Detection of Fake News using Text Classification

Roshan N. Lasrado
Computer Science
University of Victoria
roshanlasrado@uvic.ca

Parisa Moghaddam
Electrical and Computer Engineering
University of Victoria
parisamoghaddam@uvic.ca

Zaid Khan
Electrical and Computer Engineering
University of Victoria
zaid@uvic.ca

Abubakar Siddeeq
Electrical and Computer Engineering
University of Victoria
asiddeeq@uvic.ca

Donya Haghighi
Electrical and Computer Engineering
University of Victoria
dashtianihaghighi@uvic.ca

Abstract

Now-a-days, un-authentic and fake news plays a vital role in diverting the perception of a targeted audience. With the increase in usage of social platforms like Facebook, Twitter, etc, people are more inclined to absorb such fraudulent news. In addition, it is impossible to rectify these statements due to the indulgence of biased mass media platforms. Ukraine, Germany, the United States of America, China, and many other countries have a plethora of such platforms [10]. The purpose of such news is to affect the opinion of the common man in a certain manner to achieve the desired result (mostly financial and political). In this project, we will be using an ensemble of Machine learning Algorithms and techniques. Based on the experimental analysis, the algorithms predict the probability that the given content is authentic or not. We also introduce a new dataset by scraping online news fact checking sites such as PolitiFact.com for the task of fake news detection.

1 Introduction

Detection of fake news has become a focal point in the news business along with the society we live in. The process of creating unauthentic news just to attract the user for financial and political purposes has been present for a long time. Particularly in the case of social media, due to its immense reach and spreading of information at such a high rate, fabricated and false statement has acquired a tremendous potential in spreading the info and causing a greater effect in society in no time, affecting millions of users.

Obtaining news updates while using social media as the medium is like a double edge sword. Social media has its pros and cons. On one side, it gives us instant access to global information at no cost. While on the other side social media can be used for transmission of fake news. As compared to the fact, fake news spread fast and is far more influential. Due to the widespread availability of the internet, more people are exposed to such news. For instance, in 2016 more than sixty-two percent of American adults obtained news from platforms like social media as compared forty-nine percent in 2012, which shows the increase in usage of the internet with time [10].

Preprint. Under review.

Also, the effect of fake news is not only limited to an individual or a person, but it also has a lasting effect on our society in the long run. For instance, in the US presidential election in 2016 fake and fraudulent news were more popular on social media as compared to the authentic one. Which shows that people are more biased towards conspiracy as a result of which it is relatively easy to manipulate people's opinion [11]. Studies show that the probability of human detecting fake news is slightly higher than 50% obtained which is obtained over 100 experiments consisting of 1000 volunteers [12].

Detection of such news is significant and quite difficult in terms of technicality as the only way the human can detect such fraudulent news is by having vast knowledge of the respective topic. Many websites are present right now which play an essential role in exposing such news by fact-checking. Due to the presence of massive datasets and economical hardware, Machine Learning techniques such as random forests, neural networks, and AdaBoost have become very effective in solving complex classification problems [11].

The remaining sections of the paper are structured as follows. Section 2 provides a review of the related work and some background information. Section 3 provides a background regarding the data preprocessing and feature extraction techniques used. Section 4 presents the experiments conducted and the various machine learning models used for building the fake news classifier. Section 5 presents some concluding remarks.

2 Related work and background information

As fake news is a relatively recent phenomenon, at public advertising websites such as Google, Facebook game, Twitter which are the main sources of dispersing fake news, research in this area is still at first steps and started in 2016, especially after the last US presidential elections. Recent statistics show that social media is used by 62% of US adults for getting news [1, 2]. Fake information can be categorized into 3 types. The first type is fake information built up by authors. The second type is incorrect satire information, and its main purpose is certainly presenting humor to the readers. The third type is drafted reports content, the news which is not totally precise. In [3], they argued about three types of fake news which each type of fake news is an inexact or unreliable reporting representation, including serious fabrications, large-Scale hoaxes and humorous fake news. Furthermore, the authors use the various types of fake news and text analytics and predictive modeling methods in detecting them. According to the authors, different types of fake news could have an adverse effect on the effectiveness of text classification techniques.

In [4], they showed the difference between fake and honest articles is distinguishable. According to their observations, there are fewer stop-words and nouns and more nouns and verbs in fake news titles. Different features were extracted which are grouped into the complexity features compute the complexity and readability of the text, psychology features measure the cognitive process and personal concerns in texts, such as the number of emotion and casual words, and stylistic features illustrate the style of the writers and syntax of the text, such as the number of verbs and nouns. These features were used in an SVM classification model. The dataset used in this article consisted of real news from BuzzFeed and other news websites, and Burfoot and Baldwin's satire dataset [5]. The achieved accuracy of comparing real news against humorous articles was 91%. However, the accuracy fell to 71% after predicting fake news against real news. In [6], they used LIAR for the first time as a new dataset that can be used for automatic fake news detection which is significantly bigger in size, and it contains 12800 manually labeled short statements from politicalFact.com. In [7], they focused on detecting opinion spam and fake news using n-gram analysis by using different features extraction methods. The n-gram features performed better real-world data and pseudo data when applied on the fake news data. The type of data used for training has an effect on the classifier performance. They showed that by increasing the features, the majority of the classifiers achieved a higher accuracy. In addition, a growth in the n-gram size decreases the accuracy. In [8], they presented a model for identifying 360 satire and humor news articles which were examined and inspected in mainly four domains, namely, civics, science, business, and what soft news. SVM classification model was proposed based on their analysis of the satirical news. Their highest accuracy of 90% was achieved using three specific features which were Absurdity, Grammar, and Punctuation.

In [9], they presented a detection model for fake news using n-gram analysis by using different features extraction techniques. Furthermore, they considered two different features extraction techniques

and six different machine learning techniques. When they used unigram features and Linear SVM classifier, the proposed model reached its highest accuracy which was 92%.

3 Data collection, preprocessing and feature extraction

3.1 Data collection

We introduce a new dataset for the task of classification of news articles, which in this paper is limited to political statements, as real or fake. This dataset consists of news articles and political statements that are both current and are related to a wide range of issues. A robotic process automation (RPA) script is written using the UI Path tool for efficiently scraping news articles from the fact checking websites. This script with some modification could be leveraged for extracting news articles from other fact checking news websites.

The fact checking news labels the news articles into 6 categories, namely 'True', 'Mostly true', 'Half true', 'Mostly false', 'False' and 'Pants on fire' depending upon various supporting or contradicting facts and statistics. To further build up the data corpus, we extract articles belonging to 6 different domains, namely 'crime', 'environment', 'foreign policy', 'guns', 'immigration', and 'taxes'. The tuple in the dataset consists of the statement, the source of the statement, details regarding when and where the political statement was made and the ruling regarding the veracity of the statement as indicated by the above 6 categories.

3.2 Data preprocessing

Before subjecting the data to further analysis, the data is first preprocessed using techniques such as stop word removal, removal of special characters and digits, lowercasing the text, stemming and tokenization. This helps standardizing the data and makes the process of representation of data using Term frequency (TF) and Term Frequency – Inverse Document frequency (TF-IDF) more efficient.

A generic data preprocessing function is created, which helps subject both the training and the test data to the same form of processing. For representation using word embeddings, the stemming step of data preprocessing is skipped.

3.2.1 Stop word removal

Before representing the textual statements using TF, TF-IDF and word embeddings, they are first subject to stop word removal. Stop words are the most commonly used words in a language. They are generally consists of articles, prepositions, conjunctions and pronouns. Removal of stop words helps significantly reduce the amount of processing required for feature extraction. A standard list of stopwords for English language from the NLTK library corpus are employed for identification and removal of stop words.

3.2.2 Stemming

After stop words removal and tokenization, the tokens are converted into a standard form using stemming. This helps significantly reduce the amount of word types and classes in the data. This further helps to make the classification algorithms faster and efficient. For performing stemming, we use Porter Stemmer, which is one of the most widely used stemming algorithms.

However, since we are interested in utilizing temporal patterns in the textual statements for classification when using word embeddings and sequence models, we do not subject the data to stemming for these techniques.

3.3 Feature extraction

The high dimensionality of the data is a major challenge when performing classification using textual data. Even after being subject to stop word removal and stemming, there still remain a large number of terms that may cause the classification algorithms to be inefficient. Word representations such as one-hot encoding, may be simpler to compute, such representations do not provide any computational or representational benefits.

In this paper, we use the Word Embeddings, Term frequency (TF), and Term Frequency – Inverse Document frequency (TF-IDF) for representing the textual statements.

4 Experiments

4.1 Random forest and KNN

For these two algorithms, we started by hyperparameter tuning. In other words, we want to adjust some settings for the algorithm to make it do the best performance. As we know evaluating the model only based on train data could cause one of the well-known problems in the machine learning world which is overfitting. Accordingly, we applied cross-validation which is the standard procedure of preventing our model from overfitting. By using Scikit-Learn's RandomizedSearchCV method, we can define a grid of hyperparameter ranges, and randomly take a sample from the grid and perform K-Fold CV with each combination of values. For random forest and k-nearest neighbours algorithms, 5-Fold is used. Further, we will discuss more the hyperparameter tuning specific to these algorithms.

4.1.1 Random forest

In the case of random forest, these settings could be important to consider them: the number of decision trees and the number of features that are considered by a tree to split on. More precisely, n_estimators and max_features in Scikit-Learn. Furthermore, the max number of levels in each decision (treemax_depth), min number of data points placed in a node before the node is (split-min_samples_split), min number of data points allowed in a leaf node(min_samples_leaf) are also adjusted. Another observation shows that for the criterion parameter (the function to measure the quality of a split) is entropy for this problem. Best parameters are chosen based on grid hyperparameters. Based on these considerations we extracted this table. For each cell, we tuned hyperparameters and extracted accuracy. The best accuracy is 72.25% by using 1000 words and also using 3000 words and uni-gram with TF-IDF. All details about better accuracies (top ones) like the value of parameters, confusion matrix and classification report are attached in appendix 1.

Table 1: Accuracies for random forest algorithm based on TF and TF-IDF and different n-grams

N-Gram	TF-IDF				TF			
	1000	2000	3000	5000(All)	1000	2000	3000	5000(All)
Uni-gram	**0.7225**	0.71	**0.7225**	0.705	0.7025	**0.72**	0.7075	0.71
Bi-gram	0.635	0.6175	0.64	0.6275	0.6425	0.6375	0.645	0.6425
Tri-gram	0.5675	0.56	0.56	0.5575	0.57	0.5575	0.5575	0.5625
Four-gram	0.5375	0.5375	0.5375	0.5225	0.5375	0.5375	0.5375	0.5375

For random forest, TF-IDF acts slightly better than TF but it is not very significant but as a fact, uni-gram acts much much better than other grams. Another interesting thing is that by decreasing the number of words in documents (the number of attributes) we can get better accuracy.

4.1.2 KNN

For this algorithm, the parameters that are considered important to tuning are the number of neighbours (n_neighbors), Power parameter for the Minkowski metric (p) and Leaf size (leaf_size). The results are in the below table. Again parameters are tuned based on each cell in the table. The best accuracy is 69% by using 1000 words and uni-gram and TF-IDF.

Table 2: Accuracies for KNN algorithm based on TF and TF-IDF and different n-grams

N-Gram	TF-IDF				TF			
	1000	2000	3000	5000(All)	1000	2000	3000	5000(All)
Uni-gram	**0.69**	0.687	0.685	0.675	0.625	0.6675	0.632	0.6275
Bi-gram	0.572	0.585	0.51	0.527	0.5825	0.575	0.555	0.55
Tri-gram	0.535	0.532	0.547	0.537	0.53	0.542	0.545	0.54
Four-gram	0.505	0.512	0.512	0.51	0.505	0.512	0.507	0.512

All details about better accuracies like the value of parameters, confusion matrix and classification report are attached in appendix 2.

Based on these results it seems that uni-gram with 1000 keywords by using TF-IDF for KNN and uni-gram with 3000 keywords by TF-IDF for random forest perform better. These two results experimented again with different preprocessing methods. The below table represents the result.1 stand for all preprocessing techniques are used (stemming, removing numbers, removing stop words,..), 2 stands for without stemming, 3 stands for without removing numbers, 4 stands for without removing stop words, 5 stands for without preprocessing (exact data used), 6 stand for without removing punctuation and 7 stands for using Lemmatization instead of stemming.

Table 3: Different accuracies based on different preprocessing methods.

	1	2	3	4	5	6	7
KNN-1000-TF_IDF	0.69	0.685	0.685	0.667	0.65	0.692	0.68
RF-3000-TF_IDF	0.7225	0.7125	0.7125	0.7025	0.68	0.7175	0.71

According to the above table, all preprocessing methods that have been used are effective in a way that if we do not use any preprocessing method the accuracy drops down around 4%.

Although, all of our experiments in order to compare algorithms' performance are done based on the 20-80 rule. Below table is just an experiment to see the impact of different train and test sizes on random forest and KNN.

Table 4: Different accuracies based on different test and train data set sizes

	20% test-80% train	10% test-90% train	30% test-70% train	40% test-60% train
KNN-1000-TF_IDF	0.69	0.64	0.71	0.688
RF-3000-TF_IDF	0.7225	0.755	0.715	0.7

Accordingly, for the random forest, the best accuracy is 75.5% and KNN is 0.71%

4.2 SVM and Naïve Bayes

By extracting features, we tried to train a classifier to predict the category of a news. In this section, naïve Bayes classifier is used which provides a nice model for this task but among several variants of this classifier in scikit-learn, the one most suitable for text classification is the multinomial Naïve Bayes.

We achieved 70.25% accuracy. The other method of training which we used was a linear support vector machine (SVM), which is widely regarded as one of the best text classification algorithms although it's also a bit slower than naïve Bayes. The accuracy of this classifier is 71.25% which is higher than Naive Bayes method.

Table 5: SVM prediction accuracy (in %) for different features size (second row) and n-gram size

N-Gram	TF-IDF				TF			
	1000	2000	3000	5000(All)	1000	2000	3000	5000(All)
Uni-gram	0.6875	**0.7125**	**0.7125**	**0.7125**	0.6875	0.70	0.70	0.70
Bi-gram	0.6075	0.6075	0.6075	0.6075	0.6025	0.6025	0.6025	0.6025
Tri-gram	0.495	0.495	0.495	0.495	0.495	0.495	0.495	0.495
Four-gram	0.475	0.475	0.475	0.475	0.475	0.475	0.475	0.475

Table 6: Naïve Bayes prediction accuracy (in %) for different features size (second row) and n-gram size.

N-Gram	TF-IDF				TF			
	1000	2000	3000	5000(All)	1000	2000	3000	5000(All)
Uni-gram	0.6775	0.6875	0.6875	0.6875	0.6975	**0.7025**	**0.7025**	**0.7025**
Bi-gram	0.6275	0.6275	0.6275	0.6275	0.62	0.62	0.62	0.62
Tri-gram	0.4975	0.4975	0.4975	0.4975	0.495	0.495	0.495	0.495
Four-gram	0.475	0.475	0.475	0.475	0.475	0.475	0.475	0.475

4.3 Neural Networks

In this section, we experiment with 5 Neural Network architectures for classification for the news articles as real or fake. For training the networks, we use a training (80%), cross-validation (20%) and a test set (20%). We also implement learning rate decay to optimize the model.

4.3.1 Deep Neural Networks using Dropout Regularization

Table 7: Accuracies for Deep Neural Network using Dropout based on TF and TF-IDF and different n-grams

N-Gram	TF-IDF				TF			
	1000	2000	3000	5000(All)	1000	2000	3000	5000(All)
Uni-gram	0.695	0.675	0.695	0.7175	0.6875	0.7175	**0.7225**	0.7
Bi-gram	0.575	0.585	0.58	0.6075	0.5725	0.61	0.57	0.5975
Tri-gram	0.5225	0.5225	0.505	0.5275	0.52	0.535	0.515	0.5325
Four-gram	0.5025	0.5075	0.5075	0.5125	0.5025	0.5075	0.5075	0.5125

We observe that utilizing Dropout regularization provides the best accuracies, by preventing the data from overfitting the training set. This technique would be particularly useful if the training set is small or if the neural network has a large number of learnable parameters. The best accuracy achieved by this network is 72.25% using a uni-gram representation of the input.

The best configuration is achieved by using a fully connected network with 4 hidden layers with 512 layers per hidden node and with a dropout of 0.5.

4.3.2 Deep Neural Networks using BatchNorm without Droupout Regularization

Table 8: Accuracies for Deep Neural Network using BatchNorm without Dropout based on TF and TF-IDF and different n-grams

N-Gram	TF-IDF				TF			
	1000	2000	3000	5000(All)	1000	2000	3000	5000(All)
Uni-gram	0.69	**0.7075**	0.6925	0.6875	0.6825	0.6825	**0.7075**	0.6825
Bi-gram	0.5825	0.535	0.605	0.615	0.595	0.5525	0.545	0.6425
Tri-gram	0.5225	0.5125	0.5175	0.5225	0.505	0.5125	0.5125	0.525
Four-gram	0.5075	0.5025	0.5025	0.5025	0.5025	0.505	0.5075	0.51

As compared to using dropout regularization, the network without dropout tends to overfit the training data and hence result in a lower accuracy. The best accuracy achieved by this network is 70.75% using a uni-gram representation of the input. This method provides similar accuracies for both TF and TF-IDF representations.

The best configuration is achieved by using a fully connected network with 4 hidden layers with 512 layers per hidden node and with a dropout of 0.5

Table 9: Accuracies for CNN, LSTM, GRU, Recurrent-CNN using word embeddings

Network	Accuracies
CNN	0.6625
LSTM	0.6925
GRU	0.675
Bi-LSTM	0.6975
RCNN	0.7025

4.3.3 Convolutional Neural Network (CNN) (using Word Embeddings)

In this section, we implement a Convolutional Neural Network (CNN) for text classification. The use of CNN helps reduce the amount of computation required when compared to a traditional deep feed forward network. Here we observe that for the current classification task, the accuracies provided by CNN are poor as compared to other deep learning techniques.

4.3.4 LSTM, GRU, Bi-LSTM (using Word Embeddings)

In this section, we utilize sequential networks for text classification. We implement the following three networks: Long Short Term Memory (LSTM), Gated Recurrent Unit (GRU) ans Bi-directional LSTM network.

We observe that recurrent networks together with word embeddings are better able to classify text as compared to other machine learning techniques. On further investigation we find that Recurrent Networks are able to identify temporal patterns in the text such as the sentence structure. This also leads us to the finding that the news articles classified as real have a significantly different temporal patterns as compared to the texts classified as fake. The best accuracy among the three is provided by a Bi-directional LSTM (Bi-LSTM) with an accuracy of 69.75%.

4.3.5 Recurrent-Convolotional Network (RCNN) (using Word Embeddings)

In this section, we implement a implement a recurrent network followed by a convolutional network. We observe that using a combination of recurrent and convolutional networks provides the best accuracy of 70.25% among all the sequential network classifiers.

4.4 Decision Trees, XGBoost, AdaBoost

4.4.1 Decision Trees

Following the same logic, we have applied different experiments using Decision Tree as a classifier over tuned data and found that the most accurate result for this specific classifier is if we use all the samples and uni gram as TfidfVectorizer range only. Moreover, we are getting these accuracies after applying hyperparameter technique to get the best parameters. As mentioned below, there is not that much difference between the accuracy of 1000 samples and TF-ID and the accuracy using All samples and TF as a vectorizer (uni-gram).

Table 10: Accuracies for Decision Trees based on TF and TF-IDF and different n-grams

N-Gram	TF-IDF				TF			
	1000	2000	3000	5000(All)	1000	2000	3000	5000(All)
Uni-gram	0.675	0.652	0.645	0.652	**0.68**	0.675	0.667	0.682
Bi-gram	0.565	0.565	0.56	0.562	0.56	0.562	0.56	0.565
Tri-gram	0.495	0.495	0.495	0.495	0.495	0.495	0.495	0.495
Four-gram	0.475	0.472	0.475	0.472	0.472	0.47	0.475	0.475

4.4.2 XGBoost

We have also used XGBoost as a classifier and experimented with our data as it is an implementation of gradient boosted decision trees designed for speed and performance. Keeping the splitting ratio same and feature extraction technique the same, we have applied different experiments and found that TF extracted train and test data is providing more accurate results for uni-gram as range. As mentioned above, we have achieved these accuracies by applying best parameters using hyperparameter technique.

Table 11: Accuracies for XGBoost based on TF and TF-IDF and different n-grams

N-Gram	TF-IDF				TF			
	1000	2000	3000	5000(All)	1000	2000	3000	5000(All)
Uni-gram	0.70	0.695	0.695	0.695	0.705	**0.707**	**0.707**	**0.707**
Bi-gram	0.567	0.567	0.567	0.567	0.562	0.562	0.562	0.562
Tri-gram	0.48	0.48	0.48	0.48	0.48	0.48	0.48	0.48
Four-gram	0.465	0.465	0.465	0.465	0.465	0.465	0.465	0.465

4.4.3 AdaBoost

As using AdaBoost now as a classifier, TF Vectrozer extracted train and test data outrages other experiment results in terms of accuracy using uni-gram and using 1000. As mentioned above, we have achieved these accuracies by applying best parameters using hyperparameter technique.

Table 12: Accuracies for AdaBoost based on TF and TF-IDF and different n-grams

N-Gram	TF-IDF				TF			
	1000	2000	3000	5000(All)	1000	2000	3000	5000(All)
Uni-gram	0.6975	0.6925	0.6925	0.6925	**0.7023**	0.69	0.69	0.69
Bi-gram	0.57	0.57	0.57	0.57	0.565	0.565	0.565	0.565
Tri-gram	0.495	0.495	0.495	0.495	0.495	0.495	0.495	0.495
Four-gram	0.472	0.475	0.47	0.472	0.475	0.472	0.475	0.477

5 Conclusion

We address the problem of fake news detection that is still persistent and widespread having serious real world consequences. We begin with exploration of related work done in the area of opinion mining and fake news detection, which details some of the successful approaches applied in this field. This study provides a broad background about the various techniques used to identify biases in in news articles.

We also introduce a new dataset, obtained from politifact.com, containing public statements made by various leaders that have been labeled into 6 categories depending upon the veracity of their claims. The real world statements regarding a broad range domains, would help further research in the field of fake news detection, opinion mining, topic modelling, rumor detection and other NLP research. We also develop a Robotic Process Automation script for automating the process of retrieving new

datasets by scraping the Politifact website. This script with further modification could be applied to other fact checking news sources.

We then analyze this dataset using a variety of machine learning techniques, to identify the technique best applicable to this domain. We also experiment with various text representations. We observe that using unigram Term Frequency (TF) with a Deep Neural Network Architecture with dropout regularization and using Term Frequency-Inverse Document Frequency (TF-IDF) with Random Forests provides the best classification. We observed TF-IDF to be a powerful tool for finding important terms given a set of documents. The important terms as indicated by their frequencies were highly indicative of the news articles being classified as real or fake. Also utilizing word embeddings together with sequential networks also provided accurate classification. These results indicate oa difference in the temporal structuring of statements in real and fake statements. Upon manual analysis of the results by the group members, we found this difference in sentence structuring to be highly indicative.

While training the algorithms using multi-class classification, we found the observed accuracies to be very low with most of the data mining algorithms, indicative of a very high degree of correlation between various categories. To simplify and therefore to increase our chances of identifying fake news articles, we decided to convert the problem to a binary classification problem. The results thus obtained 72.25% accuracy in classification of real or fake statements. In this paper we use a broad range of standalone and ensemble data mining techniques that outperform other papers operating on similar datasets in the same domain. These results show a promise of application of machine learning techniques in classifying political statements.

6 Task Breakdown

Table 13: Task breakdown

Name	Tasks Assigned
Roshan	Data Scrapping Deep Neural network CNN, LSTM, GRU, Bi-LSTM, Recurrent-CNN Report: Proposed Approach and Models, Conclusion, Compilation
Parisa	Random Forest KNN Presentation
Donya	SVM Naive Bayes Report: Related work and background information
Abubakar, Zaid	Preprocessing Feature extraction Decision Trees XGBoost AdaBoost Report: Abstract, Introduction

References

[1] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609–616. Cambridge, MA: MIT Press.

[1] Gottfried, J., Shearer, E.: News use across social media platforms. Pew Res. Cent. 26 (2016)

[2] Gottfried, J., et al.: The 2016 presidential campaign–a news event that's hard to miss. Pew Res. Cent. 4 (2016)

[3] Rubin, V.L., Chen, Y., Conroy, N.J.: Deception detection for news: three types of fakes. In: Proceedings of the 78th ASIS&T Annual Meeting: Information Science with Impact: Research in and for the Community (ASIST 2015). Article 83, p. 4, American Society for Information Science, Silver Springs (2015)

[4] Horne, B.D., Adali, S.: This just in: fake news packs a lot in title, uses simpler, repetitive content in text body, more similar to satire than real news. In: the 2nd International Workshop on News and Public Opinion at ICWSM (2017)

[5] Burfoot C, Baldwin T. Automatic satire detection: are you having a laugh? Paper presented at: Proceedings of the ACL-IJCNLP 2009 Conference Short Papers, August 04-04, 2009; Suntec, Singapore.

[6] Wang, W.Y.: Liar, Liar Pants on fire: a new Benchmark dataset for fake news detection. arXiv preprint (2017). arXiv:1705.00648

[7] H. Ahmed, I. Traore, and S. Saad, "Detection of Online Fake News Using N-Gram Analysis and Machine Learning Techniques," in Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments, ser. Lecture Notes in Computer Science. Springer, Cham, Oct. 2017, pp. 127–138.

[8] Rubin., Victoria, L., et al.: Fake news or truth? Using satirical cues to detect potentially misleading news. In: Proceedings of NAACL-HLT (2016)

[9] H. Ahmed, I. Traore, S. Saad, "Detecting opinion spams and fake news using text classification," Secur Priv [Internet]. 2017;(November):e9. Available from: http://doi.wiley.com/10.1002/spy2.9

[10] Granik M., Mesyura V. Fake news detection using naive Bayes classifier / 2017 IEEE First Ukraine Conference on Electrical and Computer Engineering (UKRCON), Kiev, 2017, pp.900–903. 2.

[11] Metz C. The bittersweet sweepstakes to build an AI that destroys fake news www.wired.com/2016/12/bittersweet-sweepstakes-build-ai-destroys-fake-news/

[12] Rubin, V., Conroy, N. J., & Chen, Y. (2015, January). Research Gate. Retrieved April 11, 2017, from https://www.researchgate.net/publication/270571080_Towards_News_Verification_Deception_Detection_Methods_for_News_Dis course doi:10.13140/2.1.4822.8166

Appendix 1

```
Random forest
Adjusted ranges:
   rf = RandomForestClassifier(random_state=42)
    n_estimators = [int(x) for x in np.linspace(start=200, stop=2000, num=10)]
    max_features = ['auto', 'sqrt']
    max_depth = [int(x) for x in np.linspace(10, 110, num=11)]
    max_depth.append(None)
    min_samples_split = [2, 5, 10]
    min_samples_leaf = [1, 2, 4]
-----------------------------------
TFIDF
n_gram is: 1
N_words :3000
best params are: {'n_estimators': 600, 'min_samples_split': 10, 'min_samples_leaf': 1, 'max_features': 'aut
confusion matrix is:
[[158  43]
 [ 68 131]]
classification_report is:
              precision    recall  f1-score   support

       False       0.70      0.79      0.74       201
        True       0.75      0.66      0.70       199

    accuracy                           0.72       400
   macro avg       0.73      0.72      0.72       400
weighted avg       0.73      0.72      0.72       400

accuracy_score is:
0.7225
---------------------------
TF
n_gram is: 1
```

```
N_words :2000
best params tf are: {'n_estimators': 200, 'min_samples_split': 10, 'min_samples_leaf': 4, 'max_features': '
confusion matrix is:
[[152  49]
 [ 63 136]]
classification_report is:
              precision    recall  f1-score   support

       False       0.71      0.76      0.73       201
        True       0.74      0.68      0.71       199

    accuracy                           0.72       400
   macro avg       0.72      0.72      0.72       400
weighted avg       0.72      0.72      0.72       400

accuracy_score is:
0.72
```

Appendix 2

```
KNN
Adjusted ranges:
leaf_size = list(range(1, 500))
n_neighbors = list(range(1, 200))
p = [1, 2]
----------------------
TFIDF
gram is 1
w is 1000
best params tfidf are: {'p': 2, 'n_neighbors': 100, 'leaf_size': 102}
confusion matrix is:
[[133  67]
 [ 57 143]]
classification_report is:
              precision    recall  f1-score   support

       False       0.70      0.67      0.68       200
        True       0.68      0.71      0.70       200

    accuracy                           0.69       400
   macro avg       0.69      0.69      0.69       400
weighted avg       0.69      0.69      0.69       400
accuracy_score is:
0.69
```