FAKE NEWS PROJECT

REPORT

Alexander Husted Rasmussen
wqg382@alumni.ku.dk

August Vitus Pallesen
wqm205@alumni.ku.dk

Salem Zaki Dableh
dcw659@alumni.ku.dk

Sky Hovin Wong
lgz282@alumni.ku.dk

Marts 2023

1 Introduction

Today, we see fake news spreading like wildfire across the whole web. According to Tim Berners-Lee, the inventor of World Wide Web, fake news is one of the three most disturbing internet trends.[int, 2017] The possible ramifications of fake news is especially worrying in politics with, for example, Russian troll farms deliberately writing fake articles about hot topics such as gun legislation, immigration, and race relations. In 2016 right before the American election, the top 20 fake news stories circulating social media received more engagement than the top 20 reliable news articles. The specific impact is hard to measure, but studies has shown that people were more likely to vote for one party if they believed the fake articles about the opposing party. [int, 2020]

In Denmark, we have *"pressenævnet"* to ensure reliability for most known domains, but this kind of system isn't perfect and isn't well suited for global and larger scales. That is why we need models to predict whether an article is fake or reliable. Ideally these models could be used to warn users to proceed cautiously before reading an article. To address this problem, we'll make a fake news detector trained solely on the content of an article. This should make a more generic model since we don't rely on extra features such as meta_description, which some articles might not have.

2 Data Processing

2.1 Data

We use pandas dataframe when working with the dataset. The reason for this is that pandas can handle large files upto 40 GB [tow, 2018], the dataframe is made specifically for python and therefore also works well with other python libraries, [dat, 2018] and pandas dataframe has a very intuitive visualisation of how a dataframe looks.

We use CSV-files to store the data throughout the project, so that we don't have to run some of our functions, such as the cleaning process, each time we start our notebook.

2.2 Preprocessing

Duplicates
We see that 52% of the dataset consists of duplicate articles. The distribution of articles should be uniform, with every article being represented once. Therefore we remove these duplicates with the build-in *.drop_duplicates* function in pandas.

Cleaning of strings
We use regular expressions to remove unwanted characters from the strings such as {!, :, ,$, &}. Then we apply stopwords from *nltk.corpus* to remove words that don't carry useful meaning. To treat all URLs, phone numbers, and numbers the same way, we use regular expressions to convert them into the strings *'phone'*, *'url'* and *'number'*. *'number'* appear as expected frequently at 9.535 times per article on average, *'phone'* and *'url'* appears less often with a frequency of .110 for *'phone'* and .002 for *'url'*.

Stemming
To do stemming, we first have to tokenize the text using *nltk.word_tokenize* to split the strings into a list of words. We apply stem from the *nltk.stem* for every element in *list.SnowballStemmer* module. When this is done, we can start to explore the words in the dataset. We see words like Trump, state, blockchain, market, and govern appear frequently. And as seen in Fig 1 we have a slightly imbalanced dataset in favor of fake news, which means there is a high likelihood that these words are keywords often seen in fake news.

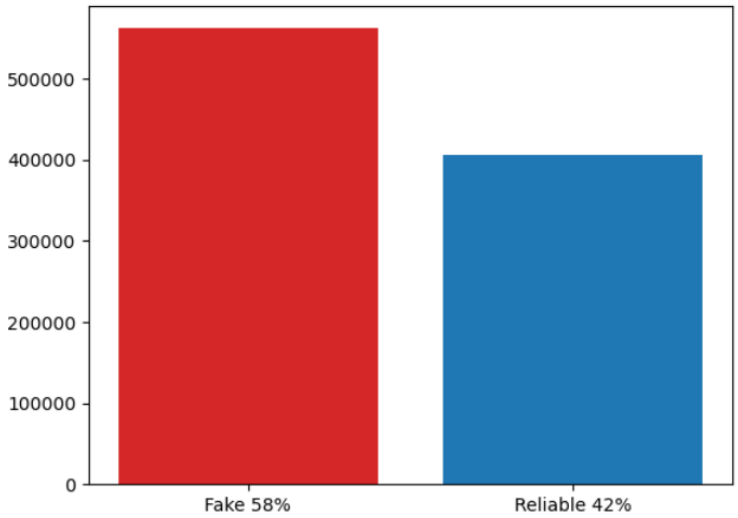


Figure 1: Baseline results with *meta_description*

Split
We split the dataset into train, test, and val with a distribution of $80\%, 10\%$ and 10%. The split fills the sets randomly to comprehend the way we created the original dataframe, meaning that the data was sorted with fake first and then reliable. The split is done by using the pandas functions $.sample$ and $.to\_csv$, this is done twice to get the three wanted files.

TF-IDF weights
To evaluate the words in the dataset and to use a way to weigh different words, we used TF-IDF weights to do this. TF-IDF weights is a statistical measure used to evaluate how important a word is in a corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. We use these weight to compute our baseline model, by using the *sklearn.feature_extraction.text* module.

3 Simple Model

Labels
We only extract labels from the dataset, where we can be confident that the label represents either Fake or Reliable. This includes *political*, which generally provides verifiable information, and *reliable*, which provides articles consistent with traditional and ethical practices in journalism, for the *reliable* training set. [fak, 2016] For the Fake, we use *fake, junksci, hate, unreliable, rumor* and *conspiracy*. We do this to preserve balance in the label distribution. If the dataset is too imbalanced in favor of *fake*, the probability of type 1 errors increases, and vice versa for *reliable* type 2 errors increases.

Baseline model
For our baseline model we chose to use a logistic regression model. Logistic regression is a well suited tool for working with binary classification[Log, 2020] which is what we are building. With binary classification we attempt to calculate the conditional probability of *Y* occurring given a *X*. In other words the conditional probability of an article being fake given an arbitrary wordvector. Mathematically it can we written $P(Y = 1|X)$, where *Y* is an element in a binary list, the value 1 equals fake and *X* represents our TF-IDF vector.

We are working with a data that have different feature attributes that we train with. In this regard we initially thought it would be beneficial to include meta-data features as these can be looked upon as extra data features, and will thus contribute in the training of the model. However we need to carefully select the relevant and useful meta-data, in order to avoid overfitting or adding more noise that could arise from using redundant or irrelevant meta-data features. Even though 76% of the rows in *meta_description* is empty/null.

Consequently we can see that including the *meta_description* feature does in fact improve the accuracy, precision and recall of our model. The results aligns with our expectation namely that when increasing the amount of relevant data-features we allow our model to train on, it will naturally improve the predictive skills of our model.

Table 1: Baseline Model results for News corpus

Fake News Corpus	**without meta**	**with meta**
Logistic regression - MSE	0,100	0,099
Logistic regression - Accuracy	0,899	0,900
Logistic regression - Precision	0,901	0,903
Logistic regression - Recall	0,899	0,896

The relatively high scores of the baseline model correlates well with the scatterplot seen below in Fig 2. The figure shows how the red and blue dots are somewhat distinctly separated, and so we can imagine how the baseline model should be able to adequately fit a line dividing the blue and red dots. This indicates that the model is able to distinguish between fake and reliable articles with a high degree of accuracy, leading to the good results.

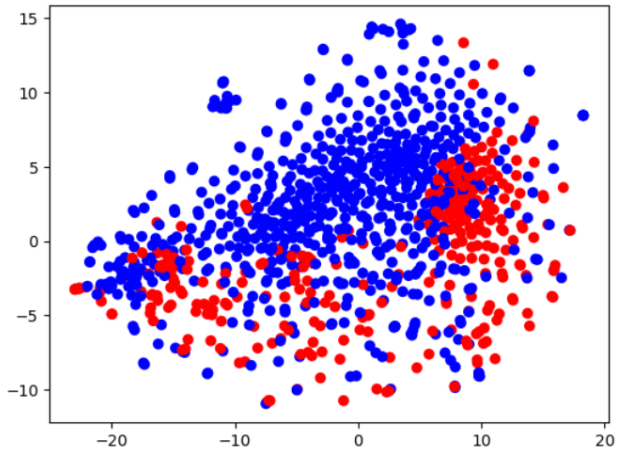


Figure 2: The graph represents the first 1000 TF IDF from *x_train* after *sklearn.manifold.TSNE* has been applied to reduce the dimensions to 2D. Blue dots represent Reliable data and red represents Fake data.

4 Advanced Model

Word embedding
For word embedding, we used Word2Vec from the *gensim.models* library. We apply the *'word2vec-google-news-300'* to each word. This method uses a continuous bag of words and a continuous skip-gram algorithm to create a 300-dimensional representation of the similarity between the words.[Goo, 2013] This enables us to compare structural and semantic relationships between words. [cov, 2023] We then take the average of all the vector representations leaving us with a distinct vector for each article's content.

Using this method reduces noise because we are using an outdated predetermined list of words, but it creates some additional problems: 1) We can't train on words such as bitcoin or blockchain, because they aren't in the list. 2) We don't include misspelled words, which is a problem because we could assume a correlation between proofreading and credibility. We still choose the Word2Vec approach because we can train the model not only on which words are in X, but also on which kinds of words are in X. This makes the model broader and more capable of dealing with unseen data.

Gradient boosting
For our advanced model, we choose *GradientBoostingClassifier* from the *sklearn.ensemble module*. Gradient boosting is a powerful model that is good with missing values, outliers, and too many of one type of unique value. [gra, 2022a]

Gradient boosting systems have two other necessary parts: a weak learner and an additive component. Gradient boosting systems use decision trees as their weak learners. Regression trees are used for the weak learners, and these regression trees output real values. Because the outputs are real values, as new learners are added into the model the output of the regression trees can be added together to correct for errors in the predictions.

The additive component of a gradient boosting model comes from the fact that trees are added to the model over time, and when this occurs the existing trees aren't manipulated, their values remain fixed. A procedure similar to gradient descent is used to minimize the error between given parameters. This is done by taking the calculated loss and performing gradient descent to reduce that loss. Afterwards, the parameters of the tree are modified to reduce the residual loss.

The new tree's output is then appended to the output of the previous trees used in the model. This process is repeated until a previously specified number of trees is reached, or the loss is reduced below a certain threshold.[gra, 2023a]

We chose this model because, as seen in Fig 3 , a lot of our data seem to be very similar (represented as points that are close to each other on the figure) and we have a few outliers ex. the single red dot at approximately (20, 30).

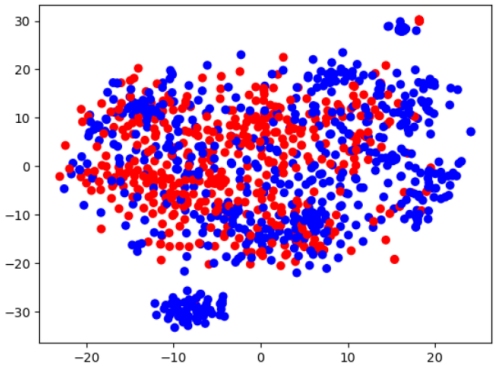


Figure 3: The graph represents the first 1000 Word2Vec from *x_train* after *sklearn.manifold.TSNE* has been applied to reduce the dimensions to 2D. Blue dots represent Reliable data and red represents Fake data.

Gradient boosting works by fitting an initial model (tree), then it creates a new tree that focuses on improving where the first model performed poorly. Then the models are combined, and the process is repeated many times.[gra, 2022b] There are several hyper-parameters we can tune for our model. Most of these values have been found using a grid search on the validation dataset. [gra, 2023b]

n_estimator
We chose $n\_estimator = 100$. This is a fairly high number of boosting stages the model has to perform. But gradient boosting is a robust model for overfitting, which means that larger numbers usually result in better performance.[gra, 2023b]

learning_rate
We chose $learning\_rate = 0.1$, which shrinks the contributions of each of the previously mentioned trees by 0.1. [gra, 2023b]

max_depth
We chose $= 4$, meaning the number of nodes on each tree is limited to 4. This value is fairly small, but since we already have a high *n_estimator*, we went with a small *max_depth* to prevent the model from overfitting. [gra, 2023b]

loss
For the loss function, we chose *'deviance'*. Which is equivalent to *'log_loss'*, since it removal in version 1.3 of *sklearn*. *'log_loss'* stands for *logistic loss function*. The role of the loss function is to estimate how good the model is at making predictions with the given data. For each article, our model calculates the probability P(X | Y=1), and the *log_loss* function tells the model how close the computed probability is to the actual value.

We choose the *log_loss* value because it's easy to interpret. The higher the *log_loss* value, the more our predicted probabilities diverge from the true value. The *log_loss* function can be generated using Bayes consistent loss function and the Table-I [Tab] be the following:

$$\phi(v) = C\left[f^{-1}(v)\right] + \left(1 - f^{-1}(v)\right) C^1\left[f^{-1}(v)\right]$$

$$= \frac{1}{log(2)}\left[\frac{-e^v}{1+e^v} log \frac{e^v}{1+e^v} - \left(1 - \frac{e^v}{1+e^v}\right) log\left(1 - \frac{e^v}{1+e^v}\right)\right] + \left(1 - \frac{e^v}{1+e^v}\right)\left[\frac{-1}{log(2)} log\left(\frac{\frac{e^v}{1+e^v}}{1 - \frac{e^v}{1+e^v}}\right)\right]$$

$$= \frac{1}{log(2)} log(1 + e^{-v})$$

The logistic loss is convex and grows linearly for negative values which make it less sensitive to outliers. And the logistic loss and binary cross entropy loss is the same up to the constant: $\frac{1}{log(2)}$

5 Evaluation

Performance
Our baseline model did worse on the LIAR set than on the corpus Tab 1. Why this is the case, is discussed further below, while looking at the scatterplots of the two datasets Fig 7.

Table 2: Baseline Model results for the LIAR set

LIAR	
Logistic regression - MSE	0.602
Logistic regression - Accuracy	0.398
Logistic regression - Precision	0.604
Logistic regression - Recall	0.195

In order to evaluate the performance of our model on the test set, we included metrics such as precision, recall and F1-score which all can be seen in the table below. The model shows a precision result of 0.755, which means that our model is fairly good at correctly predicting true positive values, that actually are true positive.

In other words the model's ability to avoid false positives. With a recall score of 0.76 it seems that the model is fairly efficient at predicting the true positives out of all the positive values (true positives + false negatives). The F1-score of 0.74 combines the precision and recall, by calculating mean of the two. With a F1-score of 0.74, we can infer that the model has an appropriate balance between the precision and recall scores.

Table 3: Advanced Model results

	NewsCorpus	**LIAR**
GradientBoostingClassifier - MSE	0.244	0.550
GradientBoostingClassifier - Accuracy	0.755	0.449
GradientBoostingClassifier - Precision	0.766	0.618
GradientBoostingClassifier - Recall	0.831	0.383
GradientBoostingClassifier - F1-score	0.740	0.475

Below can be seen a confusion for the NewsCorpus- and Liar data respectively. We observe that the model trained on NewsCorpus is significantly better at predicting true positives and negatives. In the NewCorpus matrix, the numbers in the True row is proportionally bigger than the numbers in the False row, as opposed to the liar confusion matrix.

The biggest deviant here lies within the false negatives which with 504 occurrences is considerable high (and unfavorable) in regards to the other relatively low values. The suboptimal liar-data confusion matrix well reflects the poor scores of the accuracy, recall and F1-score from before.

	Actual	
Predicted	Positive	Negative
True	58267	33014
False	17723	11848

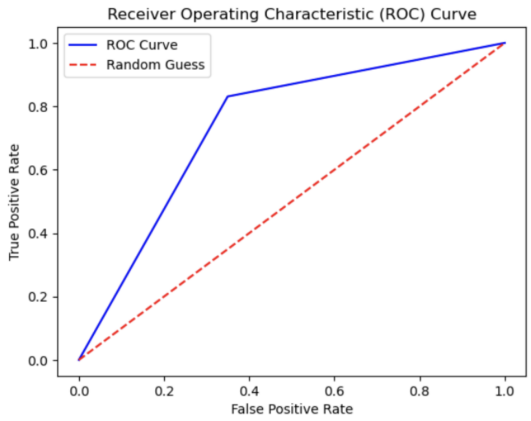
Figure 4: Confusion Matrix based on NewsCorpus data

	Actual	
Predicted	Positive	Negative
True	314	255
False	194	504

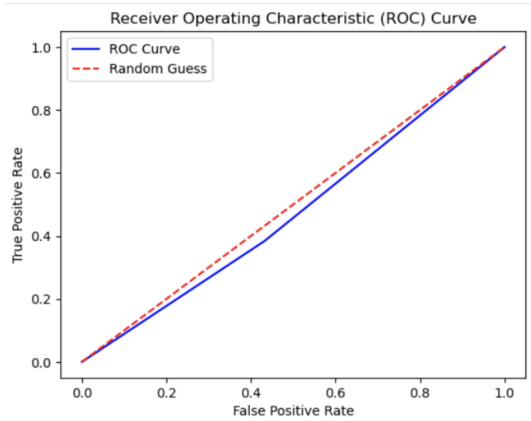
Figure 5: Confusion Matrix based on LIAR data

For the graphical representation of the performance of the model we used ROC-curves, as can be seen in Fig 6. In the ROC-curves we see the different thresholds for the True and false positives respectively. Naturally the more the curve is bends towards the upper left corner$((0,0),(1,0))$ the better the model is at predicting true positives correctly. Vice verca if the curve bends towards the bottom right corner $((1,0),(0,0))$ the model would be accurate in its prediction of false positives, which the opposite, and is not desirable.

The NewsCorpus ROC curve indicates a that the model is gradually better at predicting the true positives than random guessing would be. Differently the liar ROC curve, seems to be performing significantly worse than the NewsCorpus ROC curve. For the Liar ROC curve it seems that plain guessing will provide a more favorable result than the liar model will.



(a) ROC curve - NewsCorpus data



(b) ROC curve - LIAR data

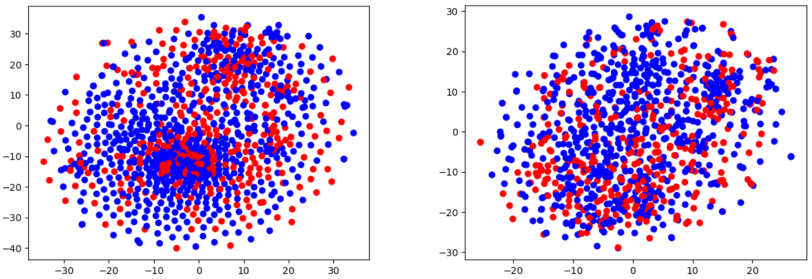
Figure 6: Represents the ROC curves based on NewsCorpus- and liar data respectively

6 Conclusion

The baseline model performs worse on the LIAR dataset, with an accuracy of 0.398 on the LIAR dataset Tab 2 and an accuracy of 0.899 on the Fakenews corpus Tab 1. The model did well with the Fakenews corpus, because there was a clear pattern distinguishing reliable data-points from fake data-points in Fig 2. Looking at Fig 7 (a), the pattern is distorted, making the model perform worse as it is not trained on such a pattern.

The advanced model also performs worse on the LIAR dataset. We see again that there are differences in how the point is scattered in Fig 3 and 7 (b), but the difference isn't as significant. We see, however, the distribution of domains in the Fakenews corpus is very poorly distributed. The top ten most frequent domain for Fake articles accounts for 78%, which is very high considering that there are 180 different domains.

The same poor distribution is seen for reliable, where the top ten articles account for 79% out of a total of 56 domains. This means that we essentially have created a model that can predict whether or not an article is in one of these domains. So when the model gets new data, such as the Lair dataset with a different set of domains, the predictions will be no better than random guessing.



(a) Scatterplot LIAR TF-IDF vector (b) Scatterplot LIAR Word2Vec

Figure 7: The graph represents the first 1000 X-values after *sklearn.manifold.TSNE* has been applied to reduce the dimensions to 2D. Blue dots represent Reliable data and red represents Fake data.

In order to improve the performance of our model, we would have to improve the data. We would need a more uniform distribution and a larger number of domains to make our model more generic and, thereby, more useful with new data. It's a well-known concept in computer science that if the model takes garbage in, it will return garbage out (GIGO), meaning having the correct data is essential to make good models.

References

The world wide web's inventor warns it's in peril on 28th anniversary, 2017. URL https://eu.usatoday.com/story/tech/news/2017/03/11/world-wide-webs-inventor-warns-s-peril/99005906/.

How fake news affects u.s. elections, 2020. URL https://www.ucf.edu/news/how-fake-news-affects-u-s-elections/.

Why and how to use pandas with large data, 2018. URL https://towardsdatascience.com/why-and-how-to-use-pandas-with-large-data-9594dda2ea4c.

6 essential advantages of pandas library – why python pandas are popular?, 2018. URL https://data-flair.training/blogs/advantages-of-python-pandas/.

several27/fakenewscorpus at v1.0, 2016. URL https://github.com/several27/FakeNewsCorpus/tree/v1.0).

Quick and easy explanation of logistic regression, 2020. URL https://towardsdatascience.com/quick-and-easy-explanation-of-logistics-regression-709df5cc3f1e.

Google code archive - long-term storage for google code project hosting., 2013. URL https://code.google.com/archive/p/word2vec/.

Word2vec explained: How computers learned to talk like we do!, 2023. URL https://www.coveo.com/blog/word2vec-explained/.

All you need to know about gradient boosting algorithm part 1. regression, 2022a. URL https://www.displayr.com/gradient-boosting-the-coolest-kid-on-the-machine-learning-block.

Theory behind gradient boost, 2023a. URL https://stackabuse.com/gradient-boosting-classifiers-in-python-with-scikit-learn/.

Gradient boosting explained – the coolest kid on the machine learning block, 2022b. URL https://towardsdatascience.com/all-you-need-to-know-about-gradient-boosting-algorithm-part-1-regression-2520a

sklearn.ensemble.gradientboostingclassifier, 2023b. URL https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html.

Loss functions for classification. URL https://en.wikipedia.org/wiki/Loss_functions_for_classification.