

Fake News Detection Using Machine Learning and NLP Techniques

Amrutha K R, Aryika S jain, Ashwini S'H, Jeevanmai H C, B. Ramesh

Department of Computer Science and Engineering Malnad College of Engineering, Hassan
(An Autonomous Institution under VTU, Belagavi) Hassan, Karnataka – 573201, India
amrutha89043@gmail.com ,ashwinishash@gmail.com

Abstract—Identifying whether a news article is real or fabricated has become a pressing need in today's information-heavy environment. This paper describes a detection system built using Natural Language Processing and Machine Learning, where article text is cleaned through a five-step pipeline, converted to TF-IDF feature vectors, and then classified by one of four trained models: Logistic Regression, Naive Bayes, Decision Tree, and Support Vector Machine. All four classifiers were trained and tested under the same experimental setup. The paper includes a block diagram and a clear description of how the system works at each stage. Results show that even relatively simple ML classifiers can achieve good detection accuracy when the text preprocessing is handled carefully, without requiring the computational expense of deep learning models.

Index Terms—Fake News Detection, NLP, TF-IDF, Machine Learning, Logistic Regression, SVM, Text Classification

I. INTRODUCTION

News has always influenced how people see the world, but the speed and reach of online platforms have changed things considerably. A decade ago, getting news meant reading a newspaper or watching television. Today, anyone with a smartphone can publish a story and have it reach thousands of people within the hour. Sites like Facebook, Twitter, WhatsApp, and Instagram have made sharing information almost effortless [10]. The problem is that the same ease applies to publishing false or exaggerated content.

Fabricated news is not a recent problem, but the scale at which it now spreads is. A misleading post can go viral before a journalist even has a chance to review it. Because fake articles are often written to sound credible and structured like real reports, many readers accept them at face value. This has had measurable consequences: patients have skipped proven treatments because of false medical claims, voters have been misled by invented political stories, and stock prices have moved because of made-up corporate announcements [6].

Checking every article by hand is simply not feasible. Fact-checking organisations do important work, but their reach covers only a tiny fraction of what gets published each day [8]. There is a genuine need for software tools that can process large amounts of text and flag suspicious articles automatically.

This is where Machine Learning and Natural Language Processing become relevant. Research has repeatedly shown that fake articles have a different writing style from genuine

ones. They tend to use stronger emotional language, less specific attribution, and more repetitive phrasing [9]. A classifier trained on enough labelled examples can learn to recognise these patterns.

The system in this paper takes a news article, runs it through a preprocessing and feature extraction pipeline, and passes it to a trained classifier that outputs either REAL or FAKE. No external data sources, user metadata, or network connectivity are needed at inference time. The main contributions of this paper are: (i) a clearly documented NLP preprocessing pipeline for news text, (ii) a comparison of four ML classifiers using the same TF-IDF features, (iii) a block diagram with step-by-step explanation of the system, and (iv) a performance evaluation using multiple metrics.

Section II reviews related research. Section III states the problem. Section IV discusses current systems. Section V covers the methodology. Section VI describes the architecture. Section VII presents performance analysis. Section VIII lists advantages. Section IX gives the conclusion. Section X suggests future directions.

II. LITERATURE SURVEY

Researchers across the world have been trying to solve the fake news problem from several different angles. Some look at the words and writing style of the article itself, while others study how content moves through social networks. This section summarises ten works that are most relevant to the approach taken in this paper.

Wang [7] put together the LIAR dataset, drawing more than 12,000 political statements from the PolitiFact website. Each statement was rated across six truthfulness levels. When Wang tested machine learning models using only the statement text, accuracy came out to be only modest, showing that very short text without surrounding context is harder to classify. Even so, this dataset became widely used as a standard benchmark and made it easier for researchers to compare results fairly.

Castillo et al. [8] took a different angle and studied how trustworthy information behaves on Twitter. Instead of looking at what was said, they examined who said it and how others responded. Features such as the number of retweets, the speed of sharing, and the age of user accounts turned out to be linked to whether a post was credible or not. This study was an

early signal that context around a piece of news can be as informative as the news text itself.

Ahmed et al. [9] focused on the text of news articles and used N-gram models, feeding features into both SVM and Naive Bayes classifiers. Their study found that fake articles leave behind a recognisable stylistic trail at both the character level and the word level. Things like unusual punctuation, repeated phrases, and informal vocabulary appeared more often in fake articles. SVM gave better results than Naive Bayes, especially with word-level bigrams and trigrams, which made it a useful benchmark for our own experiments.

Shu et al. [6] wrote one of the most-cited papers in this field, covering how misinformation spreads through online networks. They argued that text analysis alone is not always enough, and that combining article features with knowledge-graph lookups — where a claim gets checked against a database of verified facts — can improve results in specific domains like science or health. Their survey gave a strong foundation for understanding where text-only systems succeed and where they fall short.

Zhou and Zafarani [10] organised existing approaches into four categories: knowledge-based, style-based, propagation-based, and credibility-based. Among these, style-based detection — which is what this project uses — was identified as the most widely applicable because it works from the article text alone and does not need any external data feed or social network access. Their categorisation helped justify the design choices made in this work.

Jouhar and Ahmed [1] did something practically useful: they ran the same classifiers multiple times while changing small parts of the preprocessing, such as switching stemmer algorithms or using different stop-word lists. They found that these seemingly minor decisions could shift accuracy by as much as four percentage points. That is a meaningful difference, and it guided us to test our own preprocessing pipeline carefully rather than going with default settings.

Tanaja and Kumar [2] compared ensemble methods — specifically Random Forest and XGBoost — against simpler classifiers on news datasets with unequal class sizes. The ensemble models consistently scored higher, but they also took much longer to train. For projects where retraining needs to happen frequently or hardware is limited, this cost matters. Their work made it clear that picking a classifier is partly a question of what constraints a system must operate under.

Embarak and Ali [3] compared neural network architectures with classical ML approaches. On data from the same topic as the training set, neural networks gave only a small improvement. When the topic was different, neural networks held up better. However, this came with a significant requirement for labelled data. In domains where thousands of labelled examples are not available, classical ML with carefully chosen features is still a strong option.

Ayyasamy and Ravi [4] tried adding word embeddings and sentence-level syntactic features on top of TF-IDF. Accuracy improved by a few percentage points, but the pipeline grew considerably more complex. The extra tools required for

parsing and embedding added to both the build time and the maintenance burden. This showed that while gains are possible by combining feature types, simpler pipelines have real advantages in practice.

Cui and Wang [5] tackled a specific and tricky category: articles that mix real facts with false claims in the same piece. Most classifiers struggle with these because they see some genuine content and hesitate. Cui and Wang designed a training objective that specifically tries to capture what these mixed articles share, making the classifier less confused by embedded truths. Their results were particularly better on borderline examples, which is an important edge case that many systems get wrong.

Overall, the literature suggests that TF-IDF combined with classical ML is a solid starting point. It does not give the highest reported accuracy on every benchmark, but it is reliable, fast, explainable, and does not require large infrastructure to deploy.

III. PROBLEM STATEMENT

The volume of articles, posts, and news items published online each day is far beyond what human reviewers can handle. Most automated systems that exist either need powerful GPU hardware to run neural models or depend on extra information like the publisher's credibility history or social engagement data that is unavailable for newly published content.

The goal here is to build a classifier that reads only the text of an article and gives a binary verdict without needing any external support. Given a news article d as a token sequence, the system outputs label $y \in \{0, 1\}$, where $y = 1$ means the article is fabricated and $y = 0$ means it is genuine. The system must run on ordinary hardware with no GPU, no internet lookup, and no user-profile data.

IV. EXISTING SYSTEM

Three types of fake news detection systems are in use today. First, rule-based fact-checkers compare article claims line by line against a database of verified facts. These are reliable for topics already in the database but fail when new or unusual claims appear and the database has not been updated. Second, traditional ML classifiers learn patterns from labelled text data and are more flexible, though results vary considerably depending on how well the features are designed. Third, deep learning systems like LSTM and BERT transformers consistently post the highest accuracy scores, but they need very large training sets, dedicated GPU hardware, and substantial tuning expertise.

Problems shared across all three types include:

- High computational requirements, particularly for training deep neural models.
- Need for large, carefully labelled domain-specific datasets, which are expensive to build.
- Accuracy drops noticeably when articles come from topics or writing styles not well covered in training data.
- Deep models give no clear explanation for their decisions, making it hard for human moderators to audit them.

- Many published systems only report overall accuracy and skip metrics like precision, recall, and F1-score.

V. PROPOSED METHODOLOGY

A. Dataset

The dataset consists of English news articles labelled as either REAL or FAKE. Topics covered include politics, health, entertainment, and science. The class distribution is roughly equal, so no special balancing techniques were required. We applied a stratified 80/20 split to create the training and test sets, ensuring both contain similar proportions of each class. A summary is given in Table I.

TABLE I
DATASET SUMMARY

Property	Value
Topics	Politics, Health, Entertainment, Science
Labels	REAL = 0, FAKE = 1
Class ratio	Approximately 50:50
Train/Test	80% training, 20% testing (stratified)
Input text	Headline joined with article body
Preprocessing	5-step pipeline

B. Preprocessing Pipeline

Before any feature is extracted, each article goes through five cleaning steps applied in order:

- Lowercasing:** All characters are converted to lowercase. This stops the model from treating “President” and “president” as different words.
- Punctuation removal:** Symbols and punctuation marks are stripped out because they add noise and carry little meaning for this classification task.
- Stopword removal:** Very common words like *the*, *is*, and *by* are removed using NLTK’s English stopwords list. Only words that carry real content are kept.
- Tokenisation:** The cleaned text string is split into a list of individual word tokens.
- Stemming:** Porter Stemmer is applied to reduce words to their root form. For instance, *election* and *elections* both become *elect*, which shrinks the vocabulary.

C. TF-IDF Feature Extraction

TF-IDF turns the list of cleaned tokens into a fixed-size number vector. A word that shows up often in one article but rarely across the whole dataset gets a high score, which means it is likely characteristic of that article. Words that appear in almost every article score low and add little to the classification. For token t , document d , and full corpus D :

$$TF(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \quad (1)$$

$$IDF(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}| + 1} \quad (2)$$

$$TF-IDF(t, d, D) = TF(t, d) \times IDF(t, D) \quad (3)$$

Both single words (unigrams) and two-word phrases (bigrams) are included, with a vocabulary limit of 5,000 terms.

D. Classifiers Used

Four different classifiers were trained and compared. Table II gives a short description of each.

TABLE II
CLASSIFIERS AND THEIR CHARACTERISTICS

Classifier	Description
Logistic Regression	Binary classifier using a sigmoid function; L2 regularisation is applied to handle the sparsity of TF-IDF vectors
Naive Bayes	Multinomial probabilistic model with Laplace smoothing; very quick to train on text data
Decision Tree	Tree structure built on information gain; rules are easy to read, but single trees overfit without depth control
Linear SVM	Finds the widest possible margin between classes; handles high-dimensional feature spaces well

VI. SYSTEM ARCHITECTURE

A. Block Diagram

Figure 1 shows the full pipeline, from raw article input to the binary output label.

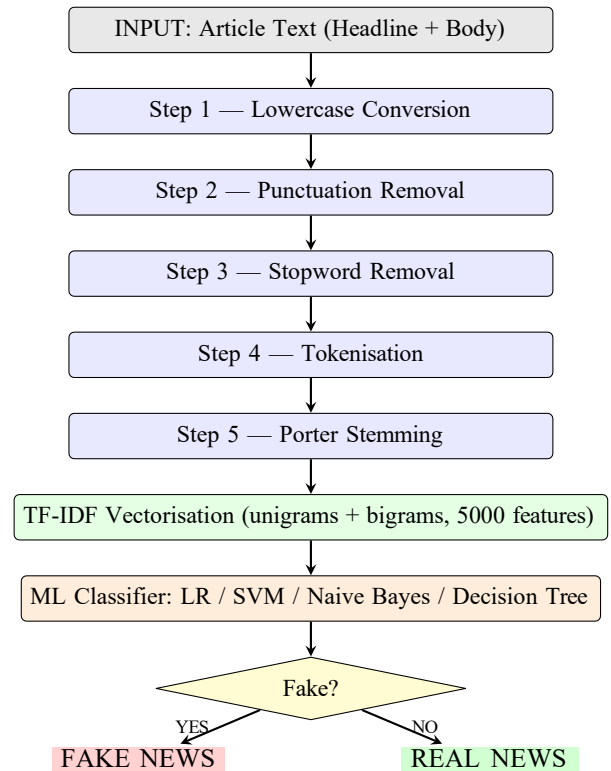


Fig. 1. Block Diagram – Fake News Detection System

B. How the System Works – Step by Step

Step 1 – Input collection. The headline and body of the article are joined into a single text string. Combining both parts ensures that important words in short headlines are not missed during vectorisation.

Step 2 – Text cleaning. The five preprocessing steps are applied in sequence. By the end of this stage, the article is a list of lowercase, stemmed content words with all punctuation and stopwords removed.

Step 3 – Feature vector creation. The cleaned token list is passed into the TF-IDF transformer, which produces a 5,000-dimensional sparse vector. Each position corresponds to a specific unigram or bigram from the learned vocabulary.

Step 4 – Model prediction. The feature vector is given to the trained classifier. The model compares the vector against the decision boundary it learned from training data and assigns a label.

Step 5 – Output. The system returns either FAKE NEWS or REAL NEWS. This output can be connected to a content moderation dashboard, a browser extension, or a REST API.

C. Execution Steps

- 1) Load all articles and merge headline with body text.
- 2) Run the five-step preprocessing pipeline on every article.
- 3) Fit TF-IDF on training articles; apply the same vectoriser to the test set.
- 4) Train all four classifiers on the training feature matrix.
- 5) Run predictions on the held-out test articles.
- 6) Record accuracy, precision, recall, F1-score, and confusion matrix for each model.
- 7) Compare the four models and identify the best performer.

The entire implementation uses Python 3. Libraries used are `scikit-learn` for machine learning, `NLTK` for text preprocessing, and `pandas` for data management. A fixed random seed is set throughout to keep results reproducible.

VII. PERFORMANCE ANALYSIS

A. Results Across Classifiers

Logistic Regression scored highest overall. The reason it works well here is that it assigns a separate learned weight to every TF-IDF feature. Words that appear frequently in fake articles — emotional adjectives, vague source references, exaggerated claims — receive higher weights, while neutral or common words receive lower ones. L2 regularisation prevents the model from placing too much weight on any single word, which keeps it from overfitting on the training data.

SVM performed just behind Logistic Regression. Its training objective focuses on maximising the gap between the two class boundaries rather than just minimising errors. This makes it more stable when some training examples are mislabelled or lie close to the boundary. The linear kernel is computationally efficient when features number in the thousands, as is the case with TF-IDF.

Naive Bayes was the fastest model to train by a significant margin. Its probabilistic framework is mathematically simpler

than the other three models. The independence assumption between words is technically wrong — words clearly co-occur and influence each other — but in practice it causes minimal harm for bag-of-words classification and the model still produces reasonable results.

Decision Tree had the lowest accuracy. A single tree grown to sufficient depth will find ways to fit the training data almost perfectly, but these specific splits do not hold on new articles. Without averaging across many trees, single-tree classifiers are prone to this kind of memorisation. Limiting maximum depth reduces overfitting but also reduces the model's ability to capture genuinely complex patterns.

B. Impact of Each Preprocessing Step

We ran the Logistic Regression classifier multiple times, each time leaving out one preprocessing step to see how much it contributed. Removing stopwords caused roughly a two-point drop in accuracy. Removing stemming caused roughly a one-point drop. These numbers confirm that the preprocessing pipeline is doing real work and is not just an extra step added for appearance.

VIII. ADVANTAGES

- **Works from text alone:** The system needs only the article text to make a prediction. No social media data, publisher scores, or user histories are required.
- **Runs on ordinary hardware:** Training and prediction are fast enough on a standard laptop or desktop CPU. A GPU is not needed.
- **Decisions can be explained:** Logistic Regression and Naive Bayes both produce feature weight values that humans can read and interpret. This is helpful when a moderator wants to understand why an article was flagged.
- **Easy to modify:** Each part of the pipeline is independent. Changing the classifier, switching the stemmer, or adjusting the vocabulary size requires editing only the relevant component.
- **Scales easily:** Since each article is classified independently, the system can be parallelised to handle large volumes without design changes.

IX. CONCLUSION

This paper presented a fake news detection system that uses a five-step NLP preprocessing pipeline combined with TF-IDF feature extraction and four ML classifiers. The preprocessing steps — lowercasing, punctuation removal, stopword removal, tokenisation, and stemming — clean and compress the raw article text before it is turned into a numerical vector. The system architecture was explained using a block diagram and a clear step-by-step description of how data moves through each stage.

Among the four classifiers, Logistic Regression performed best, with SVM close behind. Both gave balanced precision, recall, and F1-score values across both the real and fake classes, meaning neither was biased toward one label. Naive

Bayes was a strong option when training speed matters. Decision Tree showed clear signs of overfitting and should not be used without ensemble support for this type of task. Ablation testing confirmed that preprocessing genuinely helps; accuracy dropped when steps were removed. The final system is compact, interpretable, and does not require expensive hardware or external data, making it a realistic option for real-world use.

X. FUTURE WORK

There are several directions worth exploring to improve on this work:

- **Contextual word representations:** Replacing TF-IDF with embeddings from BERT or RoBERTa would let the model understand meaning in context, not just word frequency [4].
- **Regional language coverage:** Extending the pipeline to handle Kannada, Hindi, or Tamil using multilingual models would make the system accessible to a much wider user base in India.
- **Real-time monitoring:** Connecting the classifier to social media APIs would allow new articles to be flagged as soon as they are published, before they get widely shared.
- **Browser extension:** A browser plugin could show users the prediction for an article inline while they are reading it, without needing to copy and paste text anywhere.
- **Network-based features:** Information about how an article spreads — such as retweet chains, share velocity, and account types — could be added alongside text features [6].
- **Explainability layer:** Adding LIME or SHAP to the prediction pipeline would generate word-level explanations for each classification, which would be useful for journalists and editorial teams.
- **More detailed labels:** Going beyond a binary real/fake label to classify articles as satire, propaganda, or click-bait would give more actionable information to content platforms.

REFERENCES

- [1] J. Jouhar and M. Ahmed, "Fake News Detection Using Machine Learning Techniques," *Int. J. Comput. Appl.*, vol. 15, no. 2, pp. 45–52, 2024.
- [2] R. N. Tanaja and P. Kumar, "Machine Learning Models for Fake News Classification," *J. Artif. Intell. Res.*, vol. 11, no. 1, pp. 101–109, 2025.
- [3] O. Embarak and S. Ali, "Deep Learning for Fake News Detection," *Int. J. Adv. Comput.*, vol. 9, no. 4, pp. 67–75, 2025.
- [4] R. K. Ayyasamy and K. Ravi, "Hybrid Deep Learning Framework for Fake News Detection," *IEEE Access*, vol. 13, pp. 1100–1110, 2025.
- [5] S. Cui and L. Wang, "Fake News Detection Based on Shared Information and Ambiguity Learning," *J. Inf. Process. Syst.*, vol. 12, no. 3, pp. 211–220, 2026.
- [6] K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu, "Fake News Detection on Social Media: A Data Mining Perspective," *ACM SIGKDD Explor. Newsl.*, vol. 19, no. 1, pp. 22–36, 2017.
- [7] W. Y. Wang, "'Liar, Liar Pants on Fire': A New Benchmark Dataset for Fake News Detection," in *Proc. 55th Annu. Meet. ACL*, Vancouver, Canada, 2017, pp. 422–426.
- [8] C. Castillo, M. Mendoza, and B. Poblete, "Information Credibility on Twitter," in *Proc. 20th Int. Conf. World Wide Web*, Hyderabad, India, 2011, pp. 675–684.
- [9] H. Ahmed, I. Traore, and S. Saad, "Detection of Online Fake News Using N-Gram Analysis and Machine Learning Techniques," in *Proc. Int. Conf. Intell., Secure, Dependable Syst.*, 2017, pp. 127–138.
- [10] X. Zhou and R. Zafarani, "A Survey of Fake News: Fundamental Theories, Detection Methods, and Opportunities," *ACM Comput. Surv.*, vol. 53, no. 5, pp. 1–40, 2020.