Uncovering Fake News with Machine Learning

A Comparison of Algorithms and Feature Extraction Techniques

Tom Martensen
DV2542 Machine Learning
Blekinge Institute of Technology
Karlskrona, Sweden

***Abstract*—The report presents a survey of different feature extraction techniques, configurations and machine learning algorithms to predict whether a news article is fake or genuine. The problem is approached solely from a natural language processing perspective and only based on the content of the articles. In an implementation, the algorithms and techniques are implemented and compared based on their classification accuracy. Through the explorative design, the project forms a foundation for further studies in fake news detection.**

***Keywords*—fake news; text classification; feature extraction; machine learning; comparison**

I. Introduction

On October 30, 1938 Orson Welles aired his play "The War of the Worlds" about an alien invasion on the US east coast. As most of the audience joined later into the program, the reports about a mass panic caused fear. The reports—when explained later—turned out to be fake. Today, they are a very popular and early example of "fake news".

Fake news are news articles that are "intentionally and verifiably false and could mislead readers" [1]. In the last years, especially in the 2016 US presidential election, coverage on fake news has alerted major media outlets and politicians. It is believed that fake news spread by social bots and populistic political websites and have influenced the election of Donald Trump as US president.

While in earlier times misinformation in media was mostly accidental or for entertainment purposes, people with malicious agendas have discovered online journalism as an easy and cheap way to spread their agendas. A challenging task for political leaders, journalists and social networks is to detect fake news and to stop their spreading.

Regarded from a machine learning perspective, fake news detection is a binary text classification problem. The distinction can be on basis of metadata (e.g. source of the article) or the text itself. In this report, the author compares different algorithms for feature extracting from text and machine learning algorithms. The expected result is a list of trained models that classify news articles based solely on the text into genuine or fake news, ranked on their performance.

II. Related Work

With an increasing number of articles considered as fake news, researchers and the public likewise are searching for reliable ways to debunk hoaxes.

It should be noted in the first place that fake news itself is a broad term and requires a specific definition that is accepted by the research community. This requires particular attention due to the political sensitivity of the topic. In [1] and [2], the authors distinguish fake from genuine news and show sources and audiences of fake news. These articles act as a catalyst to understand fake news and their definitions are a basis for this report.

A number of reports present tools and methods to detect fake news with network analyses, thus considering only metadata for the classification task e.g. networks, conflicting viewpoints or relationships of "likes" and "comments" [3–5].

As mentioned before, this report focuses on detecting fake news only based the content text of the news itself. Text classification is in general a well-researched topic. [4, 6] for example have both researched how to detect fake news with natural language processing approaches for text classification. However, their results show room for improvement. For a general introduction to text classification and natural language processing, compare [7].

Like [6], this report will use the fake news dataset curated in [8] and a sample of genuine articles from [9]. Other sources of fake news articles have also been considered like [10]—but ultimately discarded as they are impure with satiric articles. The implementations are based on the *sklearn* library for *Python* [11] that also provides some guidance for text classification. Directives for experiment setup and interpretation stem also from [12, 13].

III. Method and Design

In this section, the author explains the sequence of actions that lead from data fetching over feature generation to the conduction of the experiment. All necessary steps are laid out here with their requirements, results and justifications to support the external and internal validity of the project. The experiment procedure shall be reproducible from the description and the attached source code.

A. *Goal and Challenge*

The purpose of the experiment shall be to compare different machine learning algorithms—preferably from different categories—and feature extraction methods for the task of supervised binary text classification. The result of the experiment shall be a ranking of the best combinations. Particularly challenging is in this context the preparation and feature extraction from text.

B. *Preprocessing*

As mentioned before, two datasets from two different sources were used in the project. Both are publicly available. To reduce the risk of a biased fake news dataset, the articles were obtained from the open data platform Kaggle [8]. The 13,000 news pieces covering different topics include text and metadata from over 240 websites and were reviewed by the Kaggle community. The Signal Media news dataset [9] consists of one million news articles and blog entries that were scraped in September 2015. Of these entries, 52,000 articles were selected in a uniform sample. These articles were considered genuine and not fake or biased in any way.

Before assigning binary labels to all articles (1: fake, 0: genuine), all texts were lowercased and stripped from non-word characters, leaving space separated words in each entry. The proportion of fake news in the experiment dataset is similar to the empirical evidence given in [1] in an attempt to reflect a real-world scenario. The selection was shuffled randomly and split into three datasets for the experiment:

1. 13,000 entries/20%: Validation set for development and hyper-parameter optimization
2. 39,000 entries/60%: Training set for final performance training
3. 13,000 entries/20%: Test set for final performance evaluation

After this preprocessing step, the entries in the dataset consist of the text of the article and the binary label for fake or genuine, respectively.

C. *Feature Generation*

For the process of feature generation, multiple algorithms were considered. These included for example *word2vec* ([14]), *glove* ([15]) or other models. However, these were discarded because of the extra effort that would have been necessary to use these opposing to their original intent—which is in word classification—in sentence or even document classification instead. Since only the text of the articles is available for the classification, algorithms that are based on the metadata were rejected as well. The particular challenging task in text analysis is that the text as a sequence of words or symbols cannot be used as input in the machine learning algorithms directly. Because text documents vary in length and—at least with no other preparation—consist of characters, a way to represent documents as vectors of fixed length and numerical values has to be found.

After the prior elimination steps and the new criteria, the bag-of-words model (also known as vector space model or bag of n-grams) seemed most promising.

In this model, the sentences or texts are split on whitespaces or punctuation as separators and represented as a multiset of their words. This removes some information about the structure of the text, the grammar and word order, but keeps the multiplicity of each word in the text. The idea is to use the number of occurrences as a feature in training the classifier. The algorithm shall be explained in a little example to familiarize with the conceptuality.

Considering two documents: *“Adam enjoys to go out. Eve enjoys to go out too.”*, *“Adam also enjoys to go swimming”* and splitting after each whitespace and punctuation, leads to the following multiset of words: *[“Adam”, “enjoys”, “to”, “go”, “out”, “Eve”, “too”, “also”, swimming”]*. This is named *“vocabulary"*. These are all words that are known to the algorithm. While this bag of words offers multiple ways to characterize the text, the most common and useful in this setting is to calculate the number of times one of the terms from the vocabulary appears in the input text. Doing this will result in an *n* long vector for each input text, where *n* is the length of the vocabulary and each entry is the number of times the term in position *i* of the vocabulary appeared in the text. In the example, the two sentences would result in these two vectors:

(1) [1, 2, 2, 2, 2, 1, 1, 0, 0]

(2) [1, 1, 1, 1, 0, 0, 0, 1, 1]

Note that as mentioned before this representation removes the information of sentence structure. A problem with this representation however, is that it overvalues very common words in languages like “to”, “a” or “the” in the English language. These so called stop-words will (in longer texts than in the example) most often have the highest term frequency and thus be noise in the input for the machine learning algorithms, as they do not provide additional information. To remove this noise, a list of these stop words can be used. These words will then be skipped, when the vocabulary is generated.

The bag of words model offers some more variation that could yield better results. This introduces for example the idea of *n-gram* models. These can be used to store some information about the context of word occurrences in text. The bag of words or uni-gram is a special case of the n-gram model where *n=1*. Consequently, a bi-gram (n=2) model vocabulary splits the text into units of two words, e.g. “John likes”, “likes to” … in the vocabulary and calculates the term frequency as mentioned above. Here as well, lists of stop words can be used to reduce noise. In the project, n-gram models were used where n was between 1 and 3. Additionally, a number of minimum occurrences was set for each model. If a word or combination of words occurred less times in the whole corpus than this threshold, it was not considered as a feature in the vector. Again, this may result in loss of information. However, if the combination did not occur enough times, it would have small influence on the training anyway and not provide additional information. Furthermore, smaller vectors obviously improve training and test speed.

Another approach to emphasize terms that may be rare but interesting and that are shadowed by very frequent terms which have not been identified as stop words, is to use the inverse document-frequency.

The tf-idf term weighting re-weights the count features into floating point values. Term frequency is the number of times a term occurred in a text compared to the maximum value of this figure (tf). The inverse document frequency (idf) depends on the whole document corpus. The values are calculated as follows:

(1) tf-idf(t,d) = tf(t,d) · idf(t), where

(2) $\mathrm{idf}(t) = \log \frac{1+n_d}{1+df(d,t)} + 1$

In these equations, n_d stands for the total number of documents and *df(d, t)* is the number of documents that contain the term. The resulting vectors are then normalized so all values are between *0* and *1*. As this transformation is an extra step after vectorizing, it was conducted in the project for the described n-gram vectors.

A more practical problem when vectorizing a large corpus with bag of words is that the mapping from the terms to integer features happens in-memory. For larger datasets, the vocabulary is obviously large too and thus, the memory requirements may exceed the machine limits. Furthermore, when fitting, even more intermediate data structures have to be used. As a global vocabulary must be maintained in vectorizing, parallelization requires so much overhead that it is not implemented in the standard library, because it would make the algorithm in effect slower.

The biggest consummator of memory is undoubtedly the dictionary that holds the mapping of vocabulary to vector indices. To phase out this problem, [16] proposed to use hash functions to map terms directly to indices. Thus, no memory is required at all to store the dictionary. By providing a sufficiently large amount of hash buckets or increasing their number in the progress, hash collisions can be avoided. While hashing enables parallelization, it does not have a bad effect on model performance, as shown in [17].

In the project, feature hashers were used for n-grams with *n* between 1 and 3 to compare the performance with the ordinary and transformed generated features. In total, nine different representations of the validation set and training set were generated. The test set was vectorized using the same vectorizers from the training set with their respective dictionary or hash function.

D. *Choice of Algorithms*

To cover and evaluate a broad field of algorithms, the available algorithms were categorized into four categories:

1. probability based classifiers,
2. functional classifiers,
3. tree based classifiers and
4. neural network classifiers.

One algorithm in each category was researched further, implemented and used in the project. In the following, the four algorithms shall be explained shortly with their general idea.

For the probability based classifiers, a *Gaussian Naïve Bayes* classifier was used. This method is very popular in text categorization and is based on Bayes' theorem of independence assumptions between the features, in this case the term frequencies. This means that the algorithms does not regard possible correlations between the features. For further information, compare [13, pp. 262-282]. The Gaussian Naïve Bayes algorithm works on continuous data, as represented in the input vectors, and assumes that the values in each class are distributed according to a Gaussian distribution. This approach was used in the experiment as a baseline to compare with more advanced models.

As an example of a functional classifier, a *Logistic Regression* model was used in the experiment. Similarly to the linear regression, this model tries to optimize a function to fit best with the available data and uses the maximum likelihood estimation (MLE)—different from the method of least square in linear regression. An example for a logistic function can be seen in figure 1. The method's codomain are values between 0 and 1 and follows this general scheme:

(1) $P(y = 1) = \frac{1}{1+e^{-z}}$, where *z* is a linear function:

(2) $z = \beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2 + \ldots + \varepsilon$,

where $x_k$are the independent variables, β_k are regression coefficients and ε is an error bias. The regression coefficients are estimated through MLE as mentioned before. It aims to provide a clear division between the probabilities of y's values.

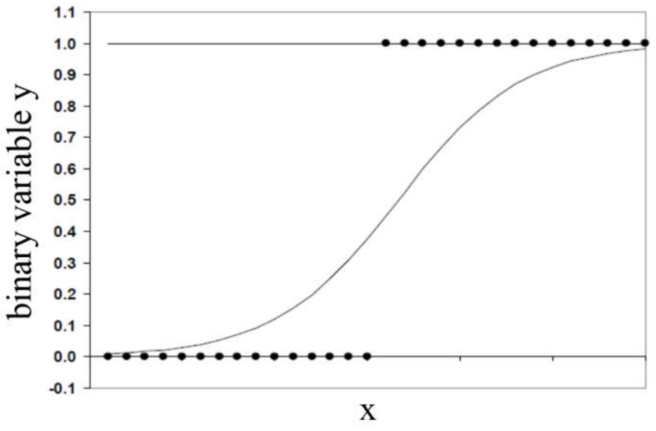

Fig. 1. Logistic function

Random Forest classifiers use tree based classifiers as a base for their algorithm. These decision trees are not correlated and grow randomly during learning. For every classification, the class that most trees assign to the input, decides the final classification. In comparison to other classification algorithms, Random Forest classifiers are considerably fast in training and the evaluation can be parallelized. Thus, it is very efficient for large datasets. Random Forest classifiers were developed to overcome the shortcomings of decision trees and are part of ensemble algorithms in Machine Learning [18].

Artificial Neural Networks (ANN) try to mimic the brain and are based on units (*neurons*) and connections between them (*synapse).* Each connection transmits a signal from one unit to another. The signal is a real number. Each unit has inputs and output that is calculated by a non-linear function using all input values. Similar to the brain, connections and units have a weight that adjusts during the learning process to stress some inputs or calculations in favor of others. Using backpropagation the functions and weights are optimized. Neural networks can have multiple "hidden" layers and are commonly used in big data settings in all domains today.

E. Hyper-Parameter Optimization

After the algorithms were selected and the features were generated according to the descriptions above, the implementation of the algorithms was needed. Fortunately, all algorithms are already implemented in the *sklearn* library and proven in practice by a community and researchers. Thus, the experiment was based on this framework.

A hindrance for beginners in the machine learning field and a challenging task in general is to find the optimal parameters for the algorithms. For different sets of data, different parameters are optimal and may influence the performance of the models.

There are multiple ways to tune these so called "hyper-parameters". To stay in the scope of the project, only the two methods implemented in *sklearn* were considered. Both approaches take a range of values for each of the algorithm parameters and search this value space for the optimum. Therefore, a performance measure has to be selected.

For this, the F-score was chosen. It is widely used in natural language processing research, can be used to assess binary classifiers and—through precision and recall—offers additional insights into the performance.

The available hyper-parameter optimization algorithms are the grid search and the randomized search. Where the grid search searches all combinations of possible values, the randomized search takes random parameter values from the available range for each iteration. The number of iterations can be specified.

Because the randomized search is faster and yields equally good results (compare [19]), it was finally used in the project. The performance of each parameter set is evaluated using a five-fold cross validation (compare [13, pp. 348-250]) and conducted for three of the algorithms with all nine data sets generated from the validation set as described in section C. As the Gaussian Naïve Bayes classifier does not have a relevant, configurable parameter, no randomized search was done for it. The results of this step can be seen in the appendix. Only the top ranked model's parameters were considered further.

F. Experiment

In the final evaluation step, the test set was—as aforementioned—vectorized using the same vectorizers that were used for the respective training set vectorization regarding the parameters. This is a necessity to have the same input vector length for the training and test set in the final experiment.

Additionally, the output from the hyper-parameter optimization must be considered now. Therefore, the parameters of the model that achieved the highest rank (best F-score) in that step are put into a configuration directly in the code of the experiment. This manual step was added to ensure the validity of the parameters, as during the hyper-parameter optimization it has happened that no optimal parameters were found in the given number of iterations. To cope with this deficiency, the hyper-parameter optimization was rerun for these configurations.

Once this was done, the evaluation experiment was run for each of the 36 combinations of the four machine learning algorithms and nine feature generation configurations. In each combination, training and test data were loaded, the model was fitted with the training data and evaluated with the test data.

As in the hyper-parametrization, the measure of model performance is the F-score. Additionally, the results of *recall* and *precision* are calculated to gain more insights. All results were fetched and are presented in table 1.

	Logistic Regression	Random Forest	Neural Network	Gaussian Naïve Bayes
n-gram 1	F: 0.85 P: 0.81 R: 0.89	F: 0.67 P: 0.68 R: 0.65	F: 0.88 P: 0.88 R: 0.87	F: 0.66 P: 0.55 R: 0.83
n-gram 1 (tf-idf)	F: 0.85 P: 0.85 R: 0.86	F: 0.70 P: 0.83 R: 0.60	F: 0.85 P: 0.88 R: 0.82	F: 0.78 P: 0.82 R: 0.74
n-gram 1 (hashed)	F: 0.84 P: 0.80 R: 0.88	F: 0.65 P: 0.66 R: 0.63	F: 0.83 P: 0.82 R: 0.84	F: 0.63 P: 0.53 R: 0.77
n-gram 2	F: 0.80 P: 0.87 R: 0.74	F: 0.60 P: 0.88 R: 0.45	F: 0.81 P: 0.89 R: 0.74	F: 0.49 P: 0.33 R: 0.97
n-gram 2 (tf-idf)	F: 0.78 P: 0.74 R: 0.83	F: 0.54 P: 0.88 R: 0.39	F: 0.80 P: 0.80 R: 0.80	F: 0.69 P: 0.56 R: 0.89
n-gram 2 (hashed)	F: 0.66 P: 0.64 R: 0.70	F: 0.53 P: 0.71 R: 0.42	F: 0.59 P: 0.51 R: 0.71	F: 0.58 P: 0.54 R: 0.62
n-gram 3	F: 0.64 P: 0.79 R: 0.54	F: 0.31 P: 0.84 R: 0.19	F: 0.62 P: 0.91 R: 0.47	F: 0.42 P: 0.26 R: 0.998
n-gram 3 (tf-idf)	F: 0.63 P: 0.75 R: 0.55	F: 0.31 P: 0.85 R: 0.19	F: 0.64 P: 0.87 R: 0.50	F: 0.42 P: 0.27 R: 0.99
n-gram 3 (hashed)	F: 0.43 P: 0.35 R: 0.55	F: 0.0 P: -- R: 0.0	F: 0.45 P: 0.40 R: 0.50	F: 0.36 P: 0.34 R: 0.38

Tab. 1. Experiment results per method and algorithm. (F: F-score, P: precision, R: recall)

IV. Results and Analysis

The results from the experiment show a broad variety in performance between the algorithms and the configurations. In the following section, some results that stand out shall be highlighted and compared to other results in the same category. The effects of high precision and recall shall also be explained in the context of the project.

As described in the section on Data Preprocessing, genuine, non-fake articles outnumber fake news articles, which are the positive samples in this setup, by factor 4. To be precise, of the 13,000 articles in the test set, 19.6% are fake (2,555 of 13,000).

The precision—the ratio of true positives of all predicted positives—of all models is therefore expected to be high. A model that guesses every test sample as negative would acquire a high accuracy. The recall—the ratio of the correctly classified positive samples—is harder to optimize. Together with the F-score that combines recall and precision, the recall shall be the benchmark for the models in this analysis. All F-score, precision and recall values from the experiment are also displayed in the diagrams in figure 2 to 4.

Analyzing the results from the Logistic Regression models, it appears that the performance was best with the different configurations of uni-grams, n-grams where n=1. However, there are no significant differences between the three configurations for this type of model. Recall reached values of more than 0.85, which means that of 100 fake news articles, more than 85 are detected. As aforementioned expected, the precision is high for these models as well. For bi- and tri-grams however, the performance decreased significantly and recall dropped to values of just over 0.50. A reason for this performance loss could be that the growth in available information produced noise in the logistic function that resulted in function parameters that did not converge well.

Similar patterns can be observed with the Neural Network. While the values for F-score, precision and recall are in similar ranges for uni-grams—with little higher precision values—bi- and trigrams seem harder to solve even for the Neural Network. The algorithm can still produce considerably high precision numbers for the normal and inversely transformed bi- and trigrams, but for the hashed n-grams the precision drops even more. This could mean that the Neural Networks have not been able to be trained properly by these data sets.

When comparing the results for Random Forests to the aforementioned methods, the performance clearly disappoints. While it can maintain relatively high precision scores, the recall values already in the uni-grams, but more obviously in bi- and tri-grams are not competitive. Uni-grams can still detect more than half of the fake news as fake, but tri-grams are only able to detect one out of five fake news articles. Worse still, in the tri-gram scenario with a hash function, the random forest classifier could not detect a single fake news articles. This shows how complex the task is and that tree-based models may reach their limits for these probability based classification tasks when the number of features is this high.

In the first intention, the Gaussian Naïve Bayes algorithm was used in the project as a benchmark for the other algorithms. During the experiment, it became clear that this algorithm can outperform the others in special settings when it comes to recall. As one can see from table 1, the trained model detects nearly all of the fake news articles as fakes for bi-grams, tri-grams and the inversely transformed tri-grams. This comes however at a cost: The false positive rate, which is the ratio between false positives and actual negatives, and the precisions are both among the worst values detected in the experiment. This means that too many "false alarms" were triggered by the model. It seems that the model overestimates the probability of a sample being positive, or—as spoken in terms of the example—categorizes too many genuine articles as fakes.

In summary, logistic regression and neural networks were able to bring the best results in terms of F-score and recall with uni-grams and outperformed the other two approaches. Adding more information about context with bi- or tri-grams does not necessarily help the performance. Inverse document frequency and using hashing functions for the n-grams did only improve the computing time and had no significant influence on the performance of the models.

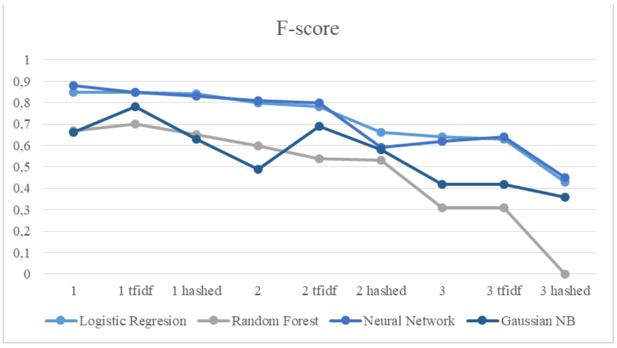

Fig 2: F-score for all algorithms and configurations

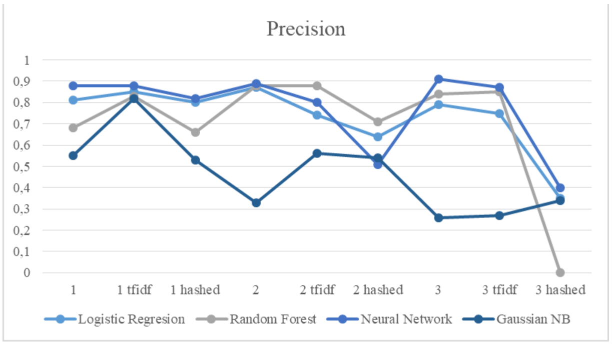

Fig 3. Precision for all algorithms and configurations

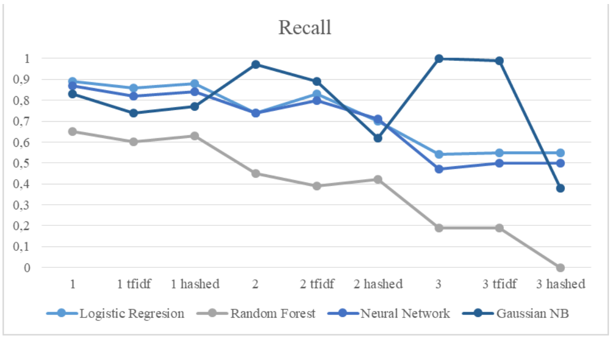

Fig 4. Recall for all algorithms and configurations

V. Conclusions

Fake news detection based only on the content of the articles has been proven as an example of binary text classification. In the project implementation and the accompanying experiment, it was shown that the combination of uni-grams and logistic regression or neural networks model show the best performance and can detect over 8 out of 10 fake news articles correctly, thus being well suited for the text classification task. It is obvious that these algorithms were better able to cope with the large number of features that they were presented with, because they can weigh some features more than others.

Another takeaway is that one should not only try to achieve a high recall, as this may come with a large number of "false alarms" as it has happened in the experiment with the Gaussian Naïve Bayes algorithms.

The study itself was presented transparent and comprehensible. Neither the fake news articles nor the genuine articles are biased in any deliberate way. All intermediate steps have been laid out, reasoned and explained. As with the implementation, the experiment can be repeated, although minor variations in test results should be expected.

VI. Future Work

The author believes that the results from this project form a good basis for future improvements. It has been shown that the analysis of fake news articles based only on the text and not the metadata is able to bring quite good results. Especially, Logistic Regression and Neural Networks seem to work fine with uni-grams. An eye should also be kept on the simple implementation of Naïve Bayes algorithms that outperformed the other algorithms for some configurations in terms of recall.

In the future, other configurations, such as *GloVe* could be added to the tests. Furthermore, it would be interesting to see how an evaluation of sentence structure, grammar or typos as features could influence the performance of the algorithms. Lastly, one can also consider to include some metadata about the articles, like the website on which it was published, as features to improve the performance of the algorithms. For an application of a fake news detection system, human participation is also an important factor that should be studied by social scientists.

VII. References

[1] H. Allcott and M. Gentzkow, "Social Media and Fake News in the 2016 Election," *Journal of Economic Perspectives*, vol. 31, no. 2, pp. 211–236, 2017.

[2] V. L. Rubin, Y. Chen, and N. J. Conroy, "Deception detection for news: Three types of fakes," *Proc. Assoc. Info. Sci. Tech.*, vol. 52, no. 1, pp. 1–4, 2015.

[3] Zhiwei Jin, Juan Cao, Yongdong Zhang, and Jiebo Luo, "News Verification by Exploiting Conflicting Social Viewpoints in Microblogs," *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 2972–2978, 2016.

[4] N. J. Conroy, V. L. Rubin, and Y. Chen, "Automatic deception detection: Methods for finding fake news," *Proc. Assoc. Info. Sci. Tech.*, vol. 52, no. 1, pp. 1–4, 2015.

[5] E. Tacchini, G. Ballarin, M. L. Della Vedova, S. Moret, and L. de Alfaro, *Some Like it Hoax: Automated Fake News Detection in Social Networks.* Available: http://arxiv.org/pdf/1704.07506.

[6] Samir Bajaj, ""The Pope Has a New Baby!": Fake News Detection Using Deep Learning," Report, Stanford University, Stanford, 2017.

[7] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval,* 1st ed. Cambridge u.a.: Cambridge Univ. Press, 2009.

[8] *Getting Real about Fake News.* [Online] Available: https://www.kaggle.com/mrisdal/fake-news. Accessed on: Feb. 13 2018.

[9] *NewsIR'16 - Signal Media News Dataset.* [Online] Available: http://research.signalmedia.co/newsir16/signal-dataset.html. Accessed on: Feb. 13 2018.

[10] W. Y. Wang, *"Liar, Liar Pants on Fire": A New Benchmark Dataset for Fake News Detection.* Available: http://arxiv.org/pdf/1705.00648.

[11] *scikit-learn: machine learning in Python — scikit-learn 0.19.1 documentation.* [Online] Available: http://scikit-learn.org/stable/. Accessed on: Feb. 13 2018.

[12] *Machine Learning course.* [Online] Available: https://www.bth.se/eng/courses/T0001661/20202/. Accessed on: Feb. 13 2018.

[13] P. Flach, *Machine learning: The art and science of algorithms that make sense of data*. Cambridge: Cambridge Univ. Press, 2012.

[14] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," [Online] Available: http://arxiv.org/pdf/1301.3781v3.

[15] Jeffrey Pennington, Richard Socher, and Christopher D. Manning, "GloVe: Global Vectors for Word Representation," Computer Science Department, Stanford, 2014.

[16] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proceedings of the 26th Annual International Conference on Machine Learning,* Montreal, Quebec, Canada, 2009, pp. 1–8.

[17] K. Ganchev and M. Dredze, "Small Statistical Models by Random Feature Mixing," in *Proceedings of the ACL-08: HLT Workshop on Mobile Language Processing,* Association for Computational Linguistics, Ed., Columbus, Ohio, 2008, pp. 19–20.

[18] T. K. Ho, "Random decision forests," in *Proceedings of the third International Conference on Document Analysis and Recognition: August 14-16, 1995, Montréal, Canada*, Montreal, Que., Canada, 1995, pp. 278–282.

[19] "Introduction," in *Introduction to Polymer Rheology*, M. T. Shaw, Ed., Hoboken: John Wiley & Sons, 2012, pp. 1–14.