Development of classification models for fake news detection

Individual Research Project Report
University College London
Javier Pascual Mesa

I. Introduction

This report aims to analyze the overall performance of the team throughout the course of our research project: evaluating ways to detect fake news spreading from Twitter. I will mainly focus on my personal contribution to the team, and how I linked my work to everyone else's.

Twitter is an extremely popular platform for spreading information and opinions on the Internet. The ease with which users can post any kind of message has given rise to the abuse of this social platform as a way of circulating opinionated and completely fake news. This is done through the different parts which make up the structure of a tweet. Over the course of the project, my responsibility mainly focused on exploring ways to analyze the textual aspect of Twitter. After having evaluated several ways in which text can reflect the truthfulness of tweets, I decided to dedicate my research to article classification from tweet URLs. The goal was to optimistically create a program that would more or less accurately conclude the veracity of certain articles spread through Twitter. The process involved a lot of research, experimentation (which will be discussed further in the paper), and collaboration with other members of the team to explore ways of merging our work together. The stages involved in the development of my classification models were:

1) Creating a dataset of both real and fake news articles, large enough to reliably create a model capable of classifying other articles not used as training data.
2) Find a way to scrape the text from online articles provided by URLs.
3) Actual development of the classifier models.

II. Personal contribution

My personal contribution to the team project was a model that could predict (within a certain degree of accuracy), how real or fake news articles were. Additionally, I used some libraries (discussed afterwards) to produce a measure of the sentiment of a tweet. Research, along with trial and error, were very important throughout all stages of development.

A. Data Gathering

First and foremost, I had to identify the sources from which I would be gathering data to train and build classification models.

I obtained the data for the training and testing of the models from different sources. Fake articles were taken from the Kaggle dataset that provided a total of 12,999 news posts classified as "Fake". To compensate for the fact that real news make up a larger percentage of total articles, the number of real posts I used for training the models was significantly higher. For the extraction of the latter, I initially turned to scraping Google News, through the use of a python wrapper for selenium and searching for articles posted in specific websites. This took some time as I had to find a way to change my IP address dynamically in order to bypass Google's "I am not a robot" captcha. An example of what the scraping program would type into the Google news search bar is "site:cnn.com politics", which would query for articles relevant to cnn.com and containing the keyword politics. I would then iterate over a list of reliable sites and scrape as many articles as I wanted. However, the approach was not always accurate and sometimes I ended up scraping articles that were not related to the sites I typed (which in turn meant that I could not guarantee the article I was extracting was reliable).

I eventually came across webhose.io, and decided to make use of it due to its simplicity and completeness in terms of customizable options available when querying for data. This API allowed me to query its database with site URLs I wanted to retrieve articles from, and thanks to this, I was able to download political articles from the most reliable news sources according to Forbes[1], which include sites like cnn.com and nytimes.com. In total, about 28,000 real articles were used in training. As the dataset of fake articles was not built by me, I additionally decided to scrape some article URLs from infamous websites that tend to post fake news[2]. Many of these sites are currently not operational, but used to be when I scraped them.

Overall, I ended up with a dataset of 40,000 news articles. However, the data had been saved across different formats, and I had to create python functions to merge it all into a common CSV file with columns "text" and "type". Throughout the entire project, I made use of the Pandas python library to create data frames from existing CSV files, which made manipulating and storing data a lot easier.

B. Data processing

The downloaded text per se was of no use until it was cleaned up. During the first attempts to create a classifier, the model's most influential features were often a sequence of digits or part of the websites HTML code (as the method used for extracting the article's text is not always perfect). For this, a number of transformations were applied:

- Removing non-alphabetical characters from the text through regular expressions. The logical reasoning for this was that, although some numbers could be useful features in the model, like "2017", for instance, many of the other numbers would be irrelevant and would make our classifier less accurate.

- Creating a list of stop-words that were often coming up as very important features in the created models, when it was obvious these words were there because of things like site structures. The words would be removed from the downloaded article text before training.
- I also considered removing all text within speechmarks, as quoted text should in theory be irrelevant to the points an article is trying to make. However, I eventually concluded that many fake sites will actually make up quotes in their posts, meaning that any words found can be crucial to correctly classify a story.

C. Model development

This part involved a lot of experimentation and research regarding the different classification models that were available. For this purpose, I made use of the scikit-learn python library, which provides a very complete set of tools for machine learning. The goal was to produce a binary classifier that would be capable of judging whether the text extracted from an article was more likely to be fake, or real. I tested at least 5 different types of classifier and tuned different parameters for each of them hoping to obtain better results.

As an argument for the classifier's constructor, I needed a matrix of vectorized features. To achieve this goal, there were two candidate vectorizers: CountVectorizer and TfidfVectorizer. The former simply counts the frequency of each word in the document, while TfidfVectorizer additionally helps account for the fact that some words may appear more than others. Tfidf transforms the frequency f from every word w in a document by applying the following transformation, where d is the total number of documents over which the transformation is being done, and c the count of documents where w appears[3]:

$$tfidf(w) = log(f+1) \times log(\frac{d}{c})$$

As expected, the TfidfVectorizer produced better results, so I used it prior to training my models. In order for the transformed vectors not to include words that appear too often, I passed a list of stopwords to the function. A minimum threshold frequency for a word to be considered was also set, allowing the process to filter out some useless or non-influential words.

The next step was selecting the most appropriate classification model/algorithm (or the most accurate one when used on the data gathered). To begin with, we had to split the collected data into training and testing data. Eventually, about 17.5% of articles in my dataset (7,000) were used for testing. This would allow us to see what percentage of the labeled articles were classified correctly, from a total of 6,000 real and 1,000 fake articles. For each trained model, a confusion matrix was plotted to display and help us evaluate its performance (see Figure 1).

The first tested classifier was a Multinomial Naive Bayes, which receives its name from using the Bayes theorem. This is a specialized Naive Bayes classifier that is more fit towards classifying text documents[4]. We use the "naive" type of algorithm, as it assumes that each word is independent from each other. What this means is that we avoid comparing entire sentences, which would otherwise lead to a very weak training (as the likelihood of a sentence appearing multiple times throughout a variety of documents is quite low). The outcome performance was actually disappointing, with an obvious imbalance in classification where most articles were being classified as "REAL". This was most likely due to the large amount of real news articles in the dataset, but after balancing the trained data, the results were still (despite some improvement) not better than ones obtained with other classification models. This was probably a consequence of one of the largest disadvantages of a Multinomial NB classifier, which is the fact that it strongly assumes the independence of words in a document, as mentioned before.

The next tested classifier was the Random Forest, which tends to increase in accuracy as more "trees" are included in the forest. The main advantages that this algorithm brings to the table include handling of Null or empty values, and the fact that it does not over-fit our classifier [5]. The accuracy seemed a bit better than with the Multinomial NB, as the percentage of accurate "FAKE" classifications increased to 59%. However, I was still not satisfied with the performance and started testing other models.

Next, I tried creating a Logistic Regression classifier[6]. Logistic regression makes use of a logistic function (known as the Softmax function), which receives the likeliness of words occuring in a document as input, and attempts to predict the target label for the particular text by scaling the values z passed to the function so they add to 1. The softmax function is as follows, where e is Euler's number:

$$\sigma(z)_j = \frac{e^{z_j}}{\Sigma_{k=1}^{k} e^{z_k}} \quad for\, j = 1, ..., k$$

This type of model finally delivered higher accuracy results. with 80% and 76% accuracy in the prediction of fake and real articles respectively. The only issue with these results was the fact that we had a decent portion of false negatives (classification of real articles as fake articles), which I wanted to improve given the fact that there is a higher proportion of real articles online.

I eventually tested a Linear SVC [7] (which I initially discarded due to the long time it took to train). This type of model is based on a Support Vector Machine, which is a machine-learning algorithm commonly used for classification purposes. The main objective consists in creating a hyperplane to divide both target classes (in this case, "REAL" and "FAKE"). The results looked very promising, as our correctly classified fake articles were only 4% less , and had about 10% more correctly classified real articles. This left us with about 90% accuracy for real articles and 76% for fake ones, seriously reducing the amount of false negatives in our test predictions. The biggest downside of using a Linear SVC for training, was the fact that it took about 3 hours to train every time I wanted to fit a new classifier with different datasets. Luckily, once the model was trained we would only be using it for predictions (and not retraining), so the issue was only time-consuming during the testing phase.

Once the models had been developed, the important data used in the building process was stored in files so they could be retrieved for future predictions without having to retrain.

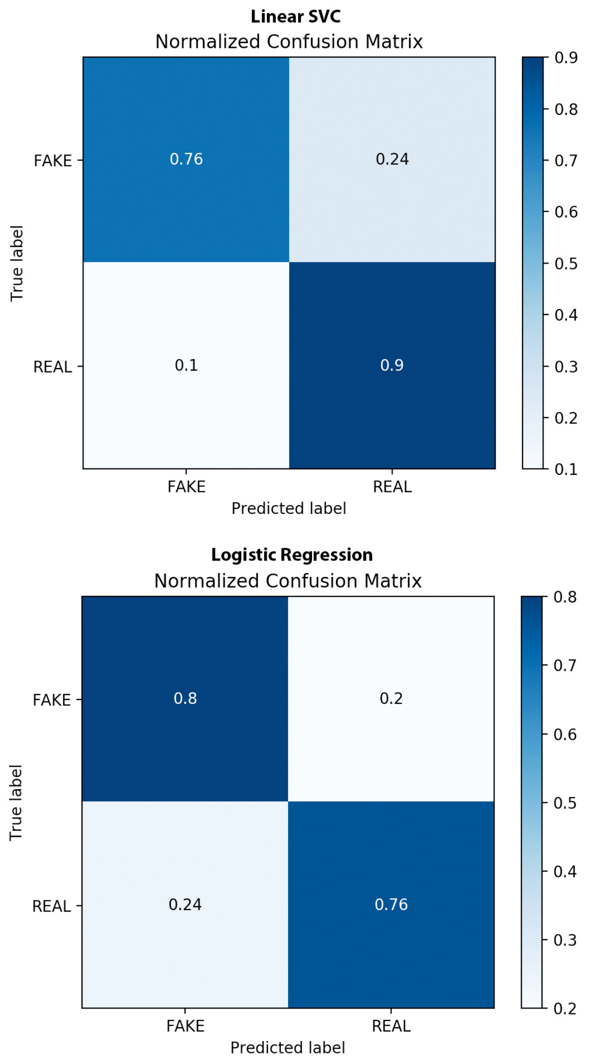

Fig. 1: Confusion matrices for test data of the two most accurate classifiers produced

D. Model application

After the models were trained, I had to find a way to apply the classifiers to different news articles spread through tweets. We wanted to retrieve the URLs from within tweets and attempt to classify the linked articles into real or fake using the models I produced. Firstly, I was responsible for finding a way to extract the article text from the URLs. For this purpose, I made use of a very useful python library known as newspaper, which would parse the HTML from the provided link into separate data like the header and the actual text of the story. This helped me convert a CSV file of URLs with labels (“FAKE” or “REAL”) into a CSV containing the article texts and labels. Nevertheless, the time taken to download text from articles was not as fast as I wanted, considering we had to do the same process for many URLs across many different tweets. So, in order to produce faster classifications of all URLs within a tweet, I had to create a pool of processes using the multiprocessing python library, which allowed me to download the text and predict the label for every article within the tweet concurrently. The complete list of predictions (along with the probability of each of them being real or fake) could then be passed on to the rest of the team’s project.

To make a more interactive way of testing my model, I developed a simple django web application that would allow users to input the URL of an article, and the page would display the predicted label for it. However inefficient this approach may seem for testing, it allowed me to distribute the link of the site to different people (like my group and friends) and get their feedback on tests they had carried out themselves using my classifier. The webapp was deployed using Heroku, which is a cloud platform that supports various frameworks and made it easier for me to push the django application online.

Through the use of the developed models, I was also able to extract what the most influential words had been in each of the learning attempts of every classifier. An example of this with the Linear SVC model can be seen in Figure 2

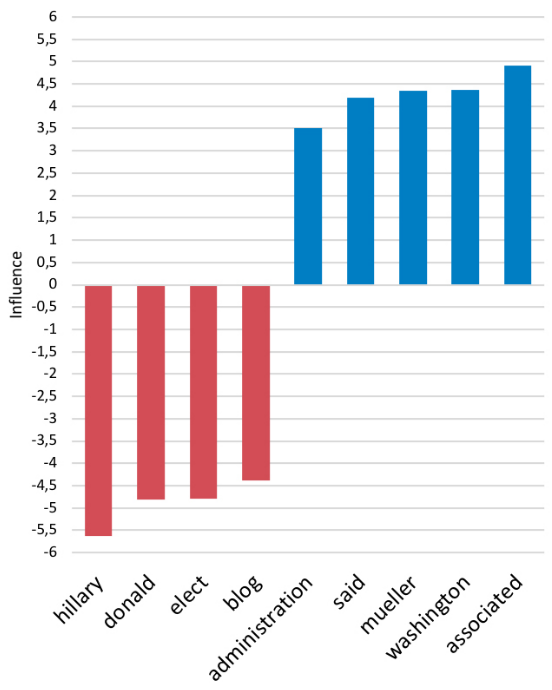

Fig. 2: Most influential words for the Linear SVC model

E. Sentiment analysis

Despite the lack of time, I wanted my part of the code to return some sort of input on the sentiment of the tweet we were currently analyzing.

My first approach was to develop a model in the same way I had done for the fake/real classification. However, I had to search for a dataset containing text excerpts that were already labeled as having positive or negative sentiment. I eventually found a pre-classified tweet sentiment dataset containing more than 1 million tweets. After playing around with the data, I developed a Multinomial NB classifier that had around an 80% success for classifying sentiment correctly into positive or negative. Unfortunately, I found out that the dataset had been partially developed automatically (by a program that would, for instance, classify a tweet as positive if it had a smiling emoticon). Automatic sentiment analysis is still something being improved on, and manually labeled datasets are definitely more reliable, so I started looking for alternatives.

Then I came across the Textblob python library, which is a wrapper around NLTK and Pattern. The Natural Language Toolkit is a python library which provides tools that ease working with language data. These tools include things like sentiment analysis. Additionally, Pattern gives Textblob a variety of morphological processing features. The results from using Textblob were very convincing, but the library was still having issues with some specific words not being recognized like “don’t”.

While reading a paper on sentiment analysis through utilizing the nltk library [8], I came across the Vader module. This is a pre-built text analyzer that returns the positivity, neutrality and negativity of a particular piece of text. At the end, it turned out to be more useful than the other approaches. Vader takes into consideration a variety of aspects from the text:

1) Sentiment of emoticons
2) Letter case of the words and punctuation
3) Booster words like ”greatly”
4) Slang words

As textual aspects like slang and emoticons are very common in Twitter, and some of the words Textblob was not recognizing were being accurately detected by Vader, I decided to make use of this module.

Figure 3 shows a table comparing the calculated polarity for each of the three alternatives tested. Polarity measures how positive or negative the text is, and will range from -1 (very negative) to 1 (very positive). It is pretty clear that the Multinomial NB ignores punctuation, as the polarity stays the same when adding exclamation marks at the end of the same sentence. In TextBlob and Vader, however, the exclamation marks serve as a way of amplifying the polarity of the text. The classified model ignores emoticons, as they have been filtered out prior to training, while the other two options take them into consideration. The determining factor in the selection of one of the two came down to a lot of testing. Eventually, I realized Vader was detecting some words that TextBlob was ignoring.

In the future, I intend to further explore the limitations and possibilities of both the VADER module and the nltk library as a whole, and possibly build my own sentiment classifier again, based on a better dataset than the ones I was able to find.

Statement	MNB	TextBlob	Vader
I love the new environmental policies.	0.56	0.32	0.64
I love the new environmental policies!!!	0.56	0.38	0.72
Does Trump think this is a joke? :(	0.23	-0.75	-0.18
I don’t like the new president	-0.21	0	-0.28

Fig. 3: Performance comparison from different sentiment analyzers on example sentences

F. Connection to team project

The final phase in my individual contribution to the research project was merging everything I had done with the code my teammates had written. To connect my code with the rest of the team’s work, I created a class that can be constructed with the name of the folder where data has been stored of a previously trained classifier. Additionally, a user can construct an empty class and train from a CSV any of the four types of classifier mentioned before. The class can also take in a tweet in JSON format (downloaded using tweepy), and extract all URLs and text from the tweet. The program then evaluates every URL concurrently, analyzing if the page it links to is considered an article through the use of a script which crawls the BlueCoat WebPulse site reviewer. This classifier returns the category a type of article falls into (politics, news, spam etc). Initially, I made my own script for scraping this site reviewer, making use of the python libraries mechanicalsoup (based on beautifulsoup), and requests. However, after a lot of testing, I came to the conclusion that BlueCoat was not very accurate when attempting to classify non-popular news sites. Hence, I discarded the code. However, Chris later told us he had made his own script for the same purpose, and we eventually ended up using it only to provide the user of our platform with some additional information.

The classifier operated by the class then predicts the veracity of each post (once they have been downloaded using the newspaper library), and returns the probability of them being real or fake. Additionally, the features (words) that have been the most influential in the classification are returned too. All of this information is packed into a JSON dump, together with the predicted sentiment obtained using the previously mentioned VADER module. A full representation of the JSON returned from my code given a tweet can be seen in Figure 4. Below is a sample code which can be run to get all information my "newsmodel" library provides for a given tweet, when passing the tweet in JSON format.

```
>>> import newsmodel as nm
>>> mod = nm.newsModel("LinearSVC")
>>> mod.testTweet("Tweet json goes here")
```

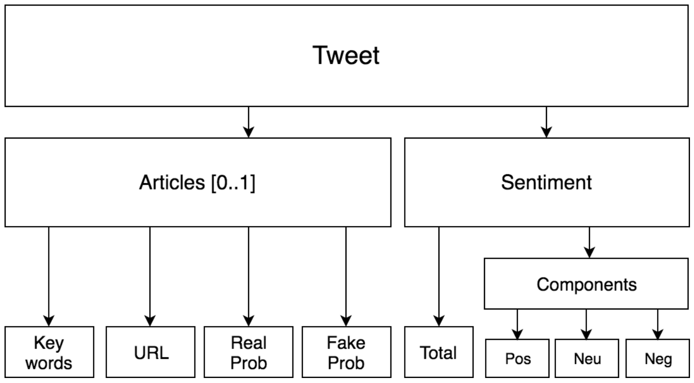

Fig. 4: JSON format of what my part of the code returns to the group project

III. Project assessment

A. Expectations

The expectations for the project were mainly based on analyzing the issue of fake news spreading through social networks from different perspectives. Each member would be responsible for undertaking key research tasks that would revolve around gathering data and making use of that data in one way or another. Eventually, we would join all programs developed to gather as much information as possible relating to fake news on Twitter, or act as an on-demand info provider. Ideally, we would be able to visualize some sort of patterns even if the end-product was not 100% viable for production.

B. Critical assessment of results

In terms of the results achieved, I believe the team tackled the problem at hand from many different angles, leading to an extensive analysis of many of the features that relate to the spreading of fake news on social networks (images, text, hashtags, etc).

From the side of image processing, we were able to store the hash values of certain images posted on Twitter in a database, and make use of hashing algorithms to compare newly obtained hashes with the ones we already had. This is extremely useful for identifying potential sources of news that repeatedly take images from already existing articles. Although this is not a clear reason for articles to be fake, it is an interesting factor to consider when evaluating the veracity of news sources.

We were also able to correctly identify the most used hashtags and the most reoccurring mentions to Twitter accounts, through analyzing the metadata of the Tweet. Additionally, URLs containing articles found in Tweets could be scraped using my program, and we could get input on the veracity and sentiment of the tweet with use of the classification models mentioned earlier.

Overall, I believe the depth of our analysis was, given the limited amount of time we had, of a considerably high standard.

C. Scope for improvements

As for potential improvements in the future, the team could build up on some areas that would further strengthen the effectiveness of the work done. For instance, the integration of each member's work into the final deliverable could have started earlier. Had each individual known a bit more about what the rest of the team was developing, the resulting web-app could have been even more cohesive.

IV. Team assessment

A. Overall assessment

Overall, the team worked very well together, and thanks to good communication and flexibility by all team members, a clear division of labor was made from the very start. A meeting was held almost every week, which served the team as a way of briefing what each of us had done during the past few days. The members would take turns to speak and get feedback from both the project supervisors and the rest of the group, which helped build everyones confidence and ensure we were all on the same page. Everyone was supportive with one another, which lead to a positive environment were everyone felt comfortable working in.

On the technical side, contribution was more or less well distributed across the team members, and everyone carried out significant research during the project to work on achieving the ultimate objective of analyzing fake news on Twitter.

B. Individual assessments

1) Christopher Hammond: Chris was one of the most determined team members during the course of the project, and was responsible for producing the python program called "Twipeline", which would gather data from Twitter using the Twitter Streaming API. Although his work was not aimed at researching possible ways to detect fake news, he provided the basis for the rest of the team to have data to test and analyze.

Chris was very active during meetings, and would constantly think of ways to perfect the system he had already built (using elastic search). It was obvious that he had previous experience working with Celery and Redis, which ended up being extremely useful for the team when collecting and processing a plethora of tweets at a time. His knowledge on concurrency also turned out to be very useful, and he always had something to say as helpful input when brainstorming ideas. For example, he brought up the multiprocessing python library which I ended up using.

As a down side, Chris was sometimes too explanatory with his briefings, and would lose focus of what the actual topic at hand was: detecting fake news. Additionally, I had difficulties trying to get him to test my code on his side. Eventually, he did test my code on the data he was collecting, but by the time I got the feedback from him we were already extremely close to the deadline.

2) Ashu Savani: Ashu was assigned the task of image analysis, and focused on detecting similarity of images being spread through tweets. For this purpose, he utilized perceptual hashing algorithms (dhash) to store hashes in our database, and later compare new hash values with the ones we had using hamming distance to find the most similar results.

In terms of strengths, Ashu was very enthusiastic about the area he had to study. His approach to finding the best hashing algorithm was very methodical, as he first evaluated the speed of all perceptual hashing algorithms, and then narrowed down by finding the most accurate one. Additionally, he was timely with his deliverables and always shared his doubts during meetings to get feedback from both supervisors and the rest of the team.

As for weaknesses, Ashu sometimes failed to communicate with the rest of the team. Both him and Chris knew each other from before, which is why I thought they sometimes talked amongst themselves about what they were doing and in a way refrained from briefing us on the progress.

3) Carlota Ortega: Carlota was responsible for analyzing the meta-data of Twitter posts. From the meta-data, she explored aspects like term frequencies of hash-tags, words, URLs and mentions. Additionally, she looked into the times at which tweets were being posted, hoping to identify any existing patterns in different types of account. Finally, she also studied the geodata of tweets and Twitter accounts.

Throughout the project, Carlota offered a lot of positivity to the team, and she constantly tried to analyze things from different angles. This can be seen from the effort put into exploring the various aspects of Twitter metadata.

In the future, Carlota could improve her performance by starting her research earlier. This in turn would have meant that she could have started testing research applications earlier, and consequently, her outcome would have been even better. Additionally, Carlota could have tried to interact more during our meetings, as it did seem at times like she feared saying something wrong. For future projects, I would say Carlota should try to get more input from team members on what she is studying/producing, as I believe we could have provided her with more useful feedback had she let us known earlier what the issues she was having were.

4) Haran Anand: Haran was mainly responsible for producing the user interface for our project. Once all projects had been connected in the background, he had to funnel all of it into an interactive web application that would help users make use of our research and developed applications.

Haran was always positive and eventually proved to have strong knowledge of web development using Django. One of us had to develop the visual aspect of the research we had carried out, and I believe Haran did do a good job at it.

I personally think his main weakness was not contributing enough to the team effort, possibly due to lack of communication. Although very useful, his deliverable was made during the last week of the semester, which left us with little time to react as the deadline was only a couple of days away. I believe we could have tackled more aspects of fake news on Twitter if he had started researching during the first couple of weeks of the project, and potentially come up with alternative ways of studying the effect of fake news on Twitter aside from the ones we were already digging into.

5) Javier Pascual, myself: As explained before in this paper, my main responsibility was looking at potential ways to exploit the text aspect of tweets and, ideally, output some sort of analysis of the latter. I mainly concentrated on the sentiment of the tweet's text and the development of veracity classifiers for articles shared through the actual tweets.

I believe I made a significant contribution to the team's project, and updated my team regularly with the milestones I was reaching in the process of developing and improving the classification models I was trying to build. I was very eager about the project, and started writing scripts to analyze tweets very early on. Finally, I was always willing to help my team with any issues or doubts they were having concerning both their part of the project, and my own.

As far as my weaknesses are concerned, I think I could have been more regular throughout the project, and I should probably work more on my time management skills. For instance, there were weeks during the semester where I would dedicate my time almost solely to the research project, and others were I concentrated on other things and almost could not work on improving the classifiers. Additionally, I should have identified the specific routes I wanted to take when applying my research to the ultimate output for the project. This is due to the fact that I began by trying many different things out, which I ended up not using or taking advantage of, and I could have made use of the lost time to work on improving the sentiment analyzer, for instance.

V. Conclusion

To sum up, results proved that it is indeed possible to identify fake news without overly complicated models, although classification is not always accurate. Linear SVC and Logistic Regression models seem to have the best performance for the particular task at hand, and with the particular dataset built. As far as sentiment analysis is concerned, it is still a very challenging topic, but many libraries already provide more or less sophisticated analysis methods to reflect the overall polarity of a text segment. Emoticons, intensifiers and punctuation are some of the textual aspects that sentiment analyzers utilize (aside from the estimated negativity or positivity words tend to carry).

The combination of software developed by the team really helps provide an insight into the range of methods that are available for examining how truthful a news source may be. Overall, the experience of researching such an important and concerning topic was very rewarding. Working as a team to attack as many relevant issues as possible helped me grow more aware of the severity of the problem, and significantly improved my research skills.

In the future, I intend to keep digging into different machine learning alternatives (like neural networks) to increase the precision of the classification. As time advances, fake news will undoubtedly become more sophisticated, and discerning them from real stories with simple classifiers will not be as easy. Additionally, I want to dedicate more time to creating my own sentiment analyzer, possibly through the combination of multiple libraries or modifying existing ones.

REFERENCES

[1] Glader P., “10 Journalism Brands Where You Find Real Facts Rather Than Alternative Facts”, 2017, Retrieved from https://www.forbes.com/sites/berlinschoolofcreativeleadership/2017/02/01/10-journalism-brands-where-you-will-find-real-facts-rather-than-alternative-facts/\#3a5aabede9b5

[2] Criipi, “Fake news sites”, 2017, Retrieved from https://gist.github.com/Criipi/a3a7357466821f2ec62ce42b2529394b

[3] Kibriya A. M., Frank E., Pfahringer B. and Holmes G., “Multinomial Naive Bayes for Text Categorization Revisited”,2004, Retrieved from: https://link.springer.com/chapter/10.1007/978-3-540-30549-1_43

[4] McCallum A. and Nigam K., “A comparison of event models for Naive Bayes text classification”, 1998, Retrieved from: http://www.cs.cmu.edu/~knigam/papers/multinomial-aaaiws98.pdf

[5] Polamuri S., “How the random forest algorithm works in machine learning”, 2017, Retrieved from http://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/

[6] Jurafsky D. and Martin J. H., “Speech and Language Processing”, Chapter 4: Logistic Regression, 2017

[7] Bambrick N., “Support Vector Machines for dummies; A Simple Explanation”, 2017, Retrieved from: http://blog.aylien.com/support-vector-machines-for-dummies-a-simple/

[8] Shravan I.V., “Sentiment Analysis in Python using NLTK”, 2016.

[9] McIntire G., “On Building a Fake News Classification Model”, 2017, Retrieved from https://opendatascience.com/blog/how-to-build-a-fake-news-classification-model/

[10] E. Ferrara, O. Varol, C. A. Davis, F. Menczer, and A. Flammini, “The Rise of Social Bots”, 2014, Retrieved from https://arxiv.org/pdf/1407.5225.pdf

[11] S. Zannettou, T. Caulfield, E. D. Cristofaro, N. Kourtellis, I. Leontiadis, M. Sirivianos, G. Stringhini, and J. Blackburn, “The web centipede: Understanding how web communities influence each other through the lens of mainstream and alternative news sources”, 2017, Retrieved from https://arxiv.org/abs/1705.06947

[12] C. Fan, “Classifying fake news”, 2017, Retrieved from http://www.conniefan.com/wp-content/uploads/2017/03/classifying-fake-news.pdf