Fake news detection

Maxime Lavech, *May 2020*

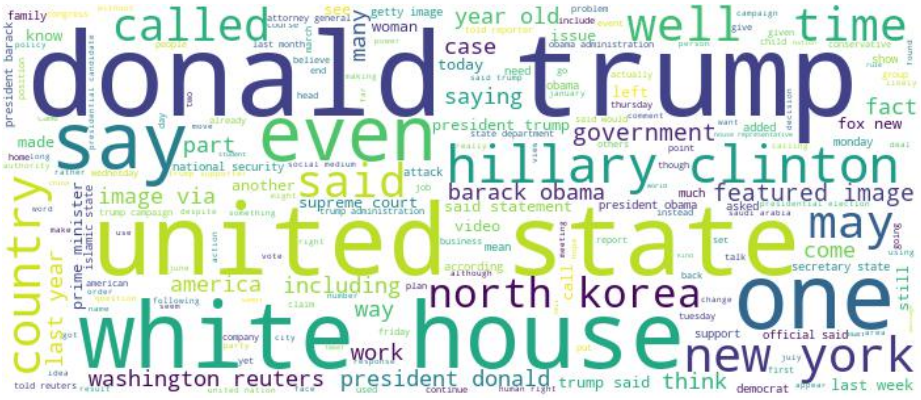

Figure 1: Word Cloud of a subset of preprocessed words in our data.

1 Background

In recent years, the quantity of information available as well as the means to access it has grown exponentially. Indeed, nowadays people have access to terabytes of data through a wide range of different sources. A non-exhaustive list would include online articles, journals, blog posts... Social media platforms have further streamlined the means of access and significantly increased the speed of transmission of the information. Further, the recent and active Coronavirus crisis has accentuated the proliferation of fake news.

Tracing sources and verifying the quality and reliability of this information has therefore become an ubiquitous task. Since tracing and weighting different sources based on reliability is not always a feasible task, can we given a news article and its corresponding metadata predict if the accessible data can be trusted or not. In other words, can we find a way of sampling out the noise and keeping only 'True' news articles?

2 Problem Statement

Given a news text and its corresponding and available metadata (title, topic, publication date), we want to be able to predict if the news sounds like a reliable source of information or not. Therefore, we want to deploy a binary classifier which given the mentioned input returns the probability of being fake. We want to train and deploy this model such that creating an endpoint with the latter

would require as little steps as possible. We provide in Figure 2 a simplified sketch of a possible deployment pipeline.

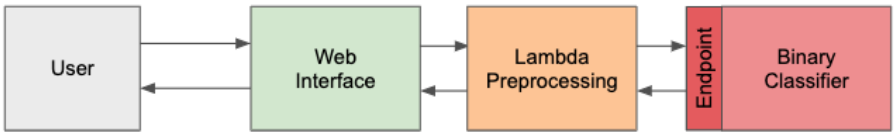

Figure 2: Deployment Pipeline Sketch.

Here the user provides a source text with available metadata at the interface. The data is processed through a lambda function and fed to the model. Predictions are then returned to the user via the web interface.

3 Dataset

3.1 Overview

The data was collected from Kaggle (`https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset`) [1] and provided by Clément Bisaillon who further acknowledged the following sources [2, 3]. The data was initially split into two **.csv* files each containing respectively the data for 'True' news and 'Fake' news. We added a label column to each dataframe and merged the data into a single dataframe. Table 1 features the first few rows of the augmented dataframe.

	title	text	subject	date	label
0	As U.S. budget fight looms, Republicans flip t...	WASHINGTON (Reuters) - The head of a conservat...	politicsNews	December 31, 2017	True
1	U.S. military to accept transgender recruits o...	WASHINGTON (Reuters) - Transgender people will...	politicsNews	December 29, 2017	True
2	Senior U.S. Republican senator: 'Let Mr. Muell...	WASHINGTON (Reuters) - The special counsel inv...	politicsNews	December 31, 2017	True
3	FBI Russia probe helped by Australian diplomat...	WASHINGTON (Reuters) - Trump campaign adviser ...	politicsNews	December 30, 2017	True
4	Trump wants Postal Service to charge 'much mor...	SEATTLE/WASHINGTON (Reuters) - President Donal...	politicsNews	December 29, 2017	True

Table 1: Header of merged dataframe

As observed in Table 1, for each observation, we have the title, the text, the topic, its publication date and the corresponding label. Our dataset contains a total of 44898 news articles with 23481 labelled as 'Fake' and the remaining as 'True'. A quick data inspection shows that there are no missing values. Further inspection will be needed before fully processing the data. For example, we would like to remove an observation with an empty text if such an observation existed. Below we provide a example of observation we may find in our data.

title: Mexico, U.S. agree not to talk publicly about wall payment
date: January 27, 2017 - **label**: True - **subject**: politicsNews

text: MEXICO CITY (Reuters) - Mexico's President Enrique Pena Nieto and his U.S. counterpart Donald Trump on Friday agreed not to talk publicly for now about

payment of the wall the American wants to build on the United States' southern border and that he says Mexico should finance. In a statement, Mexico's government said that during a phone call on Friday, the two had held "constructive and productive" talks and broached issues including the U.S. trade deficit with Mexico and the flow of illegal arms and drugs across the border. "Regarding payment of the border wall, both presidents recognized their clear and very public differences of opinion on this sensitive subject, and agreed to resolve their differences as part of a comprehensive discussion on all aspects of the bilateral relationship," the statement said. "The presidents also agreed for now to not talk publicly about this controversial issue."

3.2 Exploration of text structure for feature engineering

Before processing the text and looking at the content (*i.e* the choice of words) of the text, we want to see if there are any general patterns within the data which would suggest that some news would be fake or real based on its structure. In other words, in this section, we will be answering some of the following questions:

- Does the length of a title/text, the choice of caption, the average length of a sentence are relevant elements to classify 'True' or 'Fake' news ?
- Is the publication date relevant to determine if a document if more likely to be fake or real news ?
- Is the use of special characters in a text an indicator of its likeliness to be 'Fake' news ?

In order to answer those questions, we had to transform the original texts and corresponding titles into a set of relevant features. Note that for readability, most of the following figures will provide either a subset of those engineered features or be plotted with a a smaller sample of observations in the data. Further, the features engineered in this section will be referred to as *structural* features. For extensive plots and tables, please refer to the notebook `./udacityProject/01_DataStructureExploration.ipynb`.

Among the several features engineered, let us highlight the following three:

- *title_length* : the length of the string corresponding to the title.
- *title_uppercase* : the ratio of uppercase letters to the title's length.
- *avg_sent_length* : the average length of a sentence within a source text.

Figure 3 provides the pairwise plots of those features against each other. Note that here 500 samples from each class were sampled at random without replacement to obtain the latter plot. In particular, with the histograms, we observe that the conditional distributions of these features (conditioned on their class) do not totally overlap which suggest that these features would seemingly be

relevant in distinguishing 'True' from 'Fake' news. For example, a high ratio of uppercase letters in the title seems to indicate that the article is likely to be fake news. Similarly, when looking at the plot of *title_uppercase* against *avg_sent_length*, we observe that there is a hint of separation between 'True' and 'Fake' observations, which suggest that having both features could be important to correctly separate both classes. These pairwise plots provided several initial insights into which features could be potentially relevant for determining whether an article was fake news or not.

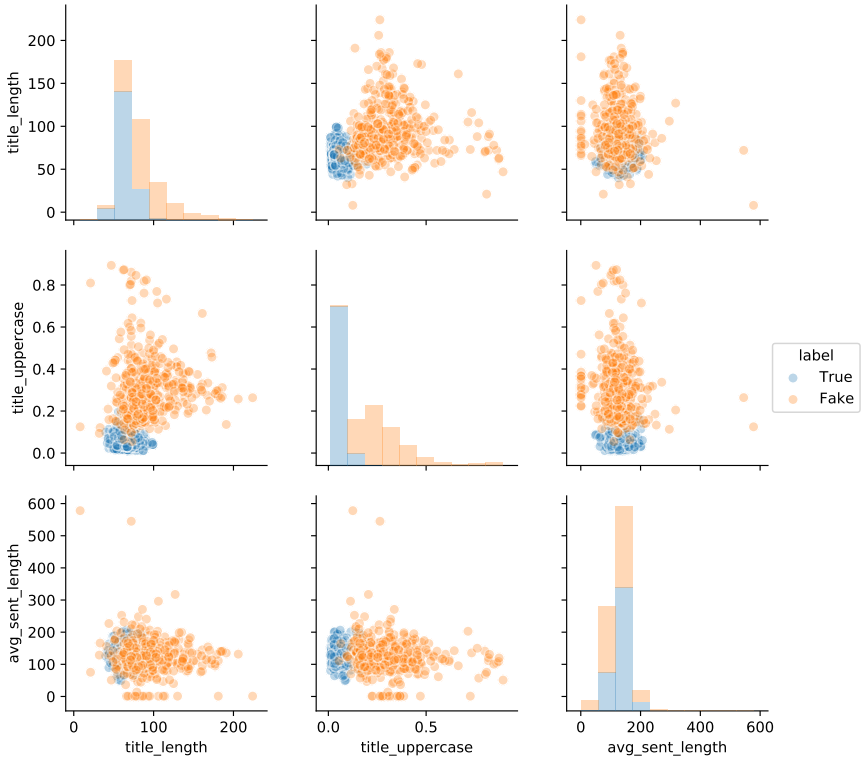

Figure 3: Pairwise plot of three engineered features.

Manual inspection of the data by random sampling of observations also suggested another type of relevant features. Indeed, articles which repeatedly use special characters such as '#' or '@' are often labelled as 'Fake'. In order to verify this hypothesis, we proceeded to visually inspecting the count of those selected characters. Figure 4 features the results of this inspection. We observe that while the use of numbers seem to be evenly distributed between 'Fake' and 'True' news articles[1], some of the characters are significantly more present in

[1]Note that there is one exception. The number '7' is approximately twice as more present in fake news articles.

fake news articles. Suggestive punctuation ('?' and '!') as well as social media commonly used hyper references symbols ('#' and '@') seem to be suggestive of fake news. We will therefore be likely to keep those engineered features for modelling as they visually seem to be correlated with the given targets.

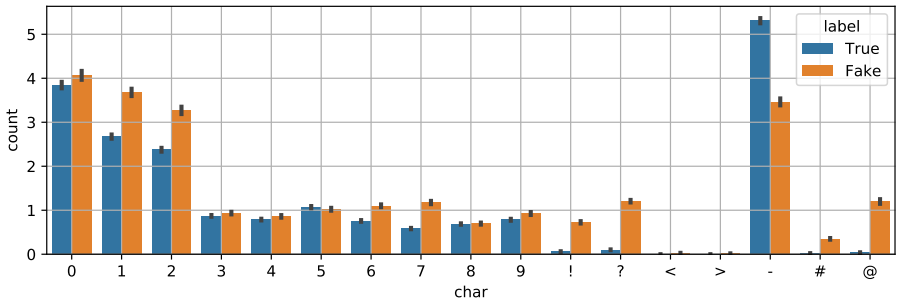

Figure 4: Average count of special characters in a news text.

Further, we hypothesized that publication date could be a relevant feature to determined whether or not an article was real or fake news. Indeed, since content is often driven by demand, and people are generally more available to access content at certain times of the year (holidays for example), we decided to explore the relationship between month of the year and target. This exploration required preprocessing the dates which came in different format. Methods used to extract and process the dates are provided in `./utils/utils_date`.

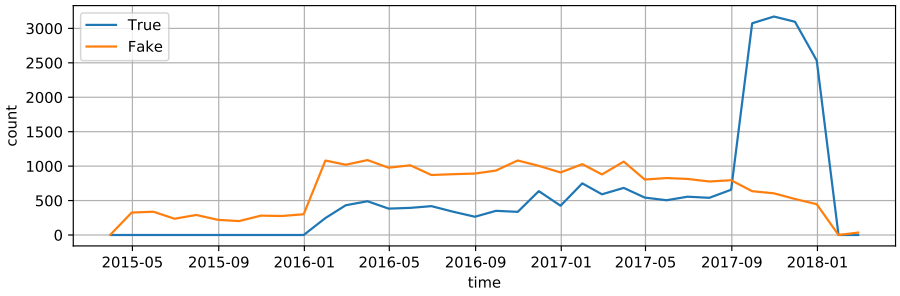

Figure 5: Number of articles over time.

Figure 5 features the count of 'True' and 'Fake' news given the month. We observe that for a period of time there are no 'True' news articles which are sampled while we can observe a high count towards the end of 2017 and the beginning of 2018. We cannot observe any form of seasonality from the way the author [1] provided the data. Indeed, we cannot guarantee that the sampling of 'True' and 'Fake' news articles was done at random and uniformly over time. Therefore, we will not include the date as a parameter in our model since this

could induce a severe bias at the modelling stage. Further, for similar considerations, we choose to not include the subject which is provided with the article since this could again lead to introducing bias.

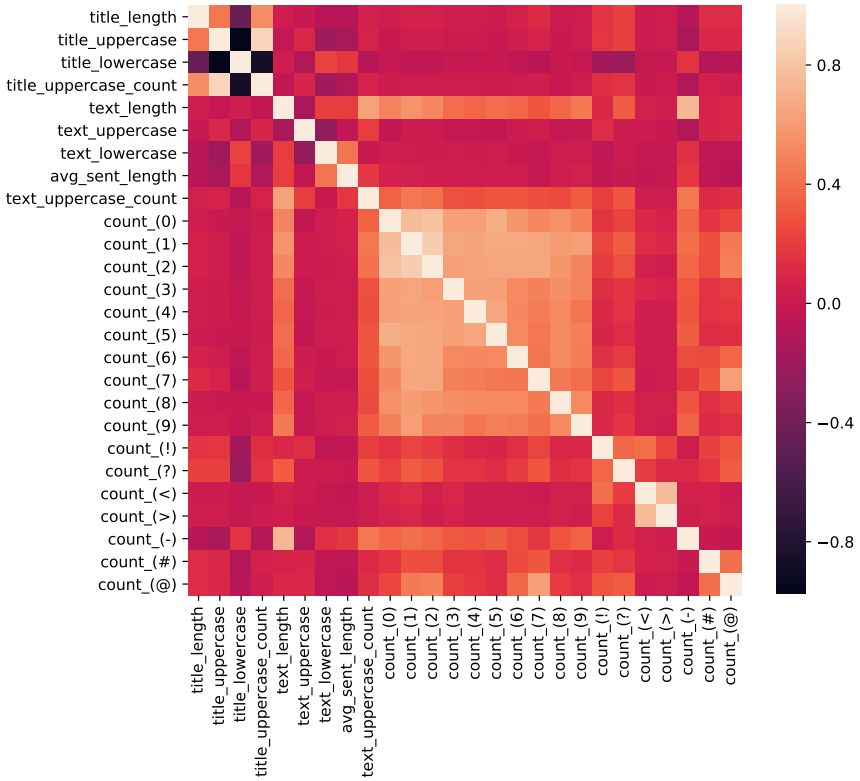

Figure 6: Feature Correlation.

Given the structural features initially engineered and the counts of special characters, we would like to select a subset of minimally correlated features to avoid any modelling issues. Indeed, several classical machine learning procedures suffer from high correlation between features. We therefore look at the correlation between our engineered features. In Figure 6, we plot a heatmap of those correlations. We just try to exclude features which share very light colors (positively correlated features) or very dark colors (negatively correlated features). First, we observe that there some strong correlations between features with prefix *title_*. Indeed, by construction these share some self-explanatory correlation. We arbitrarily choose to keep two of the four features: *title_length* and *title_uppercase*. Second, we observe that the count of numbers (e.g. *count_(0)*) are positively correlated with the *text_length*. This is probably due to the fact that the longer a string is, the more likely it is to contain some numbers. Finally, we would like to make a closing comment regarding the light square at the center

of the heatmap. This shows that the count of one character number is strongly correctly with any other. This makes sense if we start thinking of the type of values we could observe in a news article (for example, a date 31/05/2020 or an amount 123,456.89 US$). Co-appearances for numbers seems very natural.

The last and probably most important step in feature selection is making sure that we are selecting features which are actually signal in the data. For the least, within the group of selected features, we want those corresponding to signal to outnumber the ones corresponding to noise. Hence, our last step consists in selecting features which are correlated with the target. We arbitrarily choose to keep features having which are correlated by a factor of at least 0.1 in amplitude with the target. Table 2 and Table 3 respectively show the structural and character-count features satisfying this condition. In particular, we observe that *title_length* and *title_uppercase* are strongly correlated with the target.

	title_length	title_uppercase	title_lowercase	title_uppercase_count	text_lowercase	avg_sent_length	text_uppercase_count
correlation	0.581159	0.756811	-0.769785	0.548111	-0.160478	-0.165843	0.104969

Table 2: Correlation of structural feature with the target.

Note that as observed earlier the count for character '7' is correlated with the target according to our threshold. We still choose not to include this count as a structural feature for our model.

	count_(7)	count_(!)	count_(?)	count_(-)	count_(#)	count_(@)
correlation	0.129443	0.22682	0.307477	-0.170916	0.14938	0.189084

Table 3: Correlation of character-count features with the target.

In conclusion, our structural analysis of the different documents suggest that we should keep the following set of engineered features: *title_length*, *title_uppercase*, *text_lowercase*, *avg_sent_length*, *text_uppercase_count*, *count_(?)*, *count_(!)*, *count_(#)*, *count_(@)* and *count_(-)*. Relevant functions to generate these features are available in `./utils/utils_metadata.py`.

3.3 Text preprocessing

Most natural language processing and understanding tasks require some prior text preprocessing techniques to standardize the inputs before training a model. The first step in this phase is text cleaning and tokenizing. As mentioned in the previous section some structural features such as special characters or text length can be relevant in determining whether or not a news article can be classified as fake news or not. We propose a text cleaning methodology and tokenization step which will map a concatenated string (title and text) to a sequence of words while keeping some special tokens to emphasize the previously mentioned features. Below we feature an example of text transformation and highlight (when applicable) the changes made to the text. Note that the functions implemented to perform this transformation are available at `./utils/utils_text.py`

Original text

In the U.S.A, Donald Trump #potus tweeted go to www.udacity.com (@UdacitySupport) to learn more about data-science <img src = "img.jpg" alt="Udacity logo">; There is 1000 more new courses to learn from.

Acronym replacement

In the USA, Donald Trump #potus tweeted go to www.udacity.com (@UdacitySupport) to learn more about data-science <img src = "img.jpg" alt="Udacity logo">; There is 1000 more new courses to learn from.

URL replacement and parsing out html

In the USA, Donald Trump #potus tweeted go to URL_TOKEN (@UdacitySupport) to learn more about data-science _ ; There is 1000 more new courses to learn from.

Detect twitter's # and @ references

In the USA, Donald Trump TWITTER_TOKEN tweeted go to URL_TOKEN (REFERENCE_TOKEN) to learn more about data-science ; There is 1000 more new courses to learn from.

Separate compound words

In the USA, Donald Trump TWITTER_TOKEN tweeted go to URL_TOKEN (@UdacitySupport) to learn more about data science ; There is 1000 more new courses to learn from.

Strip punctuation, replace numeric and lowercase

in the usa donald trump twitter_token tweeted go to url_token reference_token to learn more about data science there is digit_token more new course to learn from

Tokenize, remove stopwords and lemmatize

[usa, donald, trump, twitter_token, tweeted, go, url_token, reference_token, learn, data, science, digit_token, new, course, learn]

One can observe that we keep track of special characters such as '-', '#' or '@' by respectively making use of the url_token, the twitter_token and the reference_token.

3.4 Exploration of text content

In Figure 7, we provide an overview of the most common processed words in 'True' and 'Fake' news. We observe in particular that while topics tackled are quite similar (overlapping words include *president, said, trump, republican*), there are some key differences which are going to be quite useful in separating both classes. For example, fake news articles seem to use a lot of urls, @-references or the term *video*. Further, while *trump* is generally used in both 'True' and 'Fake' news, his first name *donald* is more commonly observed in fake news articles.

To take this exploratory analysis a step further, we explore feature importance via L_1-regularization (or the LASSO model [4]). To this effect, we vectorize our texts by computing bag-of-words (BOW) counts and keep the 5000 most frequent words and fit a logistic regression with regularization to obtain

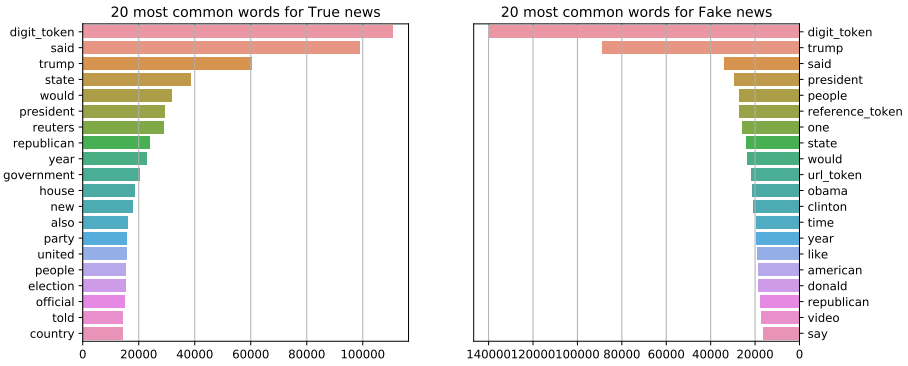

Figure 7: Most frequent words in True and Fake news.

a sparse solution which will indicate the most important features. Note that the regularization parameters was set to $\lambda = 100$ to get fewer selected features. In Figure 8 we provide a randomly selected sample of relevant words in separating both classes. Note that the implementation of the aforementioned procedure follows in part the blog post at `https://towardsdatascience.com/feature-selection-using-regularisation-a3678b71e499`.

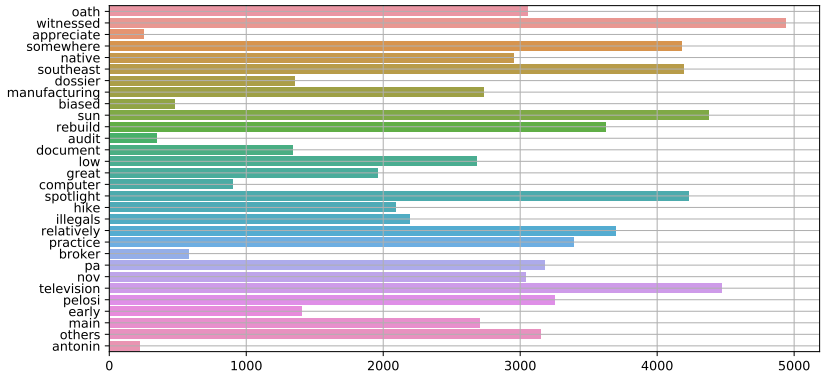

Figure 8: 30 randomly selected decisive words and corresponding frequencies.

4 Evaluation Metrics

In order to evaluate the strength of our trained estimator and compare it to a set of benchmark models, we first want to evaluate the accuracy of our classifier.

The accuracy can be defined as

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{[\hat{f}(x_i)=y_i]} \tag{1}$$

where samples from out test set are of the form (x_i, y_i), $\hat{f}$ is our trained estimator and $\mathbb{1}$ is an indicator function assigning 1 when the equality is verified and 0 otherwise. Note that since our data contains approximately the same number of positively and negatively labelled samples, we do not need to opt for any balancing strategies before measuring the accuracy.

Further, recall that our main goal was to filter out the noise. If we consider that fake news is labeled as 0 and true news as 1, we actually want to control the false positive rate. In other words, we want to make sure that the number of fake news articles classified as true is quite small. A standard metric to control the number of false positives is the precision [5] which can be defined as

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2}$$

where TP is the count of true positives and FP stands fort the count of false positive. Hence, we want to built a relatively accurate model with a very high precision. In our fake news detection task, this means that we are ready to sacrifice a few accuracy points as long as long as the articles selected are very likely to be true positives.

5 Benchmark model

In order to easily generate multiple benchmark scores we will be proposing three subsets of the data to train our benchmark model. The first will consider only the structural features engineered in Section 3.2. The second will consider the term-frequency inverse-document-frequency (tf-idf) weighted BOW features [6] generated using the transformed texts. Finally, our final training dataset will consist of the concatenation of both the structural features and the tf-idf BOW features.

5.1 Preprocessing

In order to train our benchmark model, we must first transform and format the data engineered in the previous section and further split the data into a training, validation and testing set. We start by splitting the data with 80% of the data used for training and the rest evenly split between validation and testing. We then proceed to vectorizing our data using tf-idf weights and standardize our structural features. Note that the process is done by fitting those data transformers to the training set and then transforming separately the validation and test set. This means that for example the standardizing parameters are chosen

with respect to training set and then used to transform both the validation and test set. We then concatenate structural features with weighted bag-of-words features. Prior to the modelling phase we also check that the datasets are balanced.

5.2 Naive Bayes Classifier

The Naive Bayes Classifier [7] is a classification method which uses a very simple Markov assumption on the data: independence of the features conditioned on the target label. In other words, after successfully applying the law of total probability, the likelihood of our estimator can be written as

$$p(y|x_1, ..., x_n) \ \alpha \ p(y) \prod_{i=1}^{n} p(x_i|y) \tag{3}$$

While the Markov assumption is relatively strong, optimizing a Naive Bayes Classifier is computationally efficient and results will then provide a simple baseline. Note that this estimator requires the choice of a prior. For simplicity, we choose to fit our data using default parameters for the `GaussianNB` estimator in scikit-learn [8].

5.3 Results

Results obtained after fitting the model are summarized in table 4. We fit three times the same model while keeping only a subset of the features. The first model contains only the structural features engineered earlier (Struct). The second one contains only the bag-of-word tf-idf weighted features (BOW) and finally the third model is trained with the concatenation of both (Struct + BOW).

	NB(Struct)		**NB(BOW)**		**NB(Struct + BOW)**	
	Accuracy	Precision	Accuracy	Precision	Accuracy	Precision
train	0.949	0.920	0.931	0.948	0.934	0.951
validation	**0.955**	0.927	0.928	0.943	0.932	**0.951**
test	**0.948**	0.916	0.920	0.932	0.922	**0.934**

Table 4: Results obtained after fitting Naive Bayes Classifier

The results obtained while using a simple benchmark model are already quite satisfactory. All of the models in testing obtain an accuracy of over 92%. Further, the precision score is also quite high going as low as 91.6 % for the test set of NB(Struct). This suggests that the engineered features led to highly separable classes. Highest accuracy is reached for NB(Struct) and highest precision for NB(Struct + BOW). Note that we initially created a validation set to consider the possibility of ensembling weak learners in the case that our simple benchmark model was not performing well enough. However, here the validation set can be seen as another test set and hence accounts for reproducibility of

our results. Finally, we consider that adding the structural element only lead to minimal improvements in the benchmark model. While we keep the results as a baseline to compare our next model, we decide to exclusively focus the training on the engineered bag-of-word features, since tracking the mean/variance of the structural elements (used to normalize our structural features before training) might lead to future issues in for example deploying the model to a webapp.

6 Model Deployment

6.1 Summary and introduction

In the previous section, we used a Naive Bayes Classifier to separate the data. We observed that selecting a subset of the engineered features already led to good results given the strong assumptions held on the data by the Naive Bayes model. In this section, we outline the steps to deploying a linear model to fit our BOW data. We will make use of SageMaker's `LinearLearner` instance and show that a linear estimator provides sufficiently accurate results. Note that initially, we had the ambition of implementing and deploying a recurrent neural network to capture the sequential aspect of our data. However, given the already satisfactory results, we focused on extracting meaningful information from the learning process (see Section 6.4).

6.2 A linear model for classification

The logistic regression model [7] is a linear classification model which aim is to estimate the probability class of the target Y given a set of features X. This can mathematically be written as

$$\mathbb{P}_w(Y = 1|X = x) = \sigma(w^\intercal x) = \frac{1}{1 + e^{-w^\intercal x}} \tag{4}$$

where σ is referred to as the logistic function, x is a vector of features and w are the corresponding weights. In machine learning, the weights are often optimized by maximizing the log-likelihood of this model. For simplicity, we will use Sagemaker's wrapper `LinearLearner` which simplifies the learning process, implementation and deployment.

6.3 Results

When fitting our logistic regression with the BOW data, we obtain the results featured in table 6.3. We obtain a four points increase in accuracy compared to the Naive Bayes classifier with the same data and approximately a two point increase in precision. Given the very satisfactory performance of our model, we further provide the recall [5] obtained which is above 99%. These results are a clear improvement over the previous baseline and suggest that there are some well-defined differences between 'True' and 'Fake' news articles.

	Linear Learner Model (BOW)		
	Accuracy	Precision	Recall
Validation	0.977	0.958	0.995
Test	0.970	0.945	0.995

Table 5: Linear learner results

6.4 Identifying key tokens in detecting fake news

One of the main advantage of some classical machine learning models, in particular the logistic regression model, is that the model weights provide some insights into the structure of the data and are interpretable. Hence, we would like to identify some textual cues which could be indicative of fake news to quickly be able to flag an article as we are reading it.

Since the BOW weighted features are tf-idf weights, those features are positive. In particular, this means that the sign of a coefficient tells us the direction in which a token is pulling us ('True' news for a positive weight and 'Fake' news for a negative one). Further, the magnitude of the coefficients is another indicator of the importance of that token in separating both classes.

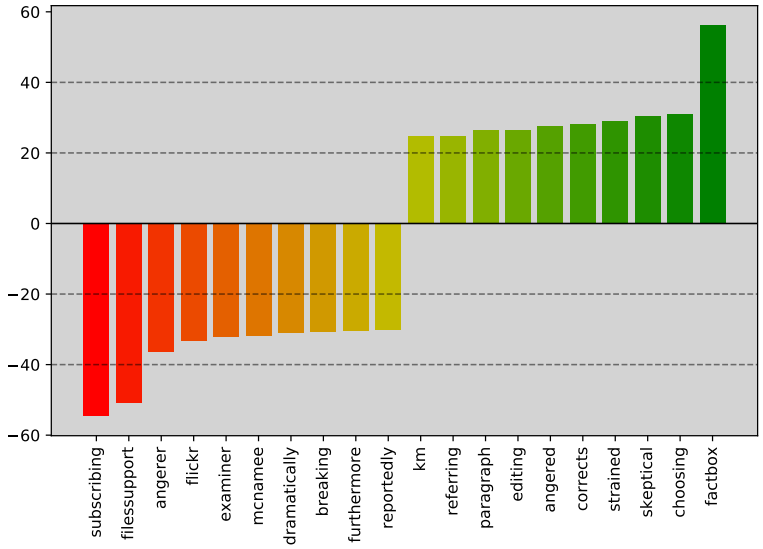

Figure 9: Some processed tokens in our vocabulary with corresponding regression weights. We kept the 10 features of highest amplitudes in each direction.

In Figure 9, we feature the weights of the linear learner for a subset of tokens. We observe that words such as *'flicker'* or *'subscribing'* seem to be indicative of a fake news article. Further, spelling mistakes and use of suggestive language as observed in *'angerer'* and respectively *'dramatically'* can point towards identifying a fake news article. On the other hand, words related with

references or citation such as *'referring'*, *'editing'*, *'paragraph'* or *'factbox'* are strong indicators of a real news article.

7 Discussion

7.1 Highly separable data ?

We obtained very satisfactory results with quite simple models. Indeed, with a linear estimator (see previous section), we managed to obtain an accuracy superior to 97% with a high precision and recall. Note that those results were obtained using a subset of the engineered features, *i.e.* the BOW tf-idf features. We may hence ask ourselves if the engineered features led to highly separable data. Figure 10 provides the t-sne projection [9] of a subset of articles in two dimensions. We also provide the empirical distribution of the data for 'Fake' and 'Real' news for both dimensions in the plot. We observe that while there is some overlap between both classes in the embedding, there is a notable distance between the distributions of the classes. This suggests that in higher dimensions, the data is likely to be even more separated.

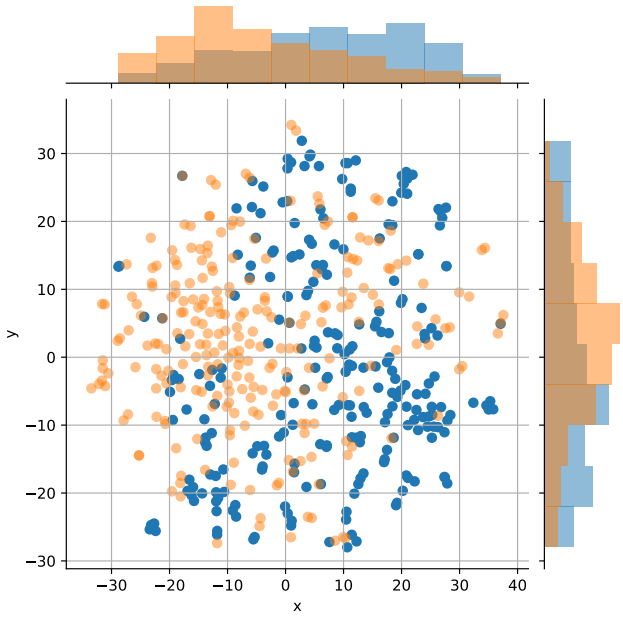

Figure 10: Subset of the bag-of-words data projected in two dimensions using t-sne. In blue, we have 'True' news and in orange, 'Fake' news.

7.2 Building a web-app

While not required, we chose to include a first version of what a web page analysing reliability of news articles would look like. We chose to kept the structure provided in the Udacity course and only provide here an update template within the framework of the project. Figure 11 features a display of what the web-app would look like. We leave the deployment of such a web app for future works.

Figure 11: Front end.

8 Conclusion

In this project, we explored the combination of some smart data engineering with classical machine learning models. We spent a significant part of the project on data exploration and preprocessing which helped us discover some peculiarities which would later on be useful in separating the 'Fake' and 'True' classes in the data. For example, we manage to incorporate some structural aspects (special characters, URLs...) into our BOW features by designing some specific transformation and tokenization processes.

Further, we deployed a linear model which outperformed our Naive Bayes benchmark model regardless of the different subsets of the data considered. Indeed, our logistic regression model had an accuracy of over 97% and a precision of 95%. Recall that one of the key goals of our project was to limit the number of false positives (*i.e.* 'True' news articles classified as 'Fake'). Hence, a standard logistic regression model was sufficient to achieve this goal.

Note that we chose not to further improve our satisfactory results by for example exploring recurrent neural network models. Indeed, while such a model would most certainly improve performance, it would come at the cost of losing interpretability. By exploring the amplitude and sign of fitted weights to our linear model, we were able to highlight some textual cues to quickly distinguish true from fake news.

While we provided a canvas to deploy a web-app using our model, this remains beyond the scope of this project and we therefore leave it as a small project for future works.

References

[1] Clément Bisaillon. Fake and real news dataset, 2020.

[2] Hadeer Ahmed, Issa Traore, and Sherif Saad. Detecting opinion spams and fake news using text classification. *Security and Privacy*, 1(1):e9, 2018.

[3] Hadeer Ahmed, Issa Traore, and Sherif Saad. Detection of online fake news using n-gram analysis and machine learning techniques. In *International Conference on Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments*, pages 127–138. Springer, 2017.

[4] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[5] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

[6] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[7] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[8] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[9] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.