Analyzing Political Bias through A User-Friendly Interface

Colin Fox Lightfoot
Advisor: James Deverick
The College of William & Mary

Abstract

Many news outlets report stories shown with biased undertones that mislead readers to believe one story over another about the same event. To help people delineate between liberally- and conservatively-biased news articles, I created a website which uses a recursive neural network with long short-term memory nodes trained to identify bias found in news articles. The network achieved an F1 Score of 0.90, and is used to provide one liberally-biased article and one conservatively-biased article side-by-side for a user to read when the user searches for a specific news story.

Acknowledgement

I would first like to thank my advisor, James Deverick, for his guidance and unwavering support throughout my undergraduate career. Without his support, this honors thesis would have never come to fruition and I would have never had such an amazing experience within the Computer Science Department. I would also like to thank Dr. Timothy Davis, Dr. Robert Lewis, and Dr. Mainak Patel for agreeing to serve on my honors committee. I give further thanks to Dr. Robert Lewis for finding the time to help me ensure my machine accepts input in order to classify the bias analyzed from scraped articles. I am grateful for the Keras API documentation and the Stack Overflow community for providing guidance on how best to approach this project and for helping me fix bugs that arose in my code. Lastly, I thank my friends and family for supporting me throughout my academic endeavors and the emotional turbulence that came with them.

Contents

1 Introduction

Many of today's news outlets report stories shown with biased undertones that lead readers to believe one perspective of a story over another. When journalists provide a one-sided view of a story, not only are they overlooking (and possibly concealing) facts, but so too may they sway their listeners to believe these partial interpretations of news. Therefore, in order to see many sides of a multi-faceted issue, one must look at multiple news sources and piece their stories together to gain a more comprehensive understanding of the reported event. Unfortunately, most people either lack the ability or the will to spend the time needed for piecing together their own version of the story before deciding which, if any, version of the story to believe.

Many researchers have begun programming machine learning algorithms for key concepts in textual analysis, especially for classifying text using semantic analysis. While there are machine learning algorithms that classify based off a few-to-several characteristics, there exist more complex algorithms which are able to classify complex objects (e.g. texts) using higher-dimensional storage and calculations to analyze many characteristics during the classification process.

Neural networks are one of these more complex algorithms, making them one of the favored machine learning algorithms for semantic analysis. Neural networks are designed to "learn," mimicking how it is believed humans learn. After learning, neural networks are designed to classify input data in a manner that follows how they were trained, involving the characteristics stated before. The results of many trained algorithms have been surprisingly accurate and have led to some unforeseen breakthroughs in classifying complex subjects. Neural networks have been able to detect nuances in texts to determine such complexities as an author's political stance on the topic.

1.1 Motivation for Thesis

Professor Deverick first proposed this idea to me when he understood I wanted to analyze text, study popular opinions on current events, and expand upon my machine learning skills. Not only is detecting political bias an intriguing concept, but it is an important skill to learn when intending to identify the truth in one's statement. Detecting political bias could also serve an important purpose in bridging the gap between Americans and how they perceive political events and figures with a click of a button. It would be important to identify if some news outlets are hiding details about an event to put their own bias into the story. Therefore I gave the idea a few days of thought which allowed the incubation of the impetus necessary to begin this project.

Almost all news outlets have an associated bias to themselves.[1] Unfortunately, most Americans do not have the time, or even the care, to look at multiple news sources to see as many sides of the story as possible. Some people go as far to view only one side of the political spectrum's news, which leads to misunderstandings on a national level and a more polarized country that is

already deeply-divided on a fundamental level.[2] Some news sites have even begun to outright try and deceive their audiences, such as Fox News with its now-abandoned "Fair and Balanced" news logo.[3] The goal of this project is to bring people to a common ground, ensuring everyone is informed on how popular liberal and conservative news outlets view an event so everyone, regardless of political stance, can understand the viewpoints of one another without the impeding reliance on one news outlet over another.

Another reason for starting this project is because Americans should have easy access publicly available knowledge, and, when analyzing complex topics like political nuances, determine the bias found in information presented or researched. Binary classifications can help make this a reality. Therefore, learning to use machine learning algorithms to detect at least political bias found in texts is a major leap towards achieving my goal.

1.2 Outline

Section 2 discusses related work. Section 3 provides a brief introduction to neural networks, including subsections on neural network sequences, preprocessing said sequences, feedforward neural networks, and backpropagation. Section 4 covers recursive neural networks, including the types of nodes they use, their known uses, and the pros and cons of using them compared to other algorithms. Section 5 discusses the necessary steps to use the project, how the project works, and how the final RNN was defined. Section 5 also provides copies of the discussed code, with comments provided alongside specific lines of code to explain confusing blocks of code. Section 6 details the techniques used to evaluate the validity of the neural network model, as well as possible reasons for the final model's level of accuracy. Section 7 discusses the conclusion reached from this project. Section 8 discusses possible work that can be built off of this project. Section 9 lists all the references used throughout this project.

2 Related Work

Neural networks have been proven able to classify the political bias of texts with astounding results. One study by Stanford researchers Arkajyoti Misra and Sanjib Basak showed that Recursive Neural Network (RNN) models with Long Short Term Memory (LSTM) nodes can be used to convincingly predict the implicit political bias found in some texts, even if there were no words that indicated any conservative or liberal ideologies.[4] They created two neural networks models based on different datasets: one dataset is extracted from speeches of US congressional floor debates known as the Convote dataset[5], while the other is a collection of sentences on multiple socio-political issues by US presidential candidates in most recent history known as the Ideological Books Corpus (IBC) dataset.[6] The first model did not perform well because there was little overlap in content which led to severe training challenges and a good portion of the data did not actually show any political bias that could be detected by a human being. The second model performed more accurately than the first.

Iyyer et. al. performed a study on both datasets as well, sorting the Congressional debates based off partisanship instead of ideology, yielding a model with higher accuracy than their other model trained with the IBC dataset.[8] Training their first model based off partisanship led to noise between moderates in either Party and views in some debates that would usually not correlate with their respective Parties, such as a moderate Republican agreeing with a liberal position on increased gun control. Iyyer et al. also acknowledged the fact that the sarcasm and idioms found throughout their dataset could potentially add noise that could also lower their model's accuracy. They also trained and tested both datasets, with bag-of-words models. Iyyer et al. found that bag-of-words models were less accurate than RNNs, and using larger datasets (or more training data) with shorter sentences are the best datasets to train RNNs. Also, their models had difficulty predicting statements with negations (e.g. "should not" compared to "should") correctly.

The conclusions reached from both groups of researchers are uncongenial when choosing which dataset to use: one found the Convote dataset created a less accurate model, while the other group found the IBC dataset created a less accurate model when the group sorted the Convote dataset a particular way. Considering the fact that the IBC dataset is fairly straightforward to implement, I chose to the IBC dataset[7] over using the Convote dataset to train my RNN model. Both groups do highlight how RNNs perform increasingly better than traditional bag-of-words models when there is an increasingly larger dataset to train machine learning models from.

3 A Brief Introduction to Neural Networks

A neural network (NN), more commonly known as an artificial neural network (ANN), is defined by Dr. Robert Hecht-Nielsen, inventor of one of the first neurocomputers, as "a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs".[9] Neural networks are mathematical algorithms, or actual hardware, that are modeled off the mammalian cerebral cortex's neuronal structure but with much less processing units compared to the billions of neurons in a mammalian brain.[10]

Neural networks are usually organized into layers that are each made up of a number of nodes that are interconnected with the other layers via weighted connections. A neural network typically contains an input layer, a number of hidden layers, and an output layer. The input layer takes in the data and sends said data to various nodes in one or more of the hidden layers through the system of weighted connections discussed above. The data is transformed through the various hidden layers until it reaches the output layer where the neural network returns an output.[10]

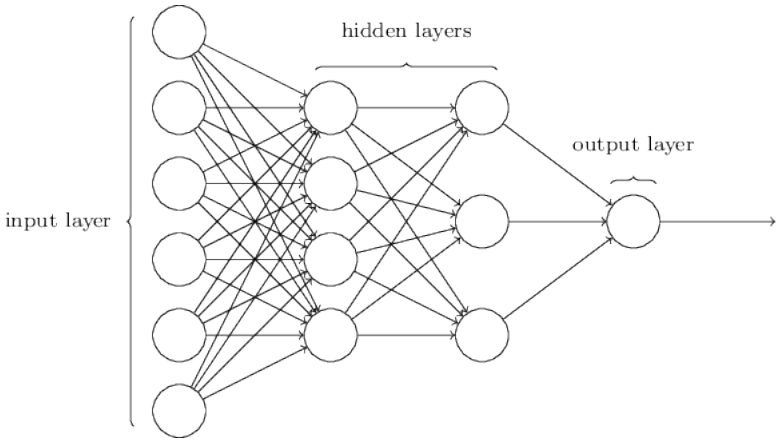

Figure 1: A basic neural network.

Each node contains an activation function in order to produce output. A node's activation function φ takes in the summation of the node's inputs and their respective weights as input and produces the node's output. [12] There are various activation functions that are chosen by the model's designer. I used the softmax activation function for the node in my final layer, defined as

$$softmax(n) = \frac{e^n}{\sum e^n}.[11]$$

The softmax activation function outputs a value between 0 and 1 inclusive, and my RNN measures loss via categorical cross entropy which needs output values

between 0 and 1 to classify each sequence. The neural network's nodes undergo training through a learning rule which modifies the weights of the connections between nodes according to the data inputted during he training of the neural network.[10] Basically, a neural network learns to recognize certain attributes by example, similarly to how we learn to classify whether a certain creature is a dog or a cat. The learning rule mainly used, and is used for this project, is known as backpropagation and is discussed in Section 3.4.

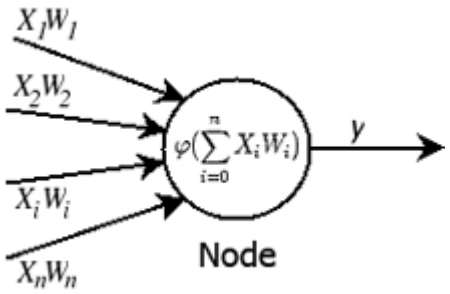

Figure 2: A basic node.

Training with this learning rule is repeatedly cyclically through cycles known as time steps or epochs (each time a network is given more input). With this learning rule, the neural network makes guesses on which of the two classifications an input is and adjust its weighted connections accordingly to become more accurate. The more epochs the more accurate the neural network model is on classifying the given dataset (note that to ensure the model is the most accurate when given input of the same type from a different dataset, verification performed by a technique known as cross validation - described in Section 5.1 - is tested against the model). [10] Once the model is trained within the researcher's accepted level of accuracy, the neural network can be used to take in data and output its defined classifications without readjusting the weights. The model can also be retrained in order to refine its accuracy.

3.1 Sequences

The input to any neural network is known as a sequence. An input sequence is represented as $(x^1, x^2, ..., x^S)$ and a target sequence is represented as $(y^1, y^2, ..., y^T)$, , where each data point x^t, y^t are real-valued vectors. The variables S, T represent the maximum allowed length of any input sequence where $S, T \geq 1$. Therefore the input and output may be single data points to countably infinite lengths and do not have to match dimensions.

A training set is a set of examples where each example composed is usually a (input sequence, target sequence) pairing. My project's dataset's examples are composed in an (target sequence, input sequence) pairing. The data points neural networks predict are denoted as $\hat{y}^t$. Sequences' x^t data points, except

for some Recursive Neural Network (RNN) models, denote data across a span of time.[13]

RNNs have recently been shown to successfully use non-temporal sequence data for analyses, such as applications to genetic data,[14] as long as each sequence has a defined order. RNNs also apply its sequencing of data with natural language processing.[13] For instance, the word sequence, "I like Ike", would map like so (assuming no preprocessing is done): x^1 = "I", x^2 = "like", x^3 = "Ike". This process of breaking word sequences about is known as tokenizing. Word sequencing is useful when one is trying to analyze the sentiment of text or categorizing text into one of two categories, such as what this project accomplishes. The strings of text are then transformed into unique numbers so the strings can be used in the model's mathematical algorithm.

3.2 Preprocessing Sequences

Preprocessing is useful to perform before word sequencing in order to train and test on similar data. For instance, without preprocessing the three sentences "I like Ike", "I like Ike." and "i like ike" would yield three different data points for "Ike" ("Ike", "Ike.", and "ike"), whereas if each sentence's uppercase characters were transformed into lowercase characters and all punctuation was striped then there would be only one data point for "Ike" ("ike"). Have a lower variety of data when already training a neural network on at least thousands of, if not more, words lowers the accuracy of neural network's predictions significantly. The lower accuracy is due to a lack of storage in a neural network to accommodate for all the different versions of words (e.g. Section 3.2's "Ike" example). Some neural networks also need to pad the input in order to train, or test, the model or use the input to generate a prediction.

RNNs are one of the neural networks that rely on padding and cutting, especially for textual data when analyzing word sequences. Padding is adding a certain character a bunch of times to either the beginning (pre-pending) or the end (post-pending) of statements in a dataset so all statements in the dataset have the same amount of characters. Cutting is analyzing only a certain amount of characters in a statement in order to ensure all statements have the same amount of characters.[15] In this project, a large amount of padding was used to ensure large text articles could be ran in the RNN model in order to make a prediction on said articles' political biases.

3.3 Feedforward Neural Networks

Feedforward networks are neural networks which prohibit cycles in directed graphs. The absence of cycles allows the layered formation seen in neural networks. The formation of layers allows output to be directed from one layer to the next, organizing the network's structure. The input x to a feedforward network initially sets the values of the lowest layer, shown in Figure 3 as the "Input Layer". Each higher layer is then successfully computed until output is generated at the topmost layer $\hat{y}$, shown in Figure 3 as the "Output Layer".

These networks are mostly used for supervised learning tasks, such as classification. RNNs are a type of feedforward networks and will be further discussed in Section 4. These types of neural networks "learn" by iteratively updating each of the weights to minimize a loss function, $L(\hat{y}, y)$, that focuses on the distance between the output $\hat{y}$ and the target y. The most popular algorithm for teaching these networks is known as the process called "backpropagation".[16]

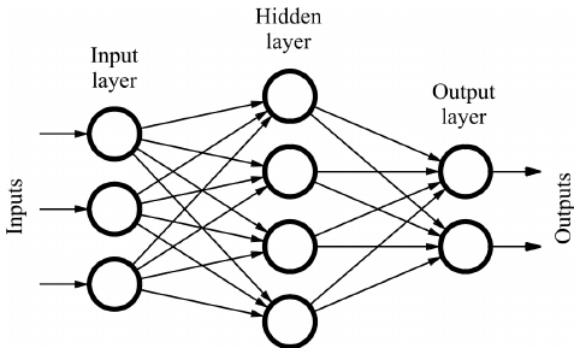

Figure 3: A basic feedforward neural network where layers are successively computed based off computations from the lower layers. [19]

3.4 Backpropagation

Backpropagation, short for "backward propagation of errors" [16] is most successful algorithm for training neural networks. This algorithm uses the chain rule to calculate the derivative of the loss function L with respect to each parameter in a network in order to perform the adjustment of weights within said network using gradient descent.[13] This is done by calculating the gradient of the network's error function with respect to the neural network's weights.

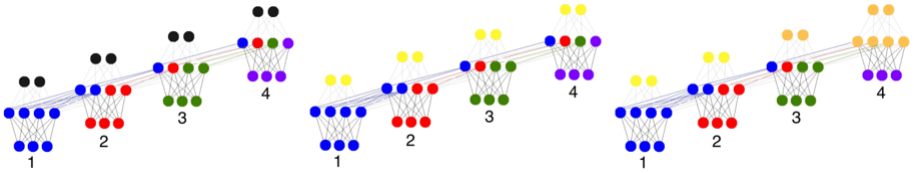

Figure 4: Black nodes are the initial prediction, errors are yellow, and the gradients are orange. The gradients are calculated starting from the final layer, Layer 4, and move to the initial layer, Layer 1, under backpropagation.[17]

The "backwards" part of the term highlights the fact that gradient calculation and weight adjustment is first performed on the network's final layer of weights with the gradient calculation and weight adjustment of the network's

first layer of weights being performed last. This allows the algorithm to reuse partial computations of the gradient from one later into the computation of the gradient in the previous layer, allowing for a backwards flow of the error information throughout the network. This backwards flow of error leads to efficient gradient computations at each layer rather than calculating the gradients of each layer separately. [16]

To calculate the gradients in a feedforward neural network, first an example is propagated forward as described in Section 3.3 to produce a value v_j at each node and outputs $\hat{y}$ at the topmost layer. The the loss function value $L(\hat{y}, y)$ is computed at each output node k. Then for each output node k, we calculate

$$\delta_k = \frac{\partial L(\hat{y}_k, y_k)}{\partial \hat{y} k} \bullet \varphi'_k(a_k).$$

And given the value δ_k for each output node, for each node in the layer before we can calculate

$$\delta_j = \varphi'(a_j) \sum_k \delta_k \bullet w_{kj}$$

where w_{kj} represents the weight for node j for incoming node k, $\varphi(x)$ is the activation function, and a_i is the product sum bias for node i.[13][16] This calculation is performed iteratively through each earlier layer to yield δ_j for every node j given the δ_k values for each node in the later layers connected to j. Each value δ_j represents the derivative of the total loss function with respect to the node's incoming activation:

$$\delta_j = \frac{\partial L}{\partial a_j}.$$

And, given the values v_j calculated during the forward pass and the values δ_j for each node connected to node j that was calculated during the backward pass, the derivative of the loss L with respect to a given weight is

$$\frac{\partial L}{\partial w_{kj}} = \delta_j v_k.$$

The loss is important to study, as the resulting scalar value is the amount weight w_{kj} is adjusted by for the current backward pass.[13]

4 Recursive Neural Networks

Recurrent Neural Networks (RNNs) are neural networks designed to make use of sequential information. Neural networks normally assume all inputs and outputs are independent of each other, but this is not useful for predicting what information might come next. RNNs are *recurrent* because they perform the same computations for every element in a sequence while the output is dependent on the model's previous epochs. RNNs can have dependent output because their nodes hold previous calculations.[18]

The figure below shows a RNN being unfolded, meaning that the amount of layers in the network is equivalent to the amount of words in the input sequence. In the figure below, x_t is the input at time step t, s_t is the hidden state at time step t, and o_t is the output at time step t. The data is stored in the hidden states s_t of a RNN and is calculated based on the previous hidden state and the input at the current time step:

$$s_t = f(Ux_t + Ws_{t-1}).$$

The output o_t of a RNN (e.g. the prediction of the next word in a sentence) would be a vector of probabilities across the remembered words of previous time steps stored in the model: $o_t = \text{softmax}(Vs_t)$.

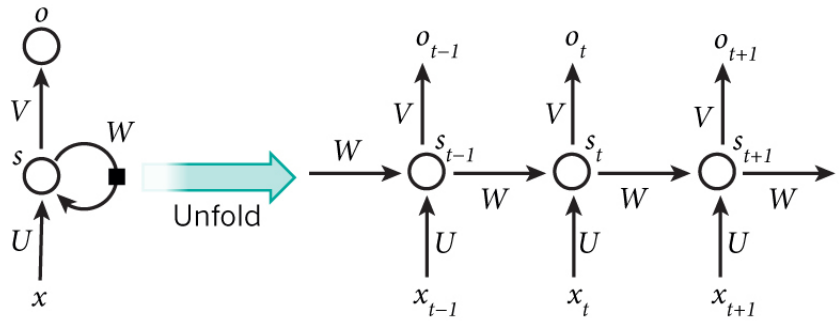

Figure 5: An unfolded recursive neural network.[18]

My model stores a large amount of words and tries to predict whether the sequence of text is largely liberally or conservatively biased based off the inputted words and outputs a scalar value where [0, 0.5) means the text is liberally biased, 0.5 means the text has no noticeable bias, and (0.5, 1] means the text is conservatively biased.[18]

4.1 Long Short Term Memory Nodes

Traditional RNNs experience gradient calculation problems during the gradient backpropagation phase where the gradient is multiplied too many times (meaning the network has too many time steps) by the weighted matrix used

with the connections between the nodes of the recurrent hidden layer. In other words, the size of the weights can impact a RNN's training. For instance, if the values in the weighted matrix are less than 1, then the multiplied gradients can approach zero quickly making the training pace incredibly slow or stop learning altogether. This predicament is known as the vanishing gradient dilemma. On the other hand, if the weighted matrices are much larger than 1, or there are many time steps, then the calculated gradients can become so large that the model's learning diverges and leads to an inaccurate model. This phenomenon is known as the exploding gradients dilemma. Both of these issues led to the creation of the Long Short Term Memory (LSTM) node that is composed of a memory cell.[20]

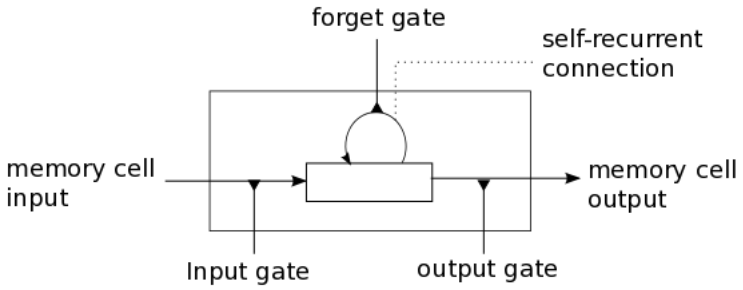

Figure 6: A simple LSTM node.[20]

A memory cell is comprised of at least these four attributes: an input gate, a self-recurrent connection (a pointer to itself), a forget gate, and an output gate. The self-recurrent connection has a weight of 1, preventing any parameters and input from causing either of the gradient dilemmas while allowing a memory cell's state to remain the same from one time step to another. Each gate controls the interactions between the memory cell and other nodes. The input gate, for instance, controls whether or not an incoming signal can alter the state of the memory cell or block it while the output gate controls whether or not the state of the memory cell can affect other nodes. The forget gate controls whether to run the self-recurrent connection and remember the previous state or not run the connection and forget its state.

The following equations detail how a memory layer of nodes is updated every time step t. The variable x_t is the input to a memory layer at time t, $W_i, W_f, W_c, W_o, U_i, U_f, U_c, U_o$, and V_o are the weight matrices that influence the nodes (as seen in Figure 5), $\varphi(x)$ represents the activation function as discussed in Section 3, and b_i, b_f, b_c, and b_o are the bias vectors received from the gates. The value for the input gate i_t at time t is computed like so:

$$i_t = \varphi(W_i x_t + U_i y_{t-1} + b_i).$$

The value for the forget gate activation function f_t at time t is computed like

so:

$$f_t = \varphi(W_f x_t + U_f y_{t-1} + b_f).$$

The value of the output gates at time t are computed with the following function:

$$o_t = \varphi(W_o x_t + U_o y_{t-1} + V_o C_t + b_o).$$

Candidate, or possible, values $\widetilde{C}$ are also calculated to readjust the state C_t at time t:

$$\widetilde{C} = tanh(W_c x_t + U_f y_{t-1} + b_f).$$

The new state of each cell C_t at time t is then readjusted:

$$C_t = i_t \times \widetilde{C}_t + f_t \times C_{t-1}.$$

Finally, the output y_t at time t is calculated with the following equation:

$$y_t = o_t \times tanh(C_t).$$

Therefore, from an input sequence $x_1, x_2, ..., x_n$ the LSTM nodes will produce an output $y_1, y_2, ..., y_n$ that is then averaged over all the time steps to create the output y for each node.[20]

4.2 Known Uses

RNNs have shown great success in many natural language processing tasks, from predicting stock prices[22] to studying an artist's lyrics and writing lyrics with the artist's lyrical style.[23] The most commonly used RNNs are ones that use LSTM nodes, like the ones this project uses. This is because LSTM-based RNNs can capture long-term dependencies, such as remembering the most frequently used words, that prove beneficial to predicting data based off of specific periods in time (i.e. stock price predictions). And, like how humans learn a language through observing a set of words multiple times, neural networks can "learn" to generate lines of text by using words or sets of words most frequently used in a dataset.

The neural Turing machine (NTM) combines RNNs with addressable memory in order to perform complex algorithmic tasks, such as sorting. The RNN part of the NTM executes the task while the Turing machine part of the NTM uses write and read heads to store and read memory from a memory buffer, such as a tape.[13]

RNNs have even been implemented to generate captions for pictures given a training set of images x and target captions y.[24] Sutskever et al. where even able to design an RNN that translated text from English to French accurately. [25] If an RNN can have a large enough dataset that provides an input x and a target y, then, from the experiments run and the results recorded, RNNs can be trained to output fairly accurate predictions on what the model designer wants to predict. But, compared to other machine learning algorithms, an RNN is not suited for certain machine learning tasks.

4.3 RNNs Compared to Other Models

SVMs and Naive Bayes models are not useful for textual analysis either. [27] This is because neither learn any structure and instead learn bag-of-words representations. Therefore, with an SVM, the only information received is knowing the words in the input and how frequently they are used. But because RNNs learn structures, you get this information as well as the context in which the most frequent words (denoted as keywords below) are used. Iyyer et al. tried comparing bag-of-words models to RNNs and found RNNs to be clearly more accurate models for textual analysis.[8] Compared to other neural networks, RNNs are more aptly designed for textual analysis.

While RNNs learn to recognize the long-term time-dependencies (contexts) in which keywords are used, CNNs learn the exact situation in which the keywords are used. This is because RNNs are time-dependent whereas CNNs are structurally-dependent. For example, if a CNN is learning to classify texts as either liberally or conservatively biased, the model looks for parts of sentences with learned keywords that it knows and pieces them together in order to base its conclusions. RNNs instead look to see the order in which the words are processed (e.g. if a keyword is preceded by a certain other keyword learned to change the initial keywords meaning), placing more importance on the latest usage of a keyword, to determine a text's bias.[26] Since CNNs must base their classifications off known word sequences rather than approximating their classifications based off the context in which certain keywords were remembered to have been labelled, they are not the best for textual analysis. Compared to other time-dependent machine learning algorithms, RNNs are still favored for textual analysis problems.

Markov chains and hidden Markov models (HMMs) are also time-dependent algorithms used for classifying input into distinct states. But their approaches to sequence classifications are limiting because they inefficiently scale in time the more states S are used at an asymptotic rate of $O(|S|^2)$. This is because more states are needed as the amount of inputted sequences increases since each hidden state in an HMM is dependent on previous states. And the table keeping track of the probability of moving between states is of size $|S|^2$. Therefore, after a lot of input is placed in a Markov model, computation becomes impractical, preventing a model from learning a large set of contexts per keyword.[13] The lack of contexts then would lead any HMM to be less accurate compared to an RNN model since an RNN model can capture many more contexts per keyword.

5 Implementation of Project

To train the LSTM-based RNN for this project, first tune the hyper-parameters to specific scalar values and then run LSTM-RNN.py. Repeat the process until you achieve possible the highest accuracy possible with similar percentages for both the evaluate() and validate() methods (this is elaborated in Section 6.1. The code will automatically store the latest algorithm as a file to be read by webdriver.py. Then, to run the project, run Webdriver.py and a Python Flask-powered local host will begin to run. Next, use a web browser to reach the local host to arrive on Website.html, the website interface for this project. Now simply type in a news topic to begin running the background tasks of finding the most popular conservative and the most popular liberal news coverage of said topic and putting the links to both side-by-side onto the website. For instance, if one were to search the term "Trump", then the project would find the first result on Google it deems to be conservatively-biased and pastes its link in the website's table's "Popular Conservative View" column. The process is the same for liberally-biased articles.

Webdriver.py is responsible for requesting and returning the website's user's search query as a link to politically-biased articles as described in the paragraph above. To complete these tasks, Webdriver.py opens up an invisible browser and copies the user's query onto the invisible browser's Google search engine. Webdriver.py then waits for Google's results page to load the "most relevant" URLs and begins reading and opening the first accepted URL, runs the articles' text through the RNN model created. Only the listed URLs in Webdriver.py are accepted and can therefore be analyzed; the list is based on the [21]'s list on the top-ranked news outlets. If the model believes the article is biased towards one side of the political spectrum than the other and the URL for the biased side has not been filled by another URL, then the spot is now filled with a URL link to the article. This process is then reiterated for each article whose URL is listed in the results and continues down the page, and sometimes even onto other results pages, until both the conservative and liberal URL link slots are filled on the website or there are no more results to analyze.

I used a multiple-layered RNN for my classification task. The layer after the input layer was an embedded layer which vectored each word in a sequence for processing. I then added a recurrent dropout layer, allowing 20% of words and their associated contexts to be forgotten at the beginning of each epoch to mimic long-term memory loss. The next layer was comprised of 300 LSTM nodes to allow the neural network to mimic long-term memory. Another recurrent dropout layer was then added with the same dropout rate to mimic short-term memory loss. Finally, a softmax activation layer was added to provide output known to closely approach its correct classes' attributed scalar value. The categorical cross entropy loss was then minimized by the 'Adam' optimizer while the batch size found to yield the most accuracy was 32. The RNN model was trained on 67% of the dataset while the accuracies and F1 Score were calculated on the remaining 33% of the dataset.

The following three subsections contain the actual code used to implement

this project, with Section 5.1 being the RNN trained and used in the project, Section 5.2 being the website displayed of which the user interacts with in order to run a search query on the project, and Section 5.3 contains the file with code that provides the functionality for opening and running the saved RNN, as well as querying and collecting data to be analyzed by the RNN. All files pertaining to this thesis are stored at this URL address: https://github.com/hydure/HonorsProject, including the final stored RNN model used by Webdriver.py.

5.1 LSTM-RNN.py

```
import numpy as np    # Linear Algebra
import pandas as pd   # Data Processing, CSV file I/O (e.g. pd.read_csv)
import re             # Regular Expression changing (CSV file cleanup)

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D,
    Dropout
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical

########################### HYPERPARAMETERS ##############################

SEED = 7                      # Fixes random seed for reproducibility.
URL = 'ibcData.csv'           # Specified dataset to gather data from.
SEPERATOR = ','               # Seperator the dataset uses to divide data.
RANDOM_STATE = 1              # Pseudo-random number generator state used
                              # for random sampling.
PADDING_LENGTH = 1000         # The amount of words allowed per piece of
                              # text.
HIDDEN_LAYER_SIZE = 300       # Details the amount of nodes in a hidden
                              # layer.
TOP_WORDS = 5000              # Most-used words in the dataset.
MAX_REVIEW_LENGTH = 500       # Char length of each text being sent in
                              # (necessary).
EMBEDDING_VECTOR_LEN = 128    # The specific Embedded later will have
                              # 128-length vectors to represent each word.
BATCH_SIZE = 32               # Takes 64 sentences at a time and
                              # continually retrains RNN.
NUMBER_OF_EPOCHS = 5          # Fits RNN to more accurately guess the
                              # data's political bias.
VERBOSE = 2                   # Gives a lot of information when
                              # predicting/evaluating model.
NONVERBOSE = 0                # Gives only results when
                              # predicting/evaluating model.
VALIDATION_SIZE = 1000        # The size that you want your validation
```

```
                            # sets to be.
DROPOUT = 0.2               # Helps slow down overfitting of data
                            # (slower convergence rate)
FILE_NAME = 'finalizedModel.h5' # File LSTM RNN is saved to so it can be
                            # used for website

############################ FUNCTIONS ################################

# Function to see what your CSV file looks after it is cleaned up
def debugAfterCleanUp(data):
    print(data)
    print(data[ data['bias'] == 'Conservative'].size)
    print(data[ data['bias'] == 'Liberal'].size)

# Checks the shape of the below four datasets
def checkShapes(X_train, X_test, Y_train, Y_test):
    print(X_train.shape,Y_train.shape)
    print(X_test.shape,Y_test.shape)

# Prints a summary of the model
def printModelSummary(model):
    print(model.summary())

# Evaluates the model
def evaluate(model, X_test, Y_test):
    score, accuracy = model.evaluate(X_test, Y_test, verbose = VERBOSE)
    print("Evaluation:")
    print(" F1 Score: %.2f" % (score))
    print(" Accuracy: %.2f%%\n" % (accuracy * 100))

# Validates the model by extracting a validation set and
# measuring the correct number of guesses
def validate(model, X_test, Y_test):

    X_validate = X_test[-VALIDATION_SIZE:]
    Y_validate = Y_test[-VALIDATION_SIZE:]
    X_test = X_test[:-VALIDATION_SIZE]
    Y_test = Y_test[:-VALIDATION_SIZE]
    score, accuracy = model.evaluate(X_test, Y_test, verbose = VERBOSE, \
                        batch_size = BATCH_SIZE)
    print("Validation:")
    print(" F1 Score: %.2f" % (score))
    print(" Accuracy: %.2f%%\n" % (accuracy*100))
    print("Getting percentage of correct guesses per political
        leaning...\n")

    conCount, libCount, conCorrect, libCorrect = 0, 0, 0, 0

    for x in range(len(X_validate)):
```

```
        result = model.predict(X_validate[x].reshape(1,
            X_test.shape[1]), \
                                batch_size = 1, verbose = VERBOSE)[0]

        if np.argmax(result) == np.argmax(Y_validate[x]):
            if np.argmax(Y_validate[x]) == 0:
                libCorrect += 1
            else:
                conCorrect += 1

        if np.argmax(Y_validate[x]) == 0:
            libCount += 1
        else:
            conCount += 1

    print("Conservative Accuracy:", conCorrect / conCount * 100, "%")
    print("    Liberal Accuracy:", libCorrect / libCount * 100, "%\n")

def save(model):
    model.save(FILE_NAME)           # Creates a HDF5 file to save the
        whole model
    print("Model saved.\n")         # (e.g. its architecture, weights,
        and optimizer rate)

############################## PREPARE DATA ################################

# Read the data from the CSV file by column
data = pd.read_csv(URL, header = None, names = ['bias', 'text'], sep =
    SEPERATOR)

# Make all characters lowercase if they are not already
data['text'] = data['text'].apply(lambda x: x.lower())

# Take out all superfluous ASCII characters
data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]',
    '', x)))

# Eliminate duplicate whitespaces
data['text'] = data['text'].apply((lambda x: re.sub(r'\s+', ' ', x)))

#debugAfterCleanUp(data);

# Preprocess texts
tokenizer = Tokenizer(num_words=TOP_WORDS, split=' ')
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)
X = pad_sequences(X, maxlen=PADDING_LENGTH)

# Declare the train and test datasets
Y = pd.get_dummies(data['bias']).values
```

```
X_train, X_test, Y_train, Y_test = \
          train_test_split(X, Y, test_size = 0.33, random_state = SEED)

#checkShapes(X_train, X_test, Y_train, Y_test)

############################ TRAIN MODEL ################################

# Define the model
model = Sequential()
model.add(Embedding(TOP_WORDS, EMBEDDING_VECTOR_LEN,
    input_length=X.shape[1]))
model.add(SpatialDropout1D(DROPOUT))
model.add(LSTM(HIDDEN_LAYER_SIZE))
model.add(Dropout(DROPOUT))
model.add(Dense(2, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', \
              metrics=['accuracy'])

#printModelSummary(model)

# Stops fitting the model when the improvement is negligible to
# help prevent over-fitting
earlyStopping = keras.callbacks.EarlyStopping(monitor='val_loss',
    min_delta=0, patience=0, verbose=0, mode='auto')

# Fit the model
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), \
          epochs=NUMBER_OF_EPOCHS, batch_size=BATCH_SIZE, \
          callbacks=[early_stopping])

print("*" * 75)

# Evaluate the model
evaluate(model, X_test, Y_test)

# Validate the module
validate(model, X_test, Y_test)

print("*" * 75 + '\n')

# Save the model
save(model)

###########################################################################
```

5.2 Website.html

```
<!doctype html>

<html>
    <head>
        <title>Fair & Balanced News</title>
        <meta charset=utf-8>
    </head>
    <body>
        <h1 style='text-align: center;'>Fair & Balanced News</h1>
        <div style="width:500px; margin: 0 auto">
            <h4 style='text-align: center;'>What news do you want to
                search for?</h4>
            <form method=post action="/results" style="text-align:
                center;">
                <input type="text" id="inputString" name="inputString"/>
                <input type=submit value='Search' name='search_btn'>
            </form>
            <h3 style='text-align: center;'>{{ errMessage }}</h3>
            <table align="center" border="1">
                <tr>
                    <th>Most Popular Liberal View</th>
                    <th>Most Popular Conservative View</th>
                </tr>
                <tr>
                    <td style='text-align: center;'><a href="{{ liberalURL
                        }}">{{ lLinkName }}</a></td>
                    <td style='text-align: center;'><a href="{{
                        conservativeURL }}">{{ cLinkName }}</a></td>
                </tr>
            </table>
        </div>
    </body>
</html>
```

5.3 Webdriver.py

```
from flask import Flask, render_template, request, redirect, url_for
from wtforms import Form, TextAreaField, validators
from keras.models import load_model
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from string import Template
import time, requests, numpy
```

```
from bs4 import BeautifulSoup

############################ GLOBAL VARIABLES ############################

# Made these global so both @app.route functions and checkURL can access
    them.
conservativeURL = ' '
liberalURL      = ' '
cLinkName       = ' '
lLinkName       = ' '
errMessage      = ' '
loadedModel     = load_model('finalizedModel.h5')
MAX_REVIEW_LENGTH = 500
TOP_WORDS = 5000            # Most-used words in the article.
NEUTRAL = 0.5               # Article is predicted to not be politically
                            # biased.

############################## FUNCTIONS #################################

# Runs article through algorithm and determines if it is politically
    biased and how.
# If politically biased and its biased URL isn't filled, fill its bias'
    URL slot.
def checkURL(url, text, linkName):
    global conservativeURL
    global liberalURL
    global cLinkName
    global lLinkName

    # Preprocess article to predict its political bias
    tokenizer = Tokenizer(num_words=TOP_WORDS, split=' ')
    tokenizer.fit_on_texts([text])
    X = tokenizer.texts_to_sequences([text])
    X = pad_sequences(X, maxlen=1000)
    print("Article has been preprocessed and a prediction is being
        made...")

    # Predict if the article is politically biased, and, if so, which
        way is it biased.
    prediction = loadedModel.predict(X)

    # Fill proper URLs based on prediction.
    if conservativeURL == ' ' and prediction[0][0] > NEUTRAL:
            conservativeURL = url
            cLinkName = linkName

    if liberalURL == ' ' and prediction[0][0] < NEUTRAL:
            liberalURL = url
            lLinkName = linkName
```

```
############################## FLASK APP ###################################

app = Flask(__name__)

class SearchForm(Form):
    inputString = TextAreaField('', [validators.DataRequired()])

@app.route('/')
def index():
    return render_template('Website.html', cLinkName=cLinkName,
        lLinkName=lLinkName, \
                        conservativeURL=conservativeURL,
                            liberalURL=liberalURL, \
                        errMessage=errMessage)

@app.route('/results', methods=['GET', 'POST'])
def results():
    global conservativeURL
    global liberalURL
    global cLinkName
    global lLinkName
    global errMessage
    global tokenizer

    # Need to clear these fields to run another query
    conservativeURL = ' '
    liberalURL     = ' '
    cLinkName      = ' '
    lLinkName      = ' '
    errMessage     = ' '

    form = SearchForm(request.form)
    if request.method == 'POST' and form.validate():

        inputString = request.form['inputString']
        driver = webdriver.PhantomJS() # Creates an invisible browser.
        driver.get('https://google.com/') # Navigates to Google.com.

        # Assigns query to Google Search bar.
        searchBarInput = driver.find_element_by_name('q')

        if inputString != '':

            # what you are searching for
            searchBarInput.send_keys(inputString + " news")
            # Hit <RETURN> so Google begins searching
            searchBarInput.send_keys(Keys.RETURN)
            time.sleep(1) # sleep for a bit so the results webpage will
                be rendered
```

```
# Scrape liberal and conservative websites that are in the
    top 10 news websites
# (top 10 according to
    http://blog.feedspot.com/usa_news_websites/'s metrics).
urls = driver.find_elements_by_css_selector('h3.r a')

# Continue mining until conservative- and liberalURL are found
while conservativeURL == ' ' or liberalURL == ' ':

    urls = driver.find_elements_by_css_selector('h3.r a')
    for url in urls:

        if conservativeURL != ' ' and liberalURL != ' ':
            break

        # Need to remove end of links that make some webpages
        # impossible to create a usuable link for my webpage.
        stoppingPoint = url.get_attribute('href').index('&')
        url = url.get_attribute('href')[29 : stoppingPoint]
        # print(url)

        # The queried search page is a url and needs to be
            skipped.
        if '?q=' in url:
            continue

        if "cnn.com" in url: # 1
            #cLinkName = "CNN Article"
            #conservativeURL = url
            linkName = "CNN Article"

            lookAtPage = requests.get(url)
            soup = BeautifulSoup(lookAtPage.text,
                "html.parser")
            paragraphs = soup.find_all('div',
                {"class":"zn-body__paragraph"})
            text = ''
            for paragraph in paragraphs:
                text = text + paragraph.text
            #print(text)
            checkURL(url, text, linkName)

        if "nytimes.com" in url: # 2
            #lLinkName = "NY Times Article"
            #liberalURL = url
            linkName = "NY Times Article"
            lookAtPage = requests.get(url)
            soup = BeautifulSoup(lookAtPage.text,
                "html.parser")
```

```
                    paragraphs = soup.find_all('p',
                        {"class":"story-body-text story-content"})
                    text = ''
                    for paragraph in paragraphs:
                        text = text + paragraph.text
                    checkURL(url, text, linkName)
                    #print(text)

                # More listed in my GitHub repository, "HonorsProject"

            if conservativeURL != ' ' and liberalURL != ' ':
                break

            # Go to the next page, if possible, to continue process.
            try:
                nextPage =
                    driver.find_element_by_link_text("Next").click()

            except:
                errMessage = "Could not find enough sources on topic."
                break

        # Optionally save a screenshot to see operation.
        # driver.save_screenshot('screen.png')

    driver.quit()
    return redirect(url_for('index'))

##############################################################################

if __name__ == '__main__':
    app.run(debug=True, use_reloader=True)

##############################################################################
```

5.4 User Interface

Figure 7: The user interface for my project.

6 Evaluation of Project

In order to tune my final model to the highest accuracy achievable given a certain size of test data, I measured how my test models fared under certain hyper-parameters. The metrics I used were each model's F1 Scores, overall accuracy, the percentage of correct conservative classifications, and the percentage of correct liberal classifications.

6.1 Measurements

A model's F1 Score is a measure of accuracy which considers both the model's precision and recall. Precision is measured as

$$\frac{true\ positives}{false\ positives + true\ positives},$$

while recall is measured as

$$\frac{true\ positives}{false\ negatives + true\ positives}.$$

Measuring the accuracy of the model by the amount of false positives and false negatives it identifies helps show the overall strength of the model in regards to not predicting Type I and II errors. The formula for calculating the F1 Score is:

$$F1\ Score = 2 \times \frac{precision \times recall}{precision + recall}.[28]$$

The documentation for recording a model's precision and recall is limited and was unable to be used, whereas their was a function, evaluate(), listed which

output the model's F1 Score for my RNN model. This function was used to calculate my final model's F1 Score, 0.90. My model's F1 Score turned out to be higher than Misra's and Basak's F1 Score when ran on the IBC dataset, 0.718 where they used a larger percentage of the IBC dataset as a training set (80% compared to my size of 67% of the dataset).[4] An F1 Score varies between 0 and 1 with the goal of getting a model to achieve a score of 1. Therefore, based on the testing dataset, the precision and recall were fairly high values, showing the model is fairly robust in classifying political bias.

But after running a cross-validation analysis on the model to ensure I did not over-train my model, the final model's accuracy against the testing dataset partition, 57.49%, turned out to be fairly low. This is because the conservative test dataset yielded an accuracy of 57.065% while the smaller liberal test dataset output an accuracy of 65.625% during the model's cross-validation. Therefore, given a subset of the IBC dataset was my model's only training data, my model can fairly predict whether or not a sentence hold an objectively liberal bias while having a somewhat hard time predicting whether a sentence has an objectively conservative bias.

6.2 Possible Noise

As Iyyer et al. noted,[8] the IBC's lack of data meant my RNN model did not have a lot of data to train off of compared to the amount of liberally- and conservatively-biased text available to the public. And, because the IBC dataset's samples were singular sentences, there were not as many keywords and patterns to record as samples with larger amounts of texts would have. The sentences were also long, allowing most sentences to likely lose their meaning as contexts and keywords were recurrently forgotten when the RNN model was learning the latest input's keywords and context. But, after only being tested against samples containing similar singular long sentences and classifying them fairly accurately, there is a high probability that the articles ran through the model when the server is running will have more than enough keywords and contexts for the RNN model to accurately classify the texts.

The model's conservative classification accuracy being lower than its liberal classification accuracy could be attributed to the fact that there were more conservative samples. Since the model can only remember so many contexts and keywords, it would most likely be more inaccurate when being tested against a larger conservatively-biased test set than it was a liberally-biased test set. This is because the greater amount of conservative samples most likely lead to more conservative contexts and keywords to remember.

7 Conclusion

In this work I have shown how an RNN with a layer of LSTM nodes can accomplish the ambitious task of determining political bias in news articles with a decent cross-validation accuracy of 57.49% and a high cross-validation F1 Score of 0.9. While there is already research on this topic, Professor Deverick and I took the idea one step further and decided to make my model easily accessible and usable to the public through a simple user interface shown in Section 5.4. This was accomplished by creating a website with a search bar that, when a news topic is typed in the search bar and the "Search" button is clicked, runs a query through Google's search engine in an invisible window and looks at each URL rendered on Google's results page matching one of the trusted websites and then scrapes through the entire article. Next, the scraped text is then run through an RNN model trained using the IBC dataset. The model outputs a value between 0 and 1, and if the output is less than 0.5, then the model is liberally-biased, and if the output is greater than 0.5 the model is conservatively-biased. If the article fulfills a bias that is not already filled, then its URL is shown on the web-page. This process continues until the most relevant liberally-biased article and conservatively-biased articles are found, providing the user with a fair and balanced opportunity to see both political views (if either exists) about the news topic in question.

Using the web-page, anyone can find a liberal and a conservative stance for any existing news topic being researched, ensuring users of the web-page access to a listing of relevant articles about a reported event that is statistically-proven to be fair and balanced. Exactly how a neural network is calculated mathematically and the reasoning for its design has also been discussed; the characteristics, along with explanation, of neural networks and what these algorithms actually output have been provided in this way as well. Furthermore, RNNs should be chosen over other models for textual analysis, especially in regards to sentiment analysis (e.g. political bias analyses). The importance of using LSTM nodes to mimic long-term memory has led to the RNN becoming one of the best, if not the best, machine learning algorithms to use for long-term contextual analysis. The design can be replicated in another experiment by following the setup previously described. The goal was to create a platform for someone to search the news in a fair and balanced way, and was accomplished by using a fairly accurate machine learning algorithm to classify articles as being liberally- or conservatively-biased on a web-page designed to perform these tasks. There is currently no other platform that accomplishes this feat and I am glad to have provided one for the public to use.

8 Future Work

When I have spare time in the future, I plan to create a larger dataset of speeches, transcripts, and statements by people identified as having mostly liberal or conservative values due to the higher accuracy achieved with models that have larger training datasets. I will then continue to train my project's RNN to see if I can achieve an accuracy greater than what either Misra and Basak or Iyyer et. al achieved with their LSTM-based RNNs given that I will have a larger dataset to work with. I may also want to base which news outlets I scrape from based on their overall trustworthiness, as described in [1], rather than scraping from Feedspot's Top USA News Websites list rankings. Already listed URLs might not need to be removed; instead, I could just add more URLs to scrape from to my current list of URLs acceptable to scrape from.

I would also like to try and classify complex objects, like sequences and images on other machine learning algorithms, such as CNN, ANN, SVM, or k-nearest neighbors, Markov-based and Naive-Bayes algorithms to compare the pros and cons between each major type of machine learning algorithm so I can more deeply understand and appreciate the different machine learning algorithms and their uses for textual analyses.

Designing and implementing a more approachable user interface is another possible future work. More article URL links could be displayed, or the URL link's articles could be displayed upon search. Another possibility is clicking on a URL link and having the link's article appear on the website.

9 References

[1] Jeff Desjardins. *The Least and Most Trusted News Sources in America* http://www.visualcapitalist.com/least-most-trusted-news-sources/.

[2] Carroll Doherty. *7 things to know about polarization in America* http://www.pewresearch.org/fact-tank/2014/06/12/7-things-to-know-about-polarization-in-america/.

[3] Michael M. Grynbaum. *Fox News Drops 'Fair and Balanced' Motto* https://www.nytimes.com/2017/06/14/business/media/fox-news-fair-and-balanced.html.

[4] Arkajyoti Misra and Sanjib Basak. *Political Bias Analysis* https://cs224d.stanford.edu/reports/MisraBasak.pdf

[5] Lillian Lee. *U.S. Congressional Speech Data* http://www.cs.cornell.edu/home/llee/data/convote.html

[6] Mohit Iyyer1, Peter Enns, Jordan Boyd-Graber, and Philip Resnik. *The Ideological Books Corpus* https://www.cs.umd.edu/ miyyer/ibc/index.html

[7] Yanchuan Sim, Brice Acree, Justin Gross, and Noah Smith. *Measuring Ideological Proportions in Political Speeches.* Empirical Methods in Natural Language Processing, 2013.

[8] Mohit Iyyer, Peter Enns, Jordan Boyd-Graber, Philip Resnik. *Political Ideology Detection Using Recursive Neural Networks* http://www.aclweb.org/anthology/P/P14/P14-1105.pdf

[9] Maureen Caudill. *Neural Network Primer: Part I* Feb. 1989.

[10] University of Wisconsin-Madison. *A Basic Introduction to Neural Networks* http://pages.cs.wisc.edu/ bolo/shipyard/neural/local.html

[11] Matlab. *Softmax Documentation* https://www.mathworks.com/help/nnet/ref/softmax.html

[12] Tony. *Neural Networks: The Node* http://www.neuraldump.com/2016/05/neural-networks-the-node/

http://www.neuraldump.com/2016/05/neural-networks-the-node/

[13] Zachary C. Lipton, Jon Berkowitz, and Charles Elkan. *A Critical Review of Recurrent Neural Networks for Sequence Learning* https://arxiv.org/pdf/1506.00019.pdf.

[14] Pierre Baldi and Gianluca Pollastri. *The principled design of large-scale recursive neural network architecturees-DAG-RNNs and the protein structure prediction problem.* The Journal of Machine Learning Research, 4:575–602, 2003.

[15] Denny Britz. *Recurrent Neural Networks Tutorial, Part 2 – Implementing a RNN with Python, Numpy and Theano* http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-2-implementing-a-language-model-rnn-with-python-numpy-and-theano/.

[16] John McGonagle, George Shaikouski, Andrew Hsu, Jimin Khim, Christopher Williams. *Backpropagation* https://brilliant.org/wiki/backpropagation/.

[17] Trask. *Anyone Can Learn To Code an LSTM-RNN in Python (Part 1: RNN)* https://iamtrask.github.io/2015/11/15/anyone-can-code-lstm/.

[18] Denny Britz. *Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs* http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/.

[19] Ramon Quiza and J. Paulo Davim. *Computation Methods and Optimization* Machining of Hard Materials, pp. 177-208.

[20] DeepLearning.net. *LSTM Networks for Sentiment Analysis* http://deeplearning.net/tutorial/lstm.html#code.

[21] Feedspot.com. *Top 100 USA News Websites* http://blog.feedspot.com/usa_news_websites/.

[22] Siraj Raval. *How to Predict Stock Prices Easily - Intro to Deep Learning #7* https://www.youtube.com/watch?v=ftMq5ps503w&t=527s.

[23] Siraj Raval. *Generate Rap Lyrics - Fresh Machine Learning #4* https://www.youtube.com/watch?v=yE0dcDNRZjw.

[24] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. *Show and tell: A neural image caption generator.* In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3156–3164, 2015.

[25] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to sequence learning with neural networks* In Advances in Neural Information Processing Systems, pages 3104–3112, 2014.

[26] J O'Brien Antognini. *RNN vs CNN at a high level* https://datascience.stackexchange.com/questions/11619/rnn-vs-cnn-at-a-high-level.

[27] Roman Trusov. *Why would any one use Recursive Neural Nets for text classification as against SVM or Naive Bayes or any traditional statistical models?*

https://www.quora.com/Why-would-any-one-use-Recursive-Neural-Nets-for-text-classification-as-against-SVM-or-Naive-Bayes-or-any-traditional-statistical-models.

[28] Adam Yedidia. *Against the F-score*
https://adamyedidia.files.wordpress.com/2014/11/f_score.pdf.