Fake news detection for Romanian

1 Project Description

1.1 Introduction

This article documents the process of using several classifiers in order to identify fake news from Romanian news web sites. The algorithms are tested on different vectorization procedures, lengths of tokenized vocabularies and test data sizes. The sought after information pertains to the accuracy and time of each classification run.

1.1.1 State of the art

There are many implementations and approaches to Fake News detection, but none could be found for Romanian at the time of this project's implementations. The article that inspired everything was **Fake news detection in social media(Kelly Stahl)**[1].

1.1.2 Motivation

Fake news as a term is defined as the deliberate spread of misinformation via traditional news media or via social media. Its purpose relates to manipulating the masses for political, social, economical or cultural purposes, creating panic or "trolling" as a way of satisfying one's ego or personal view of fighting against a particular moral concept present in current societal norms. It can start with satirical or sarcastic intentions, but snowball and spread at an out of control rate, slowly changing and adapting each time it is shared. Regardless of origin and intent, fake news can rapidly cause damage on a high scale and lasting repercussions. This project aims to take the concept of fake news detection, which is widely known and researched, and apply it to the Romanian language, which is a far less explored territory.

1.2 Method

The process makes use of Romanian fake and true news articles scraped from real and satire news sites. The text is preprocessed through tokenization and stemming, vectorized and passed through several classifiers in order to find the one that yields the best accuracy and/or time. The detection method was researched through multiple articles and the chosen programming language is Python. As soon as these details were set, many articles were read, a lot of research was done, many tasks were created and a lot of steps were collected into a diagram.

All the algorithms from the initial architecture were used, as well as some extra ones and combinations, which exceeds the original expectations.

1.2.1 Gathering Data

Finding sites that could be crawled and scraped was quite a daunting task. Real news are easy to find, there are countless organizations that have been existing for decades, in television, radio or newspaper formats. Fake news was, on the other hand, tricky. Nobody will write a bunch of false articles, admit to them being fake, then collect them together in a nice place to be found. True malicious or ill-intended posts were impossible to gather, so the next best thing was, logically, satire. Sarcastic news sites are a fraction of the news site pool, especially in Romania. This narrowed down the amount of data that could be collected, as the true corpus size had to be the same as the fake one.

There was also the question of how this restriction would apply into the real world. We used sarcastic articles and we tested them on more sarcastic articles. This can be very helpful against the rumor snowball effect that can lead to the creation of fake news, as well as the tone recognition problem faced by many people. Carefully crafted texts meant to destroy nations, create panic, mimic official and formal articles or exude authenticity would perform worse under this particular data set, but not go completely undetected. The topics chosen by the satire authors and their writings are more times than not meant to poke fun at pieces that circulate as fake news. They have their fair share of "true fake news" characteristics.

Web Sites

The final corpus has 200 000 words, used across 1200 articles, equally spread between the two categories. There were 8 sites crawled and scraped, half for true news, half for satire.

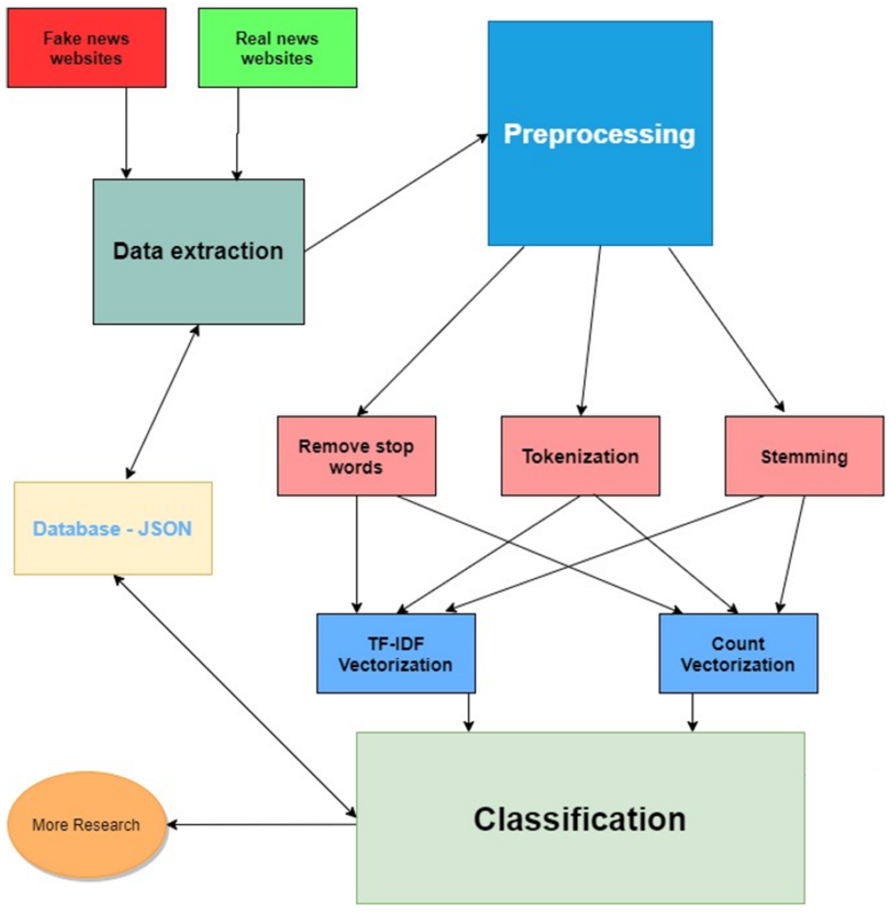

Figure 1: Initial Project Architecture

Category		True	News			Fake	News	
Web Sites	**ProTV**	**Digi24**	**Libertatea**	**Realitatea**	**TNR**	**Cațavencii**	**7lucruri**	**TimpuriGrele**
Articles	124	89	91	108	42	156	502	96
Words	41,796	69,800	41,981	46,508	7,917	98,544	055,899	37,690

Figure 2: **Web Site Corpus Distribution**

Crawling

The article link gathering was done through a process called ”crawling”, which means recursively finding and saving all the links found on a certain web page. Those saved links are used to continue the search, until no more are found or until a set limit. Some websites were tough to crawl, as they only advertised recent articles. This lead to a disproportionate distribution of total words obtained across all websites, but he main goal of finding equal amounts of true and fake news was met. The implementation was relatively easy, done using Python libraries.

Scraping

The actual downloading of the article text is called scraping. It involves using the HTML tags of the web sites in order to locate the content, which was quick as all the news sources fit into a pattern and only required one function with different inputs to scrape.

1.2.2 Preprocessing

The articles had to be preprocessed before being passed through the classifiers. The first step was combining the title and content of each piece into one string. The second step was removing the stop words, then turning the string into individual words. The last process chosen in this particular implementation was stemming, which means reducing words to a short root that represents the differently-spelled versions of a certain notion.

Stop Words

This is the beginning of preparing the extracted texts for classifiers to properly label the data further down the line. Firstly, an extensive stop words list is needed, that we can identify in our texts and have them removed. After this, the actual process is a relatively easy task, as the app only needs to compare the raw data with this list of

unnecessary words, and have them removed entirely.
Multiple sources for choosing these stop words have been used, and they can be found in the references section of this document.

Tokenization

Tokenization is the method of breaking down a large amount of text into smaller pieces called **tokens**. These tokens are extremely useful for pattern recognition and are used as a starting point for stemming and lemmatization. Tokenization here is also be used to replace sensitive data elements with non-sensitive ones. We use the method **word_tokenize()** to split a sentence into words.
For better text comprehension in machine learning applications, the performance of the word tokenizer in NLTK can be translated to a Data Frame.

Stemming

Stemming is the process of eliminating a word's suffix and reducing it to its **root word**. The main goal is to reduce each word's inflectional forms to a single base word, root word, or stem word. Inflection is the process of changing a word to convey different grammatical categories including tense, case, voice, aspect, person, number, gender, mood, animacy, and definiteness. For this project, we have employed the **Natural Language Toolkit** (NLTK) library and its **SnowballStemmer()** method, language parameter set on 'romanian' and stopwords ignore enabled, as the stopwords used were from a different library and not the NLTK native one.

All of the data is stored in **.json** files after checking for satisfactory results with the stemming, tokenization and stop words removal processes. Further down the line, the classifiers we have implemented will have these files as input for their algorithms.

1.2.3 Data Models

The models used were Bag of Words and TF-IDF. They were implemented from scratch, as a big portion of the preprocessing was as well, which wouldn't have worked with the Scikit-learn algorithms.

1.2.4 Input Formats

The current state of the project does not have a finished interface or a way to input a piece of text in order to receive a classification label, but running the code allows the user to choose different algorithms to test the corpus on.

1.2.5 Classifiers

Naive Bayes

"The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.".(Scikit-learn)

Naive Bayes is a simple statistical classificator based on the Bayes Theorem. It follows a few short steps: it determines the prior likelihood for the given labels, calculates the probability of each attribute for each label and then allocates the label with the highest score derived from the theorem.

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Figure 3: Bayes Theorem

Naive Bayes is fast and accurate, performs well with discrete variables, it is efficient on large datasets, has a low computation cost and works well on text problems.

Words		**200**				**1000**				**19518**		
Model	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**
Test	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**
Result	82.5%	82.9%	82.6%	82.9%	90.3%	89.7%	87.4%	86.6%	93.3%	93.3%	73.0%	71.8%
Time	0.01s	0.01s	0.01s	0.01s	0.09s	0.08s	0.05s	0.06s	1.76s	1.67s	0.98s	1.08s

Figure 4: **Naive Bayes**, Accuracy Table, **Average** out of **100** runs

Naive Bayes had the best time-accuracy score in the whole experiment. There are classifiers than performed better with less words, but overall it was consistently superior.

Passive Aggressive Classifier

"The passive-aggressive algorithms are a family of algorithms for large-scale learning. They are similar to the Perceptron in that they do not require a learning rate. However, contrary to the Perceptron, they include a regularization parameter C."(Scikit-learn)

Words		**200**				**1000**				**19518**		
Model	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**
Test	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**
Result	83.0%	83.4%	84.3%	84.5%	93.1%	93.3%	93.5%	92.6%	94.3%	93.8%	94.2%	94.0%
Time	0.03s	0.03s	0.03s	0.03s	0.13s	0.13s	0.17s	0.15s	2.50s	2.3s	2.79s	2.63s

Figure 5: **Passive Aggressive Classifier**, Accuracy Table, **Average** out of **100** runs

This was the best performing algorithm by far. It passed 94% accuracy using the entire vocabulary with both Bag of Words and TF-IDF in less than 3 seconds.

Logistic Regression

"Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function."(Scikit-learn)

Words		**200**				**1000**				**19518**		
Model	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**
Test	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**
Result	86.1%	85.9 %	85.5%	84.8%	93.8%	93.2%	84.2%	82.9%	93.7%	93.2 %	72.1 %	71.4%
Time	0.10s	0.09s	0.04s	0.04s	0.25s	0.22s	0.13s	0.11s	3.98s	3.84s	1.86s	2.14s

Figure 6: **Logistic Regression**, Accuracy Table, **Average** out of **100** runs

Logistic Regression had a mediocre performance, as it reached 93% at its best with the whole vocabulary at about 4 seconds per run. TF-IDF had an interesting peak when the fewest words were used.

Support-vector machines

Support vector machines (SVMs) are a class of supervised learning methods for classification, regression, and identification of outliers.

Basically, SVMs consider all of the data points and generate a line called a "Hyperplane" that separates the two groups: fake and true news. A "decision boundary" is created.

Because of the long training period, SVMs are not appropriate for large datasets, and they also take longer to learn than, let's say, Naive Bayes. They have issues with overlapping classes and are also affected by the kernel type used, which in this case was linear.

Words		**200**				**1000**				**19518**		
Model	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**
Test	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**
Result	86.1%	85.2 %	86.8%	85.9%	92.4%	92.8%	91.7%	90.0%	94.1%	92.6 %	82.3 %	80.6%
Time	0.20s	0.12s	0.06s	0.06s	0.34s	0.32s	0.61s	0.50s	84.59s	79.05s	158.10s	174.49s

Figure 7: **SVM**, Accuracy Table, **Average** out of **10** runs

This is by far the classification that took the longest, even tough it only had 10 runs. It is outperformed by other algorithms in both accuracy and time, so it is not worth it in this situation. An interesting observation is that TF-IDF starts to slowly take longer than BoW with the size increase, which is the opposite of most other algorithms.

Decision Tree Classifier

Decision Trees (DTs) are a supervised learning system for classification and regression that is non-parametric. The aim is to learn basic decision rules from data features to build a model that predicts the value of a target variable. A tree is an approximation to a piecewise constant. **Advantages** of DTs:

- decision trees need less effort for data preparation during pre-processing than other algorithms.

- the model is based on a white box. If a scenario can be observed in a model, boolean logic can easily clarify the situation.
- even if the true model, from which the data were produced, violates some of its assumptions, it still performs well.

Some of the **disadvantages** of using Decision Tree classifier:

- any minor change in the data will result in a significant change in the decision tree's structure, resulting in instability.
- the training time for a decision tree is usually longer
- because of the difficulty and time required, decision tree training is relatively costly

Words		**200**				**1000**				**19518**		
Model	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**
Test	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**
Result	80.6%	80.2%	80.9%	80.2%	88.2%	88.3%	87.1%	86.7%	89.6%	88.4%	87.2%	87.7%
Time	0.05s	0.05s	0.06s	0.05s	0.23s	0.19s	0.26s	0.22s	11.8s	9.5s	9.9s	7.0s

Figure 8: **Decision Tree Classifier**, Accuracy Table, **Average** out of **100** runs

Random Forest Classifier

"A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.".(Scikit-learn)

As the definition above states, this classifier merges different types of decision tree algorithms to form a more powerful prediction model. It is supervised and can be used for both regression and classification tasks. The steps intertwine as follows: a decision tree is built on a random amount of samples picked from the data set, the process gets repeated for a chosen number of times and, at the end, for classification, the label is assigned based on the majority vote of the trees.

Words		**200**				**1000**				**19518**		
Model	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**
Test	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**
Result	87.5%	86.3%	88.8%	88.5%	94.0%	92.5%	92.4%	92.8%	92.0%	90.0%	92.3%	92.0%
Time	0.30s	0.28s	0.35s	0.32s	0.57s	0.54s	0.59s	0.54s	7.44s	6.64s	6.34s	5.49s

Figure 9: **Random Forest Classifier**, Accuracy Table, **Average** out of **10** runs

This algorithm is a readily implemented classification method that will be tackled later down the line in this article, which is prediction model merging. It is somewhat more simple, as it only focuses on decision tree models. It performed much better than the original Decision Tree Classifier, who was stuck in the 80-89% accuracy and it took less time, making it superior on all terms.

Ada Boost Classifier

An AdaBoost classifier is a meta-estimator that starts by fitting a classifier on the original dataset, then fits additional copies of the classifier on the same dataset, but adjusts the weights of incorrectly classified instances such that subsequent classifiers concentrate more on difficult cases.
Advantages of using this classifier:

- AdaBoost is often referred to as the strongest out-of-the-box classifier because it uses decision trees as poor learners.
- it may be less prone to overfitting than other algorithms in certain situations.
- individual learners can be slow, but as long as their output is better than random guessing, the final model will converge to a good learner.

Among the **disadvantages** of this classifier, one that pertains to our project is:

- AdaBoost is vulnerable to outliers and noisy data. When the data isn't easily amenable to a particular separation plane (with acceptable results based on the model objective. For example, weak learners must be better than guessing at random), making it more difficult for the meta-learner to converge to a strong learner without overfitting. As a result, we can conclude that AdaBoost performs better when the dataset is free of outliers.

Words		200				1000				19518		
Model	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**
Test	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**
Result	86.3%	86.8%	86.2%	86.1%	93.5%	93.1%	92.8%	92.4%	93.5%	93.2%	92.9%	92.9%
Time	0.2s	0.2s	0.3s	0.3s	1.3s	1.1s	1.5s	1.3s	31.9s	28.5s	32.4s	29.3s

Figure 10: **Ada Boost Classifier**, Accuracy Table, **Average** out of **100** runs

Voting Classifier

"The idea behind the VotingClassifier is to combine conceptually different machine learning classifiers and use a majority vote or the average predicted probabilities (soft vote) to predict the class labels. Such a classifier can be useful for a set of equally well performing model in order to balance out their individual weaknesses.".(Scikit-learn)

Words		200				1000				19518		
Model	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**
Test	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**
Result	87.3%	86.4%	84.8%	85.6%	93.7%	93.9%	90.4%	89.9%	91.2%	91.1%	85.0%	85.7%
Time	0.42s	0.39s	0.42s	0.38s	0.96s	0.81s	0.94s	0.74s	12.7s	12.2s	10.0s	8.68s

Figure 11: **Random Forest, Logistic Regression, KNN**, Accuracy Table, **Average** out of **10** runs

Words		200				1000				19518		
Model	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**
Test	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**
Result	84.1%	84.2%	85.2%	84.7%	93.8%	93.3%	87.5%	85.8%	93.5%	94.1%	74.4%	73.7%
Time	0.15s	0.13s	0.09s	0.08s	0.41s	0.41s	0.34s	0.32s	7.63s	7.68.s	6.48s	6.44s

Figure 12: **Naive Bayes, Logistic Regression, Passive Aggressive**, Accuracy Table, **Average** out of **10** runs

Words		200				1000				19518		
Model	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**	**BoW**	**BoW**	**T-I**	**T-I**
Test	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**	**10%**	**20%**
Result	86.8%	86.9%	87.0%	85.8%	94.2%	93.3%	93.8%	93.4%	93.2%	94.5%	93.8%	91.9%
Time	0.45s	0.38s	0.42s	0.45s	0.90s	0.81s	0.86s	0.95s	11.3s	10.3s	10.5s	9.36s

Figure 13: **Naive Bayes, RandomForest, Passive Aggressive Classifier**, Accuracy Table, **Average** out of **10** runs

The combinations used took a long time per run and could not outperform the stellar 94% accuracy of the Passive Agressive algorithm.

1.3 Results

The best results using the **entire vocabulary** were given by the **Passive Agressive Classifier(94.3%)**. Using **200 words** only, the **Random Forest Classifier** reached **87.5%**, and the **Voting Classifier** with the **Naive Bayes, Passive Aggresive and RandomForest combination** peaked the **1000 word** tests with **94.2%**.

The fastest algorithm was **Naive Bayes** with the longest time being **1.76 seconds** for the **entire vocabulary** using **Bag of Words**.

1.4 Conclusion

The project could have included more algorithms or their combinations, which would have been the next step in the implementation. Nonetheless, many tests were done and the best accuracy achieved was more than 94%, which is a very good result based on the initial expectations.

References

[1] **Fake news detection in social media - Kelly Stahl**(`https://www.csustan.edu/sites/default/files/groups/University%20Honors%20Program/Journals/02_stahl.pdf`)

[2] **Sci-kit Learn. Classification Algorithm Library**(`https://scikit-learn.org/stable/`)