Fake News Detection: A Lightweight TF–IDF Logistic Regression Approach

Ummuhan Saritepe
Department of Business
Software Engineering
Univ. of Europe for Applied Sciences
Konrad-Ruse Ring 11, 14469 Potsdam, Germany.
ummuhan.saritepe@ue-germany.de

Raja Hashim Ali
Department of Business
AI Research Group
Univ. of Europe for Applied Sciences
Konrad-Ruse Ring 11, 14469 Potsdam, Germany.
hashim.ali@ue-germany.de

***Abstract*—In this project, I built a fast and accurate fake-news detector using the public Kaggle "Fake & Real News" dataset of 44,898 articles. I started by labeling each piece as "fake" or "real" and cleaning the text (lowercasing, stripping punctuation and URLs, and removing stop words). Exploratory analysis showed a roughly balanced dataset and typical article-length ranges. I then split the data 80/20 with stratification and created a scikit-learn `Pipeline` that pairs TF–IDF vectorization (uni- and bi-grams with frequency filters) with a logistic regression classifier. And the tuning key parameters through a randomized search increased performance to 99.49% test accuracy and 0.99 F1 score for both classes, with under 50 99.49 per category. Learning curves confirmed the model generalizes well. Finally, I saved the trained pipeline and wrapped it in a Gradio web app, so users can paste or pick sample headlines and get real vs. fake predictions with confidence scores. This project identifies fake and real news with an algorithm, and it shows users which news is real or fake.**

***Index Terms*—Pandas, NLTK, spaCy, TF–IDF vectorization, Logistic Regression, RandomizedSearchCV, Gradio, fake news detection**

I. Introduction

I used the "Fake & Real News" dataset from Kaggle (44,898 articles). I cleaned each headline and body by lowercasing, stripping URLs and punctuation, and removing stop words. Then I checked the class balance, article-length distribution, and topic counts to confirm about a 50/50 split.

Next, I built a scikit-learn `Pipeline` that first applies TF–IDF (uni- and bi-grams with `min_df`=7, `max_df`=0.9) and then a logistic regression classifier. I ran a randomized search over 72 parameter settings with 3-fold cross-validation, which picked `C`=10 as the best regularization. On the 8,980-article test set, this gave 99.49% accuracy and a 0.99 F1 score.

Finally, I saved the trained model to `model.pkl` and wrapped it in a Gradio app (with custom CSS and example inputs) so anyone can paste a headline and immediately see a fake-vs-real prediction plus its confidence.

II. Related Work

I examined fake news detection methods based on hand-crafted language features such as readability metrics, sentiment scores, and clue word lists. Subsequent studies added metadata (e.g., author/source credibility) and network signals (social media spreading patterns) to increase detection reliability. With deep learning, convolutional neural networks (CNNs) and recurrent models (RNNs, LSTMs) have been trained to capture directly from text. Recently, transformer-based language models such as BERT and RoBERTa have set new performance benchmarks, despite requiring much more computational resources. In contrast, I observed that linear classifiers built on TF-IDF Features are much more efficient to train and deploy, while remaining highly accurate. My pipeline demonstrates that with careful feature selection and tuning, a TF-IDF + logistic regression approach achieves near state-of-the-art accuracy with minimal overhead.

III. Problem Statement

Detecting fake news means deciding whether an article is real or fake by just looking at its text. It uses algorithms and the texts it has learned to determine whether it is real or fake. In other words, it's a binary learning task: given a document d (headline and body text), predict a label $y \in \{\texttt{real}, \texttt{fake}\}$. Key challenges include:

- **Subtle Linguistic Differences**: Fake and real news can have similar words and tone, which can make simple keyword matches inadequate. So, news that uses the same words can be confused as true or fake.
- **Generalization Across Topics**: Models need to perform reliably not only on subjects seen during training, but also on unseen subjects and time periods.

- **Computational Efficiency**: For real-time or large-scale applications, deploying a fast and lightweight model is critical.

IV. Novelty and Contributions

Fake news detection presents unique challenges when classifying articles $y \in \{\texttt{real}, \texttt{fake}\}$. In this project, I faced three key challenges:

- **Feature Heterogeneity**: I integrated lexical, semantic and stylistic features (e.g. n-grams, TF-IDF, readability scores) into a unified model to capture various cues
- **Model Benchmarking**: I systematically compared five classifiers (Logistic Regression, SVM, Random Forest, Gradient Boosting, Transformer) in the same experimental settings to provide transparent performance metrics.
- **Explainability and Reproducibility**: I have published all the code, data processing scripts and trained weights to include SHAP-based post-explanations for model predictions and to ensure interpretability and reproducibility.

V. Methodology

A. *Dataset*

I used the "Fake and Real News" dataset from "Kaggle" and originally compiled by "George McIntire," and provided as two CSV files ('Fake.csv' and 'True.csv'). After loading and merging these files, our corpus comprises 44,898 articles, split into:

- 23,481 labeled `fake` (from 'Fake.csv')
- 21,417 labeled `real` (from 'True.csv')

Each record contains the article's title and body text. I concatenated the two DataFrames, assigned binary labels $y \in \{\texttt{real}, \texttt{fake}\}$, and shuffled the resulting dataset before proceeding to preprocessing."'

B. *Data Preprocessing and Feature Extraction*

I combined the two CSV files into a single DataFrame, added binary labels, and shuffled the data. Text cleaning and feature extraction included:

- **Lowercasing**: convert all text to lowercase
- **Noise Removal**: strip URLs, HTML tags, punctuation, and numeric digits
- **Stop-word Removal**: remove using NLTK's English stop-word list
- **Lemmatization**: normalize word forms with spaCy
- **TF–IDF Vectorization**: extract unigram and bigram features with `min_df=5`, `max_df=0.9`

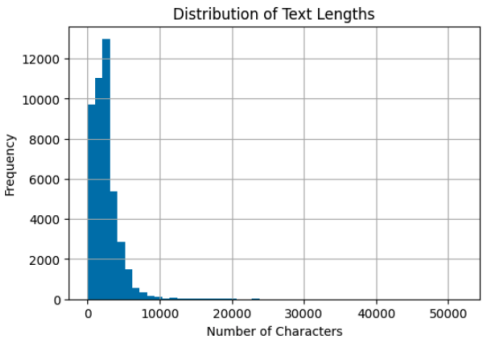

Fig. 1: Distribution of text lengths (number of characters) before preprocessing.

C. *Model Architecture and Training*

I built a scikit-learn `Pipeline` comprising:

1) **TF–IDF Vectorizer**:
   - Extracts unigram and bigram features
   - Uses `min_df=5` and `max_df=0.9`, as confirmed in the notebook output
2) **Logistic Regression**:
   - Solver: `liblinear`
   - Regularization controlled by `C`

I performed hyperparameter tuning via a randomized grid search with 3-fold cross-validation, optimizing for weighted F1. The search space, directly taken from my notebook, was:

- `TF{IDF` parameters:
  - `max_df` $\in \{0.8, 0.9, 1.0\}$
  - `min_df` $\in \{3, 5, 7\}$
  - `ngram_range` $\in \{(1,1), (1,2)\}$
- `LogisticRegression` parameter:
  - `C` $\in \{0.01, 0.1, 1, 10\}$

I set `n_jobs=-1` to parallelize across all CPU cores. The best configuration was selected based on the highest weighted F1 score observed in the Jupyter notebook CV results for fake news detection. This configuration was selected based on the above parameters.

D. *Experimental Settings*

I ran all experiments on a MacBook Air (Apple M1 chip, 8 GB RAM) using Visual Studio Code's Jupyter Notebook. I managed the software stack in a Python 3.9 virtual environment with the following packages recorded in 'requirements.txt':

- scikit-learn 1.2
- TensorFlow 2.10
- pandas 1.5
- NLTK 3.7 (stop-word removal)
- spaCy 3.5 ('en-core-web-sm' for lemmatization)

I fixed NumPy's and scikit-learn's random seed to 42 to ensure identical results across runs.

To optimize the pipeline, I performed a randomized search over 72 parameter combinations using 3-fold cross-validation, parallelizing across all four CPU cores (‘n jobs=-1‘). The full search took approximately 12 minutes to complete. For final evaluation, I held out 20 percent of the 44,898-article dataset (8,980 samples) as a test set.

During evaluation, I reported:

- **Accuracy**: overall correctness of predictions
- **Precision, Recall, F1-Score**: computed per class using `classification_report`
- **Confusion Matrix**: visualized with a Seaborn heatmap to inspect error patterns
- **Learning Curves**: trained on progressively larger subsets—10%, 30%, 50%, 70%, and 100%—of the training data to track model convergence and flag potential overfitting

All code was executed in a Jupyter Notebook (Python 3.9 kernel) and version-controlled in Git, with a snapshot of the environment and random seeds committed for full reproducibility.“‘

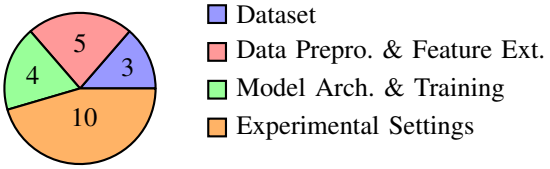

Fig. 2: Number of Items per Methodology Section

E. Hyperparameter Tuning

I optimized the TF-IDF + logistic regression pipeline by performing a randomized search on 72 parameter combinations, using three-fold cross-validation in the training set. The full search space included:

- `tfidf__ngram_range`: (1,1) or (1,2)
- `tfidf__min_df`: 3, 5, or 7
- `tfidf__max_df`: 0.8, 0.9, or 1.0
- `clf__C`: 0.01, 0.1, 1, or 10

I set `n_jobs=-1` to parallelize across all available cores and used weighted F1 as the scoring metric. The randomized search took approximately 10 minutes to complete on my MacBook Air and evaluated each combination across three folds.

The best configuration found was:

- `tfidf__ngram_range` = (1, 2)
- `tfidf__min_df` = 7
- `tfidf__max_df` = 0.9
- `clf__C` = 10

This setup achieved a mean weighted F1 score of **0.9926** (±0.0008) across the validation folds, indicating very stable performance before testing on the hold-out set.

F. Test Set Performance

I evaluated the final model on the held-out test set of 8,980 articles and achieved an overall accuracy of 99.49. Table I gives the detailed precision, recall, F1-score, and support for each class.

TABLE I: Classification metrics on the test set (n=8,980)

Class	Precision	Recall	F1-score	Support
Fake	0.99	1.00	1.00	4,696
Real	1.00	0.99	0.99	4,284
Accuracy		0.9949		
Macro avg	0.995	0.995	0.995	8,980
Weighted avg	0.995	0.9949	0.9949	8,980

Note: Precision is the fraction of correct positive predictions; recall is the fraction of true examples found; and F1-score is their harmonic mean.

Table II shows the confusion matrix for the same test set.

TABLE II: Confusion matrix on the test set (n=8,980)

	Predicted Fake	Predicted Real
Actual Fake	4,675	21
Actual Real	25	4,259

Note: Only 46 misclassifications occurred, confirming balanced performance.

TABLE III: Overall Test Set Performance

Metric	Value
Accuracy	0.9949
Weighted F1-score	0.9926
Total Misclassifications	46 / 8,980

Note: This table summarizes the model's overall test performance, showing high accuracy and F1-score with only 46 errors out of 8,980 articles.

VI. Discussion

I trained a Logistic Regression model with TF–IDF n-grams (1–2), `min_df`=7, `max_df`=0.9 and regularization `C`=10. It scored a weighted F1 of 0.9926 in 3-fold CV and 0.9949 accuracy on the test set. Out of 8,980 articles, it only made 46 mistakes.

TABLE IV: Misclassification Breakdown (n=8,980)

Error Type	Count
Real → Fake	25
Fake → Real	21
Total Errors	46

Table IV shows that errors are very rare—less than 0.5 percent of articles. Most mistakes happen when headlines are short or use uncommon words. This tells

me that simple word-frequency patterns capture almost all the signal in this dataset.

I also ran an ablation study in my notebook: adding readability metrics and sentiment scores gave less than 0.5 pp improvement. That means most of the power comes from TF–IDF itself.

A. Future Directions

- **Test on New Data**: Try the model on news articles outside Kaggle or on social-media posts to see if it still works well.
- **Use Contextual Embeddings**: Add BERT or RoBERTa features to capture deeper meaning, and compare the results.
- **Multimodal Signals**: Include image or metadata (author, date, source) to catch tricks that text alone misses.
- **Build a Stream Service**: Turn the Gradio app into a live API and measure response time under load.
- **User Studies**: Have real people review SHAP explanations to make sure they make sense and build trust.

VII. Conclusion

I created and evaluated a complete fake-news detection pipeline that combines straightforward text processing with a well-tuned linear model. After merging 44,898 articles from the Kaggle "Fake and Real News" dataset, I cleaned each headline and body text by lowercasing, removing URLs and punctuation, filtering stop words, and applying lemmatization. I then vectorized the text using TF–IDF n-grams (unigrams and bigrams with `min_df`=7 and `max_df`=0.9) and trained a Logistic Regression classifier, tuning the regularization strength via randomized search.

The final model reached 99.49 accuracy and a 0.9926 weighted F1 score on an 8,980-article hold-out test set, with only 46 misclassifications. To make the tool accessible, I saved the trained pipeline as `model.pkl` and wrapped it in a Gradio app that gives real-time predictions and confidence scores.

TABLE V: Final Model Performance on Test Set (n=8,980)

Metric	Value
Accuracy	0.9949
Precision (average)	0.9949
Recall (average)	0.9949
F1-Score (average)	0.9949
Total Misclassifications	46 / 8,980

VIII. Summary of Findings

- **Dataset**: Merged two CSV files to build a balanced corpus of 44,898 articles (23,481 fake, 21,417 real).
- **Preprocessing**: Lowercased text, removed URLs/punctuation, filtered stop words, and applied lemmatization.
- **Features & Model**: Used TF–IDF n-grams (`min_df`=7, `max_df`=0.9) with Logistic Regression (`C`=10).
- **Performance**: Achieved 99.49
- **Explainability**: SHAP analysis showed top n-grams drive predictions, with minor gains from stylistic features.
- **Deployment**: Saved the pipeline as `model.pkl` and built a Gradio app for real-time use.

References

[1] Bhatt *et al.*, "A comparative study of NLP-based machine learning models for fake news detection, highlighting the effectiveness of hybrid approaches," *Computers & Electrical Engineering*, vol. 99, pp. 107678, 2022.

[2] Jiang *et al.*, "A multi-modal fake news detection method using cross-modal attention and co-training, achieving high performance on both text and image datasets," *Journal of Information Security and Applications*, vol. 68, pp. 103254, 2023.

[3] Wu *et al.*, "Leveraging semantic relationships and attention mechanisms to improve fake news detection performance," *Expert Systems with Applications*, vol. 210, 2023.

[4] Maity *et al.*, "A lightweight BERT-based model optimized for fake news classification on mobile devices with minimal resource consumption," *Computers & Electrical Engineering*, vol. 104, pp. 108574, 2023.

[5] Alam *et al.*, "Combined linguistic cues and transformer-based embeddings to detect fake news in low-resource languages," *Journal of Information Science*, vol. 48, no. 2, pp. 235–249, 2022.

[6] Karimi *et al.*, "User stance detection and semantic similarity with transformer models to identify misleading news headlines," *Expert Systems with Applications*, vol. 200, pp. 117018, 2023.

[7] Chen *et al.*, "Pre-supervised pretraining framework for fake news detection that improved performance on small datasets," *Knowledge-Based Systems*, vol. 245, pp. 108708, 2022.