University of Nevada, Reno

Distilling Public Data from Multiple Sources for Cybersecurity Applications

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science in
Computer Science and Engineering

by

James Schnebly

Dr. Emily Hand, Ph.D., Advisor
Dr. Shamik Sengupta, Ph.D., Co-Advisor
May, 2020

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

James Schnebly

entitled

Distilling Public Data from Multiple Sources for Cybersecurity Appplications

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Emily Hand, Ph.D.
Advisor

Shamik Sengupta, Ph.D.
Co-advisor

Sergiu Dascalu, Ph.D
Committee Member

Grant Schissler, Ph.D.
Graduate School Representative

David W. Zeh, Ph.D., Dean
Graduate School

May, 2020

Abstract

The amount of data being produced every day is growing at a very high rate, opening the door to new knowledge while also bringing forth cyber breach opportunities for malicious users. In this thesis, the objective is to analyze public data to gain valuable insight for cybersecurity applications. Using public Twitter account data, a machine learning model is trained to identify bot accounts which helps lower the amount of fake news and malicious users. A survey of text summarization techniques to identify the best method for summarizing public data in the domain of cybersecurity is presented. A web application is also created to serve as a public tool for users to summarize input text of their choosing using a variety of algorithms. The contribution of this thesis is thus twofold: a model capable of identifying Twitter bots with high accuracy, and a web application for summarizing cybersecurity information from public data.

Acknowledgments

I would like to thank my advisers Emily Hand and Shamik Sengupta in addition to my parents who have always supported me. I would also like to thank my committee members Sergiu Dascalu and Grant Schissler for not only being on my committee but also for for being excellent professors.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

With an ever-increasing amount of data being produced, the difficulty of digesting data has risen significantly. There are a plethora of social media applications creating incredible amounts of data as well as informational websites, blogs, online shopping sites, cloud services, and more. Managing all of these data sources can be equated to trying to stay upright in front of a fire hose.

In one day 500 million tweets are sent, 294 billion emails are sent, 4 petabytes of data are created on Facebook, and five billion internet searches are made. It is estimated that 463 exabytes of data will be created each day globally by 2025 [1]. In 2018, the authors of [2] stated, "Over 2.5 quintillion bytes of data are created every single day, and it's only going to grow from there". This is equivalent to 2.5 exabytes, so one can see just how quickly the world's data production is rapidly increasing.

With such an abundance of data, it is no wonder that data analysis has gained significant attention in recent years. Data analysis can be thought of as building a story from data. Typically, the more data available to study, the better story it can tell. Attempting to digest and sift through all this data is rigorous, but can explain many situations or trends that we may have been previously blind to. Since there

are now more ways to collect data and store said data easily, the data can be used to help make better decisions.

With this increase in data being produced, the number of malicious users and programs has also drastically increased. The authors of [3] state "During 2018, we have seen a 350% increase in ransomware attacks, a 250% increase in spoofing or business email compromise (BEC) attacks and a 70% increase in spear-phishing attacks in companies overall. Further, the average cost of a cyber-data breach has risen from $4.9 million in 2017 to $7.5 million in 2018, according to the U.S. Securities and Exchange Commission." Now, cybersecurity [4], the practice of defending computers, servers, mobile devices, electronic systems, networks, and data from malicious attacks is more important than ever.

The wide availability of data makes it easy for adversaries to gain access to sensitive information or perform other malicious behaviors, while at the same time making it more difficult for defenders to digest information and create robust defenses for cyber-threats. The amount of publicly available data is continuing to grow and without automated methods to help digest that information, there is no way that cybersecurity can keep up with the threats brought along with this new age of data.

People working in the field of cybersecurity are overwhelmed with the amount of data that they must sift through in order to identify malicious activity. The intention is to make this a more proactive process by automating bot detection and digesting cybersecurity related information for review. Automating these methods will free up time for cybersecurity professionals to address more important problems and to more quickly identify malicious activity.

Using publicly available data, in the form of tweets, blogs, and news articles one can identify malicious activity using machine learning, as well as distill information concisely for review by cybersecurity professionals. Applying artificial intelligence or

more specifically, machine learning, to these cybersecurity issues allows for a seamless integration into the every day life of a cybersecurity analyst in this era of big data.

The goal of this thesis is to demonstrate that public data from different sources can be used to build cybersecurity applications that aid in the defense against cyber threats:

- Train a text summarizing tool to help digest cybersecurity articles
- Use social media data to identify bot accounts or fake users

The remainder of this thesis is organized as follows: chapter 2 surveys background works in text summarization and bot detection. Chapter 3 describes a machine learning model trained on public twitter data that can identify a fake user with 90% accuracy. Chapter 4 describes a web application proof of concept that summarizes user text with the summarization algorithm of the user's preference, and finally, chapter 5 concludes the thesis, further emphasizing the benefit of analyzing public data.

Chapter 2

Background

This thesis builds on work in fake user detection and text summarization. This chapter details the relevant related work in both of these areas. It also provides some background on supervised and unsupervised learning as well as the random forest machine learning algorithm.

2.1 Fake User Detection in Social Media

There has been a significant amount of work in the past decade in fake user detection. This section discusses some of the recent research done on cybersecurity applications that detect fake social media accounts.

2.1.1 SybilRank

SybilRank is an effective and efficient fake account inference scheme, which allows OSNs to rank accounts according to their perceived likelihood of being fake (sybil) [5]. It leverages the power of graph representation of social network accounts and therefore

scales very well. SybilRank is an inference scheme customized for OSNs whose social relationships are bidirectional such as Twitter. The underlying principle built upon is that sybils have a disproportionately small number of connections to non-Sybil users. The authors of [5] found that 90% of the accounts that SybilRank flagged justified suspicion.

2.1.2 FRAppE

FRAppE, or Facebook's Rigorous Application Evaluator, is a cybersecurity tool focused on detecting malicious apps on Facebook using MyPageKeeper's dataset [6]. A feature set is developed by asking questions about the application such as, "Any posts in app profile page?", "Is client ID different from app ID?", and "Number of permissions required". The feature set is then input into a support vector machine (SVM) and binary classification training takes place. When training with a 1:1 ratio between malicious and benign users, FRAppe has a 98.5% accuracy with 5-fold cross-validation.

2.1.3 Framework for Detecting Clusters of Fake Accounts

The authors of [7] describe a framework for detecting fake accounts that were all created by the same malicious user. Supervised machine learning methods such as support vector machine and logistic regression are used to classify an entire cluster of accounts as malicious or legitimate. Clusters are created by grouping on registration date and registration IP address. A large contribution of this research is the feature engineering. The authors use simple statistics to get basic cluster features, pattern recognition to get see if certain users in a cluster share commonalities, and frequency features that show how rare a certain name or description on the account is. The

model has been turned into a commercial product and has now identified more than 250,000 fake accounts since deployment.

2.1.4 Profile Characteristics with an Activity-based Pattern Detection Approach

The authors of [8] present a strategy to identify fake profiles. Using a pattern matching algorithm on screen names and an analysis of tweet update times, a highly reliable subset of fake user accounts can be identified. The analysis of profile creation times and URLs of these fake accounts reveals distinct behavior of the fake users relative to a ground truth data set. The combination of this scheme with established social graph analysis opens the door for time-efficient detection of fake profiles in social media sites.

2.2 Text Summarization

Text summarization has a long history in the field of natural language processing. This sectiion discusses the two main types of summarization, as well as some modern approaches to the problem.

2.2.1 Extractive vs Abstractive Summarization

Before the success of sequence-to-sequence models [9], most work done in the field of summarization was strictly extractive where certain phrases or words would be extracted to make a summary. While this technique works well in some cases [10, 11], it does not allow for generalization or paraphrasing. Once sequence-to-sequence models began to demonstrate promising results, abstractive summarization with RNNs (Re-

current Neural Networks) became the new frontier for machine summarization [12,13]. Abstractive summarization is where the summary is a sequence of generated words and not an exact phrase from the text. Although abstractive summarization was deemed viable by Natural Language Processing (NLP) researchers, they still had problems such as working with out-of-vocabulary (OOV) words and repeating words or phrases in the generated summary. In order to combat those problems, [14] found success in applying an attention mechanism with coverage to fix the repeating problem and a pointer-generator network to fix the OOV issues. Even with the emergence of attention and coverage, extractive methods are still popular because of their simplicity when compared to abstractive methods that often have deep neural networks.

2.2.2 Text Summarization using the TF-IDF Algorithm

TF-IDF stands for "Term Frequency - Inverse Document Frequency". It is a measure of how important a word is to a document [15]. It is important to note that the term "document" can be used to mean many different things in natural language processing but in this section, document will be synonymous with sentence.

Algorithm Overview

The general algorithm for using TF-IDF to score sentences and summarize an article begins with reading in the sentences of the article and tokenizing them. From there a frequency matrix is created for the words in each sentence. Using the frequency matrix the Term-Frequency (TF) is computed and stored. Then a table is built to keep track of how many documents a word has been found in for all words. Using said table, an Inverse Document Frequency (IDF) score is computed. For each word, we can multiply TF and IDF to get the TF-IDF score. Each sentence is then scored by summing the TF-IDF and dividing by the number of words in the sentence. To

generate the summary, all sentences with a score above a decided upon threshold are extracted as the article summary.

Term Frequency

TF is the number of times the term appears in the document divided by the total number of words in the doc.

$$TF(t,d) = \frac{n_{t,d}}{T}$$

Where t is the specific term, d is the specific document, $n_{t,d}$ is the number of times term t appears in a document d, and T is the total number of terms in the document.

Inverse Document Frequency

The IDF for a specific term is the natural log of the total documents over the number of documents with that specific term in it.

$$IDF(t) = \ln \frac{N}{d_t}$$

Where N is the total number of documents and d_t is the number of documents with term t in it.

2.2.3 Text Summarization with TextRank Algorithm

TextRank is an unsupervised machine learning text summarization algorithm that scores sentences in an article similar to that of the PageRank algorithm [16]. The PageRank [17] algorithm is a method for rating Web pages objectively and mechanically, effectively measuring the human interest and attention devoted to them. To

understand the full details of the TextRank algorithm, prior knowledge of word embeddings is essential.

Word Embeddings

In many natural language processing tasks, it is essential to be able to process words in a continuous space as opposed to a discrete one. To do this, words are embedded into a vector space using one of many existing training algorithms. These word embeddings are a vector of real numbers and similar words, words semantically related, are similar vectors. Word2vec [18] and Global Vectors (GloVe) [19] are two state-of-the-art word embedding tools.

Algorithm Overview

The general algorithm for applying TextRank to an article starts with preprocessing the input data and splitting the text into individual sentences. Using word embeddings, we create vectors for the words in each sentence. Then, we take the mean of those vectors to arrive at a consolidated vector for the sentence. The next step is to find similarities between the sentences, so we use the cosine similarity approach to achieve this. Any similarity metric can be used but for the purpose of this research, cosine similarity works best. Once the similarity matrix is calculated with all sentences the matrix is then turned into a graph with sentences as vertices and similarity scores as edges. This graph then allows for Google's PageRank algorithm to run and return the ranking of the sentences in the article. The generated summary is the top k sentences where k is an arbitrary integer equal to the number of sentences desired in the generated extractive summary.

2.2.4 Pointer-Generator Networks

Pointer-Generator Networks [14] use LSTM [20] cells with attention and coverage vectors to produce high-quality summaries. The pointer-generator is a hybrid approach to summarization with both abstractive and extractive methods. In order to fully understand the pointer-generator model as well as many of the other state-of-the-art abstractive summarizatiion models, one must understand LSTM cells, attention, and coverage.

LSTM

Long short-term Memory (LSTM) networks are a variation of Recurrent Neural Networks where the network can learn long term dependencies by knowing the cell state of the cell at the previous time-steps. The idea behind RNNs is that they can use information from the previous time-step to compute a value for the current time-step. This advantage quickly disappears as the gap between the current time-step and the dependant time-step with relevant information grows. This is where LSTMs come in, they are able to handle a greater distance between dependencies hence their name, long short-term memory networks. Information can be added and taken away from the cell state at each time-step and this way the cell has information about the previous time-steps as well as the new input for that time-step. For example, when trying to find a word to complete the phrase: “I went to Mexico so I had to learn how to speak __ .”, the dependency for the new word is nine time-steps away from the time-step we are currently on. LSTMs can handle this task because they can go back and see the word “Mexico” in its memory and output the missing word as “Spanish”.

Attention

One of the biggest advances in abstractive summarization was the concept of attention. Attention is a mechanism relating to different positions of a single sequence in order to compute a representation of the sequence. Attention mechanisms have become a necessary component of neural summarization and translation systems because of the benefit of modeling dependencies regardless of distance in the sequence [21]. Between attention and LSTMs, the quality of machine translation and summarization system outputs has increased greatly. Although there is definite growth, there are still some problems with this such as repeatedly attending to the same position in the source sequence. Essentially, the attention distribution is not changing enough after a certain phrase or word is already translated or summarized. We can combat this problem by using a coverage vector with our attention mechanism.

Coverage

Neural Machine translation (NMT) and summarization face a serious problem, lack of coverage. This idea of coverage in machine translation is where a decoder maintains a coverage vector to indicate whether a source word is translated or not. This is important for ensuring that each source word is translated into decoding. The decoding process is completed when all source words are "covered" or translated [22]. In abstractive summarization, we can use coverage and attention together in order to make sure we are attending to parts of the source text that have not been covered already in the partially produced output summary.

Algorithm Overview

The pointer generator model holds a coverage vector which is the sum of attention distributions over all previous steps. The coverage vector helps the model not repeat

itself when generating a summary. The model produces a probability, P_{gen}, which dictates whether the model should use an abstractive approach or an extractive approach for the next word or phrase in the summary. When the pointer lands on OOV words, an extractive approach is used and therefore OOV words can be in the generated summary.

2.2.5 Named Entity Extractive Text Summarization Algorithm

Named Entity Recognition (NER) is the process of finding important people, companies, and numerical information in text [23]. Text summarization using NER to score a sentence is a simple yet efficient way to build an extractive summary. The algorithm counts the number of named entities in each sentence and divides said count by the number of non-stop words in the sentence.

2.2.6 Entropy Based Summarization

The original idea for this type of extractive summarization comes from [24] where the authors propose a sentence scoring system based on the entropy of the non-stop words making up the sentence. Entropy values are based on collocation pairs (w_i, w_j) where w_i is a word in the sentence and w_j is one of the following words. Entropy values are created for each word (w_i) by looking at the uncertainty of w_j when given w_i. The sentences' "score" was the sum of the forwards and backward entropy of all non-stop words in a sentence, divided by the total number of non-stop words in the said sentence. The highest scoring K sentences are then selected as the extractive summary.

2.2.7 Text Summarization Dataset

The CNN/DailyMail is a non-anonymized text summarization dataset containing articles from both CNN and DailyMail. Each article has two features: Text, which contains the plain text article to be used as either training or testing material, and Highlight, which contains highlights about the article that can be used as a reference summary. In this experiment, only the CNN stories are used as that gave roughly 92,500 articles to work with.

2.3 Machine Learning with Random Forest Model

In order to develop a base understanding of the main concepts used in the next chapter, a brief overview of supervised and unsupervised learning and the Random Forest machine learning algorithm is described in this section.

2.3.1 Supervised Learning vs Unsupervised Learning

Supervised learning is a subset of machine learning where the user feeds the model training data which contains labels. The model learns to classify or predict the labels based on the training data that is associated with it. Once training is complete, the model uses its prior knowledge of the training data to predict labels on new data or test data. Unsupervised learning makes decisions based on the relationship between the data instance at hand and the rest of the dataset. The unsupervised learning algorithm does not have any prior information about the correct class the data instance should belong to.

2.3.2 Random Forest

A Random Forest is a meta estimator that fits a number of Decision Tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting [25]. The Random Forest algorithm is a supervised ensemble learning algorithm, meaning the algorithm is actually made up of many other basic machine learning algorithms. To predict a class, each basic machine algorithm votes for a class and after all of the basic algorithms have voted, the class with the most votes is the class the ensemble algorithm predicts. With Random Forests, the underlying algorithm used is the Decision Tree classifier, hence the “Forest" in Random Forest. The Random Forest algorithm brings randomness into the model when it is growing the Decision Trees. Instead of searching for the best attribute while splitting a node, or decision in the tree, it searches for the best attribute among a random subset of attributes. This process creates diversity among trees and allows for each tree to be built upon different attributes, which generally results in a better model [26].

Chapter 3

Random Forest Twitter Bot Classifier

Social media usage is growing faster than ever in today's world, according to PEW Research Center, 73% of U.S. adults use YouTube, 68% use Facebook and 35% use Instagram. Shortly behind these social media "staples", Twitter has quickly risen to the point at which one in every four U.S. adults uses its platform [27]. Twitter launched in 2006 and has not stopped growing since it went public. On average, around 6,000 tweets are posted every second of the day, adding up to roughly 360,000 tweets per minute [28].

A growing issue with Twitter is the amount of fake or "bot" accounts either sharing malicious content or being used to enhance the metaphorical reach of genuine accounts. These Twitter bots are easily programmed due to the Twitter API (Application Programming Interface) and can be created within seconds. This means that just about anybody can create bots to form the illusion of popularity around hot topics or political figures. To make matters worse, Twitter does not use a "captcha" when users are creating an account, therefore it is very simple to have an automated service pump out tons of bots that are then run through the open Twitter API [29]. A captcha is a program used to verify that a human, rather than a computer, is entering

data on an online form [30].

Twitter bots have the ability to alter trending topics by the sheer volume of tweets they can produce. They can make false stories appear to have more views by promoting or tweeting about a certain topic along with the associated hashtag. These bots also spell disaster for corporations and their social media presence. One person with a massive army of bot accounts could have the bots all tweet negative sentiment about a company's product or service and tag the company in the tweet. Now anyone searching this product, service or company will be fed false information due to the shear volume of tweets sent out by the attacking bots.

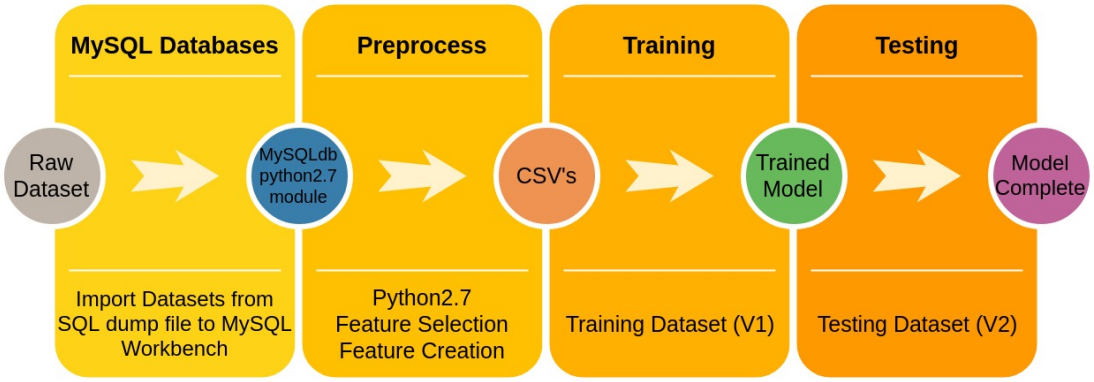


Figure 3.1: Workflow of creating the Random Forest model 3.0

Now more than ever we need a way to get rid of existing bots to protect people from being fed false information. In this chapter, we present a simple yet effective classifier model that has the ability to detect existing Twitter bot accounts using only data that is easily attainable through the Twitter API as well as attributes that are ratios of the previously mentioned data. This chapter details my contribution to bot detection using public data as follows:

- A generalized machine learning model that can detect existing Twitter bot accounts with 90.25% accuracy
- A feature set that includes both basic and derivative attributes that can be

automatically extracted from any public Twitter account

We start by obtaining data and importing it into two MySQL databases. Then, the data is preprocessed the data by querying the MySQL databases and begin the feature selection and feature creation processes. The preprocessing python program outputs CSV (Comma-Separated Values) files which hold the input data and labels that are to be fed into the machine learning model. Lastly, we feed the CSVs into the python program that trains and tests a Random Forest classifier. This workflow can be seen in Figure 3.1.

The remainder of this chapter is organized as follows: Section 3.1 dives into the process and methodology of building and training the classifier, Section 3.2 covers the results and the testing of the previously mentioned classifier and reviews and conclude the chapter.

3.1 Methodology

3.1.1 Data Collection

To train a Twitter bot classifier, we first need access to Twitter account data for both genuine accounts and bot accounts. Additionally, we also need access to as many tweets per account as possible. This task proves to be tricky as there are limitations to the free Twitter API [31]. Furthermore, there is no way to label an account as genuine or bot if just querying the API for data. Two datasets from the MIB (My Information Bubble) project [32, 33] hosted at IIT in Italy are used for our research purposes.

The first dataset was gathered in 2015 and contains verified human accounts as well as known bot accounts used to falsely inflate the number of followers for a genuine

account. The second dataset was gathered in 2017 with new known bot accounts and verified genuine accounts.

The two datasets contain basic account information such as, but not limited to: number of followers, number of tweets, when the account was created and if the account was genuine or a bot. Along with the account information there are also hundreds, sometimes thousands, of tweets that belong to each account.

For the present research purposes we use a subset of each dataset as the data is somewhat repetitive. Dataset V1 contains 445 real accounts and 3202 bot accounts from the original 2017 dataset and V2 which contains 1946 real accounts and 3457 bot accounts from the original 2015 dataset. These datasets V1 and V2 are then imported into a MySQL [34] database to make the preprocessing stage smoother. The data is imported into two databases, V1 and V2 where each database has two tables, namely Users and Tweets. The primary key shared between these two tables is the user_id which is a unique identifier given to every Twitter account.

3.1.2 Preprocessing

We select basic account attributes from the database table Users that could have an impact on the account being genuine or fake. These attributes are length of description (bio), age of account, number of followers, number of tweets, number of people the account is following (friends), number of likes and the follower to friend ratio. These attributes are chosen because they explain the account usage at a fundamental level. The basic feature subset can be seen in Table 3.1.

After the basic attributes are chosen, we create derived attributes by computing ratios. These derived attributes allow us to gain more insight on the activity of the account. These are derived features because they are not directly available through the Twitter API.

Table 3.1: Basic Feature Set

Length of Bio
Followers
Age of account
Following
Number of tweets
Number of likes
Follower ratio

Likes_age_ratio

First we create the likes to age ratio, where we divide the number of likes the account has given by the number of days since the account's creation. This feature shows how actively the account likes another account's tweet.

Tweet_age_ratio

The tweets to age ratio is created by dividing the number of tweets by the days since the account has been created. This attribute shows how often the account tweets. Now, we analyze the account's tweets by performing an inner join on the user_id between the User table and Tweets table.

Hashtag_tweet_ratio

The hashtag to tweet ratio is computed by counting all tweets in the sample that contain at least one hashtag and dividing it by the total number of tweets in our given sample. The total number of tweets in the sample is given by using the SQL aggregate COUNT() to count the number of tweets associated with a given user_id. This new ratio attribute gives insight on how often the account uses hashtags in their tweets.

URL/Pics_tweet_ratio

The url to tweet ratio is computed similarly. we use COUNT() to tally up all the tweets in the sample that contain urls and divide that integer by the total number of tweets in the sample for that particular user_id. Interestingly enough, when accounts post pictures on twitter the pictures are represented as urls, therefore this url to tweet attribute accounts for links to websites as well as pictures. This ratio shows how often the account posts urls or pictures when they tweet.

Reply_tweet_ratio

The reply to tweet ratio is computed the same way using the COUNT() aggregate, and it shows the approximate percentage of tweets posted that are replys to other accounts. This percentage is only based on the sample of tweets that are available, but provides valuable insight on the account's overall tendencies. The derived feature subset can be seen in Table 3.2.

Table 3.2: Derived Feature Set

Hashtag_tweet_ratio
URL/Pics_tweet_ratio
Reply_tweet_ratio
Tweet_age_ratio
Likes_age_ratio

The full proposed feature set can be seen in Table 3.3 along with a description of each attribute.

Once all of these attributes are queried from the SQL database or created by the above methods, we create a CSV file that serves as input for our machine learning classifier. One CSV is created for genuine accounts and another is created for known bot accounts since the information is held in separate databases.

Table 3.3: Proposed Feature Set

Length of Bio	Number of characters in bio
Hashtag_tweet_ratio	Ratio of tweets with hashtags to total tweets
Followers	Number of accounts following
URL/Pics_tweet_ratio	Ratio of tweets with URL's to total tweets
Age of account	Number of days since account creation
Reply_tweet_ratio	Ratio of replies to total tweets
Following	Number of accounts the account is following
Number of tweets	Number of tweets on the account
Tweet_age_ratio	Ratio of tweets to days since creation
Likes_age_ratio	Ratio of likes to days since creation
Number of likes	Number of likes on the account
Follower_ratio	Ratio of followers to following

3.1.3 Random Forest Implementation

To implement the Random Forest classifier, we use python3.5 as well as the libraries Scikit-Learn, pandas, and numpy [35–37]. First, we read the CSV's for the genuine accounts and the bot accounts into pandas dataframes. Then, using numpy, we add the label column to each dataframe, namely "Fake_Or_Real". The genuine accounts get a '0' in this column while the bot accounts receive a '1'. Next, we combine the two dataframes into one that is then split it into the input attributes, X, and the output labels, y. Before training the model, we must create a train set and test set.

Since we have two completely separate datasets as discussed in 3.1.1, we create three different models each using the data differently. Model 1.0 uses V1 for both the training and testing. The data is randomly split with 75% going to training while the remaining 25% is kept for testing. Model 2.0 combines both V1 and V2 and then does the same 75%-25% split for training and testing. Model 3.0 uses the entire V1 dataset for training and then tests on the entire V2.

We use the Scikit-learn RandomForestClassifier class and specify the forest to be made up of 20 Decision Trees [35]. 20 Decision Trees is decided upon after tests of 10, 20, and 50 trees. The accuracy increase from 10 to 20 was significant while the

accuracy increase from 20 to 50 was very minimal, thus leaving 20 as the optimal amount. We also specify that we want the trees to make decisions based on entropy by making the nodes, or decisions, yield more binary splits. The more binary the split, the higher the entropy. To train the models, we feed the twelve attributes X, discussed in Section 3.1.2, as well as the output label y, into the model as training input and associated output.

3.2 Results

3.2.1 Experiment Environment

The training and testing of the models presented in this chapter were trained and tested on a machine with the specifications from Table 3.4.

Table 3.4: Machine Specifications

Processor:	Intel Core i7-7700 CPU @ 3.60GHz x 8
RAM:	16 GiB
OS:	Ubuntu 16.04 LTS
Environment:	python3.5 in Spyder3

3.2.2 Model Results

Model 1.0 is tested with 25% of V1 and averaged 99% accuracy over the course of 10 runs. Model 2.0 is tested with 25% of V2 and averaged 98% accuracy over the course of 10 runs. Because of the high accuracy, we look to the decision trees produced in the random forest to see if there is any potential over-fitting in these two models. The decision trees give a visualization of each decision node and the size of the tree. The larger the tree, the more fit to the training data the model is. The goal is to have trees that can accurately classify instances with as few decision nodes as possible.

The tree needs to be smaller, or generalized, in order to be useful in a real world situation. The larger, less-generalized trees may perform well on the training data but poorly on real world data because it is too tightly fit to the training data.

As seen in Figure 3.2, the tree from model 1.0 is smaller and cleaner than the tree from model 2.0, seen in Figure 3.4. The tree from model 1.0 has less decision nodes and the splits are more binary, which means higher entropy. Since model 1.0 is more suitable for generalized usage we use its foundation to create model 3.0, previously discussed in section 3.1.3. Model 3.0 is trained and tested on completely different datasets therefore giving a more clear picture of how this model would work if deployed onto Twitter right away. Over the course of 10 runs the model produced an average accuracy of 90.25%. Figure 3.3 shows a Decision Tree from model 3.0; note the similarity between the tree from model 1.0 and model 3.0. Both models are more general than model 2.0. The purpose of figures 3.2, 3.3 and, 3.4 are to display the overall generalizability of each model. It should also be noted that the tree numbers were chosen arbitrarily and that any other tree could have been chosen to display the generalizability of each model. Models 1.0 and 3.0 are generalized compared to the overfit tree from model 2.0. This similarity between models 1.0 and 3.0 makes sense because they are trained on the same dataset. Model 1.0 is trained on a subset of V1 and tested on a different subset of V1 while model 3.0 is trained on all of V1 and tested on V2. Model 3.0 shows better real world feasibility than model 1.0 because it achieved the 90.25% accuracy on data from a different dataset. Model 1.0 was tested on a subset of data from the same dataset as its training data, therefore its accuracy cannot be transferred to a real world situation.

To evaluate accuracy and generalization of the models, a ratio of average decision nodes per tree to test accuracy is computed. Since model 3.0 is essentially a more feasible version of model 1.0, a comparison of this ratio between model 2.0 and model

3.0 is shown in Figure 3.5. Model 2.0 has, on average, 156 decision nodes per tree with an accuracy of 98%. Model 3.0 averages 43 decision nodes per tree and achieves an accuracy of 90.25%. This gives model 2.0 a ratio of .628 and model 3.0 a ratio of 2.09. These ratios show that although model 2.0 has a higher accuracy, it is not nearly as generalized as model 3.0. Model 2.0 may have a higher accuracy by 7.25% but the average amount of nodes per tree is more than triple of the average nodes per tree in model 3.0.

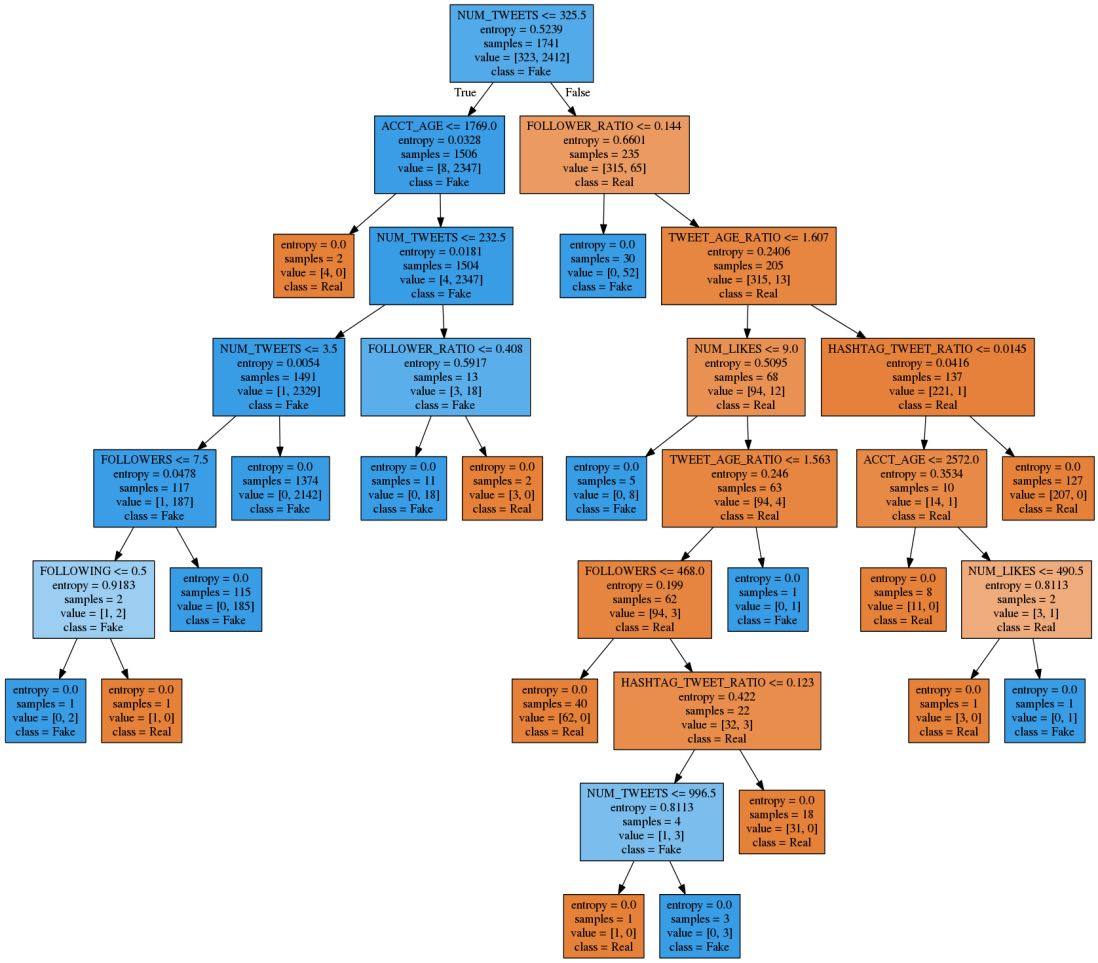


Figure 3.2: Model 1.0 Tree 15

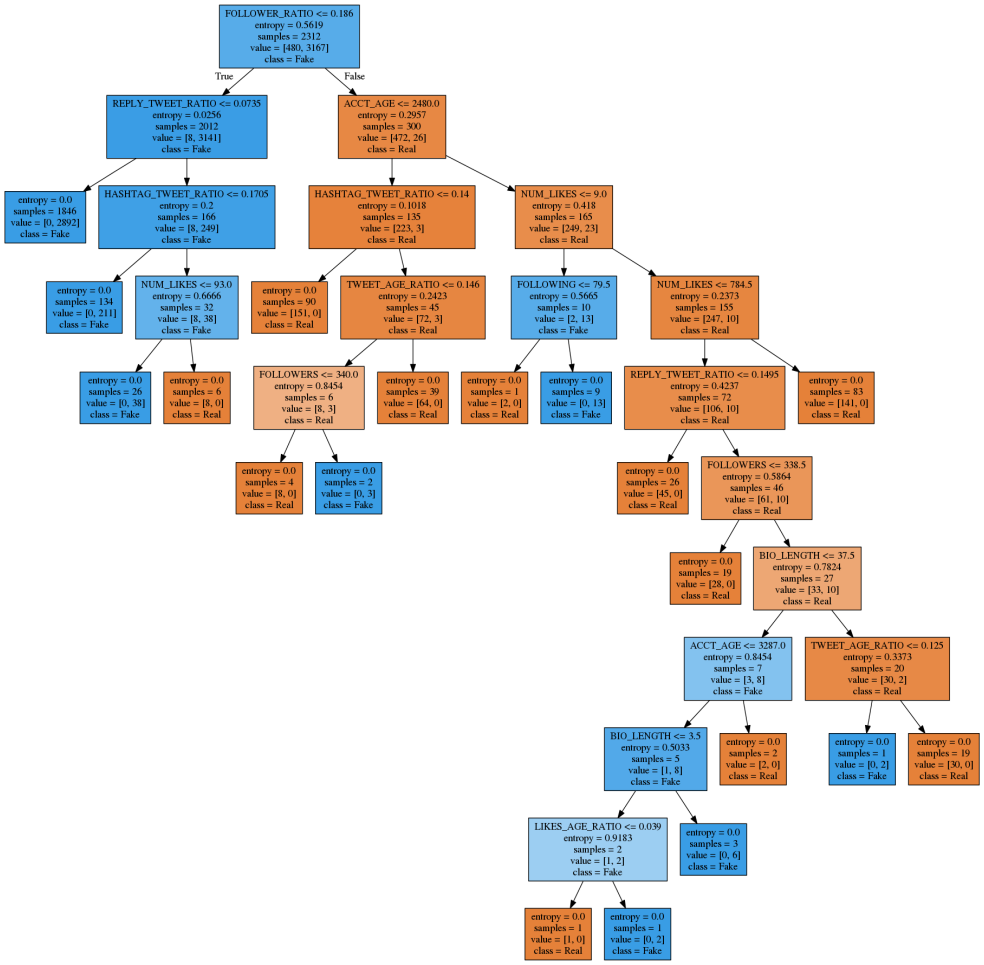


Figure 3.3: Model 3.0 Tree 0

3.2.3 Feature Results

Since our contribution in this section is not only the model, but also the features used to build it, we now look at feature importance to show our new features make a positive difference in classifier accuracy. All test runs to show feature importance are done with model 3.0 as it is the best, most generalized model of the three discussed in section 3.2.2. Table 3.5 shows the feature importance for each attribute averaged over the same ten runs that model 3.0 achieved the 90.25% accuracy. Note the feature importance was calculated by the Scikit-learn [38] feature importance built-in method.

To show that our derived features make a positive difference in regards to accuracy,

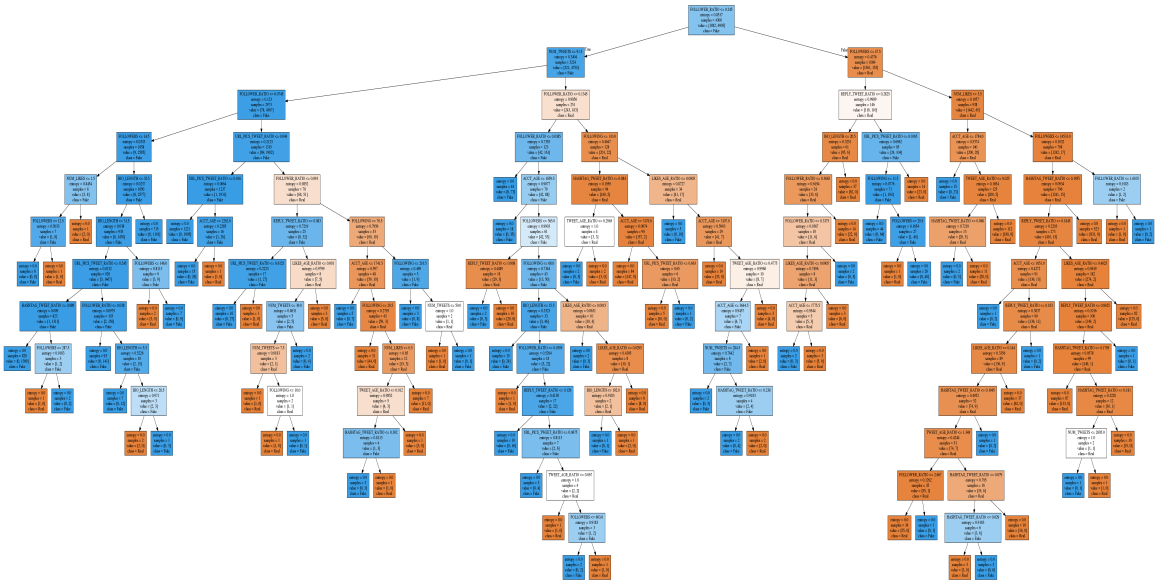


Figure 3.4: Model 2.0 Tree 19

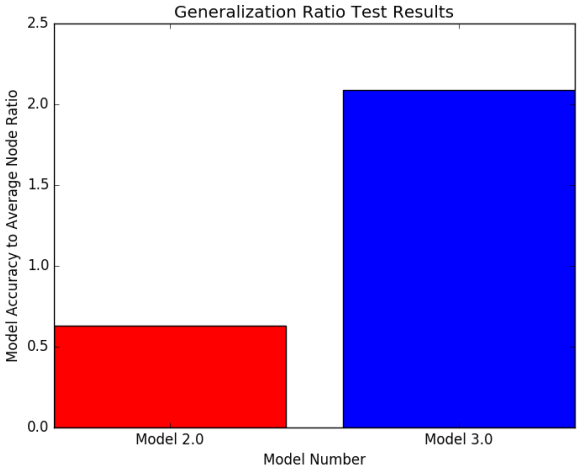


Figure 3.5: Accuracy to Generalization Ratio Test

we run versions of model 3.0 with different attributes to see the differences in accuracy as we change the attributes the model is trained upon.

First, we take the seven basic features, denoted in Table 3.5 by not having a *, and build model 3.0 on just those attributes. Over a period of 10 runs the accuracy of the Random Forest model was 81.7%. Therefore, by adding our derived ratio attributes to the model we increase its accuracy by just under 10%. From here,

Table 3.5: Feature Importance of Proposed Feature Set (where * indicates a derived feature)

Length of Bio	.004
*Hashtag_tweet_ratio	.0054
Followers	.0145
*URL/Pics_tweet_ratio	.0174
Age of account	.0189
*Reply_tweet_ratio	.0304
Following	.0511
Number of likes	.1105
*Tweet_age_ratio	.1345
*Likes_age_ratio	.1654
Number of likes	.1834
Follower_ratio	.2647

we look at the top two most important features from Table 3.5 and build a model on just those two features. When running the model on just Number of likes and Follower_ratio, the model achieved 98% accuracy on the test set. Although this accuracy was higher than the proposed model of all twelve attributes, it is not fit to be deployed as it lacks generalizability and could potentially be overfit; however, we can use this model as a baseline and see if a model using only our derived ratio attributes performs better. Over ten runs, the model trained on our five derived ratio attributes produced an accuracy of 99.6%. This test shows that these ratio attributes provide valuable insight on whether an account is genuine or fake. Using the derived features in conjunction with basic account features yields a generalized model that can classify Twitter accounts at an accuracy of 90.25%. The feature set test results can be seen in Figure 3.6. One should note that accuracy above 95% shows that the feature set provided valuable insight, but the model might not be generalized enough. Therefore, some tests show high accuracy, but have low feasibility for real world application which is why the proposed model has an accuracy of 90.25%, but is generalized.

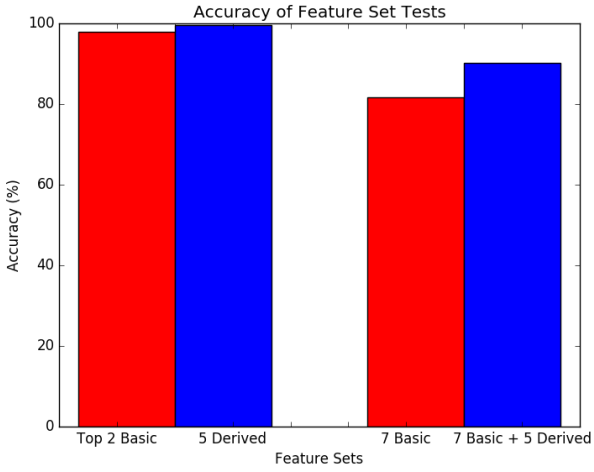


Figure 3.6: Feature Set Tests

The unstoppable growth of social media in today's world also brings forth the problems of false information spreading, malicious content spreading and fake followers for popularity. These problems often involve the use of bot accounts or automated accounts. In this chapter we have propose a machine learning classifier along with a set of features that can accurately detect these automated accounts and label them as bots. To train and test this classifier we use datasets from different years that were gathered by the IIT (Institute of Informatics and Telematics) in Italy. We selected basic features that could be accessed from the Twitter API then created five ratio features that granted additional insight on how the account was generally operated. We then tested different models that used subsets of our proposed attributes in order to demonstrate that our derived ratio features played a significant role in creating a model that was general enough to be deployed onto Twitter, but could also maintain a 90% accuracy on new data.

The next chapter details the other main contribution in this thesis, a text summarization web application.

Chapter 4

Summarization Web Application

The text summarization web app follows the microservice architecture with 4 main services: javascript's React.js running on the frontend, an API server built with python's Flask, and Firebase for user authentication and NoSQL cloud data store. This microservice architecture structures an application as a collection of services that are highly maintainable, testable, loosely coupled, and independently deployable. This means if one service goes down, the others are not dragged down with it. There are many other software development architectures between microservice and monolithic, but for the purpose of this research the modularity of microservices works best.

The overall architecture can be seen in figure 4.1. The React frontend can call out to all 3 other services via REST APIs to either get or send data. The Flask API server can pull data from the user authentication service and the data store. The main task of the API server is to summarize input data and return the summary generated by the specified algorithm.

The remainder of this chapter is organized as follows: section 4.1 details the frontend which is built using javascript framework React.js, section 4.2 discusses the

API server and the individual text summarization APIs, section 4.3 shows how user authentication and data storing is handled within the app, and section 4.4 presents how to use the text summarization feature.

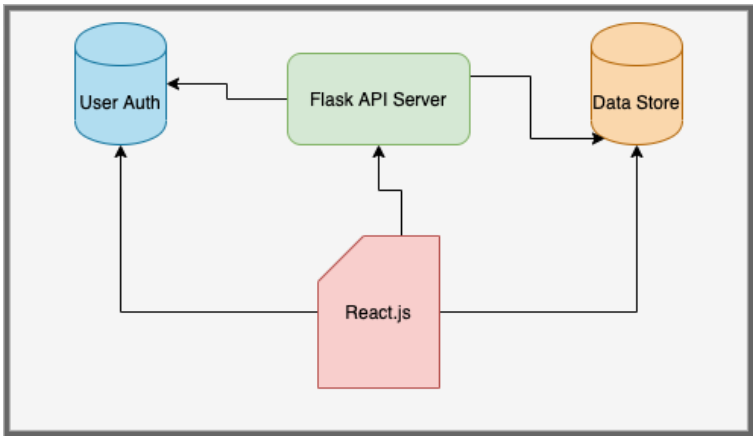


Figure 4.1: Web app architecture

4.1 React.js Frontend

In this section, React is discussed in more detail and the architecture for the frontend of the web application is presented.

4.1.1 React.js Background

React.js (React) is a javascript library for building user interfaces [39]. It was officially released on May 29, 2013, by Facebook and has since become the most popular frontend javascript framework. React is a declarative, component-based framework that is known for easy data flow and efficient rendering. It achieves this efficient render time by making a virtual copy of the DOM tree and when a component changes in React, React changes the virtual DOM tree. Then, React only re-renders the elements in the DOM that differ from the virtual DOM. This way only the necessary DOM

elements are re-rendered resulting in the need to render fewer elements.

4.1.2 Frontend Component Architecture

The React app component architecture consists of the following page components: home page, sign in page, and demo page. Each of these pages is made up of many smaller components that contain both javascript and Html in the form of what React calls "JSX". The home page component contains links to the sign-in page and the demo page in the navbar as well as links to find information on the text summarization algorithms available for demo. The home page can be seen in figure 4.2. The demo page component consists of a large input text box, a dropdown menu to select an algorithm, a summarize form submission button, and an output bordered div where the generated summary appears. The demo page can be seen in figure 4.3. The sign-in page contains two children components, one for signing in and another for creating an account. Section 4.3 discusses the logic behind both of these children components.

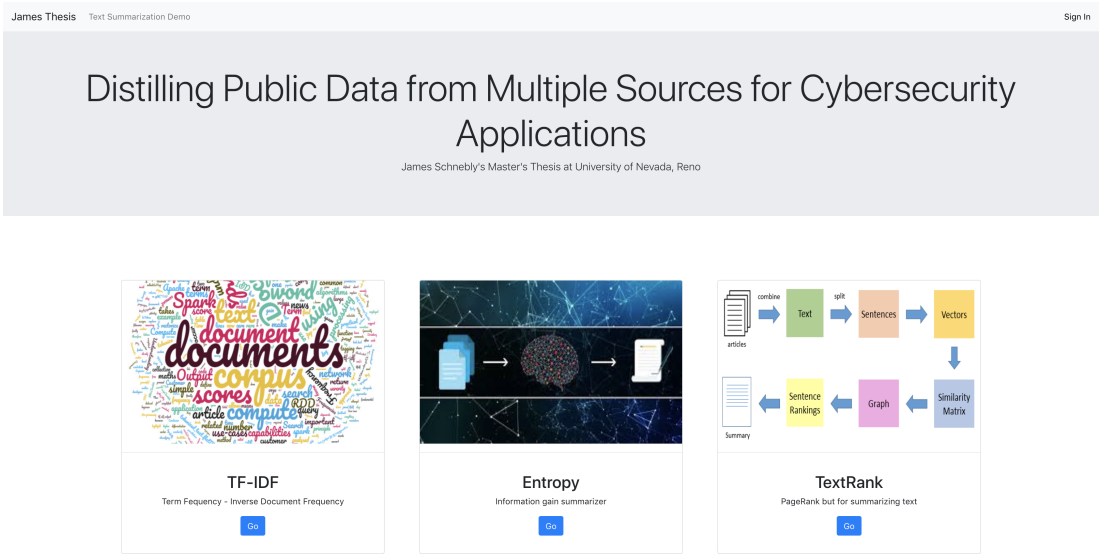


Figure 4.2: Screenshot of the home page component

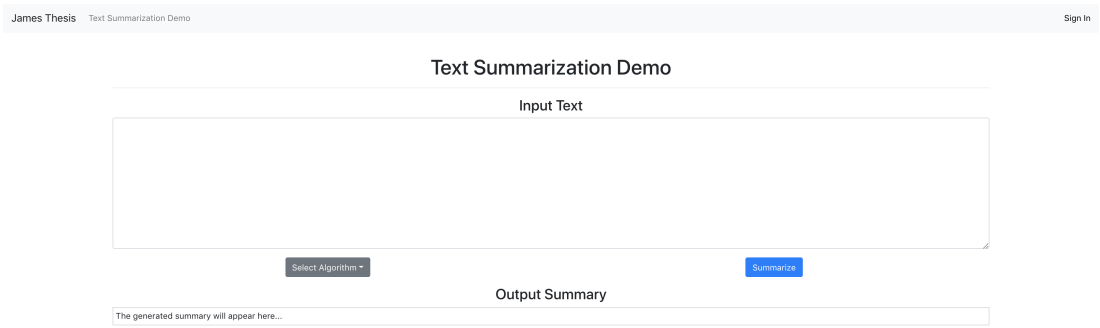


Figure 4.3: Screenshot of the demo page component

4.2 Flask API Server

In this section, the API server that is in charge of summarizing the user's input is detailed in full after discussing Flask's background.

4.2.1 Flask Background

Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications [40]. Flask is very powerful and can take many forms but in this application, it serves as the framework for the API server. There is a single file where each API route is defined as well as how it may be accessed, either a GET request or a POST request.

4.2.2 TF-IDF API Implementation

The TF-IDF API is under the route "/TF-IDF/" and can be accessed by selecting the TF-IDF algorithm from the dropdown on the demo page. When the summarize button is clicked, React fires off an API request to "/TF-IDF/" with the input text. The API takes the input text, summarizes it using the TF-IDF algorithm, and returns

the generated summary. The following subsections detail the implementation of the TF-IDF API on my API server.

Tokenize the sentences

To get the data into sentences from the raw text, `sent_tokenize` from nltk.tokenize is used. To find the total number of documents, the python3 function `len` is called on the array of sentences returned by `sent_tokenize`.

Create the Frequency matrix per sentence

For each sentence, an empty dict object is created and then `word\_tokenize` is used to get a list of word tokens. For each word in the list, we run the string *lower* method and use the `PorterStemmer` from nltk to get the stem. If the word is a stop word, go to the next word in the list, else, either add it to the dict object as a key with a value of 1 or if the key already exists we simply increment the value by 1.

After doing this for each word in a sentence, we assign the dict object as a value with a key of the first 15 characters of the sentence. As one can see in figure 4.4, the key is the first 15 characters of the sentence and the value is a dict with a specific word as a key and the frequency within the sentence as the value.

Calculate Term-Frequency Matrix

For this, we use the frequency value from the matrix made in the last step and and count the number of words in the sentence with the *len* function. As one can see in figure 4.5, each word now has a TF value.

Key	Type	Size	Value
1) First Americ	dict	24	{'1':1, ')':1, 'first':2, 'american':3, 'corpor':2, ':':1, '~':1, '885 ...
2) Facebook: ~5	dict	15	{'2':1, ')':1, 'facebook':2, ':':1, '~540,000,000':1, 'thi':1, 'news': ...
4) Flipboard: ~	dict	30	{'4':1, ')':1, 'flipboard':3, ':':1, '~150,000,000':1, 'content':1, 'a ...
A more expensiv	dict	13	{'expens':1, 'option':1, 'secur':1, 'catalogu':1, 'data':1, 'rather':1 ...
A rolling code	dict	14	{'roll':1, 'code':1, 'app':1, 'text':1, 'messag':1, 'form':1, 'mobil': ...
According to th	dict	13	{'accord':1, 'thi':1, 'varoni':1, 'report':1, ',':1, '57':1, 'percent' ...
Another Faceboo	dict	26	{'anoth':1, 'facebook':1, 'app':1, 'backup':1, 'titl':1, 'pool':1, 'al ...
Are you sure yo	dict	8	{'sure':1, 'pay':1, 'enough':1, 'attent':1, 'potenti':1, 'cyber':1, 'r ...
As Facebook fac	dict	14	{'facebook':1, 'face':1, 'scrutini':1, 'data':1, 'stewardship':1, 'pra ...
But Check Point	dict	30	{'check':1, 'point':1, 'research':1, 'found':1, 'vulner':1, 'could':1, ...
But if you read	dict	19	{'read':1, 'thi':1, 'data':2, 'breach':1, 'list':1, 'think':1, ',':1, ...
Check Point Res	dict	13	{'check':1, 'point':1, 'research':1, 'didnt':1, 'need':1, 'creat':1, ' ...

Figure 4.4: Frequency matrix

Creating a table for documents per words

Next, we run through each value in each frequency dict and create or increment the entry for that word for every document it is found in. This new dict is called the count_doc_per_words. This can be seen in figure 4.6.

Calculate IDF matrix

Using the formula from section 2.0.2, the variable calculated earlier holding the total number of documents, and the count_doc_per_words table, the IDF matrix is created. The IDF matrix looks similar to the TF matrix, but the values per word are the IDF values.

Calculate TF-IDF matrix

Using both the TF matrix and the IDF matrix, we multiply the entries in the respective matrices to get the TF-IDF score for each word.

Key	Type	Size	Value
1) First Americ	dict	24	{'1':0.041666666666666664, ')':0.041666666666666664, 'first':0.0833333 ...
2) Facebook: ~5	dict	15	{'2':0.06666666666666667, ')':0.06666666666666667, 'facebook':0.133333 ...
4) Flipboard: ~	dict	30	{'4':0.03333333333333333, ')':0.03333333333333333, 'flipboard':0.1, ': ...
A more expensiv	dict	13	{'expens':0.07692307692307693, 'option':0.07692307692307693, 'secur':0 ...
A rolling code	dict	14	{'roll':0.07142857142857142, 'code':0.07142857142857142, 'app':0.07142 ...
According to th	dict	13	{'accord':0.07692307692307693, 'thi':0.07692307692307693, 'varoni':0.0 ...
Another Faceboo	dict	26	{'anoth':0.038461538461538464, 'facebook':0.038461538461538464, 'app': ...
Are you sure yo	dict	8	{'sure':0.125, 'pay':0.125, 'enough':0.125, 'attent':0.125, 'potenti': ...
As Facebook fac	dict	14	{'facebook':0.07142857142857142, 'face':0.07142857142857142, 'scrutini ...
But Check Point	dict	30	{'check':0.03333333333333333, 'point':0.03333333333333333, 'research': ...

Figure 4.5: TF matrix

Score the sentences

Here, to score the sentence as a whole, we add up the TF-IDF of each word and then divide it by the number of words in that sentence. This gives each sentence a TF-IDF score that can be used to rank the pseudo importance of the sentence. The dict object with these scores can be seen in figure 4.7.

Find the threshold

After experimentation, 1.3 multiplied by the average sentence score was found to be the best threshold.

Generate the summary

Filter out the sentences that have a score lower than the threshold and return the sentences with a score higher than the threshold. This extractive summary based on the TF-IDF algorithm is what is returned from the API.

Key	Type	Size	
bank	int	1	1
base	int	1	2
becom	int	1	1
befor	int	1	2
best	int	1	1
biggest	int	1	4
billion	int	1	1
blow	int	1	1
bounti	int	1	1
breach	int	1	16

Figure 4.6: Count Doc per Word Dict

4.2.3 TextRank API Implementation

The TextRank API is under the route "/text-rank/" and can be accessed by selecting the TextRank algorithm from the dropdown on the demo page. When the summarize button is clicked, React fires off an API request to "/text-rank/" with the input text. The API takes the input text, summarizes it using the TextRank, and returns the generated summary. The following subsections detail the implementation of this algorithm on my API server.

Preprocess

For preprocessing, punctuation and stop words are removed from the sentences after being passed through a `toLower()`function. Since the later computations will be done with word embeddings, no more preprocessing is necessary.

Key	Type	Size	Value
1) First Americ	float	1	0.06540727569377086
2) Facebook: ~5	float	1	0.08739949534722283
4) Flipboard: ~	float	1	0.05151762437124659
A more expensiv	float	1	0.10774571269976949
A rolling code	float	1	0.10257308769809556
According to th	float	1	0.1008228386972747
Another Faceboo	float	1	0.05675511029293437
Are you sure yo	float	1	0.19759334932361838
As Facebook fac	float	1	0.09049338303277117
But Check Point	float	1	0.05038136079700572

Figure 4.7: Scored Sentences

Get Word Embeddings

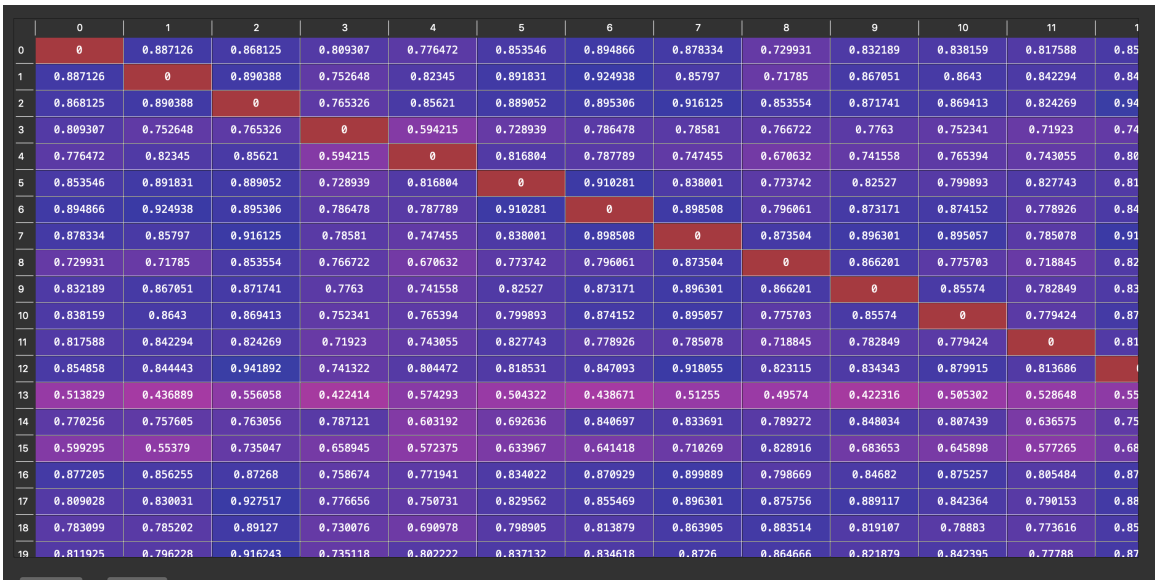
The GloVe word embeddings contain 100 dimensional vector representations of 6 billion words. Using the GloVe word embeddings, each word is substituted with the respective vector of length 100. Each word vector in the sentence is used to build an average vector that represents the sentence.

Build Similarity Matrix

With the cosine_similarity function from sklearn we generate a similarity matrix by comparing each sentence's vector to every other sentence's vector. An example similarity matrix can be seen in figure 4.8.

Apply PageRank and Present the Summary

Here the aforementioned similarity matrix is turned into a graph, using networkx, with the nodes representing the sentences and the edges representing the similarity scores



	0	1	2	3	4	5	6	7	8	9	10	11	1
0	0	0.887126	0.868125	0.809307	0.776472	0.853546	0.894866	0.878334	0.729931	0.832189	0.838159	0.817588	0.85
1	0.887126	0	0.890388	0.752648	0.82345	0.891831	0.924938	0.85797	0.71785	0.867051	0.8643	0.842294	0.84
2	0.868125	0.890388	0	0.765326	0.85621	0.889052	0.895306	0.916125	0.853554	0.871741	0.869413	0.824269	0.94
3	0.809307	0.752648	0.765326	0	0.594215	0.728939	0.786478	0.78581	0.766722	0.7763	0.752341	0.71923	0.74
4	0.776472	0.82345	0.85621	0.594215	0	0.816804	0.787789	0.747455	0.670632	0.741558	0.765394	0.743055	0.80
5	0.853546	0.891831	0.889052	0.728939	0.816804	0	0.910281	0.838001	0.773742	0.82527	0.799893	0.827743	0.81
6	0.894866	0.924938	0.895306	0.786478	0.787789	0.910281	0	0.898508	0.796061	0.873171	0.874152	0.778926	0.84
7	0.878334	0.85797	0.916125	0.78581	0.747455	0.838001	0.898508	0	0.873504	0.896301	0.895057	0.785078	0.91
8	0.729931	0.71785	0.853554	0.766722	0.670632	0.773742	0.796061	0.873504	0	0.866201	0.775703	0.718845	0.82
9	0.832189	0.867051	0.871741	0.7763	0.741558	0.82527	0.873171	0.896301	0.866201	0	0.85574	0.782849	0.83
10	0.838159	0.8643	0.869413	0.752341	0.765394	0.799893	0.874152	0.895057	0.775703	0.85574	0	0.779424	0.87
11	0.817588	0.842294	0.824269	0.71923	0.743055	0.827743	0.778926	0.785078	0.718845	0.782849	0.779424	0	0.81
12	0.854858	0.844443	0.941892	0.741322	0.804472	0.818531	0.847093	0.918055	0.823115	0.834343	0.879915	0.813686	
13	0.513829	0.436889	0.556058	0.422414	0.574293	0.504322	0.438671	0.51255	0.49574	0.422316	0.505302	0.528648	0.55
14	0.770256	0.757605	0.763056	0.787121	0.603192	0.692636	0.840697	0.833691	0.789272	0.848034	0.807439	0.636575	0.75
15	0.599295	0.55379	0.735047	0.658945	0.572375	0.633967	0.641418	0.710269	0.828916	0.683653	0.645898	0.577265	0.68
16	0.877205	0.856255	0.87268	0.758674	0.771941	0.834022	0.870929	0.899889	0.798669	0.84682	0.875257	0.805484	0.87
17	0.809028	0.830031	0.927517	0.776656	0.750731	0.829562	0.855469	0.896301	0.875756	0.889117	0.842364	0.790153	0.88
18	0.783099	0.785202	0.89127	0.730076	0.690978	0.798905	0.813879	0.863905	0.883514	0.819107	0.78883	0.773616	0.85
19	0.811925	0.796228	0.916243	0.735118	0.802222	0.837132	0.834618	0.8726	0.864666	0.821879	0.842395	0.77788	0.87

Figure 4.8: Similarity matrix

between sentences. This makes it easy to apply the PageRank algorithm as networkx has a pageRank() method that can be run on a graph object. Once PageRank has returned the sentence rankings, the top k sentences are returned as the PageRank extractive summary.

4.2.4 Entropy API Implementation

The entropy API is under the route "/entropy/" and can be accessed by selecting the entropy algorithm from the dropdown on the demo page. When the summarize button is clicked, React fires off an API request to "/entropy/" with the input text. The API takes the input text, summarizes it using the pretrained entropy dictionary, and returns the generated summary. The entropy dictionary is trained prior to the launch of the API server and therefore the trained entropy dictionary stays on the server for summarization use. The training data consists of all of the sentences in the CNN half of the CNN/DailyMail text summarization dataset. The highlight feature was disregarded as it was technically not part of the news article. The following

subsections detail the training of the entropy dictionary and the "/entropy/" route on my API server.

We begin building this model in a similar manner as the previous models by preprocessing the input data. For each sentence in each article of the "training data", stop words and punctuation are removed leaving each sentence as a list of sequential non-stop words. We refer to this as the cleaned training sentences. To build the vocabulary from the cleaned training sentences, every word becomes a vocabulary word and is added to the vocab list. That is, all non-stop words found in the training data make up the vocabulary. A count of how many times each vocabulary word is found is also created as these values are needed at a later point in time to calculate probabilities.

Next, the window size is decided upon and the count lookup table is created. The window size for this experiment is of size 4. The count lookup table tracks collocated word pairs and increments the count when w_1 and w_2 are found within the window. Once the count lookup table is created, the forward's entropy is calculated for every word in the vocabulary. The vocabulary and the entropy calculations serve as the trained model as the summarizer only depends on the entropy of the vocabulary words to score sentences.

The equation for forwards entropy of a vocabulary word is:

$$E_F(v_i) = -\sum_{j=1}^{j=|V_J|} p(v_i)\hat{p}(v_j|v_i)\ln[p(v_i)\hat{p}(v_j|v_i)] \tag{4.1}$$

where $p(v_i)$ is the probability of a certain vocabulary word v_i, $|V_J|$ is the number of unique words that are collocated with v_i, and $p(v_j|v_i)$ is the probability of v_j being collocated with v_i. The formal definition of $p(v_j|v_i)$ is:

$$\hat{p}(v_j|v_i) = \frac{C(v_i, v_j)}{\sum_{j=1}^{j=|V_J|} C(v_i, v_j)} \tag{4.2}$$

where $C(v_i, v_j)$ is the count in the count lookup table where v_i is the first word in the collocated pair, and v_j is the other. Once all words in the vocabulary have an entropy, the entropy dictionary is saved with the pickle module in order to load the trained model into memory without having to retrain every time a new summary is needed.

To generate a summary of a test article, the input sentences are cleaned of stop words and punctuation in order to yield a sequence of non-stop words. From there, every word token is looked up in the entropy model and the individual word entropy is summed over all of the word tokens in the sentence. The summed entropy is then divided by the number of non-stop words in the sentence resulting in the sentence score. Therefore, the equation for the sentence score is:

$$Score(t_i) = \frac{\sum_{k=1}^{k=|t_i|} E_F(w_k)}{|t_i|} \tag{4.3}$$

where t_i is the sentence being scored, $|t_i|$ is the total number of non-stop words in the sentence, and $E_F(w_k)$ is the entropy value for word token number k in sentence t_i.

4.2.5 Named Entity API Implementation

The Named Entity API is under the route "/named-entity/" and can be accessed by selecting the NER (Named Entity Recognition) algorithm from the dropdown on the demo page. When the summarize button is clicked, React fires off an API request to "/named-entity/" with the input text. The API takes the input text, summarizes it using the NER, and returns the generated summary. The implementation of this

algorithm on my API server is simple.

First, preprocessing is done to read in and clean the sentences of punctuation. Then, using Spacy's nlp() function, we are able to get the named entities from each sentence. The number of named entities in the sentence is divided by the number of non-stop words in the sentenced to return the sentence score. The top k scoring sentence is chosen where k is equal to 20% of the input text sentence length. This list of length k sentences is returned to the frontend as the summary.

4.3 Firebase User Authentication and Data Store

This section goes over how the web app handles user authentication and data storing.

4.3.1 Firebase Background

Firebase **cite** is a comprehensive app development platform that brings the power of a backend server to your application. Firebase has tools for user authentication, data storing, machine learning, growth analytics, hosing, and more. Firebase is simple to set up and allows for more time to be spent on working on the tools of the application and not boilerplate coding a database integration or token management.

4.3.2 Implementation

Using Firebase, user authentication is done elegantly with their react API. Firebase gives a JSON object containing the app authentication variables as well as various configuration variables. Using the Firebase javascript library, we now have access to functions like signInWithEmailAndPassword() and signInWithGoogle(). These functions handle all authentication, encryption, and user management behind the

scenes. The functions return a user object with values such as displayName and token. These values can now be used in the frontend of the web app if needed. The data store is also very elegant as Firebase has a NoSQL database service called Firestore. Importing Firestore from the Firebase library allows one to perform the CRUD operations on JSON objects in the database.

4.4 Text Summarization Demo Usage

In order to use the demo page, a user must be signed in. To sign in or sign up, the user must navigate to the sign in page by either clicking the link in the top right corner of the page or directly by accessing the /signin/ route. Once on this page, the user can sign into a previously made account or create a new account. After signing in, the user can access the text summarization tool by clicking the link that says "Text Summarization Demo" at the top of the page in the navigation bar.

On the demo page, the user may paste in or type the text they wish to be summarized, select an algorithm from the drooppdown box, and click the summarize button. Once the text is summarized, the generated summary is displayed under the "Output Summary" header text.

Chapter 5

Conclusion and Future Work

In recent years the amount of data produced has skyrocketed with social media bursting into the scene as well as other services beginning to collect data such as phone manufacturers or app developers. With this abundance of data being collected by nearly every company, there are bound to be valuable insights to be gained by analyzing the data. Examining this data can be difficult depending on how much data there is and how fast it is being produced but sometimes a small subset of this data is enough to create a useful tool or gain valuable information hidden in the data.

As more important data is transferred and stored online, more opportunities for cyber breaches and malicious software arise. Cybersecurity-based applications can be built to help fight cyber threats and we can use public data to build the models on which those applications revolve around. Creating cybersecurity tools with machine learning allows for autonomous defense which opens up time for other more involving cybersecurity tasks.

We survey text summarization algorithms to observe their ability to work with public data. First we describe TF-IDF summarization. Then, we look into TextRank with cosine similarity. Bridging the gap between extractive and abstractive summa-

rization is the Pointer-Generator network. Switching back to extractive techniques, we detail simple named entity scoring system that can sometimes outperform other more complex techniques. Lastly, an entropy-based sentence scoring system was then discussed.

Using a small portion of public twitter data to train on, we developed a machine learning model that classified fake Twitter accounts with a 90% accuracy. In addition to the model, we also created new derived ratio features in which 2 of them proved to have significant feature importance. This allows cybersecurity professionals to detect malicious Twitter accounts autonomously and therefore, they can direct their attention to other pressing cybersecurity issues.

To allow the others to learn about different text summarization techniques, we built a web application using React.js and Flask that allows a user to enter in text, select their desired summarization algorithm, and view the result. This web application allows anyone to quickly and efficiently summarize cybersecurity articles in order to effectively digest the article.

In the future, we would like to add more algorithms to the API server so there are more options for clients to demo. In addition, we would like to add pretrained neural network summarizers to this application but in my time scope, it is not possible. In the future we would like to train the entropy algorithm on the entire CNN part of the CNN/DailyMail dataset which contains upwards up of 90,000 news articles. This would increase the vocabulary of the entropy summarizer and therefore hopefully increase the quality of the entropy summarization algorithm on the demo page of the web app. In terms of the twitter bot summarizer, we would like to test other algorithms and see how the F1 scores compare instead of just using accuracy. Additionally, combining this random forest model with other machine learning classification models could prove to be a higher quality model. Another path to explore

with the twitter bot detection model is adding an additional model to review the text from account tweets.

The goal of this thesis is to show how public data can be distilled in such a way that we can learn a new insight or create a valuable tool in the realm of cybersecurity. This goal was achieved by training the Twitter bot classifier and building the text summarization web application.

Bibliography

[1] J. Desjardins, “How much data is generated each day?,” World Economic Forum. [Online]. Available: https://www.weforum.org/agenda/2019/04/how-much-data-is-generated-each-day-cf4bddf29f/. [Accessed: 05-May-2020].

[2] I. Ahmad, “How Much Data Is Generated Every Minute? [Infographic],” Social Media Today, 15-Jun-2018. [Online]. Available: https://www.socialmediatoday.com/news/how-much-data-is-generated-every-minute-infographic-1/525692/. [Accessed: 05-May-2020].

[3] StackPath. [Online]. Available: https://www.industryweek.com/technology-and-iiot/article/22026828/cyberattacks-skyrocketed-in-2018-are-you-ready-for-2019. [Accessed: 05-May-2020].

[4] usa.kaspersky.com. [Online]. Available: https://usa.kaspersky.com/resource-center/definitions/what-is-cyber-security. [Accessed: 05-May-2020].

[5] Cao, Qiang, et al. "Aiding the detection of fake accounts in large scale social online services." Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12). 2012.

[6] M. S. Rahman, T.-K. Huang, H. V. Madhyastha, and M. Faloutsos, “FRAppE,” Proceedings of the 8th international conference on Emerging networking experiments and technologies - CoNEXT 12, 2012.

[7] C. Xiao, D. M. Freeman, and T. Hwa, “Detecting Clusters of Fake Accounts in Online Social Networks,” Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security - AISec 15, 2015.

[8] S. Gurajala, J. S. White, B. Hudson, and J. N. Matthews, “Fake Twitter accounts,” Proceedings of the 2015 International Conference on Social Media & Society - SMSociety 15, 2015.

[9] Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems. 2014.

[10] Narayan, Shashi, Shay B. Cohen, and Mirella Lapata. "Ranking sentences for extractive summarization with reinforcement learning." arXiv preprint arXiv:1802.08636 (2018).

[11] Zhou, Qingyu, et al. "Neural document summarization by jointly learning to score and select sentences." arXiv preprint arXiv:1807.02305 (2018).

[12] Sherstinsky, Alex. "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network." arXiv preprint arXiv:1808.03314 (2018).

[13] Lipton, Zachary C., John Berkowitz, and Charles Elkan. "A critical review of recurrent neural networks for sequence learning." arXiv preprint arXiv:1506.00019 (2015).

[14] See, Abigail, Peter J. Liu, and Christopher D. Manning. "Get to the point: Summarization with pointer-generator networks." arXiv preprint arXiv:1704.04368 (2017).

[15] Rajaraman, A.; Ullman, J.D. (2011). "Data Mining" (PDF). Mining of Massive Datasets. pp. 1–17. doi:10.1017/CBO9781139058452.002. ISBN 978-1-139-05845-2.

[16] Mihalcea, Rada, and Paul Tarau. "Textrank: Bringing order into text." Proceedings of the 2004 conference on empirical methods in natural language processing. 2004.

[17] Page, Lawrence, et al. The pagerank citation ranking: Bringing order to the web. Stanford InfoLab, 1999.

[18] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013.

[19] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global vectors for word representation." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.

[20] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[21] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems. 2017.

[22] Tu, Zhaopeng, et al. "Modeling coverage for neural machine translation." arXiv preprint arXiv:1601.04811 (2016).

[23] Nadeau, David, and Satoshi Sekine. "A survey of named entity recognition and classification." Lingvisticae Investigationes 30.1 (2007): 3-26.

[24] Ravindra, G., N. Balakrishnan, and K. R. Ramakrishnan. "Multi-document automatic text summarization using entropy estimates." International Conference

on Current Trends in Theory and Practice of Computer Science. Springer, Berlin, Heidelberg, 2004.

[25] SK-learn RandomForestClassifier, https://tinyurl.com/SKRanFor

[26] Towards Data Science. (2018). The Random Forest Algorithm – Towards Data Science. [online] Available at: https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd [Accessed 20 Dec. 2018].

[27] Pew Research Center: Internet, Science & Tech. (2018). Social Media Use 2018: Demographics and Statistics. [online] Available at: http://www.pewinternet.org/2018/03/01/social-media-use-in-2018/ [Accessed 20 Dec. 2018].

[28] twitter-statistics, (2018). [online] Available at: http://www.internetlivestats.com/twitter-statistics/rate [Accessed 20 Dec. 2018].

[29] Quartz. (2018). Twitter has a serious bot problem and Wikipedia might have the solution. [online] Available at: https://qz.com/1108092/twitter-has-a-serious-bot-problem-and-wikipedia-might-have-the-solution/https://qz.com/1108092/twitter-has-a-serious-bot-problem-and-wikipedia-might-have-the-solution/ [Accessed 20 Dec. 2018].

[30] Techterms.com. (2018). Captcha Definition. [online] Available at: https://techterms.com/definition/captcha [Accessed 20 Dec. 2018].

[31] Makice, K. (2009). Twitter API: up and running. Sebastopol, CA: O'Reilly.

[32] The Paradigm-Shift of Social Spambots: Evidence, Theories, and Tools for the Arms Race, S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, M. Tesconi.

WWW '17 Proceedings of the 26th International Conference on World Wide Web Companion, 963-972, 2017

[33] Fame for sale: efficient detection of fake Twitter followers, S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, M. Tesconi. arXiv:1509.04098 09/2015. Elsevier Decision Support Systems, Volume 80, December 2015, Pages 56–71

[34] Widenius, M. and Axmark, D. (2002). MySQL reference manual. Beijing: O'Reilly.

[35] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[36] Travis E, Oliphant. A guide to NumPy, USA: Trelgol Publishing, (2006).

[37] Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010)

[38] Scikit-learn.org. (2018). 1.13. Feature selection — scikit-learn 0.20.2 documentation. [online] Available at: http://scikit-learn.org/stable/modules/feature_selection.html [Accessed 20 Dec. 2018].

[39] "React – A JavaScript library for building user interfaces," – A JavaScript library for building user interfaces. [Online]. Available: https://reactjs.org/. [Accessed: 05-May-2020].

[40] "Welcome to Flask" Welcome to Flask - Flask Documentation (1.1.x). [Online]. Available: https://flask.palletsprojects.com/en/1.1.x/. [Accessed: 05-May-2020].