Detecting Fake News with Supervised Learning

Benjamin R. Perucco

December 30, 2020

1 Definition

1.1 Project Overview

Our world is highly interconnected and it is of paramount importance that citizens are informed objectively and truthfully about issues that influence and shape our world (like issues on geopolitics or climate change). The internet lead to a rise of news media (for example social media, news portals, etc.) to report on these stories. The vast amount of (online) news articles available yield a new phenomenon called fake news. Fake news is false or misleading information presented as news and can reduce the impact of real news [1].

1.2 Problem Statement

This work is about answering the question whether machine learning (ML) can be applied to classify news articles as truthful or fake. There are several papers available where Ahmed et al. classified news articles [2] or hotel guest reviews [3] to be truthful or fake. Ahmed et al. used natural language processing (NLP) models to transform text into a structured, mathematical representation and achieved accuracies up to 92% [2].

Based on a set of features $\{f_1, f_2, f_3, ...\}$ extracted from news articles, a machine learning algorithm needs to be trained to find the relationship $\hat{\theta}$ between the a priori known classification of the news articles y ($= 0$ for truthful and $= 1$ for fake) and the set of extracted features $\{f_1, f_2, f_3, ...\}$. The relation can be written as

$$\hat{y} = \hat{\theta}(X) \tag{1}$$

where $\hat{y}$ is the predicted class label by the trained function $\hat{\theta}$ applied to a feature matrix X that holds the set of features $\{f_1, f_2, f_3, ...\}$. $\hat{\theta}$ must be trained on available data, such that

$$\sum_{i=1}^{n} (y - \hat{y})^2 \tag{2}$$

is minimal. Here it is assumed that we have n news articles where it is already known whether they are truthful or not.

A first step to be worked on is to extract a feature matrix X from a set of news articles (or also called documents). How reliable is the prediction of class labels on unseen news articles once the function $\hat{\theta}$ has been found?

1.3 Metrics

Equation 2 is rather used for regression problems where the predicted output is a continuous variable. For classification problems like news article classification into a truthful (= 0) or a fake class (= 1), the accuracy is a more suitable metric. This can be nicely demonstrated by the confusion matrix

$$\begin{bmatrix} & 1 & 0 \\ 1 & \text{TP} & \text{FP} \\ 0 & \text{FN} & \text{TN} \end{bmatrix}. \tag{3}$$

The entries in the row indicate predicted class labels and the entries in the columns represent the real class labels. TP stands for true positives, TN for true negatives, FP for false positives and FN for false negatives. To asses the performance of $\hat{\theta}$, the accuracy a is considered, which can be calculated as

$$a = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FN} + \text{FP}}. \tag{4}$$

2 Analysis

2.1 Algorithms and Techniques

News articles must be converted into a structured, mathematical representation in order to be used by machine learning techniques. The following chapters discuss how we can convert text into numerical features.

2.1.1 n-gram Model

The n-gram is usually defined as a contiguous sequence of words with length n. For example, if $n = 1$, we speak of a unigram that contains only single words. Or if $n = 2$, we denote this as a bigram which is built on two adjacent words.

Consider the following text: “Sometimes we eat green apples, and sometimes, the apples we eat are red.” Based on a unigram (1-gram), we obtain a set of tokens: {’sometimes’, ‘we’, ‘eat’, ‘apples’, ‘green’, ‘and’, ‘the’, ‘are’, ‘red’}. We can derive a frequency array of tokens in the text: [2, 2, 2, 2, 1, 1, 1, 1, 1]. For the bigram (2-gram), another set of tokens is obtained: {’sometimes we’, ‘we eat’, ‘eat green’, ‘green apples’, ‘apples and’, ‘and sometimes’, ‘sometimes the’, ‘the apples’, ‘apples we’, ‘eat are’, ‘are red’}. The corresponding frequency array of tokens in the text is: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]. Please note that punctuation is not considered when converting text into tokens.

In order to a build frequency arrays for a set of texts (or documents), a common vocabulary needs to be built of which the n-gram model is underlying principle.

2.1.2 Vocabulary

Consider a corpus D which contains a set of documents $\{d_1, d_2, d_3, ..., d_n\}$. Then a vocabulary F is a set of tokens $\{f_1, f_2, f_3, ..., f_m\}$ extracted from the corpus D. Mind that a token is created based on the n-gram model. Usually for a set of tokens, only the m mostly occurring tokens in a corpus D are considered. In the following, the tokens are denoted as features as these construct the features (or independent variables) of a machine learning model.

2.1.3 Definitions

Let $\sigma(d_i, f_j)$ denote the number of occurrences of feature f_j in document d_i. Then a feature matrix S can be built, where

$$S = \begin{bmatrix} \sigma(d_1, f_1) & \sigma(d_1, f_2) & \sigma(d_1, f_3) & ... & \sigma(d_1, f_m) \\ \sigma(d_2, f_1) & \sigma(d_2, f_2) & \sigma(d_2, f_3) & ... & \sigma(d_2, f_m) \\ \sigma(d_3, f_1) & \sigma(d_3, f_2) & \sigma(d_3, f_3) & ... & \sigma(d_3, f_m) \\ ... & ... & ... & ... & ... \\ \sigma(d_n, f_1) & \sigma(d_n, f_2) & \sigma(d_n, f_3) & ... & \sigma(d_n, f_m) \end{bmatrix}. \tag{5}$$

An element $\sigma(d_i, f_j)$ in matrix S (representing the document d_i and feature f_j) is abbreviated using the notation σ_{ij} for simplicity.

2.1.4 Term Frequency Model

The matrix S could be already used for machine learning. Features usually need to be normalized in machine learning to increase performance. Therefore, the term frequency (TF) model normalizes matrix S leading to matrix $\hat{S}$. An element $\hat{\sigma}_{ij}$ of matrix $\hat{S}$ is written as

$$\hat{\sigma}_{ij} = \frac{\sigma_{ij}}{\sum_{j=1}^{m} \sigma_{ij}}. \tag{6}$$

Or spoken in plain language: the number of occurrences of a feature f_j in a document d_i is divided by the total number of occurrences of all features $\{f_1, f_2, f_3, ..., f_m\}$ in the same document d_i. So we end up with a representation where the importance of each feature can be compared to other features in the same document (relative importance).

2.1.5 Inverse Document Frequency Model

The inverse document frequency (IDF) model is used to define the feature importance not just in a document d_i but also compare its importance to the whole corpus D. A matrix E is introduced, where an element ϵ_{ij} of matrix E is 1 if $\hat{\sigma}_{ij} > 0$ (or $\sigma_{ij} > 0$). An inverse normalization is applied to the matrix E resulting in a vector $\hat{e}$. An element $\hat{\epsilon}_j$ of vector $\hat{e}$ is calculated as

$$\hat{\epsilon}_j = 1 + \log\left[\frac{|D|}{\sum_{i=1}^{n} \epsilon_{ij}}\right]. \tag{7}$$

Or spoken in plain language: in a corpus D which comprises of a set of documents $\{d_1, d_2, d_3, ..., d_n\}$, it is counted in how many documents the feature f_j appears. This number is used to divide the number of documents $|D|$ in a corpus D. Consider two examples: if a feature f_j occurs in each document $\{d_1, d_2, d_3, ..., d_n\}$, we divide the number of documents $|D|$ in a corpus D by the same number. So equation 7 results in 1, weighting feature f_j as 1. On the other hand, if a feature f_j occurs only in one document, equation 7 results in a much larger number, thus increasing the weight of feature f_j in the corpus D.

Finally, a matrix $\hat{P}$ is obtained as the element-wise product of the term frequency matrix $\hat{S}$ and the inverse document frequency vector $\hat{e}$, written as $\hat{P} = \hat{S} \odot \hat{e}$. For an element $\hat{\pi}_{ij}$ of matrix $\hat{P}$, this is written as

$$\hat{\pi}_{ij} = \hat{\sigma}_{ij} \cdot \hat{\epsilon}_j, \quad \text{element-wise for } j = 1, 2, 3, ..., m. \tag{8}$$

This term is denoted as the term frequency inverse document frequency (TF-IDF) model.

2.2 Data Exploration

A set of truthful and fake news articles is available on kaggle.com [4]. The data was collected from real world sources. Truthful articles were obtained from Reuters and fake news articles were gathered from unreliable websites that were flagged by Politifact which is a fact-checking organization. These data sets contain different types of articles on different topics, the majority of articles focus on political and world news topics [3].

There are 44,898 articles in total in the corpus. 21,417 of the articles are classified as truthful and 23,481 are classified as fake, so both class labels are roughly balanced. Approximately 6,169 of the articles are not unique. The duplicates are being removed after text processing discussed in chapter 3.1.

2.2.1 Visual Analysis of Top Features

A first glimpse of the top 20 unigrams per class is given in figure 1. It can be observed from the unigrams that probably the most discussed topic in the set of news articles is the presidential election in the U.S. during the year 2016. Despite some unigrams that are used frequently in both classes of articles (like said, trump, presid, republican, etc.), we also see distinct unigrams only used in one class of articles.

3 Methodology

3.1 Data Preprocessing

Raw text has to be brought into a clean state in order for feature extraction. Several text cleansing steps are performed like removal of introductory statements, dates, hyperlinks, numbers and stop words. Finally, so called stemming is performed to end up with the stem of a word (for example cat should identify such strings as cats, catlike, or catty [5]).

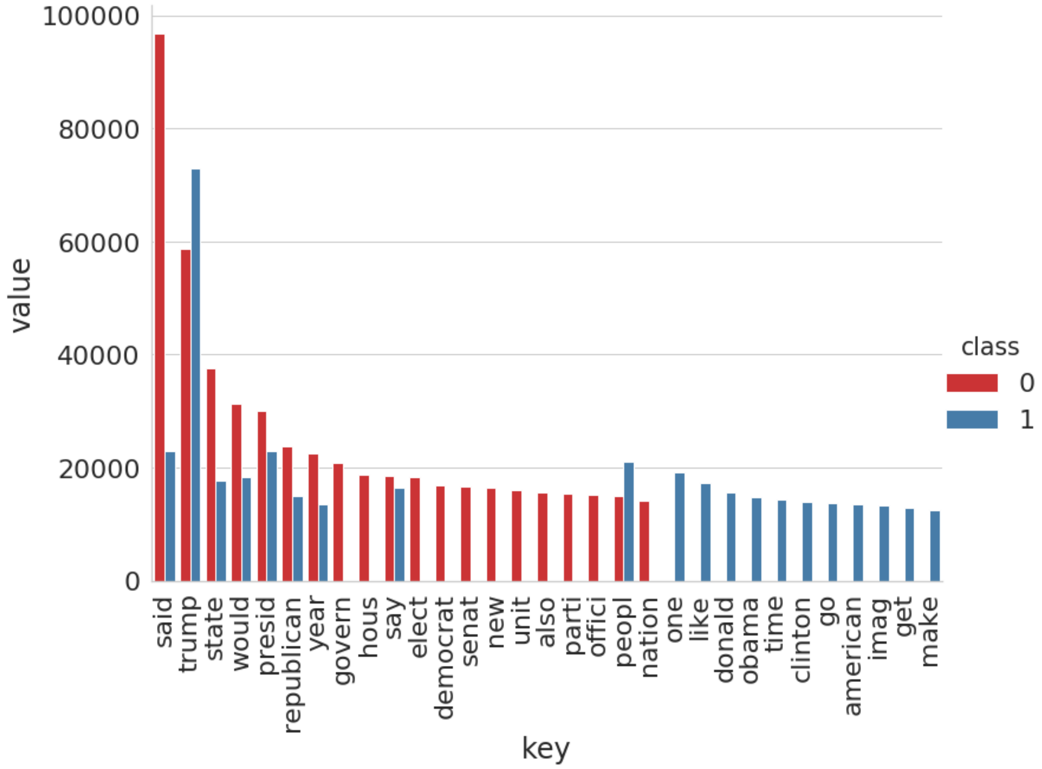

Figure 1: Frequency of the top 20 unigrams per class extracted from the corpus.

3.2 Implementation

3.2.1 Feature Extraction Method

The indices of the corpus are randomized and splitted into three different data sets as displayed in the table 1 below.

Name	Purpose	Size	TF	IDF	TF-IDF
Training	Model training	60%	$\hat{S}_{\text{train}}$	$\hat{E}_{\text{train}}$	$\hat{S}_{\text{train}} \odot \hat{E}_{\text{train}}$
Test	Hyper parameter tuning	20%	$\hat{S}_{\text{test}}$	$\hat{E}_{\text{train}}$	$\hat{S}_{\text{test}} \odot \hat{E}_{\text{train}}$
Validation	Model validation	20%	$\hat{S}_{\text{valid}}$	$\hat{E}_{\text{train}}$	$\hat{S}_{\text{valid}} \odot \hat{E}_{\text{train}}$

Table 1: Data preprocessing and feature extraction methods applied.

It is good practice not to use the same data for model training and validation. For hyper parameter tuning a test data set is used to avoid over fitting.

Term frequency (FT) feature matrices are calculated for each set according to equation 6. In comparison, the inverse document frequency (IDF) feature matrix is only calculated for the training set according to equation 7.

The reason is that the validation and test set should simulate the performance behavior in case of new and unseen data. Therefore, the IDF feature matrix is estimated on a training set and the resulting matrix $\hat{E}_{\text{train}}$ is used for the transformation of all sets into the TF-IDF feature space according to equation 8.

Contrary, the TF feature matrix describes the relative importance of features in a single document. This is independent of other documents in the corpus and therefore, we can derive three matrices $\hat{S}_{\text{train}}$, $\hat{S}_{\text{test}}$ and $\hat{S}_{\text{valid}}$ for the three sets.

3.2.2 Modeling

SageMaker [6] is used as the training and deployment platform for the machine learning models. Five machine learning models are based on the sci-kit learn framework [7] because of its simplicity to try out quickly different models while having a common interface. The following sci-kit learn models k nearest neighbor (knn), support vector machine (svm), logistic regression (log), gradient boosting (gbc) and multi layer perceptron (mlp) are evaluated. In comparison, the built-in XGBoost algorithm (xgb) in SageMaker is used as a performance reference [8]. Furthermore, the TF and TF-IDF feature extraction methods are applied. The feature size is varied between 125 and 250 features. Unigram and bigram models are used for the creation of the common vocuabulary.

3.3 Refinement

Hyper parameter tuning is performed on the training set and cross checked on the test set. The objective target accuracy (see equation 4) has to be maximized. Tuning is performed on a `ml.c5.2xlarge` instance where 8 hyper parameter combinations are evaluted in parallel. This results in 384 model to be trained (6 models × 2 feature extraction methods × 2 different feature sizes × 2 different n-grams × 8 models for hyper parameter tuning).

In the following table 2, the model hyper parameters as well as its varied ranges are shown.

Model	Parameter	Type	Range
knn [9]	`n neighbors`	int	[3, 15]
	`weight`	cat	'uniform', 'distance'
	`p`	int	[1, 8]
svm [9]	`random state`	int	1
	`kernel`	cat	'poly'
	`C`	float	[0.001, 3.0]
	`degree`	int	[2, 3]
log [11]	`max iter`	int	10,000
	`C`	float	[0.001, 3.0]
gbc [12]	`random state`	int	1
	`learning rate`	float	[0.001, 0.5]
	`n estimators`	int	[100, 1,000]
	`max depth`	int	[2, 10]
mlp [13]	`random state`	int	1
	`activation`	cat	'relu'
	`max iter`	int	1,000
	`learning rate`	float	[0.001, 0.1]
	`start size`[1]	int	125
	`end size`[1]	int	2
	`hidden layer size`	int	[1, 5]
xgb [8]	`num round`	int	100
	`rate drop`	float	'0.3
	`tweedie variance power`	float	1.4
	`eta`	float	[0, 1]
	`min child weight`	int	[1, 10]
	`alpha`	int	[0, 2]
	`max depth`	int	[1, 10]

Table 2: [1] Note that start and end size are not available as parameters in the multi layer perceptron model in scikit-learn. Based on the hidden layer size and its start and end size, a linear interpolation is performed for the number of neurons in between in case the hidden layer size is higher than 2.

4 Results

4.1 Model Evaluation and Validation

4.1.1 Influence of Parameter Variation

There are 48 parameter variations studied in total using hyper parameter tuning in SageMaker [6] (6 models × 2 feature extraction methods × 2 different feature sizes × 2 different n-grams). The influence of these different parameters is demonstrated below in figure 2.

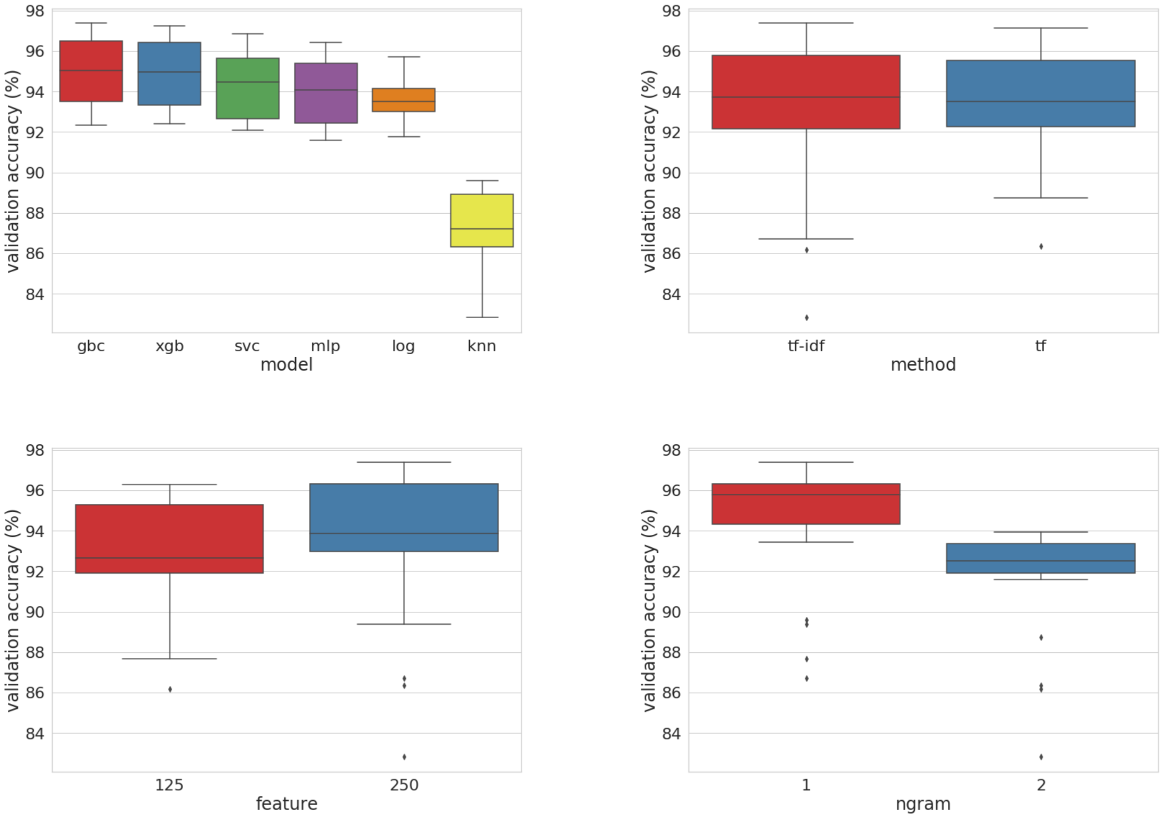

Figure 2: The influence on the accuracy of varied models, feature extraction methods, feature sizes and n-grams is demonstrated.

4.1.2 Accuracy Performances

The parameter combinations leading in the best five accuracy performances are shown in the table 3 below.

Model	Method	Features	n-gram	Accuracy on test set (%)	Accuracy on validation set (%)
gbc	TF-IDF	250	1	96.96	97.37
xgb	TF-IDF	250	1	96.80	97.24
gbc	TF	250	1	96.78	97.15

Table 3: Parameter combinations leading in the best three accuracies.

4.1.3 Detailed Results

Detailed results on achieved accuracies, model hyper parameters and other varied parameters can be found on the Github repository.

4.2 Justification

Except for the knn model, we see very good performances leading to accuracies well above 90%. This is in line with the finding of Ahmed et al. [2]. Even tough a much smaller feature size is used in this study, accuracies are similar or even better (250 vs. $\geq$ 1,000). The tree based models (gradient boosting [12] and XGBoost [8]) work best and lead to the best three accuracies (see table 3). Ahmed et al. [2] found that a LSVM method is optimal.

It can be observed that both feature extraction methods TF and TF-IDF lead to almost equivalent results with a slight advantage for the TF-IDF method. Also, there seems to be almost a tie between the tested feature sizes (125 vs. 250). To save computational ressources, one should choose a feature size as low as practical. Furthermore, a unigram model leads to better results than a bigram model. The difference is quite significant as seen in figure 2.

5 Conclusion

5.1 Reflection

Machine learning models are able to classify articles into two class labels of truthful and fake articles really well. It can be shown that an accuracy of 97% can be achieved. But one has to ask the question what the methodology has been to decide which article is fake and which article is truthful. For example all truthful articles are from Reuters and fake articles were classified by Politifact. Even though this methodology is consistent for the set of articles analyzed in this work, it does not necessarily mean that new and unseen articles from other sources (others than Reuters for example) can be classified equally well by a machine learning model. This is one of the fundamental remaining question to be answered in a further study.

5.2 Improvement

Further investigations can be done in order to achieve an even better classification and increase training and prediction computation speed:

- With such a large corpus of 44,898 documents, it can be difficult to achieve a text processing that is clean from text not related to the article (for example hyperlinks, references to twitter accounts, etc.). A second thorough investigation must dive deeper into the news articles to remove any unwanted stuff that remained from the present text processing implemented.
- The methodology of article classification has to be reviewed and one has to clarify how well the classification performs on completely different sources of news articles (as mentioned in chapter 5.1).
- Calculating the correlation between features and focusing on features with low correlation could help to reduce the number of features and thus save compuation cost. The same could be achieved by applying principal component analysis (PCA).

References

[1] Fake news, December 30, 2020, wikipedia.org,
https://en.wikipedia.org/wiki/Fake_news

[2] Ahmed H., Traore I., Saad S. (2017) Detection of Online Fake News Using N-Gram Analysis and Machine Learning Techniques. In: Traore I., Woungang I., Awad A. (eds) Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments. ISDDC 2017. Lecture Notes in Computer Science, vol 10618. Springer, Cham.
https://doi.org/10.1007/978-3-319-69155-8_9

[3] Ahmed, H, Traore, I, Saad, S. Detecting opinion spams and fake news using text classification, Security and Privacy, 2018,
https://doi.org/10.1001/spy2.9

[4] Fake and real news dataset, December 30, 2020, kaggle.com,
https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset

[5] Stemming, December 30, 2020, wikipedia.org,
https://en.wikipedia.org/wiki/Stemming

[6] Amazon SageMaker, December 30, 2020, amazon.com,
https://aws.amazon.com/sagemaker

[7] scikit-learn - machine learning in Python, December 30, 2020, scikit-learn.org,
https://scikit-learn.org/stable

[8] Amazon SageMaker - XGBoost Algorithm, December 30, 2020, amazon.com,
https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost.html

[9] scikit-learn - k-nearest neighbors vote classifier, December 30, 2020, scikit-learn.org,
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[10] scikit-learn - c-support vector classification, December 30, 2020, scikit-learn.org,
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[11] scikit-learn - logistic regression classifier, December 30, 2020, scikit-learn.org,
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[12] scikit-learn - gradient boosting for classification, December 30, 2020, scikit-learn.org,
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

[13] scikit-learn - multilayer perceptron classifier, December 30, 2020, scikit-learn.org,
https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html