DATA SCIENCE - FINAL PROJECT

EXAM REPORT

Daniel Kjeldsen
vsq693

Emil Nielsen
rjq923

David Folting
hfc635

June 9, 2022

Part 1 - Know your data

Representation

We designed our database with the purpose of it being consistent and easily maintainable. In the database each many-to-many relationship was divided into two one-to-many relationships. The transitive functional dependency from `domain` over `url` to `id` is removed from *article* by adding the *url* table. We added an `id` to the *author* table, because an author might change his name, which should be easy to do, or we might add another author with the same name as an existing author, and we want to be able to distinguish between those. Now, having `id`'s for *article* and *author*, we did the same for *keyword*, *meta_keyword* and *tag* in order to keep our database consistent and allow for easy changes to those entities. Also, we divided `url` into `sld`(second-level domain), `tld` (top-level domain), `slug` and `ssl` (which is 'yes' if the website has en SSL certificate and 'no' otherwise). We end up with the following database design:

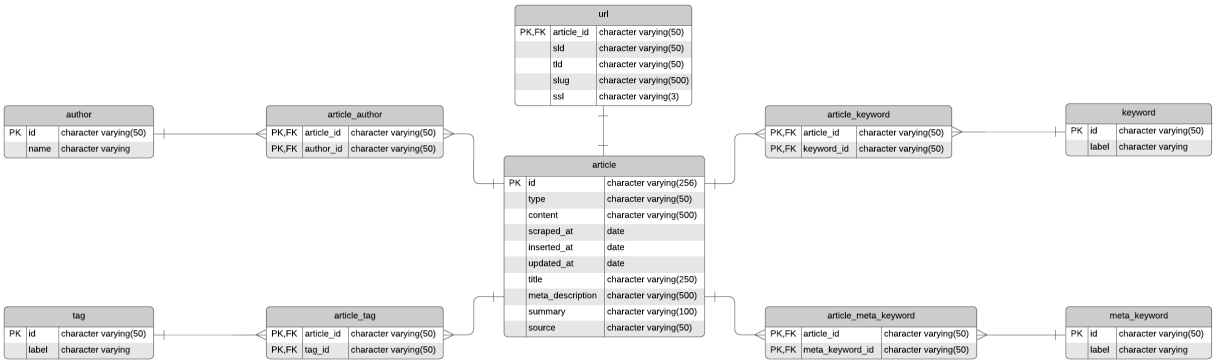

Figure 1: FakeNewsCorpus - ER Diagram

Problems

Looking at the dataset we identified a high amount of missing data. When columns had only sporadic entries or were empty we replaced the entries with `null`. We encountered a lot of commas that were not meant to separate entries which also caused some problems. Also, the representation of dates in `content` were typically different from one article to another.

Key Properties

The most common `type` of articles was 'political' followed by 'bias', 'fake' and 'unreliable' (Figure 2). Looking into the distribution of what `slds` where most responsible for articles categorized as 'fake', we observed that 'beforeitnews' accounted for an extremely high proportion (Figure 3). We also looked into what kind of articles the different authors

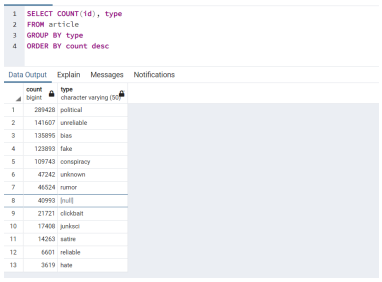

Figure 2: Distribution over types

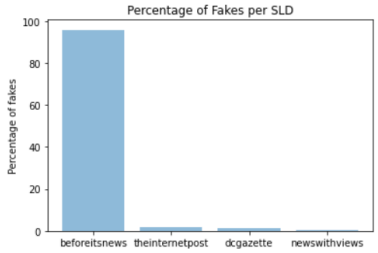

Figure 3: Fake distribution over sld

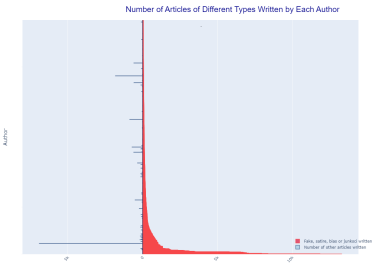

Figure 4: Articles by authors

wrote (see Figure 4 where the red lines are number of articles considered to be fake news and blue lines are all other articles). The pattern showed that authors either wrote what we defined as being fake or other types, but very rarely both. Both of these discoveries indicate that our meta-data contains important information correlating with the `type` of the article. We also discovered that some of the columns were not useful because they either were empty or did not vary at all (Figure 5, 6, 7).

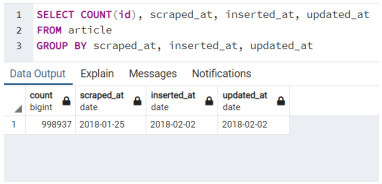

Figure 5: Dates query

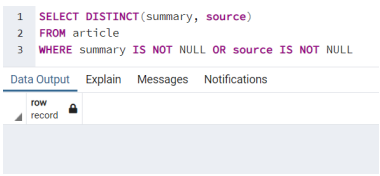

Figure 6: Summary, source query

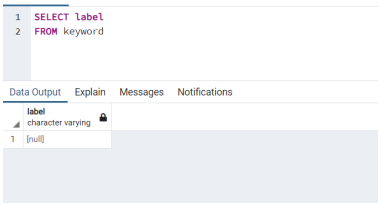

Figure 7: Keyword query

Experiences with WikiNews

We had some problems with handling the data. The first part of the problem was technical and dealt with separating the data. The second part concerned what meta-data we should scrape and how to categorize it. We quickly decided that `published` (date), `title` and `domain` were relevant. `content` was harder to pin down since not all WikiNews articles (called *pap_article* for "politics-and-conflicts article" in our database - yes, wrongly abbreviated) had the same internal structure, which made it harder to capture a type of logic that makes us able to decompose `content` further. We therefore stored all scraped text from each article in their respective `content` entry.

ER Diagram and Statistics

We scraped the following meta-data; `published`, `date`, `title`. If we had the time we would have created an additional meta-data field called `keywords`, which would consist of the ten most common words in the articles - after the data had been cleaned. In total we have 1212 articles. Below is an ER diagram of our simple database design (Figure 8). We have inserted a surrogate key as `id` to keep consistency with the FakeNewsCorpus design. Also, we queried the database showing the number of articles (Figure 9), and the distribution over the years the articles were published (Figure 10).

pap_article		
PK	id	integer
	title	character varying(100)
	published	date
	content	character varying(500)

Figure 8: WikiNews - ER Diagram

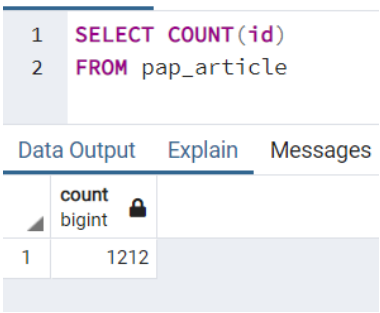

Figure 9: Entry count

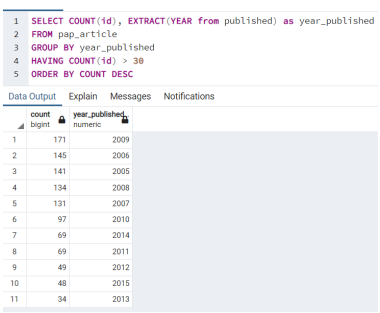

Figure 10: Distribution over years

Combined View and Choice of Data

When integrating data from the FakeNewsCorpus dataset and the WikiNews dataset into a single view we choose the columns that are shared by both datasets (`id`, `title`, `content`). Naturally, we also included `type`, and because that column does not exist in the Wikinews dataset, we set the `type` value for all articles from Wikinews to 'political'. Lastly, we also included a column called `dataset` because articles from different datasets might have overlapping `ids` and we wanted to be able to distinguish between those.

```
CREATE VIEW all_news AS
    SELECT CAST(id AS INT), title, content, type, 'FakeNewsCorpus' AS dataset
    FROM article
    UNION ALL
    SELECT id, title, content, 'political' AS type, 'WikiNews' AS dataset
    FROM pap_article
```

Data Output | Explain | Messages | Notifications

```
CREATE VIEW

Query returned successfully in 48 msec.
```

Figure 11: Creating our view

```
SELECT id, type, dataset, title
FROM all_news
ORDER BY id
LIMIT 10
```

Data Output | Explain | Messages | Notifications

	id integer	type character varying	dataset text	title character varying
1	1	political	WikiNews	civil defence thwarts israeli air strike on gaza refugee camp
2	2	political	WikiNews	188 fatah supporters permitted to enter israel
3	2	rumor	FakeNewsCorpus	is life an illusion researchers prove reality doesn t exist if you re not looking at it
4	3	political	WikiNews	arafat memorial ceremony in gaza cancelled following attacks on fatah leaders
5	4	political	WikiNews	audio tape released of missing israeli soldier
6	5	political	WikiNews	bbc reporter alan johnston is released in gaza
7	6	political	WikiNews	bbc reporter could be released within next few hours
8	6	hate	FakeNewsCorpus	donald trump
9	7	political	WikiNews	beirut car bomb targets hamas official
10	7	hate	FakeNewsCorpus	donald trump

Figure 12: Querying our view

We decided to not merge the data after a long discussion. In favor of merging was the idea that more data, especially data we had a rather certain knowledge of being reliable, would allow a more finely-grained classification model. Also because the WikiNews are political in nature, which is the same as our most common article type. Against merging was the fact that, if we merged the dataset then an even higher portion of our total dataset would consist of articles of the 'political' `type`. This could make our predictor consider 'REAL' and 'political' to be too strongly correlated and then predict other types of articles as 'FAKE' even though they actually were 'REAL'.

Classification

We opted for using "$github.com/several27/FakeNewsCorpus$" as a reference. Some of the types were obvious and gave us an initial classification of 'FAKE' ('fake', 'conspiracy', 'satire') and 'REAL' ('reliable', 'political', 'clickbait'). That left us with 'hate', 'bias', 'junksci', 'conspiracy' and 'unreliable'. Our dilemma was that even though some articles see the world differently or even has a negative opinion of for example race, in some instances it is more a debate and therefore a personal value question instead of a science question. We therefore decided to remove some of the most questionable categories and adopted a conservative approach by being more accepting of type 2 errors instead of type 1. We ended up classifying 'fake', 'conspiracy', 'satire', 'bias' and 'junksci' as 'FAKE', 'clickbait', 'political', 'reliable' and 'state' as 'REAL' and we considered 'unreliable' and 'hate' to be unclassifiable and removed all entries having one of those two types before going forward.

Part 2 - Establish a baseline

A baseline is a simple model that act as a reference point for our more advanced model later on. It contextualizes the results by serving as a benchmark for the more advanced model, but can also give valuable information about the data by seeing which type of baseline models respond best to the data. For our baseline models we chose monkey, logistic regression, naive Bayes and k-nearest neighbors and ran it on `content` only. We used only the default parameters to keep the baseline as simple as possible. We transformed `content` by stemming, removing stopwords and then used count vectorization. Finally, we split the data into train and test sets in a 80/20 split. We ran the models several times and consistently got results in the ranges in the table below.

Model	Accuracy
Monkey	53-55%
Naive bayes	76-78%
Logistic regression	84-86%
K-nearest neighbors	73-75%

Interpreting the scores, we saw the highest success rate for logistic regression, followed by naive bayes, then k-nearest neighbors, and finally monkey. This indicated that the data had some kind of relationship. In naive bayes the feature weights were set independently based on how much it correlated with the `type`, whereas in logistic regression the weights were set together. The difference between the two models appeared when some features correlated. If a feature recurred frequently and correlated with another feature, naive bayes would then give them both a strong weight, whereas logistic regression would compensate by weighting them together. K-nearest neighbors predicts using the euclidean distance between points, and the success rate of this algorithm indicated that there existed some geometrical separation between 'FAKE' and 'REAL' data. In our dataset we had switched out numbers in `content` for 'num', and same with dates etc. We therefore have a highly repetitive element which probably accounts for the difference in the baseline results.

We then considered possibilities for extending our baseline. According to our analysis (in Part 1) we had very strong correlation between `article.type` and `url.sld` and `author.name`. We thought adding these to the model would increase its predictive power. We conducted an extended baseline with these features added and got an accuracy of 94% for logistic regression, 89% for naive bayes and 85% k-nearest-neighbours, which is a clear improvement. The performance by monkey was 54% which corresponded to our previous results.

Part 3 - Create a Fake News predictor

We ended up creating a fake news predictor based on a support vector machine (SVM). Our baseline showed that logistic regression performed better than naive bayes, hinting that a linear method - such as SVM - could be beneficial. A SVM algorithm fits the best hyperplane to a binary classification problem. The reason we believed SVM to be an upgrade compared to logistic regression is due to the structure of our data. Where logistic regression assumes that the data has clear independent variables when fitting, the SVM is better suited for unstructured or semi-structured data. This is due to SVM being based on geometrical properties of the data compared to logistic regression that has a more statistical approach.

We conducted hyperparameter tuning on our predictor, but also on the baselines.

Model	Parameter	Parameter	Parameter	Accuracy
Naive bayes	Alpha = 1.0	Fit_prior = True		Score: 77-79%
Logistic regression	C = 0.1	Penalty = l1	Solver = lbfgs	Score: 85-87%
K-nearest neighbours	K = 10			Score: 76-78%
Support-vector machine	C = 1.0	Gamma = 1.0	Kernel = Linear	Score: 87-89%

All models improved with the hyperparameter tuning and the most optimal parameters for each model are showed in the table above. In regards to the SVM the kernel handled how the model was fitted to the data by either fitting linearly or non-linearly. The parameter C is concerned with regularization, which means defining a penalty associated with misclassification. Lastly is gamma, which concerns how tight or loose we fit the data to the training set.

Overall our more advanced model showed a slight increase in efficiency compared to logistic regression and a higher jump compared to naive bayes and k-nearest neighbors. To test whether our predictor were quantifiable better than our best baseline model we ran a unpaired t-test between logistic regression and SVM. The models were executed ten times each. Our initial null hypothesis was that the distributions did not vary significantly.

Statistics = 6.16634	Pvalue = 0.00027

We could therefore reject the null hypothesis which indicated that we had a true difference in our models.

Our two main options for a more advanced model was neural networks (NN) and SVM's. Both algorithms can in theory solve the same problems, by fitting linearly and non-linearly, and are well suited for text classification. We picked SVM's because they can reach the same results on smaller subsets of data compared to NN's, if there is a pattern of separation in the data. Running on more than 100.000 samples gave us exponentially higher running times. We simply did not have the time to run full NN's, tune them properly and then run enough trials to conduct proper t-tests. The high success rate from the baseline also supported our assumption of a pattern enabling us to run on a smaller subset.

Part 4 - Performance beyond the original data set

When exploring the LIAR dataset we used the columns, `label` and `statement`, because we believed that they strongly corresponded with the `type` and `content` columns from the FakeNewsCorpus dataset. We classified the entries in the `label` column the following way; 'true' and 'mostly-true' as 'REAL', 'false' and 'pants-fire' as 'FAKE' and we deemed 'barely-true' and 'half-true' to be unclassifiable.

We did not retrain our models but instead used the vocabulary of FakeNewsCorpus to create our vector for the LIAR dataset. We merged the 'train.tsv', 'valid.tsv' and 'test.tsv' files into a single file that we used for testing our pre-trained models. We ran a couple of trials and below are the results from one of the trials that we considered to be a 'typical' one.

Accuracy Scores		
Models	FakeNewsCorpus	LIAR
Naive Bayes	78.82%	54.01%
Logistic Regression	86.35%	54.46%
K-nearest neighbours	77.21%	44.23%
Support-Vector	87.85%	54.92%

It is clear that there is a massive difference between the performance on the FakeNewsCorpus dataset and on the LIAR dataset. No model has a performance on the FakeNewsCorpus dataset that is less than 20 percentage points more than on the LIAR dataset. It is especially noteworthy that all our accuracies on the LIAR dataset are below 56%. This is interesting because - using our classification - a little more than 56% of the entries in the LIAR dataset are considered 'REAL'. This means that a model outputting 'REAL' every single time would have a performance a little above 56% which is better than all of our models. We were actually able to get this result when limiting the size of our `CountVectorizer` but we did not consider a model giving only one output to be interesting.

Part 5 - Discussion

To be honest, our model had a miserable performance on the LIAR dataset. This could be caused by several things. Firstly, our model might have overfitted and thus not captured a more general truth about fake news. Avoiding overfitting when using SVM's is much about tuning parameters - especially the parameter C which deals with regularization. We picked the parameters in the model as those that gave the best result on the FakeNews training set. It could maybe have been better to go with a lower regularization parameter in order to have the model generalize better.

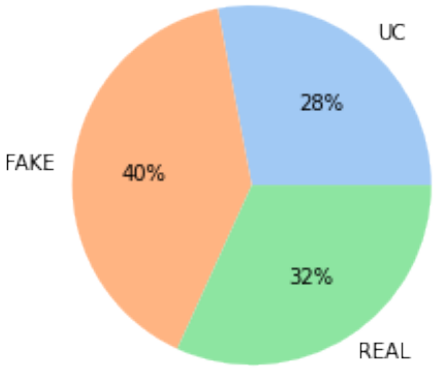

Figure 13: FakeNewsCorpus

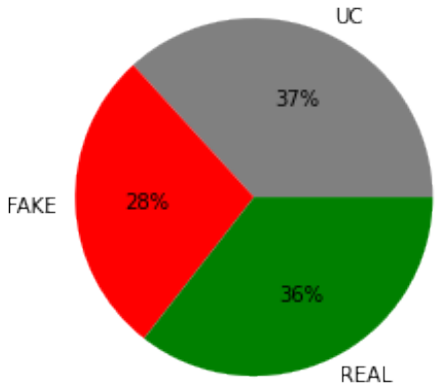

Figure 14: LIAR

Another reason might be the data itself. The LIAR dataset deals only with statements and not the content of an article. This means there are a lot fewer words and hence less to predict the data with. Again opting for a more generalize model might necessitate either using meta-data, or find a better way to make the compared data more similar. An idea

could be to preprocess the data such that certain keywords are extracted on all articles and those keywords are given a higher weight.

When classifying the LIAR dataset we noticed that the majority of entries were considered to be 'REAL' (after removing all unclassifiable), which is in contrast to the FakeNewsCorpus dataset where most of the entries were 'FAKE' (Figure 13 and 14). This is a rather interesting observation because all our models predicted most of the entries in LIAR to be 'REAL'. We hypothesised that URLs and emails would not be present in statements, and since our model mainly predicted 'REAL' for the LIAR dataset, then 'FAKE' articles would contain significantly more URLs and emails. Running a SQL query on the number of URL or emails present in 'FAKE' and 'REAL' showed our hypothesis did not hold, as the distribution between 'REAL' and 'FAKE' is relatively even (Figure 15).

```sql
SELECT COUNT(id), CASE
                    WHEN type IN ('fake', 'satire', 'junksci', 'bias', 'conspiracy') THEN 'FAKE'
                    ELSE 'REAL'
                  END AS classification,
                  CASE
                    WHEN content LIKE '%url%' OR content LIKE '%email%' THEN 'YES'
                    ELSE 'NO'
                  END AS contains_url_or_email
FROM article
WHERE type IN ('fake', 'satire', 'junksci', 'bias', 'conspiracy', 'political', 'clickbait', 'reliable', 'state')
GROUP BY classification, contains_url_or_email
```

Data Output | Explain | Messages | Notifications

	count bigint	classification text	contains_url_or_email text
1	372803	FAKE	NO
2	28399	FAKE	YES
3	286175	REAL	NO
4	31575	REAL	YES

Figure 15: Query on 'url' and 'email' in content by type

We expected our model to transfer badly to the LIAR dataset, but not to perform worse than just picking 'REAL' for all entries. This indicates that the quality of the data is too low or that there at least needs to be less discrepancy between using `statement` and `content`. Also, we thought that maybe the context of the statements in the LIAR dataset are mostly political, and since we trained our model to classify articles of 'political' type to be 'REAL' then it makes sense that when that model is used on the LIAR dataset, it almost only outputs 'REAL'.

Our predictor returned high accuracy scores for the FakeNewsCorpus dataset, and these improved when we included the chosen meta-data. However, our model did not transfer well to the LIAR dataset. This could be because our model was wrong or the data was faulty. The reason the model could be wrong did not seem to be the choice of SVM, but instead in how we preprocessed the data and how we selected types to be either 'FAKE' or 'REAL'. The LIAR dataset has a political touch with only `statement` whereas articles had `content` in the FakeNewsCorpus dataset. In the FakeNewsCorpus dataset we classified all political articles as 'REAL'. This could indicate that we operated with a different definition of being 'FAKE' in the two datasets. With that point in mind, our analysis is that the most obvious fault for the poor generalization is due to the data.

Conclusion

To summarize our experience working with this project, we started by designing our database to be clear and consistent and we only used data from the FakeNewsCorpus to train our models. The best baseline performance was $\sim$86% by logistic regression on the test part of FakeNewsCorpus and it dropped considerably on the LIAR dataset. The same can be said about our more advanced model, where we chose Support-vector machine. We came up with different hypothesises for the drop in performance and managed to disprove one. When using meta-data our baseline performed considerably better, so it is reasonable to believe that the same could apply to the LIAR dataset, although we never tested that. Finally, we concluded that the large performance drop was primarily because the difference in the data was too large and only using `statement` and `content` limited the performance. However, our classification might also have had a negative effect.