FINAL PROJECT

KU-id: mgr829
University of Copenhagen
mgr829@alumni.ku.dk

KU-id: pvf168
University of Copenhagen
pvf168@alumni.ku.dk

KU-id: wzr263
University of Copenhagen
wzr263@alumni.ku.dk

July 5, 2022

ABSTRACT

In this final project we combine the work we have done in the previous milestones and exercises. We further build on top of the prior work creating a relational database schema for the data scraped from the WikiNews site and develop both baseline and extended models that try to predict if an article is fake or not. As a baseline model we used logistic regression (LR) with a bigram vectorization of the corpus. We further developed a multitude of models making use of Term Frequency - Inverse Document Frequency (TF-IDF), bigram and trigram vectorization combined with classification using k-Nearest Neighbours (KNN), Decision Trees (DT) and Support Vector Machines (SVM) to see if we can improve the accuracy of prediction compared to the baseline model. The accuracy of the baseline model on the dataset without metadata is 0.871 i.e. 87.1% with an F1-score of 0.843. The best performing model on the dataset without metadata was using TF-IDF vectorization and SVM. The accuracy of this model was 0.916 i.e. 91.6% with an F1-score of 0.903. These models have been trained and tested on a subset of 75.000 randomly chosen articles of the FakeNewsCorpus. Cross domain performance is tested on the LIAR dataset and shows a decline in overall performance. The accuracy of the baseline model is 71.9% with an F1-score of 0.421. Even though 71.9% might seem reasonable it is misleading as there is a large type II error for all models. This causes most of the models cross-domain accuracy to lie around 70% which is close to the actual ratio of REAL/FAKE in the LIAR dataset. The source code of the project can be found on the following link.

Keywords NLP · SQL · FakeNewsCorpus · LIAR dataset · TF-IDF · n-grams · k-Nearest Neighbours · Logistic regression · Support Vector Machines · Decision Tree Classifier · Fake news classification

1 Know your data

1.1 Representing the FakeNewsCorpus

The design of the E/R diagram in Fig. 1 takes into account that the variables `author`, `tag` and `meta_keywords` are entity sets i.e. an article can have a large number of `authors`, `tags` and `meta_keywords`, that is, they are not atomic and it would thus not make sense to have them as direct attributes of an article. The `domain` is also an entity set, but in this case if the domain changes the changes do not need to be applied to all records in the article relation - only on the domain relation. In this case there is a functional dependency between the `domain` and the `type` of the article, that is, if we know the domain, we can conclude which type the article has. Furthermore we omit the variables `keywords`, `source` and `summary` as these variables only contain `Null` i.e. no observations across all articles. As `meta_keywords` does contain observations we do include this as shown in the figure. As the timestamps for `scraped_at`, `inserted_at`, and `updated_at` are already atomic we let these be attributes of an article.

1.2 Inherent problems with the data

After exploring the data, we found that there were a number of inherent problems. We will exemplify these with some of the queries from App. C. First, there were a significant number of duplicates in the content field as the same content had been posted to different URLs on the same site such that they shared the same metadata and content but with

differing URLs. To account for this we removed all duplicates of the content field such that the database only contains unique content fields and at the same time only keeping one URL pointing to the content. This reduced the number of articles by approximately 370.000.

Second, there were a number of problems with the data containing information about the author of the article. As seen in App. C List. 3 only a few of the 30 most common author names actually contain the name/URL of the author/authoring site. Furthermore a few of the author names repeat themselves because they differ by one space symbol e.g. the author name "Posted On" and "␣Posted On" differ because of the space character in the start. This can be due to the way webpages differ and the way the scraping was done.

Third, (Rubin et al., 2015) defines nine requirements for a fake news corpus: (i) Availability of both truthful and deceptive instances; (ii) digital textual format accessibility; (iii) verifiability of ground truth; (iv) homogeneity in lengths; (v) homogeneity in writing matters; (vi) predefined time frame; (vii) manner of news delivery; (viii) pragmatic concerns; (ix) consideration for language and cultural differences. From inspecting the data we find that some of these do not hold for the FakeNewsCorpus. For example the two homogeneity requirements do not hold as there is a great variability in length of articles including that there is a great difference in the writing style of the articles within each of the labels. Furthermore, the labels in the FakeNewsCorpus are given to each article depending on website origin. This may be misleading as articles should not solely be labelled according to which website they are posted on, author of the article or publisher (Oshikawa et al., 2018). (Shu et al., 2020) have found that the majority of publishers who have previously published fake news only publish one piece of fake news which can cause data to be misleading if labels have been created on the basis of the publisher.

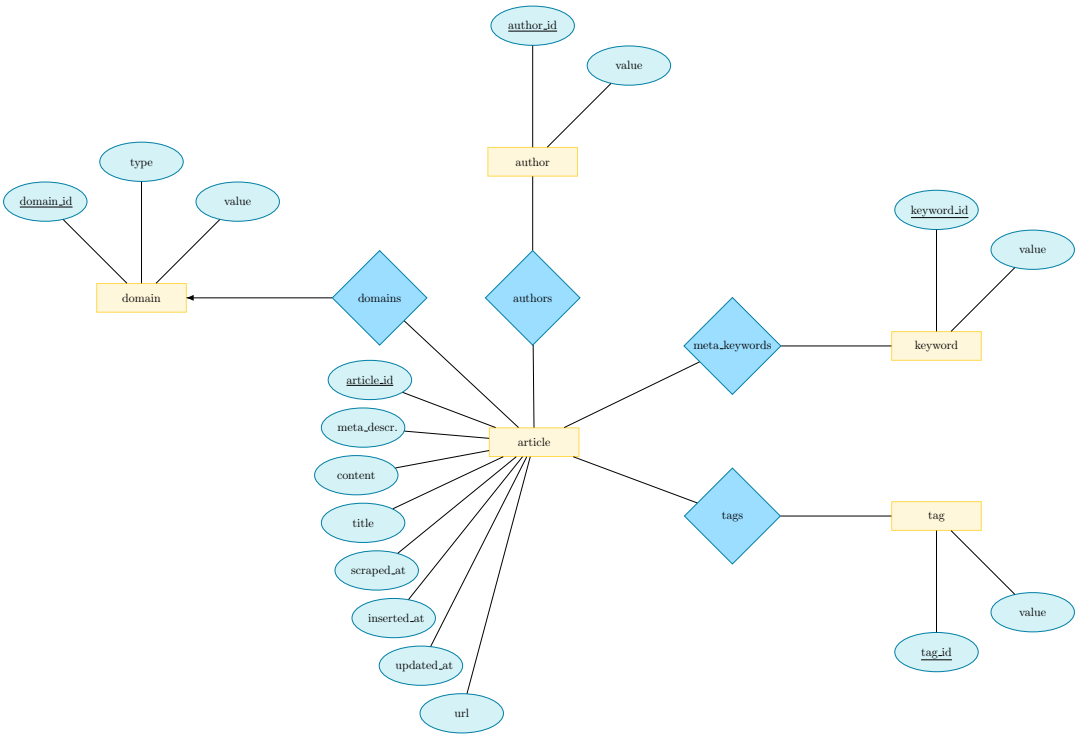

Figure 1: Entity-Relationship model of the FakeNews database. The underlined variables signify which of the attributes work as keys.

1.3 Key properties of the data

From App. C List. 1 we see that most of the articles do not contain any keywords but for the ones that do the keywords are mostly concerning politics. From App. C List. 2 it is seen that the average length of the articles is around 3.590 characters, the longest article is 100.000 characters and the shortest article is 41 characters. Furthermore, most of the tags concerns politics or race. From App. C List. 7 we can see that the most common article type is political followed by fake. The least common article type is hate.

1.4 Experiences with scraping

Inspecting the Wikipedia News page there were a few inherent attributes which occurred on each news page i.e. headline, publishing date, text, and sources. We focused on these and extracted a dataset containing data in each category. We started by collecting the links and headline for each news article in the assigned range of letters given our group number. This was done by finding all the links in the parsed `html` that we got as output by using the package BeautifulSoup and then appending the links to a list which was converted to a pandas dataframe. This made it possible for us to iterate through the list and collect the data from each individual newspage and add each of the individual data attributes. A description of the whole process including the generation of keywords for each article can be found in App. D.

1.5 Creating a relational database schema to represent scraped data

For the WikiNews data we created a relation between the entity sets with separate atomic values. The design of the E/R diagram in Fig. 2 takes into account that the variables `article`, `author` and `keywords` are entity sets. As the `publish_date`, `url` and `content` are already atomic we let these be attributes of an article. Furthermore the values of `author` and `keyword` are also atomic.

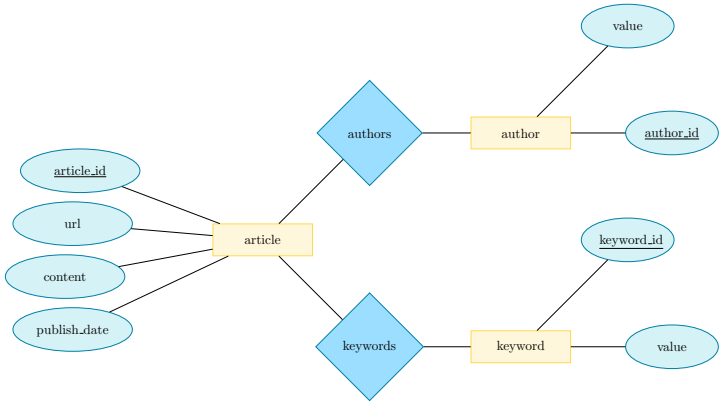

Figure 2: Entity-Relationship model of the WikiNews database. The underlined variables signify which of the attributes work as keys.

1.6 Basic statistics on scraped data

From App. C List. 4 we see that the average length of the articles is 2.612 characters and that the longest article contains 35.011 characters. Furthermore, we can see that most articles were published in 2006 with few being published after 2012. In App. C List. 5 we see that the keywords generated from the text mostly concern politics and society.

1.7 Creating a view that integrates the article information from both sources

In order to create a joint view of the two datasets, we read the two different schemas into the same database as `public` and `public2`. This allows us to create a view as seen in App. C List. 6 which works similar to a relation table, without altering the tables below and is recomposed when called. To make the schemas fit we created a new attribute for the WikiNews articles called `label` which is the type, that we set to "reliable". As both datasets are structured around article id's which will not be unique in the joint view, we create a new dimension `db` to specify the dataset origin, which can be used in the view-relation as part of the primary key. A table with `id, db, title, author, type` and `keywords` from both datasets is obtained and can be used for queries and modelling.

1.8 Data used to train and test the coming models

The fake label constitutes approx. 34% of the articles. Therefore we choose not to include the WikiNews data to avoid further imbalancing of the data assuming that the WikiNews articles are reliable. Off all the articles remaining after removing duplicates we have randomly picked 75.000 articles (60.000 for training and 15.000 for tests). We have chosen to randomly pick 75.000 articles such that the assumption that the data is independent and identically distributed still holds. Before feeding into the models the raw text data required some preprocessing. We started by removing URLs, digits and special characters from the texts. Thereafter we removed stop words and stemmed the texts.

1.9 Creating two classes (FAKE/REAL) in the FakeNewsCorpus

The FakeNewsCorpus contains a number of different labels regarding the articles as seen in App. C List. 7. Some of these labels refer to articles that are more verifiable than others. In our selection of the dataset in a FAKE/REAL split, we chose the FAKE label to contain articles labled fake, conspiracy, unreliable and junksci. These labels contain articles ranging from being fabricated to articles claiming truthfulness of pseudoscience, metaphysics, and other scientifically false claims. We chose not to include rumor as fake as it lies in between being real or fake and might have been created with a foundation in a true statement. Furthermore following (Nasir et al., 2021) and (Zubiaga et al., 2017) a rumor is considered as unconfirmed information rather than false information which by definition is deemed false.

2 Establish a baseline

2.1 Choosing baseline models

We seek to establish a baseline which should outperform what (Skiena, 2017) calls "the monkey" (random guessing) and "the sharp" (cleverly choosing a distribution that makes no sense but gives high accuracy. Outperforming the monkey may be manageable. However, a challenge may be the imbalanced dataset - if "the sharp" chooses all articles to be reliable, then this would result in an accuracy of about 70%.

Having established these rudimentary basics for evaluation of our performance metrics, we choose logistic regression (LR) as our baseline model for several reasons. Firstly, it is easy to implement and update with more data. Secondly, it has clear probabilistic interpretations (see App. B) and thirdly it works well with larger datasets like ours as outliers pose little problem, yielding low asymptotic error. It does too poorly with non-linear data and multi-dimensional datasets to be considered an advanced model in this context, but it performs well on binary classification problems (Skiena, 2017).

We are starting out by using only features from the main text in the content field, but we do so using three different vectorizers on the stemmed Bag of Words (BoW), that is n-grams with both 2 and 3 words as well as Term Frequency - Inverse Document Frequency (TF-IDF) frequencies for the tokens (App. A). As expected the baseline models with LR does quite well with accuracies approaching 90% and fairly equal precisions and recalls (especially in the macro weighted versions to compensate for the imbalance in the dataset) combining to F1-scores of 84-89%. More on these metrics in section 3.

2.2 Effect of including metadata features to the model data

We also experimented with including other metadata than the content, which might lead to a decrease in performance with multi-dimensional data when using LR (Skiena, 2017). Contrary to intuition performance improves overall perhaps due to a larger sample of word vectors when including the title. The accuracy score for LR with bigram vectorizers improves from 0.871 to 0.873, with trigram vectorizers from 0.883 to 0.888 and with TF-IDF from 0.906 to 0.935 combined with an increase in F1-score across all models. We also tried to run the model on the titles alone, based on a theory that these short texts might resemble the statements in the LIAR dataset more. However, performance drops dramatically for the FakeNewsCorpus prediction, and makes no visible difference for the LIAR dataset.

3 Create a Fake News predictor

3.1 Quantifying performance of the Fake News predictor

In order to challenge our baseline, we try out a few different machine learning approaches with different advantages and disadvantages. For each of our vectorizers, we run models using linear Support Vector Machines (SVM), decision trees (DT) and k-Nearest Neighbours (KNN), with 1, 3, 5, 7 and 10 nearest neighbours as classifiers (App. B). Without the metadata a close run was observed between the different models. Here our baseline model also performed well. However, the TF-IDF vectorizer combined with the linear SVM model achieved the best overall performance with an accuracy of 0.916 as well as high precision and recall, with an F1-score of 0.903. This will therefore be our prime choice for fake news detection. Results can be seen in Table 1. For a full table with metrics for each model please follow this link.

3.2 Choice of approach

Accuracy is the most intuitive metric measure representing the percentage of true calls out of the total number of calls. This however does not take into account if these are achieved by random guessing or proposing that everything is

reliable or fake news. Precision helps by calculating the ratio of true positive predictions out of the actual positives. As fake news datasets, are often imbalanced or skewed, a high value for this can easily be obtained by making fewer positive predictions (Shu et al., 2017). Recall is the ratio of true positive divided by the sum of true positives and false negatives, causing a trade-off between precision and recall for skewed datasets and bad prediction (Skiena, 2017). Combining the harmonic mean of these in the F1-score is therefore an overall well-performing measure. All these measures should be as close to one as possible. (Shu et al., 2017)

DT are, much like our baseline LR model easy to understand and interpret. It is an independent model, not much affected by irrelevant features or missing values, which should be a benefit (Khanam et al., 2021). However, like LR it can be prone to overfitting, which may explain why the performance was close to our baseline model especially for trigram vectorization. While DT perform decently, KNN are trailing behind our baseline performance. This is true for all vectorizers, but especially for the TF-IDF's, having even lower accuracy than "the sharp" (< 0.7) and a recall and F1-score of approx. 0.5. Using KNN's was worth a try, as they generally work well for explorations with no assumptions about the data and constantly evolving as more data is added (Skiena, 2017). An explanation for the poor performance of the KNN's may be due to the imbalanced data and high-dimensional features.

SVM does share the theoretical similarity with LR, in that it constructs a set of hyperplanes which can be used as a decision boundary for classification. Since we are performing a binary classification problem, which is preferred for LR (Skiena, 2017) this may explain why our baseline LR model is only outperformed by the SVM's.

We conclude that the linear SVM model has the best performance, beating our baseline LR model. Top performance is reached with the TF-IDF vectorizer as mentioned reaching an accuracy of 0.916 and a weighted F1-score 0.903. This confirms the popularity of SVM's in text classification and NLP as it is very well suited for the high-dimensionality of handling BoW, with a high accuracy and nice theoretical guarantees against overfitting (Oshikawa et al., 2018) (see App. B).

4 Performance beyond the original dataset

4.1 Cross-domain performance on the LIAR dataset

In this section we enter cross-domain application of our fake news detector, creating a new set of challenges and trade-offs on performance (Janicka et al., 2019, Silva et al., 2021). What we have done is to run all the models, we have trained, including our baseline LR and our best-performing linear SVM model on the LIAR dataset, repeating all the experiments with this as our test data.

4.2 Comparison of results

Viewing the accuracy of the models performance on the LIAR dataset, the level of performance is generally lower. Instead of scaling around mid 80% accuracy, it is in the lower 70% with some models even lower. Regarding the precision, recall and F1 scores we find that the precision and recall metrics are all just around 0.5 and the unweighted F1-score right about 0.42 for most models. Thinking closer about this, we notice how the accuracy of most models are in fact very close to 0.722. The confusion matrices further reveal that these predictions are completely skewed for most of the models in the experiment, predicting that all articles in the dataset are reliable. In this case not outperforming the "monkey and the sharp". Metrics and confusion matrices for each model can be found on this link.

5 Discussion

5.1 Discrepancy between the performance on the test set and on the LIAR set

As we saw in the last section the performance of our model on the LIAR dataset was poor compared to the FakeNews-Corpus. As "the sharp" is someone who knows the evaluation system, we are using, and can pick the baseline model which will do best according to it. With this knowledge he will try to make the evaluation statistic look bad by achieving a high score with a useless classifier (Skiena, 2017). As our dataset is somewhat imbalanced, with around 70% of articles classified as reliable, such an agenda can be achieved by predicting all the articles in the dataset as reliable. In this case it will have an accuracy of just over 70%. This seems to be the case for many of our models performance on the LIAR dataset, with low F1-scores implying low precision and/or recall.

This discrepancy in performance is not surprising considering the differences in our datasets. While the FakeNewsCorpus consists of all sorts of shorter and longer blogposts and articles from vast sources in full, the content of the LIAR dataset are short statements collected by PolitiFact, mostly from mainstream news sources (and twitter), which is curated by

			Without metadata				With metadata		Metadata only†			
			FakeNewsCorpus		LIAR dataset		FakeNewsCorpus		FakeNewsCorpus		LIAR dataset	
Vectorizer		Classifyer	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score
TF-IDF		LR	0.906	0.890	0.719	0.425	0.935	0.925	0.794	**0.745**	0.679	0.453
		KNN	-	-	-	-	-	-	-	-	-	-
		$k=1$	0.691	0.500	0.721	0.421	0.783	0.756	0.730	0.661	0.700	0.459
		$k=3$	0.680	0.468	0.722	0.419	0.798	0.761	0.738	0.641	0.718	0.432
		$k=5$	0.677	0.461	0.722	0.419	0.809	0.767	0.672	0.627	0.602	0.496
		$k=7$	0.676	0.457	0.722	0.419	0.813	0.767	0.676	0.625	0.616	0.495
		$k=10$	0.674	0.452	0.722	0.419	0.809	0.750	0.701	0.617	0.668	0.481
		DT	0.873	0.859	0.655	**0.483**	0.920	0.912	0.754	0.721	0.607	**0.497**
		SVM	**0.916**	**0.903**	0.718	0.428	**0.940**	**0.932**	**0.795**	0.743	0.689	0.447
n-grams	$n=2$	LR	0.871	0.843	0.719	0.421	0.873	0.846	0.723	0.587	0.702	0.446
		KNN	-	-	-	-	-	-	-	-	-	-
		$k=1$	0.776	0.757	0.466	0.456	0.787	0.769	0.502	0.496	0.432	0.428
		$k=3$	0.838	0.812	0.700	0.454	0.839	0.814	0.509	0.504	0.438	0.433
		$k=5$	0.850	0.821	0.710	0.434	0.855	0.828	0.718	0.590	0.703	0.470
		$k=7$	0.857	0.828	0.717	0.426	0.860	0.832	0.718	0.588	0.707	0.459
		$k=10$	0.861	0.830	0.721	0.420	0.866	0.836	0.721	0.583	0.714	0.433
		DT	0.837	0.818	0.700	0.448	0.855	0.837	0.722	0.588	0.711	0.448
		SVM	0.879	0.853	0.722	0.419	0.893	0.872	0.723	0.586	0.713	0.445
	$n=3$	LR	0.883	0.858	0.722	0.419	0.888	0.865	0.666	0.430	**0.721**	0.434
		KNN	-	-	-	-	-	-	-	-	-	-
		$k=1$	0.862	0.836	0.719	0.423	0.866	0.842	0.390	0.333	0.293	0.252
		$k=3$	0.878	0.852	0.721	0.419	0.883	0.859	0.390	0.334	0.294	0.253
		$k=5$	0.879	0.853	0.720	0.419	0.884	0.860	0.390	0.333	0.294	0.253
		$k=7$	0.881	0.855	0.721	0.419	0.885	0.861	0.389	0.331	0.293	0.251
		$k=10$	0.880	0.854	0.721	0.419	0.885	0.860	0.388	0.331	0.294	0.253
		DT	0.883	0.858	**0.723**	0.419	0.890	0.867	0.666	0.430	0.720	0.434
		SVM	0.878	0.854	0.721	0.419	0.886	0.864	0.665	0.430	0.719	0.435

Note: The acronyms are as follows Term Frequency-Inverse Document Frequency (TF-IDF); Logistic Regression (LR); Support Vector Machines (SVM); Decision Tree Classifier (DT); (KNN) k-Nearest Neighbours. To accommodate the small imbalances in the data the above measures are in macro average meaning that they are computed by taking the arithmetic mean.
† The only metadata used is the title of the text.

Table 1: Results of running multiple models on the FakeNewsCorpus

professional journalists prior to entry (Wang, 2017, Zhou and Zafarani, 2020). It is therefore expected, that there are differences in both length and writing style in the datasets. This also means differences in what constitutes fake news and how our models detect these differences. Further, both the FakeNewsCorpus and LIAR dataset work with different non-binary classification scales turned binary for our purposes in this assignment. The different overlapping categories, and which labels we have classified as fake news might also account for some of the difficulties in matching prediction on the datasets. We can further question forcing the datasets into a binary classification. As (Oshikawa et al., 2018) argues it is a "fatal error" if a classifier predicts True news as False, but it is a much smaller error predicting True news as Mostly True. A road to progress on cross-domain detection might therefore be in more detailed classification as this becomes more practical and achievable. (Oshikawa et al., 2018)

5.2 Overall lessons learned during the project

We identify the main factors we could change for difference (and possible progress) in performance as (i) Using even more data and more balanced datasets, (ii) making changes to the way we classify the categories of the datasets into a binary fake/reliable classification, and (iii) training the model on more diverse types of articles.

As for (i), in the end we used a dataset of 75.000 data points to train the model (60.000 for training and 15.000 for tests). With better access to advanced computers there is even more data in the FakeNewsCorpus to include. However, our performance on the FakeNewsCorpus suggests that this is a sufficient amount of data to train on for this task avoiding both underfitting and overfitting. The main issue with the dataset is of course the considerable imbalance between fake news and reliable news. We had a few options to adjust for this. For instance we could have chosen to include the WikiNews articles in the training data. This would only change the balance a few percent, however, and in the wrong direction making the share of non-fake articles greater. However, for more general purpose fake news detection this might be a good set to include, maybe also improving the performance on the LIAR dataset (see point (iii)). (ii) We could also change the distributions of fake/reliable news by doing the binary classification differently, e.g. including more categories as fake aside from the categories fake, conspiracy, unreliable and junksci and exploring the discrepancy in performance on the LIAR dataset further. Whether the chosen categories are reasonable is another discussion. It would also be worth analysing more in depth if our binary-classification maps well to the categories of false and pants-on-fire we have chosen as "fake news" from the LIAR set. Extending our aim of the fake news classification to a general purpose model we should also (iii) include more diverse kinds of articles in our training data. We mean this in the sense of the nine requirements for fake news detection corpus introduced in section 1.2. Texts can have different lengths, style, time frames and manner of news delivery. Different languages and cultures also comes with inherent contextual differences. A homogeneity on all such statistics in the corpus is important, and cross-domain detection can only be successful if our model is trained on a dataset representative for all the domains we want it to detect fake news on. (Oshikawa et al., 2018) Zooming out in this way we might consider the FakeNewsCorpus as only one datapoint in a coordinate system out of conceivable thousands of datapoints each containing an entire domain-specific dataset. In this picture, obviously it is not possible to train a general purpose model on just one datapoint. However, the same analogy of doing a best fitting line through multiple datasets also explain why this is bound to be a trade-off in performance on the domain-specific datasets. To end the discussion and conclude on the task of making a predictive fake news model based on the FakeNewsCorpus, in that specific domain we achieved good performance equivalent to state of the art levels of 90% as demonstrated in (Janicka et al., 2019).

References

Maria Janicka, Maria Pszona, and Aleksander Wawer. Cross-domain failures of fake news detection. *Computación y Sistemas*, 23(3):1089–1097, 2019.

Z Khanam, BN Alwasel, H Sirafi, and M Rashid. Fake news detection using machine learning approaches. In *IOP Conference Series: Materials Science and Engineering*, volume 1099, page 012040. IOP Publishing, 2021.

Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. URL `probml.ai`.

Jamal Abdul Nasir, Osama Subhani Khan, and Iraklis Varlamis. Fake news detection: A hybrid cnn-rnn based deep learning approach. *International Journal of Information Management Data Insights*, 1(1):100007, 2021.

Ray Oshikawa, Jing Qian, and William Yang Wang. A survey on natural language processing for fake news detection. *arXiv preprint arXiv:1811.00770*, 2018.

Victoria L Rubin, Yimin Chen, and Nadia K Conroy. Deception detection for news: three types of fakes. *Proceedings of the Association for Information Science and Technology*, 52(1):1–4, 2015.

Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1):83–112, 2017.

Kai Shu, Amy Sliva, Suhang Wang, Jiliang Tang, and Huan Liu. Fake news detection on social media: A data mining perspective. *CoRR*, abs/1708.01967, 2017. URL `http://arxiv.org/abs/1708.01967`.

Kai Shu, Deepak Mahudeswaran, Suhang Wang, Dongwon Lee, and Huan Liu. Fakenewsnet: A data repository with news content, social context, and spatiotemporal information for studying fake news on social media. *Big data*, 8(3): 171–188, 2020.

Amila Silva, Ling Luo, Shanika Karunasekera, and Christopher Leckie. Embracing domain differences in fake news: Cross-domain fake news detection using multi-modal data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 557–565, 2021.

S.S. Skiena. *The Data Science Design Manual*. Texts in Computer Science. Springer International Publishing, 2017.

William Yang Wang. "liar, liar pants on fire": A new benchmark dataset for fake news detection. *CoRR*, abs/1705.00648, 2017. URL `http://arxiv.org/abs/1705.00648`.

X Zhou and R Zafarani. A survey of fake news: Fundamental theories. *Detection Methods, and Opportunities*, 2020.

Arkaitz Zubiaga, Maria Liakata, and Rob Procter. Exploiting context for rumour detection in social media. In *International conference on social informatics*, pages 109–123. Springer, 2017.

Appendix A: Vectorizers

Count Vectorization and n-grams

One of the vectorizers we make use of is the count vectorizer. Let x_{nt} be a token/word in location t in document n. Let the dictionary D be the set of unique tokens in the text such that all the tokens in the dictionary compose the vocabulary of the corpus. Then the n'th document can be represented as a D-dimensional vector $\tilde{\mathbf{x}}_n$ where $\tilde{x}_n$ is the number of times a word v occurs in n such that

$$\tilde{x}_n = \sum_{t=1}^{T} \mathbb{I}(x_{nt} = v)$$

with T being the length of document n. The process can be illustrated as in Figure 3. Three documents are given to the vectorizer which created a dictionary and created a $D \times N$ matrix for the number of features D and number of documents N.(Murphy, 2022)

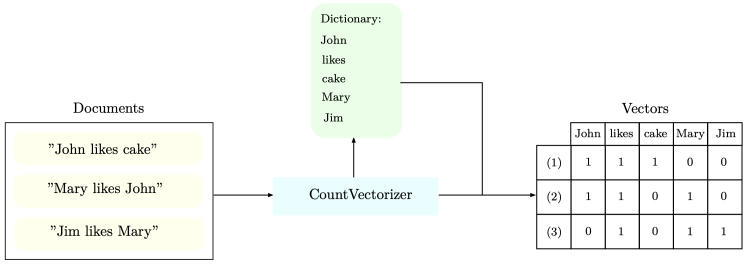

Figure 3: Illustration of count vectorization

The figure above shows the situation where the dictionary is created using a unigram i.e. a single token. Instead of using just a single token it is also possible to use strings of tokens that follow each other. That is, instead of using the single tokens "John", "likes" and "cake" to create the dictionary it is created by the subsequences "John likes" and "likes cake" instead if using a bigram. This process is also known as n-grams where n signify the number of tokens which have to be included for each entry in the dictionary.

Term Frequency-Inverse Document Frequency (TF-IDF)

Using the word count vector to represent documents come with the drawback that frequent words can have a large influence because the word count for these are higher than non-frequent words even though the non-frequent words may carry a higher semantic meaning. A solution to this problem is to take the logarithm to the word count as to reduce the impact of words that occur many times within a document. To reduce the impact of words that occur many times across all documents we use the inverse document frequency which is defined as $IDF_i = \log \frac{N}{1+DF_i}$ where DF_i is the number of documents with term i. (Murphy, 2022) Combining the two transformations gives:

$$TF\text{-}IDF_{ij} = \log(TF_{ij} + 1) \cdot IDF_i$$

The figure below shows how this process unfolds. First the term and inverse document frequencies are calculated and then combined to the TF-IDF vectorization.

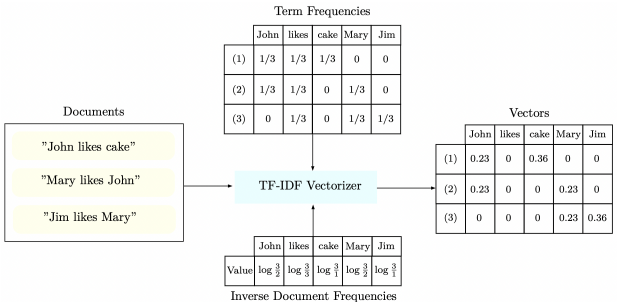

Figure 4: Illustration of TF-IDF vectorization.

Note that in Figure 4 we have for simplicity's sake defined the TF as

$$TF = \frac{\text{frequency of term } t \text{ in document } d}{\text{total number of terms in the document}}$$

and the IDF as

$$IDF = \log \frac{\text{total number of documents}}{\text{number of documents containing the term } t}$$

Appendix B: Classifyers

Logistic Regression (LR)

Logistic regression is a commonly used model for classification. In our models we have used binary logistic regression as we only have two classes. Binary logistic regression is defined as

$$p(y|\mathbf{x};\boldsymbol{\theta}) = \text{Ber}(y|\sigma(\mathbf{w}^T\mathbf{x}+b))$$

where $x \in \mathbb{R}^D$ is a D-dimensional input vector, $y = \{0,1\}$ is the class label, σ is the sigmoid function, $\mathbf{w}$ are the weights, b is the bias and $\boldsymbol{\theta}(\mathbf{w}, b)$ are the parameters. It follows that

$$p(y=1|\mathbf{x};\boldsymbol{\theta}) = \sigma(a) = \frac{1}{1+e^{-a}}$$

where $a = \mathbf{w}^T\mathbf{x}+b$ is equal to $\log(\frac{p}{1-p})$ for $p = p(y=1|\mathbf{x};\boldsymbol{\theta})$. (Murphy, 2022)

For simplicity we will only look at the linear classifyer in this appendix. We encourage the reader to read chapter 10 in (Murphy, 2022) for a more in depth walkthrough. The sigmoid function in the above gives the probability of the class label being $y = 1$. If the loss for making a wrong classification for both classes are the same then it would be optimal to predict $y = 1$ if class 1 is more likely than class 0. It follows that

$$f(\mathbf{x}) = \mathbb{I}(p(y=1|\mathbf{x}) > p(y=0|\mathbf{x})) = \mathbb{I}\left(\log\frac{p(y=1|\mathbf{x})}{p(y=0|\mathbf{x})} > 0\right) = \mathbb{I}(a>0)$$

where $a = \mathbf{w}^T\mathbf{x}+b$. The prediction function follows such that

$$f(\mathbf{x};\boldsymbol{\theta}) = b + \mathbf{w}^T\mathbf{x}$$

This function creates a linear hyperplane that separates the $\mathbb{R}^D$ space such that the plane creates a decision boundary. If the linear hyperplane separates the two classes without making any errors in classification on the training set the data is linearly separable.(Murphy, 2022)

To find the weights $\mathbf{w}$ the logistic regression we have to optimize the loss by minimizing it. The optimization problem we would like to solve is

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\text{argmin}}\ \mathcal{L}(\mathbf{w})$$

where $\mathcal{L}(\mathbf{w})$ is the log-loss function which in this case is

$$-\frac{1}{N}\sum_{n=1}^{N}(y_n \log\mu_n + (1-y_n)\log(1-\mu_n))$$

where $\mu_n = \sigma(a_n)$ is the probability of class 1 and $a_n = \mathbf{w}^T\mathbf{x}_n$.(Murphy, 2022) We use the stochastic average descent method for optimizing the loss. We use stochastic average descent as it is faster than other solvers for large datasets especially when the number of samples and the number of features are large.(Schmidt et al., 2017) For a deeper understanding of the stochastic average descent method we refer to (Schmidt et al., 2017).

Support Vector Machines (SVM)

Similarly to logistic regression a support vector machine constructs a set of hyperplanes which can be used as a decision boundary for classification. A good separation is achieved by the hyperplane which has the largest distance to to the nearest data points for any given class. It follows that the larger the margin the lower the error of the classifyer. Figure 5 shows an example of margin boundaries (dashed lines) which are called support vectors. If the a problem is not linearly separable then the support vectors are equal to the data points within the margin boundaries. In the following we assume that the class labels are equal to $y = \{-1,1\}$ and not $y = \{0,1\}$ as before and denote the target labels as $\tilde{y}$. (Murphy, 2022)

In the following consider the binary classifyer $\text{sign}(\mathbf{w}^T\mathbf{x} + w_0)$. We want to maximize this such that it is correct for most samples. The support vector classifyer has to solve the following problem

$$\min_{\mathbf{w},w_0,\xi}\frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N}\xi_n \ \text{ s.t. } \ \xi_n \geq 0, \ \ \tilde{y_n}(\mathbf{x}_n^T\mathbf{w}+w_0) \geq 1-\xi_n$$

This is also called a primal problem as it has $N + D + 1$ variables subject to N constraints. We have to maximize the margin, by minimizing $||\mathbf{w}||^2 = \mathbf{w}^T\mathbf{w}$ (i.e. maximizing $1/||\mathbf{w}||$ is equal to minimizing $||\mathbf{w}||^2$), while getting penalties when a sample is misclassified or within the margin boundary. Ideally, the value $\tilde{y_n}(\mathbf{x}_n^T\mathbf{w} + w_0)$ would be ≥ 1 for all samples which would mean that there is perfect prediction. Here C is the parameter which determines how many points are allowed to violate the margin constraint. (Murphy, 2022)

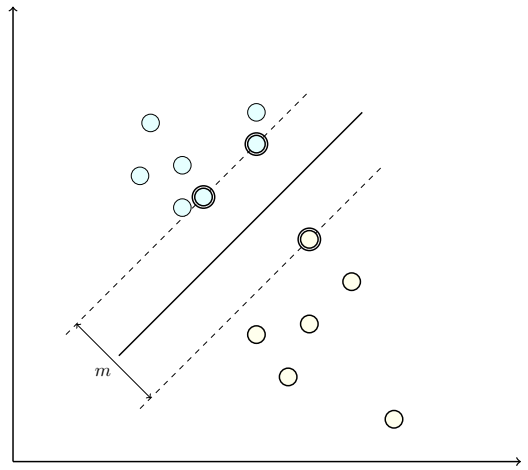

Figure 5: Illustration of a Support Vector Machine trained with samples from two classes. Samples on the margin m are called the support vectors.

For every primal problem it is possible to derive a dual problem. The dual problem to the primal problem is

$$\max_{\boldsymbol{\alpha}} \left(-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j\tilde{y}_i\tilde{y}_j\mathbf{x}_i^T\mathbf{x}_j + \sum_{n=1}^{N}\alpha_n \right) \quad \text{s.t.} \quad \sum_{n=1}^{N}\alpha_n\tilde{y}_n = 0,\ 0 \leq \alpha_n \leq C \text{ for } n = 1..N$$

where $\mathbf{x}_i^T\mathbf{x}_j$ is the kernel method. In our case we have chosen a linear kernel. If $\alpha_n = 0$ then the sample point is ignored. If $0 < \alpha_n < C$ then $\xi_n = 0$ i.e. the sample point lies on the margin. If $\alpha_n = C$ then the point is inside the margin and if $\xi_n \leq 1$ then the point is correctly classified and if the opposite $\xi_n > 1$ is true the point is misclassified. (Murphy, 2022)

The prediction function of the model is

$$f(\mathbf{x}, \hat{\mathbf{w}}, w_0) = \hat{\mathbf{w}}^T\mathbf{x} + \hat{w}_0 = \sum_{n\in\mathcal{S}}\alpha_n\tilde{y}_n\mathbf{x}_n^T\mathbf{x} + \hat{w}_0$$

where $\hat{w}_0$ is the bias. (Murphy, 2022)

Decision Tree Classifier (DT)

Given training vectors $\mathbf{x}$ and a label vector y a decision tree recursively partitions the feature space such that the samples with the same labels or similar target values are grouped together. The tree is constructed such that it consists of a sequence of decision rules. At each node i the feature dimension d_i for the input $\mathbf{x}$ is compared to a specific threshold t_i. If d_i is above or below t_i then the input is either passed down to the right or down to the right of the tree. The leaves of the tree specifies the prediction such that if the input falls into a specific leaf after it has passed through the height of the tree the leaf that the input ends at decides its label. As an example given the input $\mathbf{x}$ the first node checks whether x_1 is less than some threshold t_1. If this is the case then it moves down the tree and the next node checks whether x_2 is less than some threshold t_2. If this is the case then it enters the left leaf node as seen in Figure 6 and thus get the label that that region is defined as having. (Murphy, 2022)

Following the example above the output for region 1 is equal to

$$w_1 = \frac{\sum_{n=1}^{N} y_n\mathbb{I}(\mathbf{x}_n \in R_1)}{\sum_{n=1}^{N}\mathbb{I}(\mathbf{x}_n \in R_1)}$$

Using this the tree prediction function can be defined as

$$f(\mathbf{x};\boldsymbol{\theta} = \sum_{j=1}^{J} w_j \mathbb{I}(\mathbf{x} \in R_j)$$

where R_j is the j'th node region defined by the feature dimensions that are used at each split, w_j is the predicted output for the node and $\boldsymbol{\theta} = \{(R_j, w_j) : j = 1..J\}$, where J is the number of nodes. (Murphy, 2022)

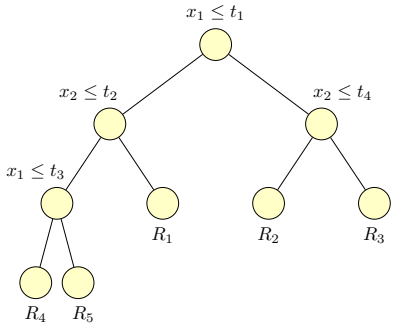

Figure 6: Illustration of a decision tree.

The loss function that has to be minimized is given by

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \ell(y_n, f(\mathbf{x}_n;\boldsymbol{\theta})) = \sum_{j=1}^{J} \sum_{\mathbf{x}_n \in R_j} \ell(y_n, w_j)$$

As this is not differentiable because of the discrete structure of the tree a greedy approach has to be taken which grow a tree one node at a time. Suppose that we are in node i such that $\mathcal{D}_i = \{(\mathbf{x}_n, y_n) \in N_i\}$ is the set of samples that has reached this node. The aim now is to split the set into a left and right branch such that the error in each subtree is minimized. Assume that the j'th feature is in $\mathbb{R}$ then it is possible to split the set by comparing j to a threshold t. Let the set of possible thresholds be $\mathcal{T}_j$. For each threshold in $\mathcal{T}_j$ the left and right partition can be defined as $\mathcal{D}_i^L(j,t) = \{(\mathbf{x}_n, y_n) \in N_i | x_{n,j} \leq t\}$ and $\mathcal{D}_i^R(j,t) = \{(\mathbf{x}_n, y_n) \in N_i | x_{n,j} > t\}$. Once $\mathcal{D}_i^L(j,t)$ and $\mathcal{D}_i^R(j,t)$ have been found for each j and t in node i the best feature j_i and threshold value t_i can be determined such that all coming inputs will be assessed from these. It follows that

$$(j_i, t_i) = \arg \min_{j \in \{1,\ldots,\mathcal{D}\}} \min_{t \in \mathcal{T}_j} \left(\frac{|\mathcal{D}_i^L(j,t)|}{|\mathcal{D}_i|} c(\mathcal{D}_i^L(j,t)) + \frac{|\mathcal{D}_i^R(j,t)|}{|\mathcal{D}_i|} c(\mathcal{D}_i^R(j,t)) \right)$$

The cost function $c(\mathcal{D}_i)$ which assesses the cost at node i can be computed by first finding the empirical distribution over labels for node i and then computing the Gini index afterwards. The empirical distribution is given by

$$\hat{\pi}_{ic} = \frac{1}{|\mathcal{D}_i|} \sum_{n \in \mathcal{D}_i} \mathbb{I}(y_n = c)$$

Following from this the Gini index is given by

$$G_i = \sum_{c=1}^{C} \hat{\pi}_{ic}(1 - \hat{\pi}_{ic}) = \sum_c \hat{\pi}_{ic} - \sum_c \hat{\pi}_{ic}^2 = 1 - \sum_c \hat{\pi}_{ic}^2$$

where $\hat{\pi}_{ic}$ is the probability of a random entry belongs to class c. This means that the Gini index is equal to the expected error rate. (Murphy, 2022)

k-Nearest Neighbours (KNN)

To classify an input $\mathbf{x}$ using the k-Nearest Neighbour classifier we find the K closest sample points to $\mathbf{x}$ in the training data, denoted $N_K(\mathbf{x}, \mathcal{D})$. When the K closest points have been found we then look at their labels and find a distribution of labels for the subregion around $\mathbf{x}$. This can be computed using

$$p(y = c|\mathbf{x}, \mathcal{D}) = \frac{1}{K} \sum_{n \in N_K(\mathbf{x}, \mathcal{D})} \mathbb{I}(y_n = c)$$

whereby the distribution of the majority label is returned. (Murphy, 2022)

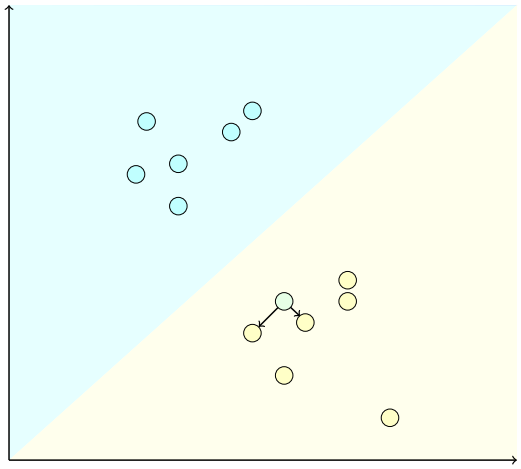

Figure 7: Illustration of a k-Nearest Neighbour classification with $k = 2$.

As seen in the figure above the two closest points to the green point, which in this case is **x**, both have the *yellow* label which given the above would make the model predict with probability 1 that the green dots labels is also *yellow*. Similarly if the green dot had three closest neighbours where two were *blue* and one was *yellow* the model would predict the green dot to have the *blue* label as it would have a $\frac{2}{3}$ probability of being *blue*.

Appendix C: SQL queries

```
SELECT value,
COUNT(meta_keyword) AS count
FROM meta_keyword
NATURAL JOIN meta_keywords
NATURAL JOIN article
GROUP BY value
ORDER BY count DESC
LIMIT 10;
```

```
         value          | count
------------------------+--------
  null                  | 547847
  '                     |   5612
  'vdare.com'           |   5296
  'Big Government'      |   5206
  'Donald Trump'        |   4321
  'National Security'   |   3929
  'Politics'.           |   3788
  'Russia'              |   3533
  'News'                |   3031
  'blog'.               |   2955
```

```
SELECT value,
COUNT(domain) AS count
FROM domain
NATURAL JOIN domains
NATURAL JOIN article
GROUP BY value
ORDER BY count DESC
LIMIT 10;
```

```
        value         | count
----------------------+--------
 beforeitsnews.com    | 106412
 dailykos.com         | 100588
 express.co.uk        |  44096
 wikileaks.org        |  27961
 abovetopsecret.com   |  23462
 shadowproof.com      |  19753
 sputniknews.com      |  19399
 breitbart.com        |  18358
 dailycaller.com      |  12092
 rawstory.com         |  10592
```

Listing 1: SQL-queries showing the 10 most common `meta_keywords` (LHS) and the 10 most common `domains` (RHS) in the FakeNews database

```
SELECT AVG(LENGTH(content)) FROM
    article;

SELECT article_id, LENGTH(content)
FROM article
ORDER BY LENGTH(content)
DESC LIMIT 1;

SELECT article_id, LENGTH(content)
FROM article
ORDER BY LENGTH(content)
ASC LIMIT 1;
```

```
           avg
-----------------------
 3590.5336776040453070

 article_id | length
------------+--------
     289492 | 100000

 article_id | length
------------+--------
       9874 |     41
```

```
SELECT value,
COUNT(tag) AS count
FROM tag
NATURAL JOIN tags
NATURAL JOIN article
GROUP BY value
ORDER BY count DESC
LIMIT 10;
```

```
                value          | count
-------------------------------+-------
  View Tags                    | 27534
  Russia                       | 11799
  American Pravda Articles     |  9787
  Race/IQ Articles             |  9785
  Race/Crime Articles          |  9534
  Donald Trump                 |  9007
  9/11 Articles                |  7658
  Syria                        |  6184
  Israel                       |  5584
  Sylvester Stallone           |  5267
```

Listing 2: SQL-queries showing the average length of the articles, the `article_id` of the longest and shortest article and their respective lengths (LHS) and the 10 most common `tags` (RHS) in the FakeNews database

```
SELECT name,
COUNT(name) AS count
FROM author
NATURAL JOIN authors
NATURAL JOIN article
GROUP BY name
ORDER BY count DESC
LIMIT 100;
```

```
       name        | count          name       | count           name      | count
-------------------+------  -------------------+------  -------------------+-------
  Created_At       | 81681   Posted On          | 68145    About The Author| 15556
  N_Comments       | 81681   beforeitsnews.com| 31627    Sarah Rumpf      | 11720
  Popular Tags     | 81681   wikileaks.org      | 27961   Michael Snyder   | 10752
  N_Stories        | 81681   abovetopsecre[..]| 23462   About The Author |  9340
  Comment Count    | 81681    Http              | 20142   wikispooks.com    |  9328
  Story Count      | 81681   sputniknews.com.  | 18768    Paul Kersey      |  8020
  Nickname         | 81681    Posted On         | 17541    Susan Wright     |  6305
  Joined           | 81681   Happy Cog Stu[..]| 17258   katehon.com       |  6242
 Backgroundurl[..]| 81678    Daily Kos         | 17258    The Saker        |  6184
  Showtags Pop[..]| 81678    Www.Happycog.Com| 17258   shadowproof.com.  |  6019
```

Listing 3: SQL-query showing the 30 most common author names in the FakeNews database

```
SELECT AVG(LENGTH(content)) FROM
    article;

SELECT article_id, LENGTH(content)
FROM article
ORDER BY LENGTH(content)
DESC LIMIT 1;
```

```
          avg
-----------------------
 2611.8073248407643312



 article_id | length
------------+----------
       1538 |  35011
```

```
SELECT extract(YEAR FROM
    publishing_date),
COUNT(*) AS count
FROM article
WHERE publishing_date IS NOT NULL
GROUP BY extract
ORDER BY count DESC LIMIT 10;
```

```
 extract | count
---------+-------
    2006 |   241
    2005 |   210
    2007 |   191
    2008 |   141
    2009 |   105
    2010 |    95
    2011 |    59
    2012 |    46
    2014 |    24
    2017 |    20
```

Listing 4: SQL-queries showing the average length of the articles, the `article_id` of the longest article and its length (LHS) and the 10 years where most articles have been published (RHS) in the WikiNews database

```sql
SELECT meta_keyword,
COUNT(meta_keyword) AS count
FROM meta_keyword
NATURAL JOIN meta_keywords
NATURAL JOIN article
GROUP BY meta_keyword
ORDER BY count DESC
LIMIT 10;
```

meta_keyword	count
united states	202
president	163
prime minister	156
government	148
party	145
talk page	129
correction	98
people	96
police	89
tuesday	88

```sql
SELECT split_part(name, '/',3) "
    domain",
COUNT(*) AS count
FROM author
GROUP BY "domain"
ORDER BY count DESC LIMIT 10;
```

domain	count
news.bbc.co.uk	502
en.wikinews.org	230
www.nytimes.com	198
www.cnn.com	192
www.reuters.com	165
www.washingtonpost.com	150
www.guardian.co.uk	135
www.bbc.co.uk	128
www.google.com	107
english.aljazeera.net	97

Listing 5: SQL-queries showing the 10 most common `meta_keywords` (LHS) and the 10 most common `author` domains (RHS) in the WikiNews database

```sql
CREATE OR REPLACE VIEW public.advancedfakewikiview
    AS
    SELECT article.article_id,
    article.db,
    article.title,
    author.name AS author,
    domain.type,
    meta_keyword.value AS keyword,
    article.content
   FROM article
     JOIN authors ON article.article_id = authors.article_id
     JOIN author ON authors.author_id = author.author_id
     JOIN domains ON article.article_id = domains.article_id
     JOIN domain ON domains.domain_id = domain.domain_id
     JOIN meta_keywords ON article.article_id = meta_keywords.article_id
     JOIN meta_keyword ON meta_keyword.keywords_id = meta_keywords.keywords_id
UNION ALL
 SELECT article.article_id,
    article.db,
    article.title,
    author.name AS author,
    article.label AS type,
    meta_keyword.meta_keyword AS keyword,
    article.content
   FROM public2.article
     JOIN public2.authors ON article.article_id = authors.article_id
     JOIN public2.author ON authors.author_id = author.author_id
     JOIN public2.meta_keywords ON article.article_id = meta_keywords.article_id
     JOIN public2.meta_keyword ON meta_keyword.meta_keyword_id = meta_keywords.
         meta_keyword_id;
```

Listing 6: SQL-query creating a view of the FakeNews and WikiNews databases

```
SELECT type,
COUNT(domain) AS count
FROM domain
NATURAL JOIN domains
NATURAL JOIN article
GROUP BY type
ORDER BY count DESC
LIMIT 12;
```

```
    type     | count
-------------+--------
 political   | 206012
 fake        | 110614
 bias        |  93205
 conspiracy  |  61403
 rumor       |  44096
 unknown     |  33686
 unreliable  |  32537
 clickbait   |  19334
 junksci     |  11468
 satire      |   7513
 reliable    |   5222
 hate        |   3112
```

Listing 7: SQL-query showing distribution of labels in the FakeNews database

Appendix D: Scraping WikiNews

We create our own news dataset by scraping it from the web. We will be looking at the *Politics and Conflict* section of the Wikinews site and scrabing it using the python library *BeautifulSoup* and use *pandas* to ease the work with the labeled data.
The main packages we use to collect the data are:

- *Beautiful Soup* is a Python package for parsing HTML and XML documents. It creates a parse tree for parsed pages that we can use to extract data from HTML.
- *Pandas* is a Python package that provides fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive.
- *YAKE* (Yet Another Keyword Extractor) is a light-weight unsupervised automatic keyword extraction method which rests on text statistical features extracted from single documents to select the most important keywords of a text. If you have not installed yake on your machine then uncomment `!pip install yake` in the first line of code below.
- *re* (regex) which is a module that provides regular expression matching operations

Inspecting the Wikipedia News page it is inherent that there are a few items which occur on each news page i.e. headline; publishing date; text; sources. We will therefore focus on these four items and extract a dataset which contains a mapping of each of these categories.
We will start by collecting the link and headline for each news article in the assigned range of letters given our group number.
This is done by finding all the links in the parsed html that *BeautifulSoup* outputs and then appending the links to a list which in the end is converted to a pandas dataframe. Collecting all the links will make it possible for us to iterate through the list and collect the data from each of the newspages.
In this case we have split each action/addition to the dataframe into a its own problem. If we were to rewrite the code we would probably collect all the pieces within one loop such that each page is only parsed once and not every time we want to add another element. This would also lessen the running time of the code. But for the sake of understanding we have split the code up.
If you want to see how the dataframe looks after each codeblock has been executed then uncomment the code `display(df)` in the codeblock where you wish to see the dataframe after execution.

```
!pip install yake

import requests
from bs4 import BeautifulSoup
import re
import pandas as pd
import yake
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning) #removes future warnings regarding regex changes


choosing = "ABCDEFGHIJKLMNOPRSTUVWZABCDEFGHIJKLMNOPRSTUVWZ"[10%23:10%23+10]
choosing = list(choosing)

liste = []

for elem in choosing:
  response = requests.get('https://en.wikinews.org/w/index.php?title=Category:Politics_and_conflicts&from=' + elem)
  contents = response.text

  soup = BeautifulSoup(contents, 'html.parser')
  soup_findclass = soup.find_all('a', attrs= { "title" : re.compile(r"^" + elem)})

  for link in soup_findclass:
    linket = link.get('href')
    teksten = link.text

    tuple = (linket, teksten)
```

```
    liste.append(tuple)

df = pd.DataFrame(liste, columns = ['Link', 'Overskrift'])
df = df.loc[df['Link'].str.startswith('/wiki/')]

#display(df)
```

The date the article was published is important for the context of the news and thus a good reason to include in our dataset. We do this in the same way as above by looping through all the articles and finding the title of the parsed html with the "id" "publishDate" as this "id" is a container for the publishing date.

```
dates = []

for index, row in df.iterrows():
    url = 'https://en.wikinews.org' + row['Link']

    response = requests.get(url)
    contents = response.text

    soup = BeautifulSoup(contents, 'html.parser')

    try:
         soup_finddate = soup.find('span', attrs= { "id" : "publishDate"})["
            title"]
    except TypeError:
        soup_finddate = "1900-01-01"

    dates.append(soup_finddate)

df.insert(2, "Udgivelsesdato", dates, True)

#display(df)
```

The sources the article was written on the basis of is important for verification and understanding. The sources are thus a good reason to include in our dataset. We do this in the same way as above by looping through all the articles and finding the link contained in the "class" "external text" as this "class" is a container for the external sources. If no link is given in the sources we do not deem the source relevant as it is not possible to falsify/verify. Following this reasoning such sources are not included.
Furthermore this is quite important if we want to assess if this is a trusted newssource. If the text itself is constructed from trusted newssources then we may assume that the text itself is trustworthy. If the sources are not credible sites then it may be the case that the text is not credible. Having the news sources thus gives of the opportunity to label the data with a trusted/not trusted label if we consider each of the pages that the news is compiled from.

```
source_link = []

for index, row in df.iterrows():
    url = 'https://en.wikinews.org' + row['Link']

    response = requests.get(url)
    contents = response.text

    soup = BeautifulSoup(contents, 'html.parser')

    try:
         soup_link = [a['href'] for a in soup.find_all('a', attrs = {"class" : "
            external text"}, href=True)]
    except TypeError:
        soup_findsource = "None"

    source_link.append(soup_link)

df.insert(3, "Kilder", source_link, True)

#display(df)
```

The text of the article is important as it gives us a deeper understanding of the event that has happened and is thus essential for the dataset. We add the text by using the same course of action as above by looping through all the articles and finding all the paragraphs of the parsed html. Furthermore we remove some of the standard sentences WikiNews includes as paragraphs in the html as well i.e. "Have an opinion on this story? Share it!". These sentences are not relevant for later analysis of the text and are thus removed.

```
text = []

for index, row in df.iterrows():
    textString = ""
    url = 'https://en.wikinews.org' + row['Link']

    response = requests.get(url)
    contents = response.text

    soup = BeautifulSoup(contents, 'html.parser')

    try:
         for para in soup.find_all("p"):
            textString = textString + (para.get_text())

         soup_findtext = textString
    except TypeError:
        soup_findtext = "None"

    text.append(soup_findtext)

df.insert(4, "Text", text, True)

df['Text'] = df['Text'].str.replace('Have an opinion on this story? Share it!', '')
df['Text'] = df['Text'].str.replace('Share this:', '')
df['Text'] = df['Text'].str.replace('Public domain\n', '')
df['Text'] = df['Text'].str.replace('false\n', '')
df['Text'] = df['Text'].str.replace('This page is archived, and is no longer publicly editable.', '')
df['Text'] = df['Text'].str.replace('Please note that due to our archival policy, we will not alter or update the content of articles that are archived, but will only accept requests to make grammatical and formatting corrections.', '')
df['Text'] = df['Text'].str.replace('Note that some listed sources or external links may no longer be available online due to age.', '')
df['Text'] = df['Text'].str.replace('Got a correction? Add the template {{editprotected}} to the talk page along with your corrections, and it will be brought to the attention of the administrators.', '')

#display(df)
```

An easy way to get an overview of what a text contains is by using keywords. For this reason we have chosen to extract the most common keywords in each text using the YAKE package which is quite useful for this purpose. As described in the beginning YAKE is a light-weight unsupervised automatic keyword extraction method which rests on text statistical features extracted from single documents to select the most important keywords of a text. YAKE finds the statisitcal most important words in the text and creates a list of these words. This list is then added to each of the rows in the dataframe for the row containing the relevant text YAKE has processed. This will make it easier for us to seach the database and to do subject analysis of the news if we wanated to.

```
df['Text_lower'] = df['Text'].str.lower()
df['Keywords'] = None

for index, row in df.iterrows():
    text = row['Text_lower']

    kw_extractor = yake.KeywordExtractor()

    language = "en"
```

```
    max_ngram_size = 3
    deduplication_threshold = 0.9
    numOfKeywords = 20

    custom_kw_extractor = yake.KeywordExtractor(lan=language, n=max_ngram_size,
        dedupLim=deduplication_threshold, top=numOfKeywords, features=None)
    keywords = custom_kw_extractor.extract_keywords(text)

    row['Keywords'] = keywords

for index, row in df.iterrows():
    firstElements = []
    tuple = row['Keywords']
    firstElements = [a_tuple[0] for a_tuple in tuple]
    row['Keywords'] = firstElements

df = df.drop(columns='Text_lower')

#display(df)
```

And finally saving the scraped articles in a .csv-file with all the attributes found

```
df.to_csv("wikinews.csv",index=False)
```