FAKENEWS PROJECT

TECHNICAL REPORT - GITHUB: HTTPS://GITHUB.COM/KRISTIANDAMPEDERSEN/FAKENEWSPROJECT

Kristian Dam Pedersen
tgz790

Joshua Niemelä
gmw103

Jákup Lützen
qkz344

March 30, 2023

Introduction

Fake news has increasingly entered the public discourse in recent years. The Oxford English Dictionary coined the term in 2013, and since then the use of the term has only gotten more and more common. We've encountered novel challenges with regards to the spreading of false information, both through informal means (such as social media) and increasingly through traditional media as well. In this technical report, we'll throw our hat into the ring, by building a fake news classifier using 3 different complex models (a large deep neural network, a smaller DNN and a model based on XGBoost). We'll start by going over our data cleaning and processing. We will then use logistic regression to create a simple model that we can use as a baseline. We will then go over our choice of complex models, before we take a look at the results and compare performance across the different models. Additionally, we will assess the generalisability of our models to unfamiliar domains by evaluating them on a new dataset, LIAR.

1 Data Exploration

1.1 Importing the FakeNewsCorpus dataset.

The first challenge was simply downloading and unpacking the files on our Linux systems. We ended up concatenating the files into a temporary zip file using `cat` and then unzipping the file using `unzip`.

Trying to read the data in python proved to be a challenging task, as the .csv file seemed to be corrupted or malformed with rows containing different amounts of columns. Some article content included `\r` (a carriage return), which was interpreted as "newline", and thus, a new row in the middle of arbitrary news content. A single command '`sed 's/\r/ /g' in.csv > out.csv`' replacing the carriage returns with spaces (to preserve word boundaries) turned out to solve this issue.

1.2 Larger than memory datasets

Due to the size of our dataset, we had to use different methods for handling our data. Throughout this report we used 3 different methods: SQL Databases, chunk-wise streaming and brute force (access to a large enough server). After getting the data into a readable CSV file in the previous step, we decided to load it into an SQL database for easier handling and exploration. In later sections we will use the other two methods as they are more appropriate for specific machine learning tasks.

1.2.1 Summary of the dataset

In this section we'll present some summary statistics. The dataset contains 8.528.956 articles. The articles are sorted into the following categories: [bias, clickbait, conspiracy, fake, hate, junksci (junk science), political, reliable, rumor, satire, unknown, unreliable]. The number of articles in each category is distributed as seen in figure 1. The dataset contains many articles from just a few publishers, as can be seen in figure 1.

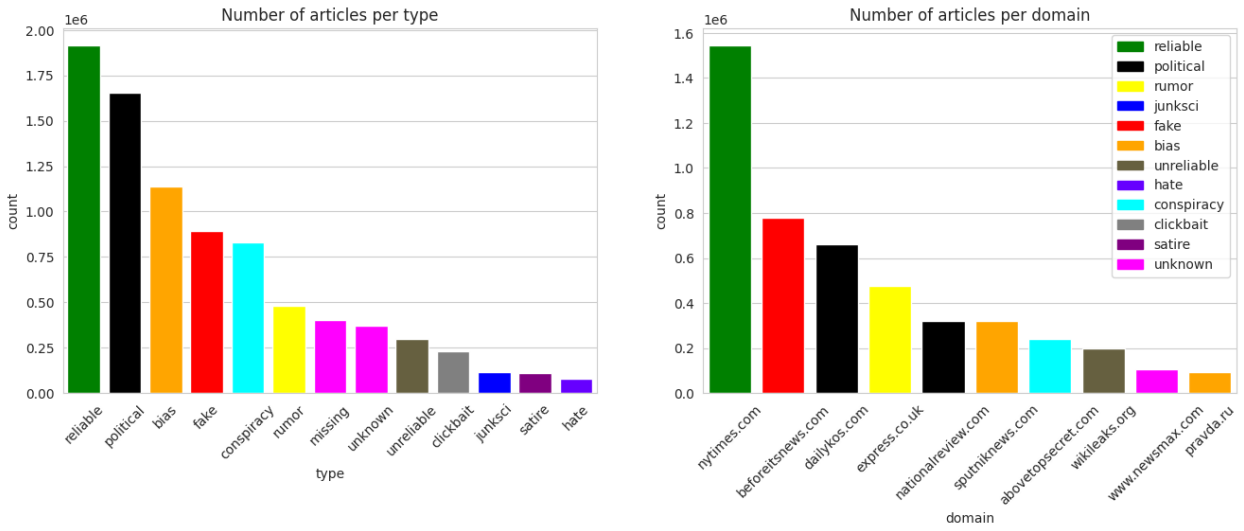

Figure 1: Article type and domain distribution

1.2.2 Vocabulary sizes

As an exploration, we decided to calculate the vocabulary size of a sample of the dataset. We did this by first tokenising the sample, and then counting the unique tokens. We then repeated this process after removing stopwords, and finally after stemming the tokens. The results are shown in table 1. As we can see the vocabulary decreases slightly after removing stopwords, and significantly after stemming. As we see, removing a list of common words doesn't reduce the vocabulary much, but reducing words to their root does.

Data	vocabulary size	vocabulary size (lowered)	% decrease	% decrease (lowered)
Sample data	21016	18011	0%	0%
Removed stopwords	20674	17865	1.63%	0.81%
Stemmed	12654	12703	39.8%	29.5%

Table 1: Vocabulary sizes

1.2.3 Classifying truth per domain

A central aspect of the dataset, is the way it classifies news articles in terms of reliability. The creators of the dataset have chosen to classify the articles per domain, meaning that all the articles from any given domain will be given the same level of credibility.

1.2.4 Duplicated articles

Another thing we realised quite quickly, is that the dataset contains a significant amount of duplicated articles as seen in table 2.

As we can see, this is due to different popups that the webscraper has encountered as it was trying to scrape the dataset. These duplicated articles were simply removed before we trained our model.

Table 2: Duplicated articles and their domain

domain	type	content	count	% of corpus
nationalreview.com	political	Plus one article on Google Plus (Thanks to al...	273388	3.2%
wikileaks.org	unreliable	Tor Tor is an encrypted anonymising network t...	169259	2.0%
sputniknews.com	bias	Dear readers, we are excited to announce that ...	102671	1.2%
⋮				

1.2.5 Lower dimensionality

We have made a 2D UMAP representation of the dataset, which is a form of non-linear dimensionality reduction. We've trained the model unsupervised to get the inherent distribution of the data without any labelling bias.

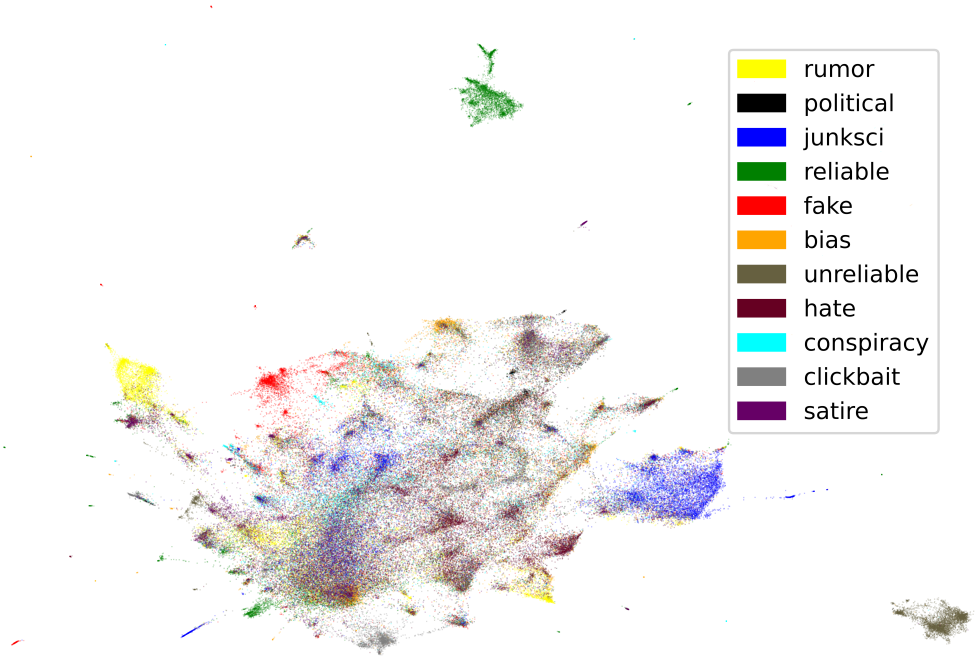

Figure 2: UMAP representation

The plot, figure 2 contains a large cluster which contains a rough mix of everything which UMAP couldn't separate as well as some more concentrated clusters of junk science and conspiracy articles. On the boundaries of the big cluster, we have some more concentrated clusters that the model was able to separete, this includes reliable and fake. Away from the main cluster, we have a large cluster of almost entirely reliable content the model was easily able to separate. As we see in figure 1 NY Times is by far the most represented domain, and all the articles coming from this domain is marked as reliable. Consequently, the large reliable cluster we see here consists of the articles from NY Times. We were able to determine this, by querying the UMAP model for the specific coordinates of various articles from a selection of sources. Other clusters far apart include unreliable and rumors.
The clear deliniation of the reliable data served as part of our justification for the binary grouping of *reliable* and *unreliable*.

Data pipeline

1.2.6 What features to focus on

For the models, we chose to primarily focus on the content of the articles as predictors for whether a given article is fake or not. Whilst we were tempted to include "domain" initially, after the discovery discussed in section 1.2.3, we decided against it. We also chose to ignore the titles of the articles, as these where often simply reflections of the content.

1.2.7 Grouping

We decided that we wanted to build a binary classifer, that classifies wether an article is "reliable" or not. This choice can be seen as a rather arbitrary one, but it is dependent on the hypothetical use case. In our usage, we imagine a parental filter, that would only expose children to predominantly reliable content. Because of our focus on the reliability of the article, we chose to encode all "reliable" articles as 1's, whilst the rest of the types ("fake", "bias", etc.) got encoded as 0's. With this approach we will be (perhaps unfairly) strict in our judgement of a given article, but we reason that it is better to accidentally label a reliable article as false, than it is to accidentally label a false article as reliable.

1.2.8 Splitting the dataset

We decided to split our data into into a training, validation and testing set, with a 80-10-10 split. And then balanced the training data.

1.2.9 Tokenisation

We tokenised the text by first splitting it into individual words, removing stop words and using lemmatisation to further reduce the vocabulary by reducing the words into their base form.

1.2.10 Feature extraction using TF-IDF

After tokenising the content column, we were then able to train a TF-IDF (Term Frequency - Inverse Document Frequency) model on it. This model looks at how often a given token (word) appears in an article, and comparing to how many times that token appears accross the entire dataset. By training this model, we essentially output a large vector, which we can use to encode the tokens into a large matrix that can be fed into our models.

1.2.11 Dimensionality reduction using Truncated Single Value Decomposition (SVD)

After encoding the data using our TF-IDF model, we were left with quite a large input space, which meant that it would be useful for us to reduce the dimensionality of our input tensor. In other words, we wanted to group similar elements (or flags) together, such that the models we build have a more clear delineation between inputs. To do this we used truncated single value decomposition (SVD). SVD works by factoring a matrix with the hopes to extract most impactful elements of it.

1.2.12 Individual model differences

Whilst the above steps are broadly applicable to all the models we built, there was some individual difference from model to model. These are seen in figure 3.

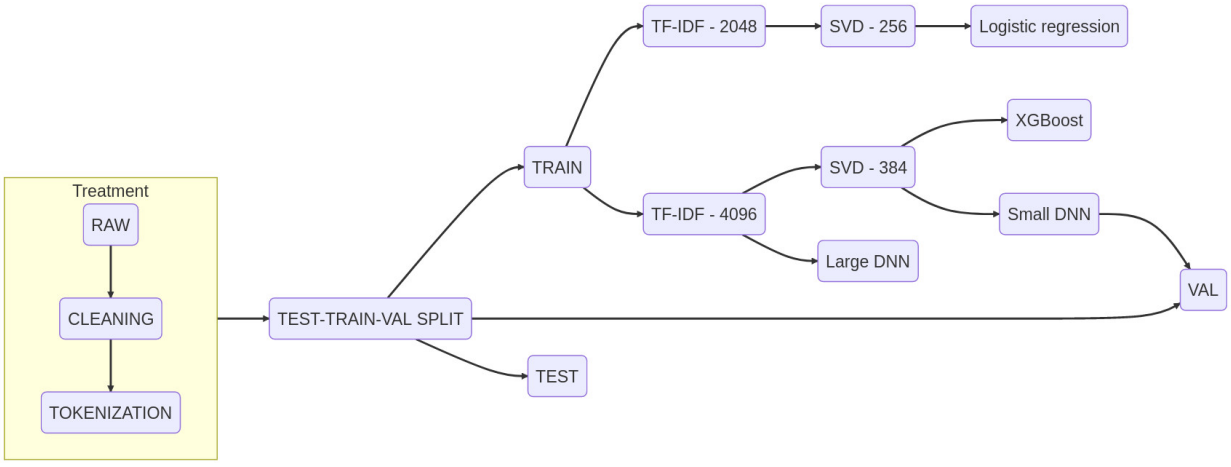

Figure 3: Pipeline

There were a few reasons for the differences. **Logistic regression**: For our simple model we initially set the TF-IDF and SVD parameters quite low. We set our TF-IDF model to output a 2048 dimensional vector that get reduced to 256

dimensions through SVD. **XGBoost and Small DNN**: we chose to increase the dimensionality such that our TF-IDF ouputted a 4096 dimensional vector, that then got reduced to 384 dimensions. **Large DNN** : We chose to omit the dimensionality reduction from the large DNN, in order to see wether a larger model would improve performance.

2 Simple model: Logistic Regression

For our baseline, we chose to use a logistic regression model. Much of the heavy lifting in developing this model took place in the previous sections, so once we reached this step we chose to just use a relatively naive implementation, using the default arguments provided by *scikit-learn*. This resulted in the following hyperparameters for our model. **Penalty**: l2. This form of penalty adds a penalty that is equal to the square of the magnitude of the coefficients of the model. In other words, it tries to limit the influence that any single coefficient has on the model. **Solver**: lbfgs (limited memory BFGS). This solver is one of the most general logistic regression solvers, and is therefore a natural fit when we don't wish to direct it further.

3 Complex models

3.1 XGBoost

XGBoost is an ensemble learner, meaning it is a collection of smaller models. The way this algorithm is trained, is we start with a decision tree. In every boosting round, which is done 10 times, we find the weights responsible for incorrect classifications, boost them (making them more important for the new tree), and attempt to minimize the loss. This means every subsequent tree is trying to correct the errors of the previous tree.

3.2 Large DNN

This model uses TF-IDF 4096 without further dimensionality reduction. It is a simple feed-forward network with the layers 1024, DO, 512, DO, 256, DO, 128, DO, 64, 1 where DO is a drop-out layer of 0.2, meaning every epoch it resets 20% of the weights. The idea with this funnel-shaped network was that it would reduce the dimensions on its own and learn how to classify the dataset with minimal loss of data. Shortcomings of the model: The model is on the larger side (71 MiB), was slow to train and has a slow inference speed.

3.3 Small DNN

This model is a more traditionally shaped feed-forward network. Since the data was passed through dimensionality reduction already, it means the initial input space can be smaller. The layers are: 384, DO, 512, DO, 512, DO, 128, DO', 16, 1. DO' is a dropout of 0.1 and DO is the same as in the large DNN. The initial SVD step made this model much smaller, and faster at inference.

4 Results and evaluation

In this section we will discuss the performance of both our simple and complex models, and discuss how these stack up against eachother. We will discuss the accuracy and performance on both the FakeNewsCorpus dataset and the LIAR dataset.

4.1 Relative performance on FakeNewsCorpus

Let's begin by looking at our models performance on the FakeNewsCorpus dataset. As we see from table 3, all of our models performed admirably when put against the test set, with our more complex models pulling ahead comfortably.

Table 3: Model performance on FakeNewsCorpus dataset

Model	Logistic Regression	Large DNN	Small DNN	XGBoost
Precision (Reliable / Unreliable)	0.83 / 0.94	0.95 / 0.98	0.92 / 0.98	0.87 / 0.95
Recall (Reliable / Unreliable)	0.87 / 0.92	0.95 / 0.92	0.95 / 0.96	0.89 / 0.94
F1-Score	0.91	0.97	0.96	0.93
Size	15.6 MiB	71 MiB	35 MiB	27.1 MiB

4.2 Generalisability to other datasets (LIAR)

However, whilst these results are impressive, they also hint that our models might be overfitted to our domain. This is likely due to the nature of the dataset we used for training not being general enough, which means a broader selection of data sources and data sets would be needed to achieve a more generalisable model. This issue is highlighted in particular by the relatively terrible accuracy achieved on the LIAR dataset as seen in table 4.

Table 4: Model performance on LIAR dataset

Model	Logistic Regression	Large DNN	Small DNN	XGBoost
Precision (True / False)	0.17 / 0.84	0.18 / 0.84	0.18 / 0.84	0.17 / 0.84
Recall (True / False)	0.13 / 0.88	0.07 / 0.94	0.12 / 0.89	0.19 / 0.83
F1-Score	0.76	0.80	0.77	0.73

4.2.1 Performance visualisation

These different metrics can be difficult to parse when evaluating on unbalanced datasets. This is where visual aids can come in handy. Confusion matrices and ROC curves effectively show how much our models struggled with predictions outside their native domain.The confusion matrices are split up into two columns and two rows. For a perfect model we want the matrix to have a perfectly colored in diagonal, such that its perfect at predicting true positives and true negatives. However as we see just columns, it means that our models have zero predictive power. This sentiment is echoed by the ROC plots.

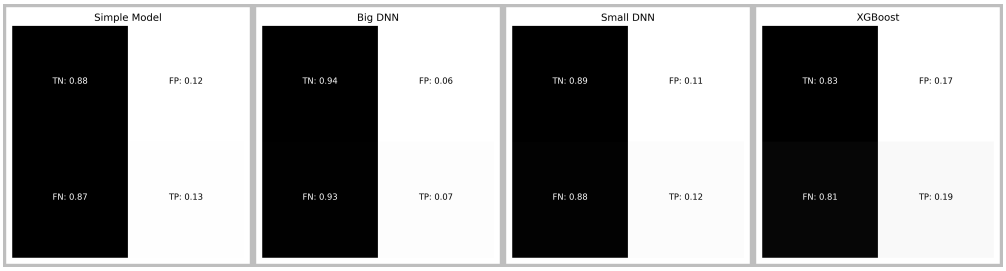

Figure 4: Confusion matrices for models predicting on LIAR dataset

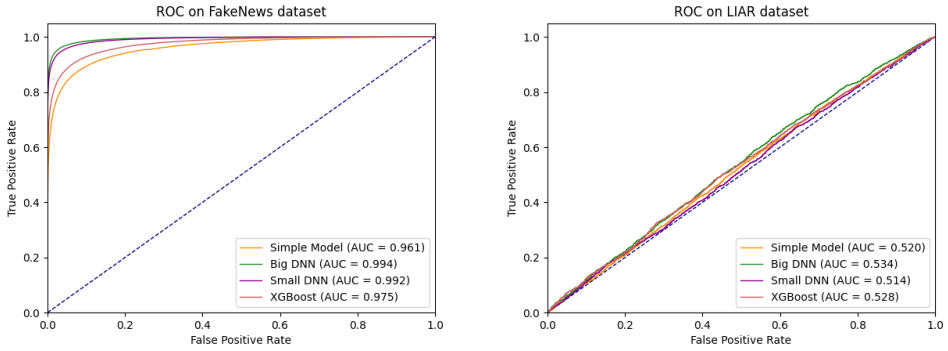

Figure 5: ROC comparison - FakeNewsCorpus and LIAR

4.2.2 Inseparability of LIAR on fitted models

We ran LIAR through the same TF-IDF, SVD and UMAP pipeline to see where they were located in the clusters, we would hope to see some sort of separation of the various levels of truthiness if we want to have any hope in predicting truthiness of the LIAR dataset. As seen in the UMAP representation (figure 6), the 6 classes of LIAR are interleaved, which means we will not be able to separate the classes nor make any meaningful inferences on the dataset.

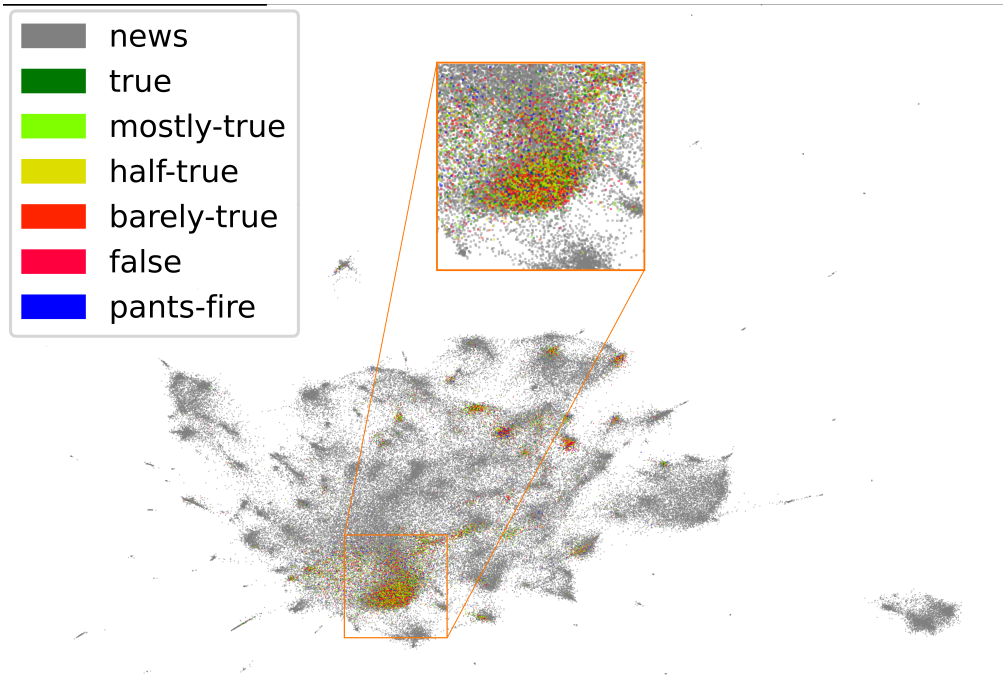

Figure 6: UMAP - LIAR overlayed on FakeNewsCorpus

5 Conclusion

5.1 Dataset issues

As has been highlighted throughout this report, the FakeNewsCorpus dataset has many shortcomings, chief among which is the classification of reliability based on domain. As we see, our classifiers trained on this dataset, generalise quite poorly outside of their original domain.

5.2 TF-IDF limitations

Another key limitation is the use of TF-IDF model for embedding the content. The limitation is that TF-IDF does not provide us with local context. It instead provides us with a single embedding representing the important keywords of the entire document, without understanding the semantics of individual sentences. Here, a more complex word embedding model would be a sensible alternative to try.

5.3 Ethical considerations with fake news classification

The subpar performance on the LIAR dataset highlights the difficulty in creating a proper fake news classifier. When trying to generalise a model, questions about what consitutes reliable, fake or misleading news quickly arises. Does a publication raising awareness of the real side-effects of the HPV vaccine count as reliable? And how does an algorithm distinguish this from a publication spreading fake statistics of the corona vaccine? Whilst there are publishers that we all (generally) accept as being reliable, and others that are generally percieved as being unreliable, the exact line delineating the two is hard (if not impossible) to strongly define.

In this technical report we described the exploration of three different models for fake news classification. We started by outlining the data import, exploration and processing we applied to the FakeNewsCorpus dataset. The processing included both tokenisation, feature extracting using TF-IDF and dimensionality reduction through single value decomposition. We build a logistic regression model to serve as our baseline, and then constructed three more complex models (two DNNs and an XGBoost model). Whilst all of the models performed admirably on the FakeNewsCorpus dataset, they all failed to generalise when we tested them against another dataset (LIAR). As discussed in our evaluation section, this is likely due both to the quality of our dataset and the inherent difficulties in constructing fake news classifiers in general. In the end we conclude that our approach has failed to build a general fake news classifier, and that more sophisticated tools need to be incorporated in order to achieve generalisable success.