Grundlæggende Data Science - Fake News Project

Lars V. Thorup
Department of Computer Science
Universitetsparken 1
rlp419@alumni.ku.dk

Victor V. Jørgensen
Department of Computer Science
Universitetsparken 1
kft410@alumni.ku.dk

Daniel Friis-Hasché
Department of Computer Science
Universitetsparken 1
rcb933@alumni.ku.dk

Vitus N. Legarth
Department of Computer Science
Universitetsparken 1
gfn320@alumni.ku.dk

March 28, 2025

The entire project can be found at: GitHub

Contents

1 Part 1: Data Processing

1.1 Task 1 & 2: Cleaning

We start by loading a subset of the FakeNewsCorpus dataset using the libraries `pandas` & `tqdm` for loading and keeping track of how far we have come with loading the set. After this, we use the `cleantext` library to clean the dataset. We do this by applying a lambda function on the content of every article (each row of the loaded dataset) with the cleantext library if the given row's content column consists of a string. If not, we replace it with NaN's which we then drop. By doing this, we only actually drop a single article (when doing it on the 995,000 articles). Next we begin the cleaning process which consists of tokenization, removing stopwords & stemming. We clean the column "content" as it is where the actual article text is stored in the csv file.

After cleaning the entire content, we use `clean_words` to convert all articles into a list of words, where we also choose to convert all non-string elements to NaN's and drop them.

At each step of our cleaning, we compute how many articles we have, so we can keep track of how many are dropped and when they are dropped.

After cleaning the small dataset given, we did the same for the large dataset.
Note: the `data_cleaning.ipynb` file was originally set up to clean for 100,000 rows, so we could use it for the exploration, but we have since changed it to run for 10,000 rows, to accomodate for the limited storage on GitHub.

1.1.1 Reduction rates

After counting the number of unique words before and after removing stopwords and applying stemming, the reduction rate is calculated to 26.84% meaning that the unique word count shrinks by 26.84% after removing stopwords and applying stemming. It is worth noting that more than one quarter of the vocabulary is therefore either stopwords or inflections of the same word.

1.2 Task 3: Data Exploration

1.2.1 Data Representation

From the many different approaches to representing our data, we chose to go with `Pandas` as our representer and data extractor. We chose to represent our data using the dataframe class. We went with this option as it has options for separating our initial load file into chunks and account for low memory machines. Besides easy loading, it would also be easier to keep the dataset in memory instead of exporting and importing new files for each change we make, allowing for an easier load on machines running our program.

In the dataset itself, we discovered that a row had some problematic and missing data, which resulted in the validation column (e.g. "fake", "reliable", "clickbait", etc.) not having the actual type of the article, but instead a time for when it was scraped. The error we encountered was printed out as:

```
Error on row:  ['908192', '', 'Financials   7:50am EST BRIEF-Al Tawfeek Co for Financial
    Leasing Q3 profit...]
```

(Most of the output has been cut off as it would simply be too long)

1.2.2 Word frequency

To explore the data we counted the frequency of the words in different cases. Before cleaning the data, after cleaning the data, only on articles classified as fake and only on articles classified as reliable. After doing data exploration it appears that the 37 most frequent words that appear in content, before removing stopwords and applying stemming are all stopwords. In our code we also made some plots to visualize that the word frequency follows Zipf's law that states when a list of measured values is sorted in decreasing order, the value

of the n-th entry is often approximately inversely proportional to n. This can be seen as the curve of the word frequencies flattens significantly when removing stopwords. Which then also shows that these are the words that appear the most. We sadly do not have room to show these in our report, but they can be seen in our code on GitHub. After removing stopwords we get an idea of which more meaningful words are frequently present in the articles from the fake news corpus. Words like 'Trump', 'president','state', and blockchain are all fairly common as they are all on among the 20 most frequent words. This tells us that Trump, the US, and technology are popular topics in news articles at the moment.
Furthermore, we made a comparison of the most frequent words among the reliable sources and the fake sources, we did this as we expected and hoped to see a notable difference that could tell us which words could indicate fake news. It turned out that each category, fake/reliable, consists of almost the same words in almost the same frequency.
We can therefore reject our hypothesis that there is a notable difference between fake and reliable news, regarding used words.

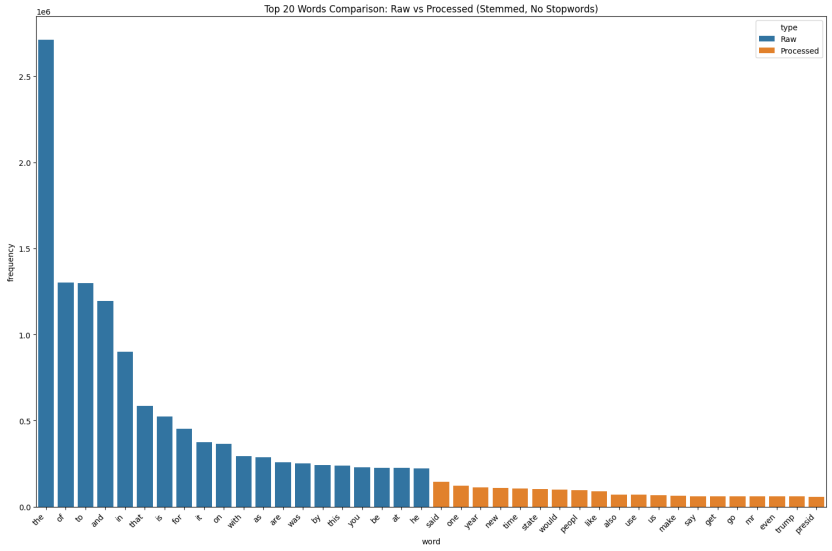


Figure 1: Word frequency as bar chart

1.2.3 Explored domains

To further explore the data we filtered the data for reliable and fake articles and then checked which domains were most present in both categories.
It shows that the domains that have the most fake and reliable articles, respectively, have none of the opposite kind. This information can be used to determine which domains you expect to get fake or real news from.

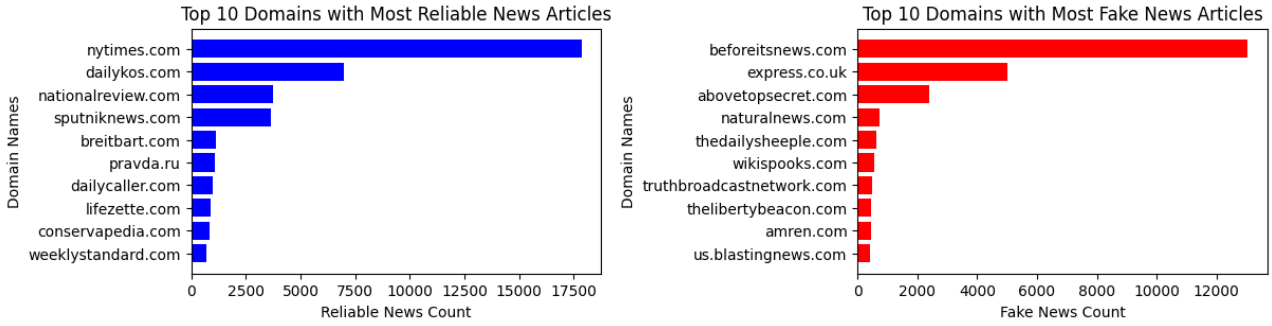


Figure 2: Real/fake domains as bar charts

2 Part 2: Simple Logistic Regression Model

2.1 Grouping of labels

Initially, the dataset contains the following labels: conspiracy, satire, reliable, unreliable, junksci, unknown, political, fake, hate, clickbait, bias and rumor. As a part of this exercise, these will have to be grouped into 'reliable' and 'fake'. For this classification, we have chosen the following conversion matrix:

'**clickbait**': Reliable, '**bias**': Reliable, '**reliable**': Reliable, '**political**': Reliable, '**conspiracy**': Fake, '**satire**': Fake, '**junksci**': Fake, '**fake**': Fake, '**hate**': Fake, '**rumor**': Fake, '**unreliable**': Skip, '**unknown**': Skip

We have chosen the reliable categories as follows: Clickbait, bias, reliable, political. Clickbait is categorized as reliable, since a clickbait article is not inherently fake, their headers are just exaggerated, and we do not use the headers. Biased articles presents reliable information but might just not give the whole truth, but are true nonetheless. Reliable articles are inherently reliable. Political articles are the same case as biased articles.

Unreliable categories are: Conspiracy, satire, junksci, fake, hate, rumor. Conspiracies are not necessarily based in truth and make some leaps in logic, so these are fake. Satire is based in truth but makes jokes that might not be, so these are categorized as fake. Junksci is science but not based on evidence, so this is fake. Fake is obviously categorized as fake. Hate is categorized as fake, since hate articles' primary objective is not to spread truth, but to spread hate, which might or might not be based on facts. Rumors are categorized as fake, since they are based on hearsay, which might be exaggerated or even fake.

Two categories are omitted entirely from the training and test set: Unreliable and unknown. Unreliable might seem like an obvious 'Fake' category, but the description in the corpus is: "Sources that may be reliable but whose contents require further verification." So it means they are incomplete and might be both fake and real. Unknown is also categorized as 'Skip', since the articles are not known to be real or fake.

2.2 Training of the simple model

To train the logistic regression model, the library SKlearn is used. The data is split between a training set and a validation set (85/15). The training set is used to train the model, while the validation set is used to monitor the performance and ensure that the model does not get overfit.

Synthetic Minority Oversampling Technique (SMOTE) is used to rebalance the dataset. The initial dataset after preprocessing has a distribution of 64%/ 36% in favor of reliable news articles, and SMOTE resamples it to 50%/ 50% by oversampling the fake news articles.

CountVectorizer from SKlearn, with the hyperparameter `'max_features'` set to 10.000, is used to vectorize all the words in the articles. This means the logistic regression is trained on the top 10.000 most occurring words in the articles.

The model achieves an F1-score of 84% on the validation set.

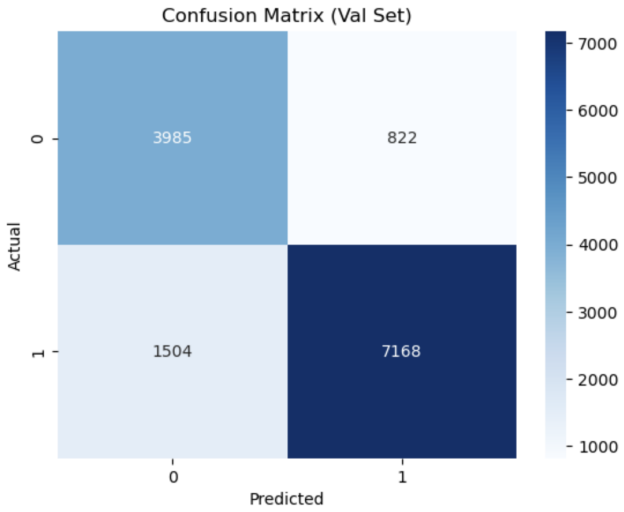


Figure 3: Confusion matrix for the predictions of the test set via the logistic regression model.
1: Reliable, 0: Fake.

2.3 Inclusion of metadata in training

Including metadata from articles can be very helpful in determining the reliability of the article. There is often a correlation between the news anchor and author publishing the article and the content of the article.

In the exploratory data analysis, it was discovered that there is a strong correlation between the domain and the type. This was used to train a variation of the logistic regression. The model variant achieved an impressive F1-score of 1.00, which is much better than the previous model, which achieved an F1-score of 0.84.

This is not very surprising, given that the exploratory data analysis showed that the reliability of an article can be determined almost exclusively by which domain it was published from.

However, if we train exclusively on the domain and omit the text of the article entirely, the F1-score is still 1.00, so it does not train on the text at all. This is problematic, since now the model does not categorize articles as fake or real news but as news anchors.

2.4 BBC News Articles

Including the BBC articles we gathered in exercise 2 did not improve performance, but decreased the F1-score to 0.82. We did therefore not choose to keep this for the advanced model.

3 Part 3: Advanced Model

3.1 Data Splitting & Vectorization

After loading the dataset, we looked at the distribution of articles we had for each category, to get an overview. We then split the data using `train_test_split` to separate the data into **train** and **test** subsets with an 80/20 % distribution.

After splitting up our data, we fitted our data with a vectorization technique, more specifically Sci-kit's TF-IDF Vectorizer. We tested a large amount of max-feature sizes, and discovered that more often than not a larger maximum resulted in a better performing neural network - at the cost of longer training time.

3.2 ROS

When looking at the previous distribution of fake/reliable, we discovered that the percentage of reliable articles was at 66.07%, meaning that for every fake article, we would have 2 reliable ones. As discussed in a previous assignment, an uneven distribution of types, can lead to a bad performing model, since it might be extraordinarily good at guessing which are reliable by just guessing every article as reliable. We solve this issued by using an over sampler, specifically a Random Over Sampler (ROS). ROS takes balances the dataset by duplicating the type which has a lower count (fake in our case), but does so randomly. We chose to go with ROS over e.g. SMOTE or ADASYN due to sheer computation time, with minimal difference in performance.

3.3 MLP

After fixing our imbalance in the training set, it was time to train a classifier model. We opted for the `MLPClassifier` (Multi Layer Perceptron Classifier) from Sci-Kit Learn.

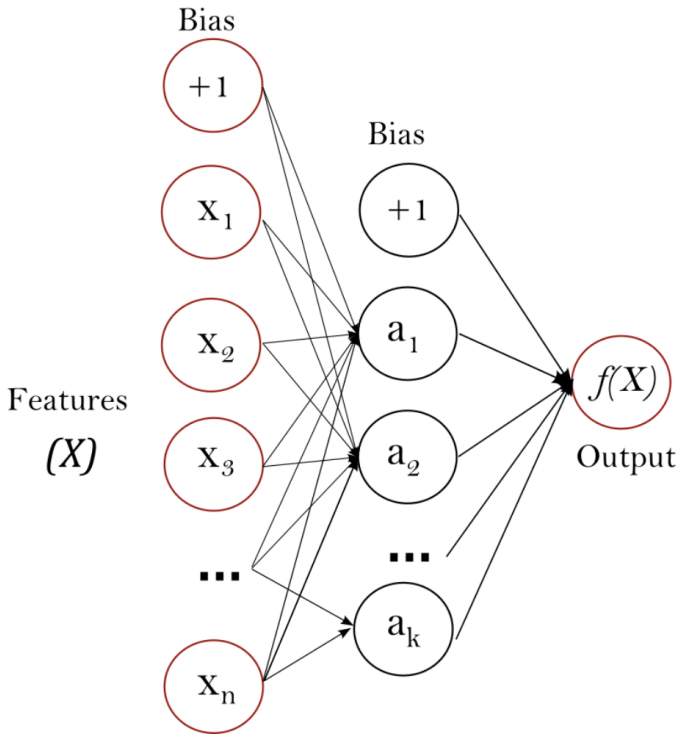


Figure 4: One hidden layer MLP. Image source: `scikit-learn`

From the many options the model has, we specifically used the `hidden_layer_sizes` to change the amount of layers and neutrons in said layers, `tol` (Tolerance On Loss) to set a tolerance for when our model (which uses the "adam" solver as its Gradient Descent for our weight optimization) should stop its iterations. Our classifier uses an adaptive `learning_rate` where the `tol` defines how much the loss of our model can increase before it should terminate. If set too low, the model might end up going for infinite iterations and would never stop, but if set too high, it might terminate before we get a well-functioning model. We therefore also set a maximum for how many iterations it could run, and defined that if nothing much changes [1] in 4 iterations, it would terminate.

[1] I.e. the change are increasing, or decreasing minimally defined from the tol

3.4 Predictor & confusion matrix

Lastly, after training our classifier for many iterations, we use it on our `X_test_vectorized`[2] data to predict whether or not a given new article is reliable or fake. We then show our results in a confusion matrix and print out a classification report to see both individual F1-scores and the Macro Average score, which gives us:

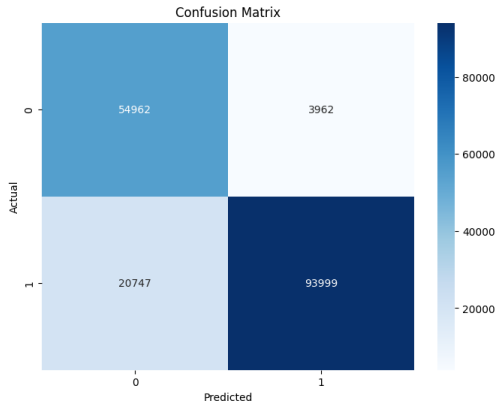


Figure 5: Confusion Matrix for Advanced model run with 995,000 rows

	precision	recall	f1-score	support
fake	0.73	0.93	0.82	58924
reliable	0.96	0.82	0.88	114746
accuracy			0.86	173670
macro avg	0.84	0.88	**0.85**	173670
weighted avg	0.88	0.86	0.86	173670

Table 1: Classification report with `macro avg` score of 85%

4 Part 4: Evaluation

To evaluate the model, the LIAR dataset can be used, which is a dataset of simple statements ranked on a scale of false to true. This has both benefits and drawbacks. It is beneficial because a very intelligent model would be able to also classify statements as true or false. The drawback is that these are both a simple models, so it would not be a surprise if they underperform on another format.

Simple model	Fake News Corpus Test Set	LIAR Dataset
F1 Fake	0.79	0.70
F1 Reliable	0.88	0.16
F1 macro average	0.84	0.43

Table 2: Comparison of the simple model to the fake news corpus and LIARS dataset

Advanced Model	Fake News Corpus Test Set	LIAR Dataset
F1 Fake	0.82	0.44
F1 Reliable	0.88	0.52
F1 macro average	0.85	0.48

Table 3: Comparison of the advanced model to the fake news corpus and LIARS dataset

[2]From the TF-IDF vectorizer

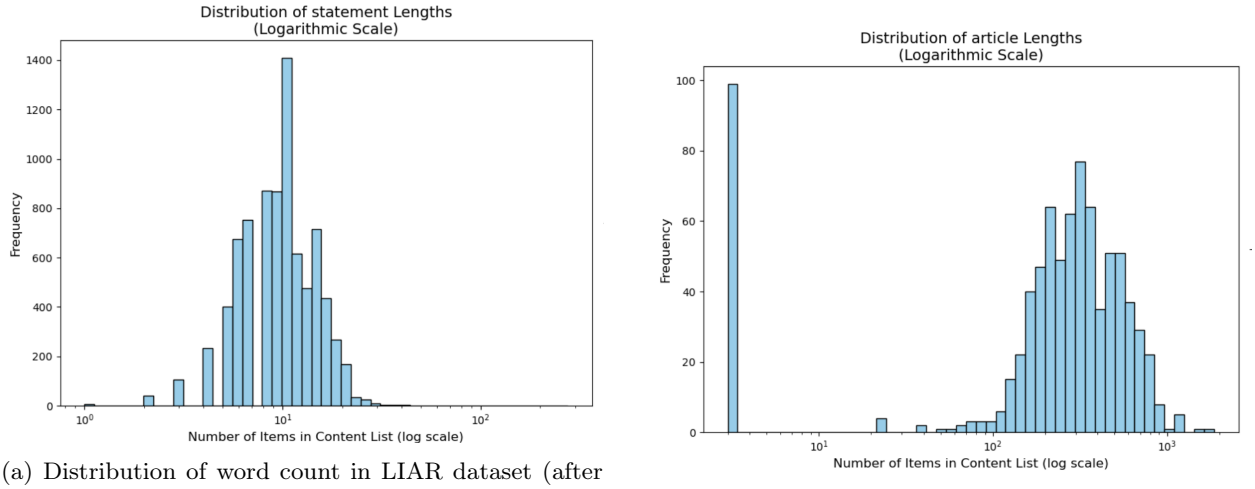
Note results from Fake News Corpus from both the Simple and Advanced models are from the full set of 995,000 and not a small snippet.

The model performs significantly worse on the LIAR dataset. An F1-score of 0.43 compared to the previous 0.84 on the simple model and an F1-score of 0.48 compared to the previous 0.85. On the confusion matrix, it can be seen that it predicts almost all statements from the LIAR dataset as fake news.

5 Part 5: Conclusion

5.1 Discrepancy between test set and LIAR set

As can be seen in Figure 5 , the average lengths of a statement are far less, with an average word count of 10 than in the articles, where the average length count is 320.



(a) Distribution of word count in LIAR dataset (after cleaning)

(b) Distribution of word count in articles (after cleaning)

Figure 6: Comparison of word count distributions

This massive change in data is bound to create some problems, if not accounted for. The problems might be caused by; I), our use of CountVector(), which converts the words in an article into a vector. Short statements make very sparse vectors though, with minimal information. II), If some of the words in the statements are not included in the 10.000 most common words used for the training, the model has no idea what to do. III), The model might have found a correlation between length and classification, which makes statements seem less trustworthy.

5.2 Lessons learned and considerations

Models do not generalize well if they are not trained on a generalized dataset. Furthermore, the models we trained on large datasets did not necessarily outperform the models trained on smaller datasets. This might suggest that our methodology and/or model is wrong.

For better performance on the LIAR dataset, short statements will have to be included in the training set, but this might also decrease the accuracy on normal articles. So a new model, which can capture more complicated relations, might be the best improvement.

Choosing the correct model, parameters and oversampling method often requires either sacrificing performance or execution time, which results in either a model that does not perform as well as it could, or a model that takes an extremely long time to train. Finding a good combination can prove especially difficult.