
FAKE NEWS PROJECT

FUNDAMENTAL DATA SCIENCE

📧 **Emil V. Klinker**

Machine Learning & Data Science
University of Copenhagen
KU-username: dkz250
dkz250@alumni.ku.dk

📧 **Simon Henriksen**

Machine Learning & Data Science
University of Copenhagen
KU-username: xw346
xw346@alumni.ku.dk

📧 **Simon Hvidfeldt**

Machine Learning & Data Science
University of Copenhagen
KU-username: zgh166
zgh166@alumni.ku.dk

📧 **William S. Saugmann**

Machine Learning & Data Science
University of Copenhagen
KU-username: sgz655
sgz655@alumni.ku.dk

March 28, 2025

Link to GitHub repository: <https://github.com/Simo259k/DS-Fake-News-Project>

Keywords Data Retrieval · Data Preprocessing (cleaning, tokenization, stopword removal, stemming) · TF-IDF · Logistic Regression · SGDClassifier · Binary Classification · F1 Score · Confusion Matrix · Sample Bias / Domain Shift · Data Quality · Exploratory Data Analysis (EDA)

1 Data Processing

FakeNewsCorpus: Preprocessing & Vocabulary Analysis

In this project, we explored a subset of the FakeNewsCorpus to practice data retrieval, cleaning, and preprocessing at scale. We began with a small sample file, `news_sample.csv`, to familiarize ourselves with key libraries and techniques. Using `pandas`, we loaded the dataset into a `DataFrame`, which allowed for convenient manipulation of columns, handling missing values, and concatenating text fields. We employed the `re` module (regular expressions) to replace noisy patterns—such as dates, URLs, numeric values, and emails—with standardized placeholders.

Next, we used NLTK's `word_tokenize` to split the text into tokens and removed stopwords using NLTK's English stopwords list. This step reduced the token count by approximately 35.6%. We then applied the Porter stemmer from NLTK to consolidate word variations (e.g., “running,” “runs,” “ran” → “run”). In our small sample, the final token count remained unchanged, suggesting minimal variation. To support exploration, we used `collections.Counter` to analyze token frequencies and gain insights into the corpus.

Scaling Up and Data Splits

For Task 2, we scaled the preprocessing pipeline to the full FakeNewsCorpus dataset, which consists of approximately 995,000 articles. Due to the dataset's large memory footprint, we processed the CSV file in chunks of 25,000 rows using the `chunksize` parameter in `pandas`. This chunk-wise method allowed us to apply the same cleaning, tokenization, stopwords removal, and stemming procedures without overwhelming our system. After processing, we concatenated the chunks into a single cleaned `DataFrame` and saved the result to a new CSV file.

We also made plots, which shows us how the rank-frequency distribution shifts once common stopwords and noise are removed, resulting in a more balanced set of high-frequency tokens. (See Figures 7 and 8)

Exploratory Data Analysis

The exploratory analysis aimed to gain an initial understanding of the dataset's structure and potential issues. Although using `.shape` was not feasible due to memory constraints, we used `.head(5)` to inspect the data. We then performed targeted analyses, including:

- Identifying missing values in key columns.
- Counting distributions of domains, types, and authors.
- Plotting barplots for the most frequent words in the content—both overall and grouped by type. (See Figures 10 to 22)
- Counting special tokens (e.g., `<URL>`, `<DATE>`, `<NUM>`, `<EMAIL>`) and creating word frequency barplots for the raw dataset. (See Figure 9)
- Detecting textual anomalies or artifacts, such as error messages or extremely short/long articles.

Our analysis revealed several inherent problems: significant missing values (particularly in `tags`, `meta_description`, and `authors`), and an imbalance in domain distribution—`nytimes.com` appeared nearly twice as often as the second-largest source. Additionally, the frequent occurrence of the token `num` (due to regex cleaning, see figure 9) led us to remove it from the final dataset to avoid artificial bias.

Data Splits

Finally, we split the cleaned dataset into three subsets for machine learning:

- 80% for training (`X_train`, `y_train`).
- 10% for validation (`X_val`, `y_val`).
- 10% for final testing (`X_test`, `y_test`).

This was implemented using `train_test_split` from `sklearn.model_selection`. We first separated 20% of the data and then evenly split it into validation and test sets, using a fixed random seed (`random_state=42`) to ensure reproducibility.

Together, these steps transformed the raw dataset into a structured, cleaned, tokenized, and normalized corpus that is ready for further linguistic, statistical, or machine learning analysis.

2 Simple Logistic Regression Model

When deciding which label should be grouped as "Fake" or "True", we made a few criteria that they need to uphold. For the "True" categorization, we wanted the labels such as "Credible". They adhere to ethical journalistic standards, ensuring thorough source verification before publication. The other labels that come closest to this categorization are "Political", and "Clickbait". "Political" is generally derived from verified sources of factual information, and "Clickbait" often originates from credible sources, but the title presents information in a misleading way. We won't be doing any analysis of the column `title`. So we reasoned that the article itself can be considered as reliable.

For the "Fake" categorization we looked for labels, which twist or misleads in a greater way than "Clickbait". These labels are "Fake News" and "Extreme Bias". Due to its reliance on highly questionable foundations, often associated with propaganda or fake news, "Extreme Bias" is classified as "Fake", and "Fake News" is also categorized as "Fake" due to its misleading or intentionally deceptive nature. The rest of the labels has been removed. They all encompass features or elements that make them unfit for a binary classification such as "Fake" or "True".

By grouping it this way we end up with 238.115 "Fake" and 440.494 "True" labeled 27. This might present some limitations when trying to implement our simple- and advanced model. The model may become biased towards "True" more often, and we risk overfitting since the models sees more true news, it might struggle with finding patterns for fake news.

We will now be implementing and training a simple logistic regression classifier using a fixed vocabulary of the 10,000 most frequent words extracted from the content field, as the input features. We won't be applying TF-IDF weighting to the features. When implementing this model we used some additional hyper-parameters. These are `max_iter=500`, `random_state=42`, `solver="saga"` and `n_jobs=-1`. These hyper parameters are used to determine the maximum number of iterations used when fitting the model. Insuring that we get the same result across different calls. Run the logistic regression faster, both through the solver "saga", which sci-kit recommends for large data sets, and by using all available processors.

With this model we were able to achieve an F1-score of 0.9239. In this model, we have removed "num" from the data frame. When exploring the cleaned data, we noticed that this particular word absolutely dominated every category. (See Figures 11 to 22). We therefor reasoned, that this word doesn't provide any insightful value to the model.

Additionally, we tried incorporating `meta_keywords` in our simple model. This is the only meta-data feature, which is consistently present throughout the data, with a missing percentage of 3.90%, where the rest were missing in almost half. By implementing this additional feature, we were able to push our F1-score to 0.9252. For this model we used the same hyperparameters as in the model without meta-data features.

We finally tried to incorporate the extra reliable data that we got during Graded Exercise 2. To do this, we first had to convert it into the same format as `cleaned_file.csv`. I.e. creating corresponding columns, labeling each article as "True", cleaning the file and removing "num" from the data. After this we could simply combine the rows from the two files. Running our simple logistic regression model with these additional articles yielded a F1-score of 0.9245. We again used the same hyperparameters. Even though this improved our model, we will not be using the additional scraped data for our advanced model. The reason for this, is the gap in years. The `FakeNewsCorpus` dataset is from 2020, but the additional data is from 2025. We were afraid that the gap in years would introduce unnecessary noise. It is reasonable to assume there have been an evolution in topics of news, and the language between 2020 and 2025. By not including the additional data we can maintain a more controlled dataset, where there is no risk of overfitting the model with features or topics that are only relevant today.

3 Advanced Model: TF-IDF and SGDClassifier

To develop a more advanced model beyond logistic regression, we considered several restrictions and goals for improving classification performance. After exploring and testing various approaches, we implemented a model based on TF-IDF and `SGDClassifier`, both available through the scikit-learn library.

TF-IDF Feature Extraction

The TF-IDF (Term Frequency–Inverse Document Frequency) vectorizer transforms text data into numerical features by measuring how important a word is to a document relative to a corpus. Term Frequency counts the number of times a word appears in a document, while Inverse Document Frequency reduces the weight of words that appear in many documents, thus emphasizing more informative, rarer terms.

We chose TF-IDF because it prioritizes distinctive word patterns rather than frequently occurring terms. To optimize performance, we limited the feature extraction to a subset of 50,000 articles. Words appearing in fewer than 5 documents or in more than 60% of documents were excluded. These thresholds help avoid both rare noise and overly common, non-discriminative terms. When extracting features from more than 250,000 articles, we observed longer runtimes without any meaningful performance improvements, and in some cases, reduced accuracy due to class imbalance in word usage. Additionally, we configured the TF-IDF vectorizer to extract unigrams, bigrams, and trigrams using `ngram_range=(1,3)`. This allows the model to capture not only individual words but also short sequences, which is essential for detecting contextual patterns in fake news. Our parameter settings were further validated using `GridSearchCV`, which tested 180 combinations of TF-IDF restrictions on a sample of 50,000 articles. The hypertuning through `GridSearchCV` has been implemented alongside our logistical argumentation for these parameters, which resulted in the given parameters for our advanced model.

Classification with SGDClassifier

After vectorizing the text using TF-IDF, we applied the `SGDClassifier`. This is a linear model trained using stochastic gradient descent—an efficient, iterative method suited for large-scale, high-dimensional, and sparse data. One of its main advantages is its low memory usage and scalability. The classifier supports several loss functions, including hinge loss (for linear SVMs) and log loss (for logistic regression). For our fake news classification task, the TF-IDF and `SGDClassifier` combination offers an interpretable, efficient, and effective solution for handling large datasets like the FakeNewsCorpus.

Handling Class Imbalance

We tested dataset balancing techniques such as SMOTE (oversampling) and `imblearn`'s undersampling methods. However, these approaches often resulted in higher accuracy for fake news compared to true news, which we deemed undesirable. In practical applications, it is more acceptable to misclassify fake news as true than to falsely label true news as fake. Therefore, we chose to retain the original class distribution. Model performance was evaluated using standard metrics, including accuracy, recall, and F1-score, providing a comprehensive assessment of classification performance in the presence of class imbalance.

Conclusion

The implemented model effectively combines preprocessing, contextual feature engineering, and a scalable linear classifier. This pipeline delivers solid classification performance and serves as a reliable foundation for future extensions in fake news detection and broader text classification tasks.

4 Evaluation

The Simple Model on the FAKE dataset has accuracy of 0.89, with a F1-scores of 0.84 for label 0 and 0.92 for label 1. The Advanced Model has a slightly worse accuracy of 0.87 and slightly lower F1-score for label 1. Overall, the Simple Model outperforms the Advanced Model by a small margin.

Label	Precision	Recall	F1-score
0	0.88	0.80	0.84
1	0.90	0.94	0.92
Accuracy			0.89
Macro avg	0.89	0.87	0.88
Weighted avg	0.89	0.89	0.89

Figure 1: Model Assessment - Simple Model on FAKE dataset

Label	Precision	Recall	F1-score
0	0.88	0.80	0.84
1	0.86	0.92	0.89
Accuracy			0.87
Macro avg	0.87	0.86	0.86
Weighted avg	0.87	0.87	0.87

Figure 2: Model Assessment - Advanced Model on FAKE dataset

The Simple Model on the FAKE dataset misclassifies 4,670 fake news samples as true and 2,611 true news samples as fake. The Advanced Model improves performance, reducing false positives to 3,124 and false negatives to 1,681, leading to a better balance between precision and recall.

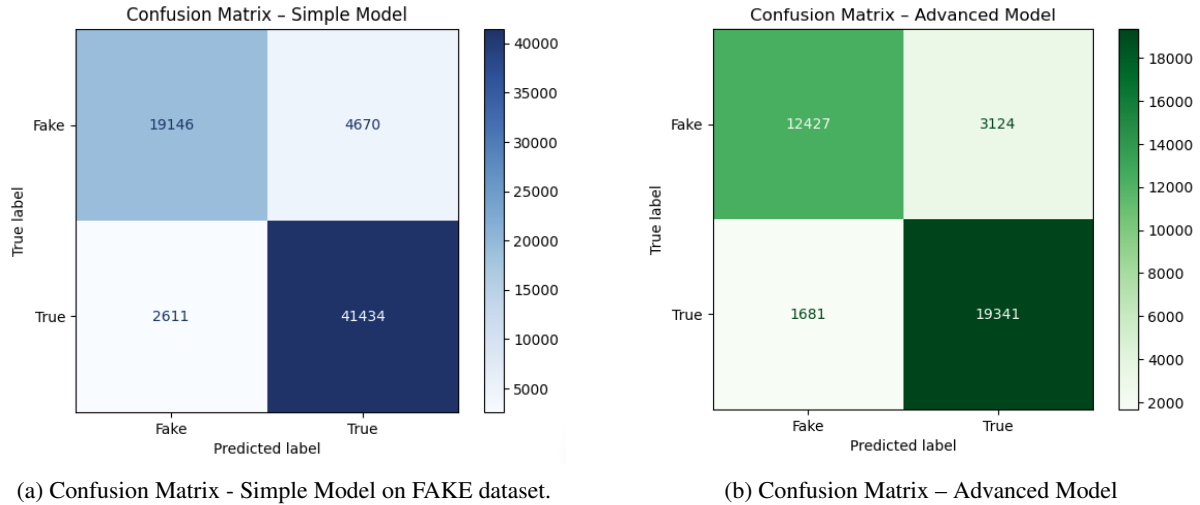


Figure 3: Comparison of Simple and Advanced Models on FAKE dataset.

For the LIAR dataset the simple model has an accuracy of 0.41, and poor F1-scores. 0.24 for label 0, and 0.52 for label 1. On the other hand, the advanced model has an accuracy of 0.57, with a F1-scores of 0.68 for label 1, and 0.33 for label 0. While both models don't perform particularly well, the advanced models performs noticeably better for the LIAR dataset, when compared to the Simple Model.

Label	Precision	Recall	F1-score
0	0.72	0.15	0.24
1	0.37	0.90	0.52
Accuracy			0.41
Macro avg	0.54	0.52	0.38
Weighted avg	0.60	0.41	0.34

Figure 4: Simple model on LIAR testset

Label	Precision	Recall	F1-score
0	0.49	0.25	0.33
1	0.59	0.81	0.68
Accuracy			0.57
Macro avg	0.54	0.53	0.50
Weighted avg	0.55	0.57	0.53

Figure 5: Advanced model on LIAR testset

The Simple Model misclassifies 699 fake news samples as true and 46 true news samples as fake, when testing on LIAR testset. This shows poor recall for the fake class. The Advanced Model improves slightly, reducing false positives to 257, but increases false negatives to 86.

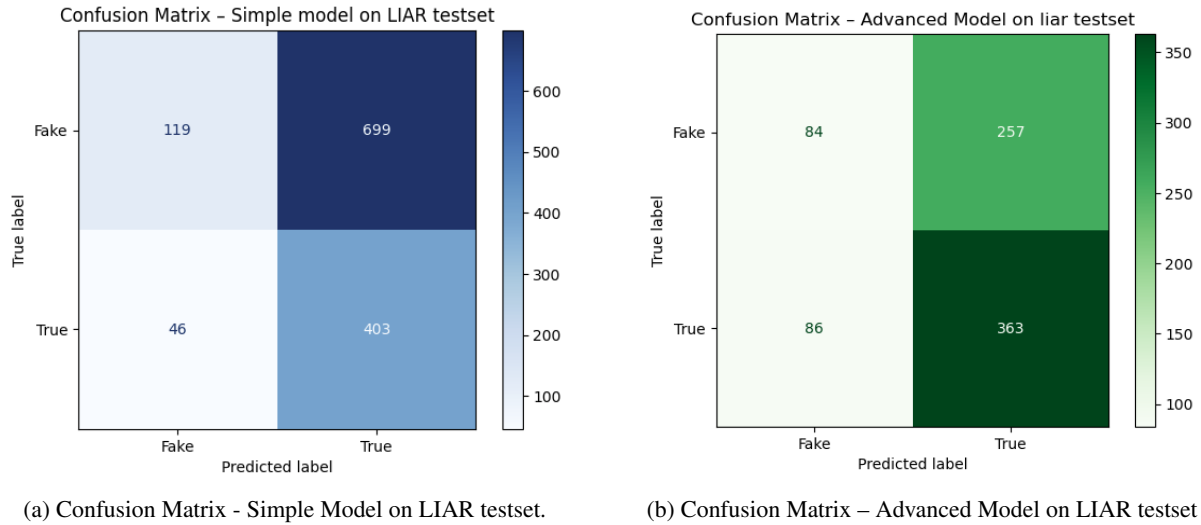


Figure 6: Comparison of Simple and Advanced Models on LIAR testset.

Both the Simple and Advanced Models perform significantly worse on the LIAR dataset than on FakeNews. For example, the Simple Model’s weighted F1-score drops from 0.89 on FakeNews to 0.34 on LIAR, while the Advanced Model’s score decreases from 0.87 to 0.53. On the LIAR test set, the Simple Model misclassifies 699 fake articles as true, demonstrating poor recall for the fake class.

In contrast, on the FakeNews dataset, both models exhibit strong performance—evidenced by high accuracy and balanced confusion matrix results (with the Simple Model correctly identifying 19,146 true negatives and 41,434 true positives, and the Advanced Model achieving 19,341 true positives and 12,427 true negatives). However, both models tend to classify fake articles as true, indicating an inherent bias. These discrepancies likely arise from the LIAR dataset’s shorter, more condensed statements and different vocabulary distributions, which challenge TF-IDF-based approaches.

5 Conclusions

Our experiments revealed a marked performance gap between the FakeNews and LIAR datasets. On the FakeNews test set, both our Simple and Advanced Models achieved high accuracy and robust F1-scores—weighted averages of approximately 0.89 and 0.87, respectively—with balanced confusion matrix results. In contrast, when applied to the LIAR dataset, performance dropped dramatically: the Simple Model’s weighted F1-score fell to 0.34 while the Advanced Model achieved 0.53. For instance, on LIAR, the Simple Model misclassified 699 fake news samples as true, demonstrating notably poor recall for the fake class.

This discrepancy can be largely attributed to inherent differences in the data. The LIAR dataset consists of shorter, more condensed statements with a distinct vocabulary, posing a challenge for TF-IDF-based approaches that were tuned on longer news articles. Moreover, both models exhibited a bias toward predicting articles as true—a tendency that becomes especially problematic when faced with a domain that diverges from the training data.

Our analysis underscores the critical role of data quality and curation in fake news detection. While our models perform admirably on datasets similar to their training set, their limited generalization across domains points to the importance of addressing sample bias and inductive bias during evaluation. This project has demonstrated that strong real-world performance requires more than impressive test set metrics; it necessitates robust preprocessing, careful dataset design, and potentially richer text representations.

Future improvements might involve incorporating advanced methods such as word embeddings or neural networks, which can better capture semantic context and mitigate domain shifts. Ultimately, our work suggests that progress in fake news detection hinges not only on model sophistication but also—perhaps more crucially—on improving data quality and ensuring that the data is representative of the target domain.

References

- [1] scikit-learn developers (BSD License). *Glossary of Common Terms and API Elements*. Available at: <https://scikit-learn.org/stable/glossary.html>.
- [2] NLTK Project. *NLTK — Natural Language Toolkit*. Available at: <https://www.nltk.org/>.
- [3] pandas via NumFOCUS, Inc. *pandas Documentation*. Available at: <https://pandas.pydata.org/docs/index.html>.
- [4] The Matplotlib development team. *Matplotlib*. Available at: <https://matplotlib.org/>.
- [5] Michael Waskom. *Seaborn*. Available at: <https://seaborn.pydata.org/>.
- [6] Python Software Foundation. *collections — Container datatypes*. Available at: <https://docs.python.org/3/library/collections.html>.
- [7] NumPy team. *NumPy*. Available at: <https://numpy.org/>.
- [8] Python Software Foundation. *csv — CSV File Reading and Writing*. Available at: <https://docs.python.org/3/library/csv.html>.
- [9] Python Software Foundation. *re — Regular expression operations*. Available at: <https://docs.python.org/3/library/re.html>.
- [10] Python Software Foundation. *sys — System-specific parameters and functions*. Available at: <https://docs.python.org/3/library/sys.html>.
- [11] GeeksforGeeks. *Stochastic Gradient Descent Classifier (SGDClassifier)*. Available at: <https://www.geeksforgeeks.org/stochastic-gradient-descent-classifier/>.
- [12] GeeksforGeeks. *SVM Hyperparameter Tuning Using GridSearchCV*. Available at: <https://www.geeksforgeeks.org/svm-hyperparameter-tuning-using-gridsearchcv-ml/>.
- [13] GeeksforGeeks. *Understanding TF-IDF: Term Frequency-Inverse Document Frequency*. Available at: <https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/>.
- [14] Let's Data Science. *TF-IDF Explained for Text Analysis*. Available at: <https://letsdatascience.com/tf-idf/>.

Additional Reference Information

The references listed above represent the primary online resources directly cited in our report. In addition, we consulted numerous additional pages and links within these sites (e.g., documentation and explanatory articles for scikit-learn, NLTK, pandas, matplotlib, seaborn, etc.) to deepen our understanding of the libraries and techniques used. While these supplementary resources were instrumental in informing our approach, only the main references are included in our reference list.

6 Figures

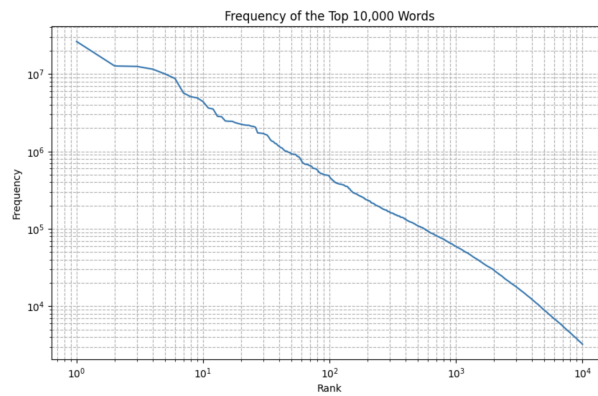


Figure 7: Distribution of Top 10,000 most frequent words before cleaning.

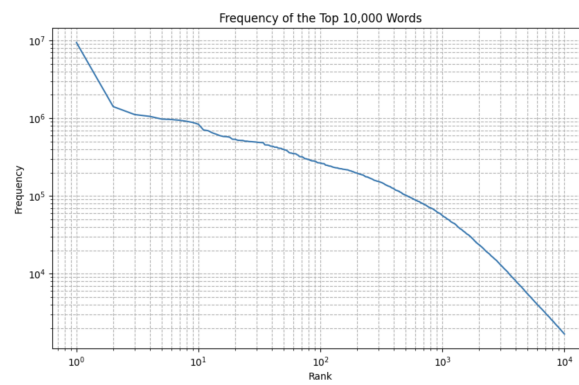


Figure 8: Distribution of Top 10,000 most frequent words after cleaning.

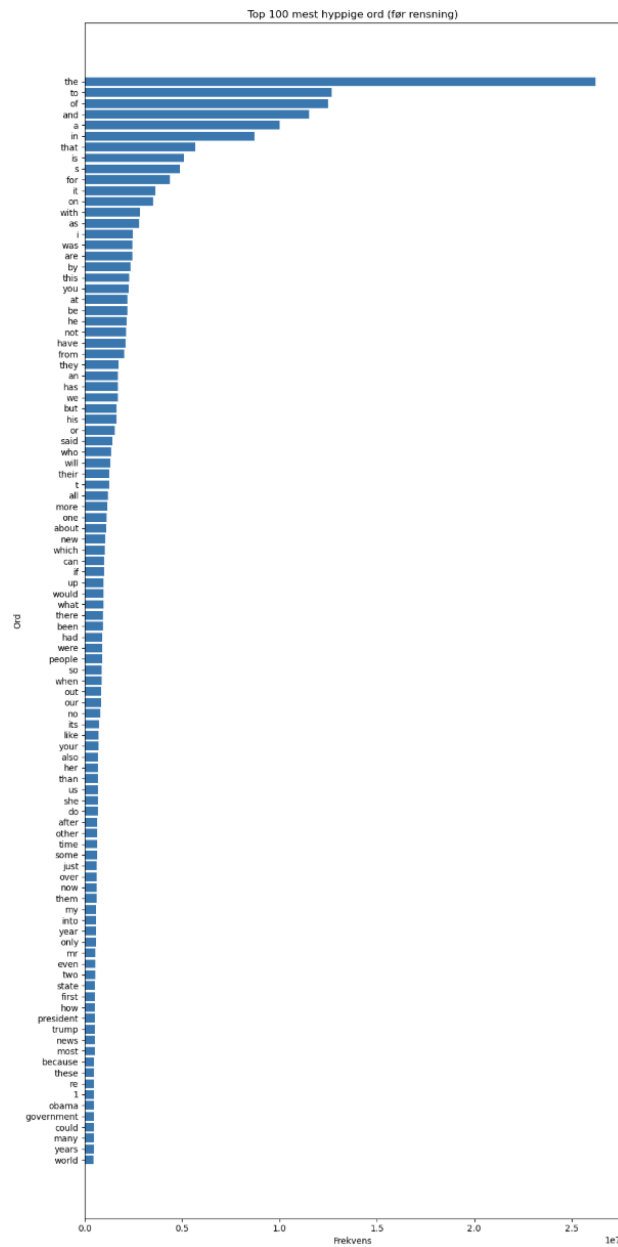


Figure 9: Distribution of Top 100 most frequent words before cleaning.

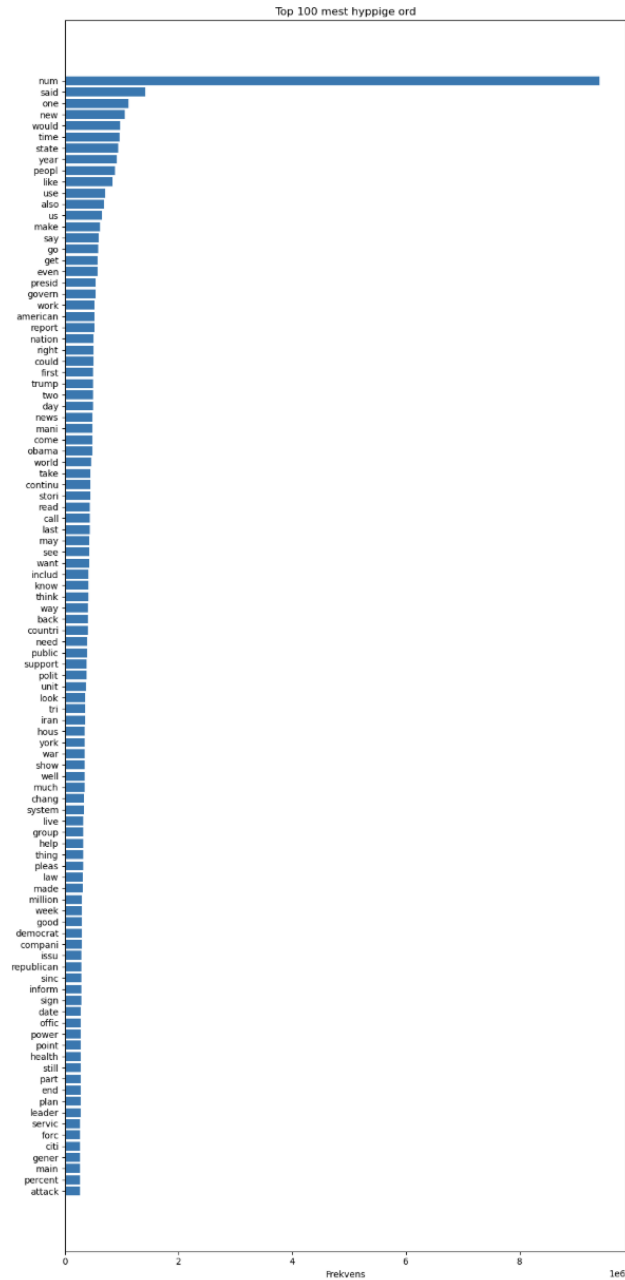


Figure 10: Distribution of Top 100 most frequent words after cleaning.

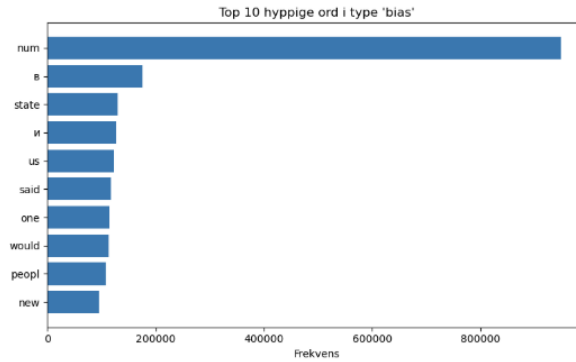


Figure 11: Distribution of Top 10 most frequent words in the type bias.

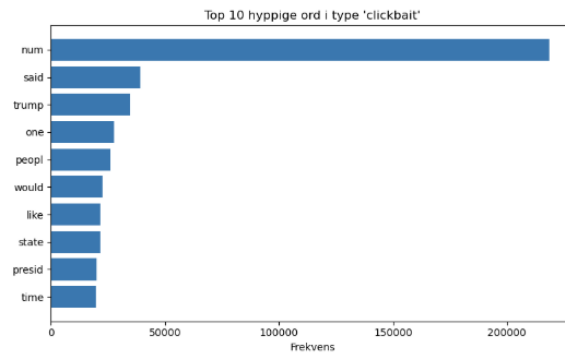


Figure 12: Distribution of Top 10 most frequent words in the type clickbait.

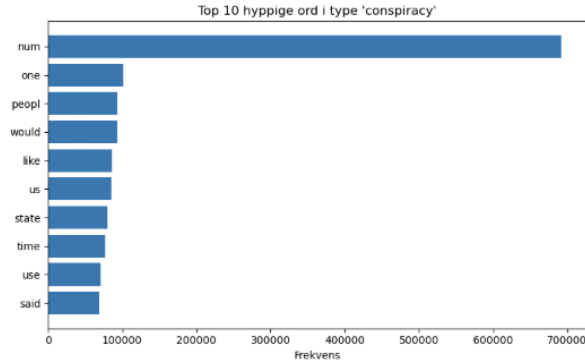


Figure 13: Distribution of Top 10 most frequent words in the type conspiracy.

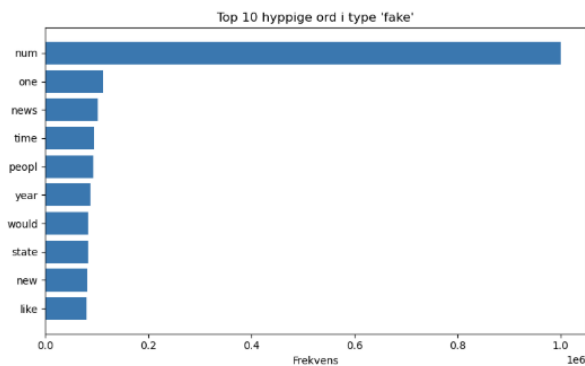


Figure 14: Distribution of Top 10 most frequent words in the type fake.

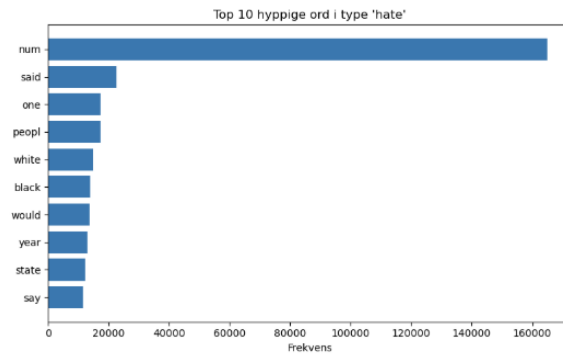


Figure 15: Distribution of Top 10 most frequent words in the type hate.

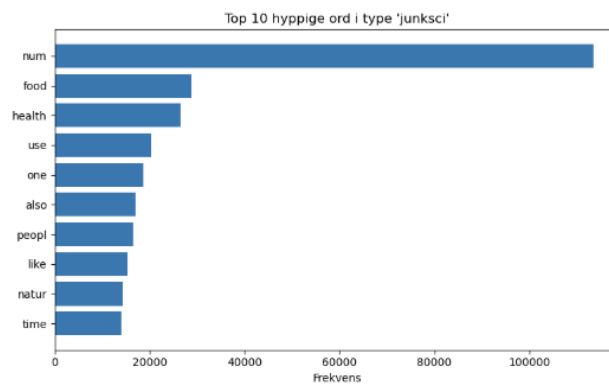


Figure 16: Distribution of Top 10 most frequent words in the type junksci.

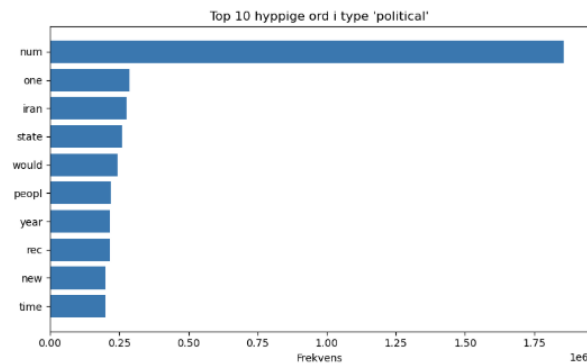


Figure 17: Distribution of Top 10 most frequent words in the type political.

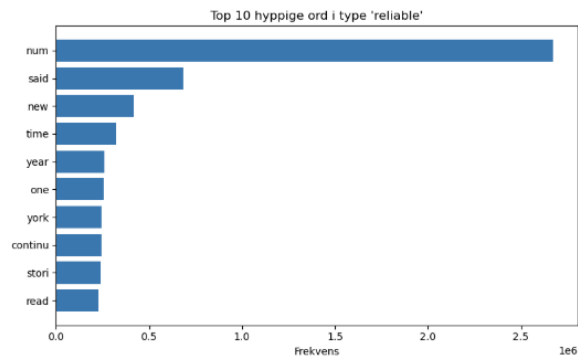


Figure 18: Distribution of Top 10 most frequent words in the type reliable.

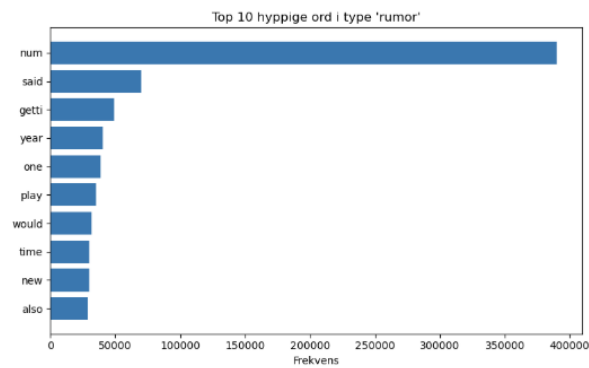


Figure 19: Distribution of Top 10 most frequent words in the type rumor.

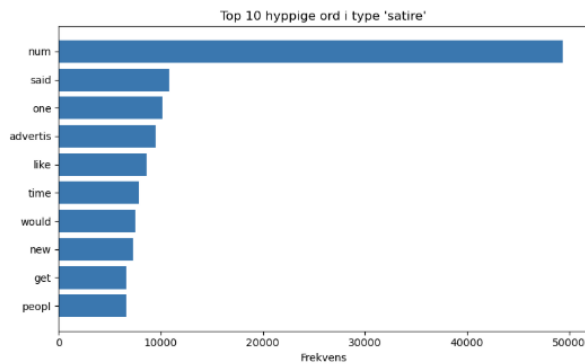


Figure 20: Distribution of Top 10 most frequent words in the type satire.

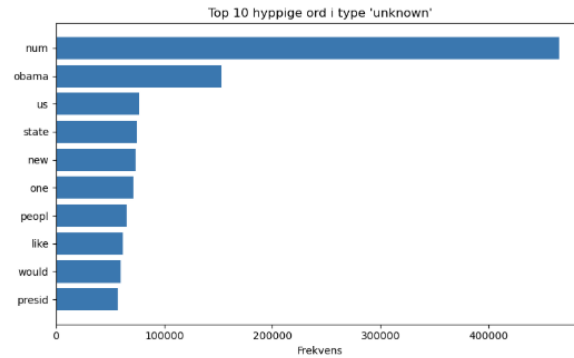


Figure 21: Distribution of Top 10 most frequent words in the type unknown.

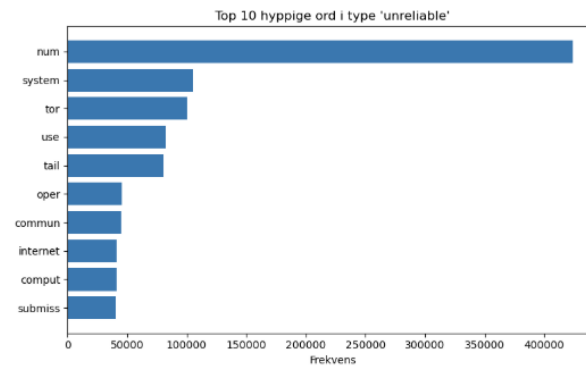


Figure 22: Distribution of Top 10 most frequent words in the type unreliable.

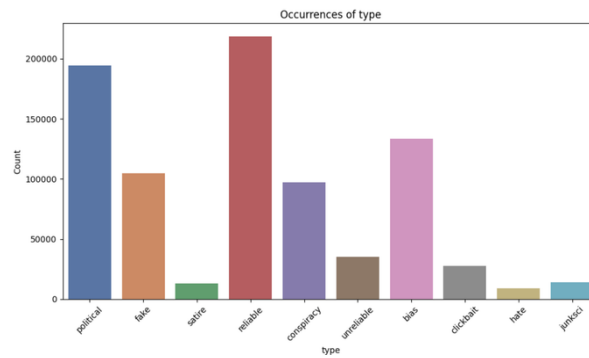


Figure 23: The occurrence of each type before sorting.

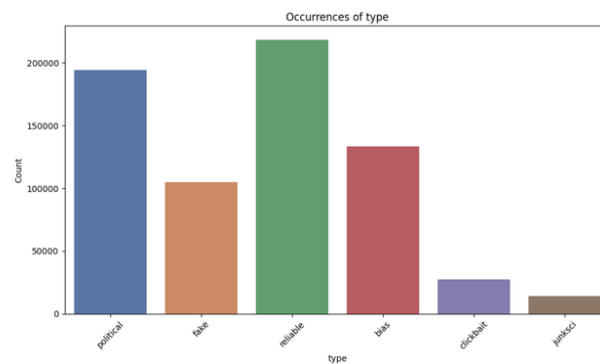


Figure 24: The occurrence of each type after sorting.

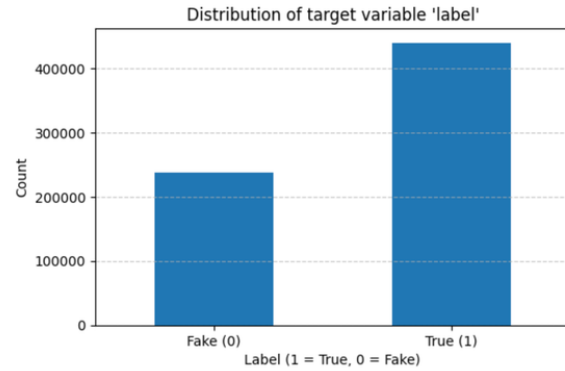


Figure 25: Distribution of the binary target variable `label1`, where 0 = Fake and 1 = True.

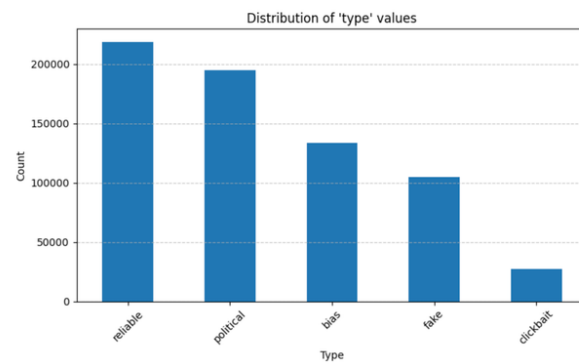


Figure 26: Distribution of articles across the original type categories.

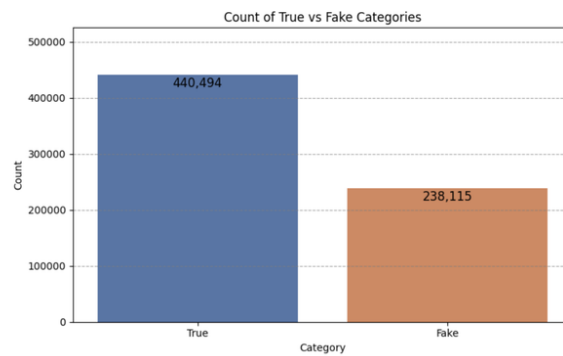


Figure 27: Count of articles grouped into True vs. Fake.

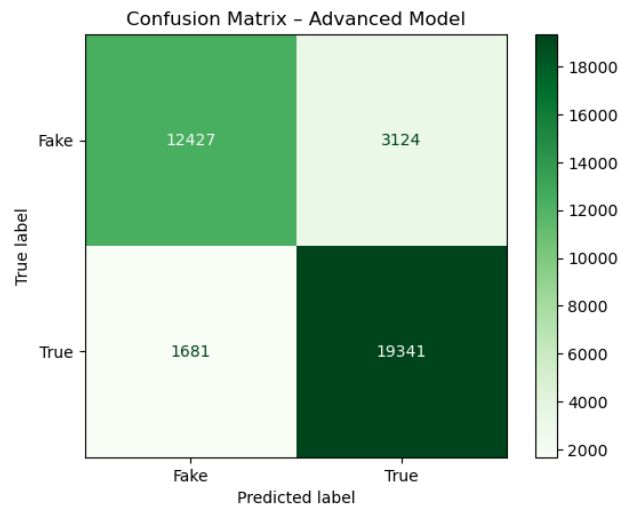


Figure 28: Advanced Model Confusion Matrix.

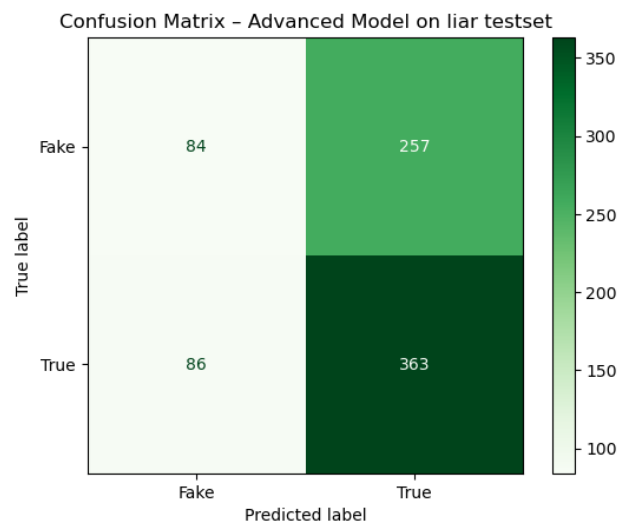


Figure 29: Advanced Model Confusion Matrix on LIAR testset.

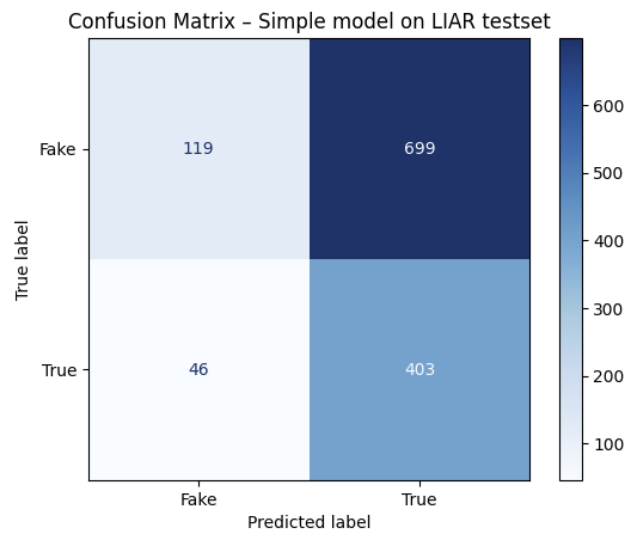


Figure 30: Simple Model Confusion Matrix on LIAR testset.

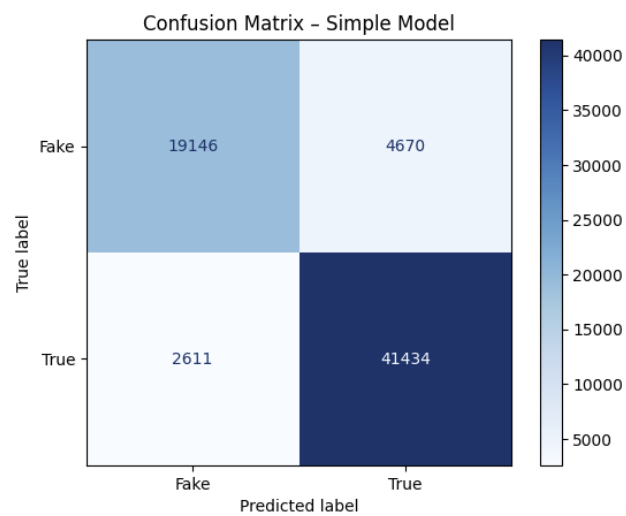


Figure 31: Simple Model Confusion Matrix.