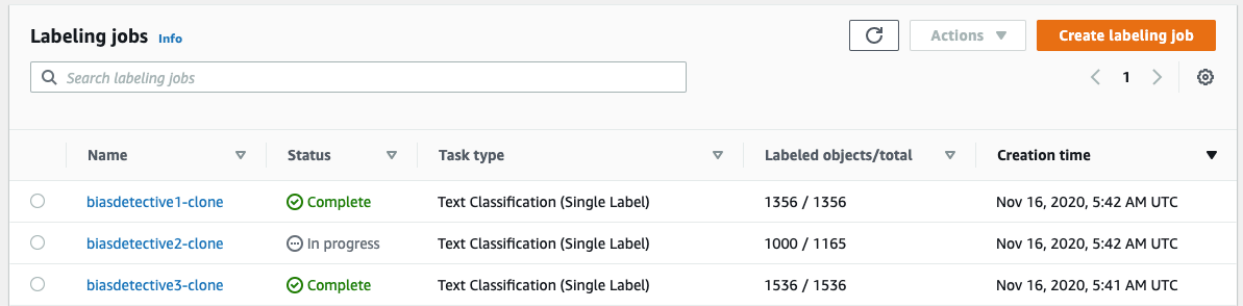


Figure 1: Data collection, filtering and labeling workflow

AWS SageMaker

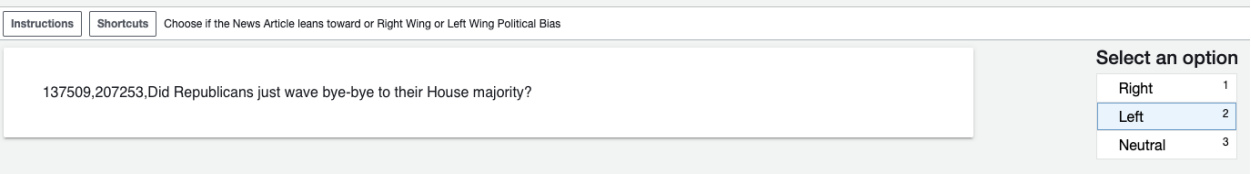
With AWS SageMaker, large labeling jobs can be broken up and assigned to public or private workforces. SageMaker increased the speed and accuracy of our labeling process. The jobs were broken up and assigned across all three team members. Each team member then recruited politically independent 3rd party 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.





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

Description: ~4,000 filtered, cleaned and labeled headlines were created 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:

Category + 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 Republican party. The Reuters writer Lucas Jackson used the terms

mourn, 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

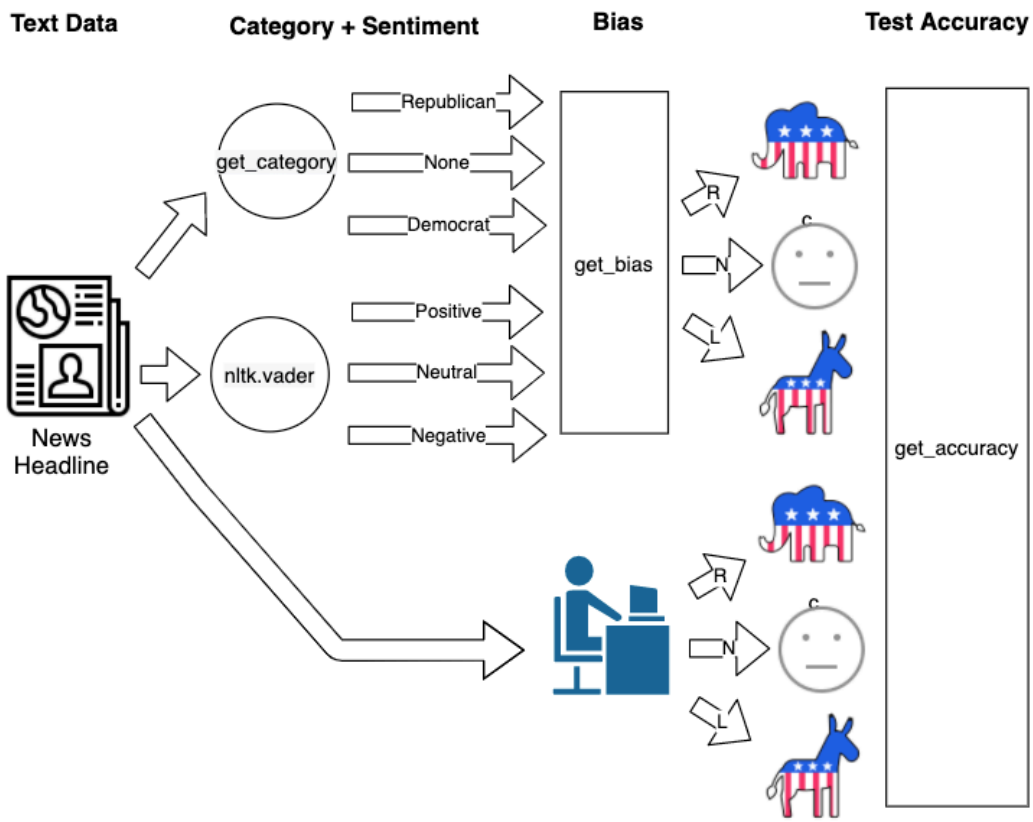


Figure 2: Combining Category and Sentiment to determine Bias

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. For example: "Game of Thrones: Republicans hate it, Democrats love it". 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 subject categories. 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 difficult to accurately label bias and accurately predict bias.

Current Results:

Test Dataset: biasdetective1.csv Classification Accuracy: 44%

Test Dataset: biasdetective2.csv Classification Accuracy: 42%

Test Dataset: biasdetective3.csv Classification Accuracy: 47%

More advanced supervised and deep learning models discussed later in the report yield more accurate results.

Support Vector Machine Approach

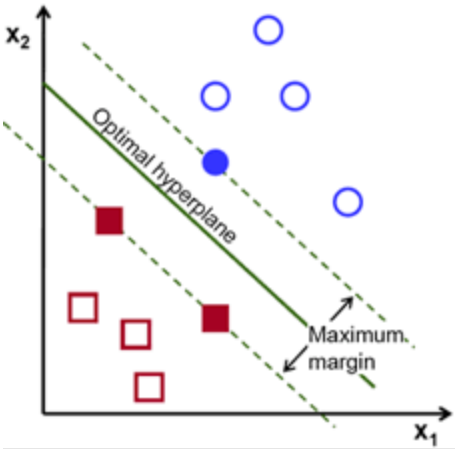
Language: Python

Dependencies: numpy, panda, glove, spacy, sklearn

Primary Team Member: Anh Nguyen

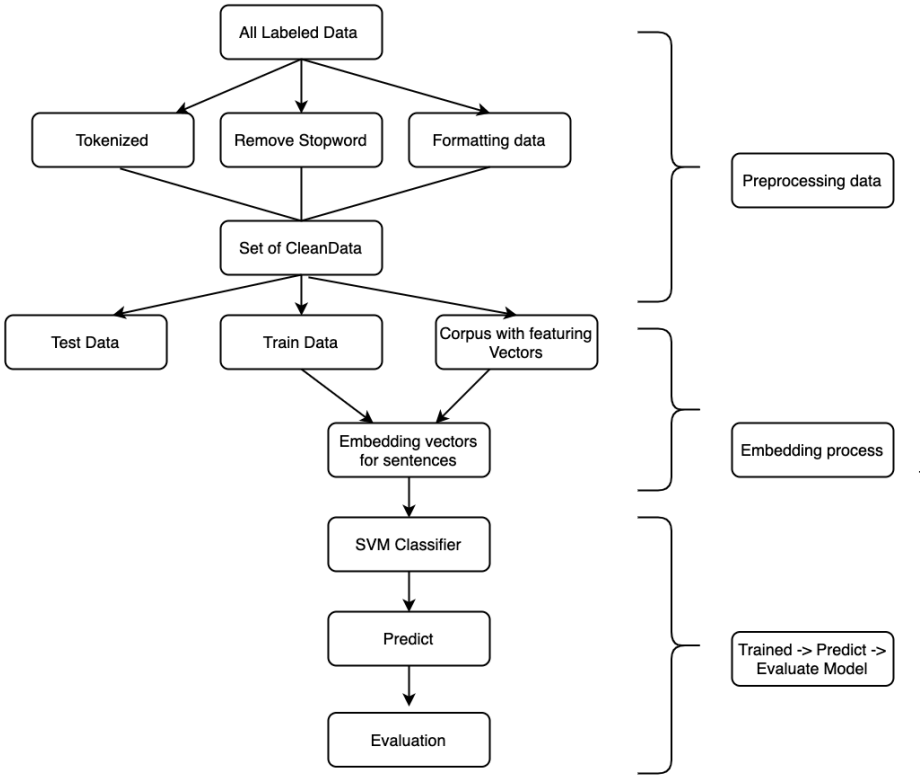
Overview:

Support Vector Machine (SVM) is a Discriminative Classification introduced in one of our lectures in Text Categorization. SVM uses classification algo to separate classes of data point then find the maximum margin of hyperplane between data points in their associated classes. The method below also includes some NLP methods + Vector Embedding to improve dataset and support model’s accuracy



Technical Approach:

Follow by diagram below



Overall Diagram

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

Goal of this is to conduct a more meaningful patterns and themes for the text data using spacy. The main method for this data processing is tokenization was breaking news titles into token, 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

Another method that I have tried is removing stopwords from sentences. However, it brings me to uncompleted sentences that are difficult to translate and understand.

```
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

OutPut:

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

X : word in titles, Y : labels

2. Word embedding & Magnitude (Using Glove)

Building Corpus

After preprocessing data, I have built 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.

```
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 I'm using in this is 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.

Vectorizing titles

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 for preprocess + SVM classification

```
Accuracy : 0.514379622021364
Precision: 0.5306949522525308
Recall   : 0.514379622021364
F1 score : 0.4751926328128259
```

Mean Embedding Vector preprocess + SVM classification

```
Accuracy : 0.5308134757600658
Precision: 0.4542249353200501
Recall   : 0.5308134757600658
F1 score : 0.48172315509176467
```

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. I started to doubt the truthfulness of our labelled data.

To assert the validity of our model pipeline, I 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
```

Challenges:

NLP: Still facing challenging with precise deep Semantic Analysis & Bias in Labelled Data:

Deep Learning: Recurrent Neural Network Classification Approach

Language: Python

Dependencies: TensorFlow

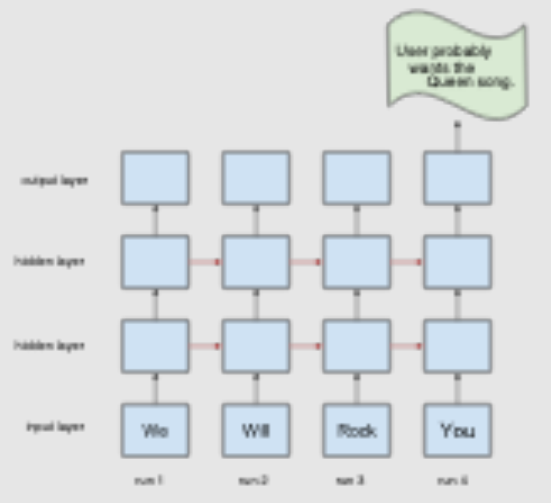
Primary Team Member: Rafay, Muhammad

Overview:

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

Technical Approach:

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:



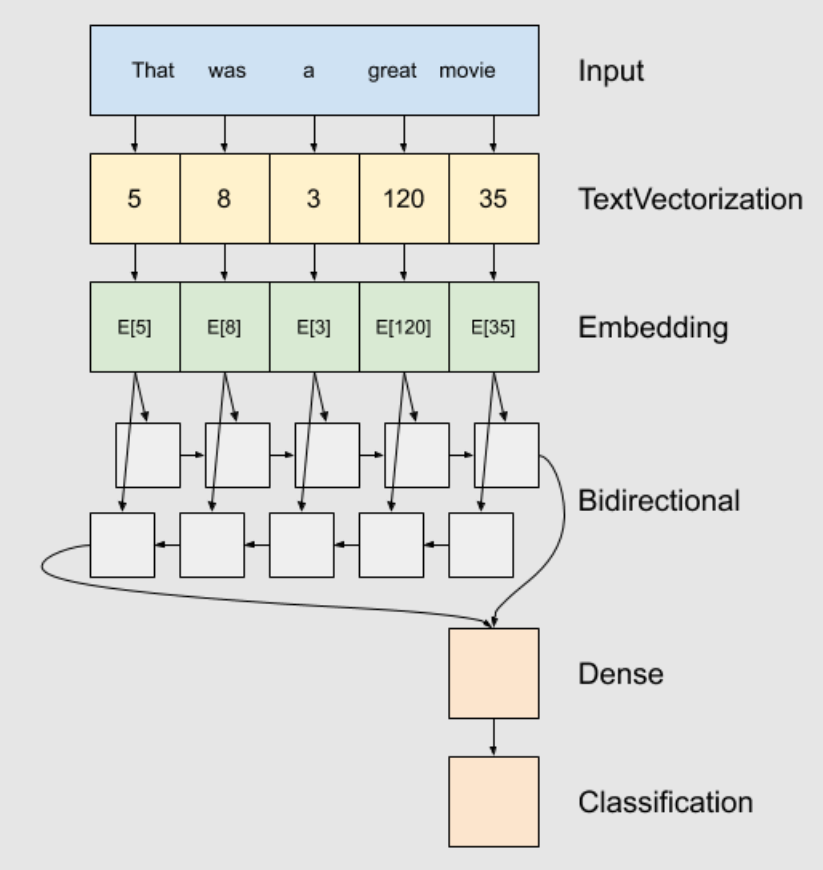
So, it means RNN is suitable for classifying sequence data and news headline is a sequence.

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 similar to the one given below:



Current Results:

Our current classification accuracy is 60% with the vanilla RNN and improved to 62% when we added two LSTM (Long Short-Term Memory) layers. Although the accuracy is not good enough for practical usage but this is the best, we have got among the models we tried.

Challenges:

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"

So, we need to involve feature engineering that would help the model learn the hidden meaning behind the headlines.

Video Demonstration

Rules + Sentiment Model 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

Instruction for Testing SVM:

$make install – installing all necessary packages in local dir

$make data – build corpus

$make train – train model

$make test – test result

Remaining Tasks

In the next two weeks, we will continue to improve our accuracy.

We look forward to use the BERT model a recent break thorough in machine learning. The BERT is a non-directional model instead of reading the words in a sequence it processes each token in context of all token before and after.

Secondly, another bottle neck in improving accuracy is the accuracy of the data labeling. We suspect that some of the data has been incorrectly labeled as well that is confusing our classifier.

Thirdly, we have about 4000 instances in our training data increasing the amount of training data might improve the results further.

So, we are exploring new datasets to use with our model.

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://github.com/plasticityai/magnitude#file-format-and-converter