International Journal of Scientific Research & Engineering Trends
Volume 12, Issue 5, Sep-Oct-2026, ISSN (Online): 2395-566X

Fake News Detection Using AI

Janani A., Nivetha D., Prabhavathi U., Sumathi S.
Department of Artificial Intelligence and Data Science, Sengunthar Engineering College, Tamil Nadu, India

Abstract—The rapid circulation of news through websites and social-media platforms makes it difficult for readers to distinguish trustworthy reporting from fabricated or misleading content. This paper presents an AI-based fake news detection system that applies Natural Language Processing (NLP) and machine learning to classify textual news content into predefined categories such as fake and real. The workflow includes data preparation, text cleaning, feature extraction using Term Frequency–Inverse Document Frequency (TF-IDF), model training, prediction, and presentation of the result through an application interface. Logistic Regression is considered as the baseline classifier because it is efficient for sparse text features and provides a practical reference for text-classification tasks. The proposed system is intended to support early screening of suspicious news, not to replace source verification or professional fact-checking. The paper describes the design, implementation workflow, evaluation metrics, risks, and future improvements. Since verified experimental logs and final test measurements were not supplied for this manuscript, numerical performance is not fabricated; the results section specifies the measurements that the team should insert after running the final model. The work provides a structured, reproducible approach for building and evaluating a news-text classification application.

Index Terms—Fake news detection, artificial intelligence, machine learning, natural language processing, TF-IDF, Logistic Regression, text classification.

I. INTRODUCTION

Online platforms have changed how people discover, read, and share news. Information can reach large audiences within minutes, but false claims and misleading articles can spread just as quickly. Readers may share content without checking the publisher, date, supporting evidence, or original source. Manual verification is essential, yet reviewing large volumes of content can be time-consuming. An automated screening tool can help identify items that deserve closer inspection.

Fake news detection can be formulated as a text-classification problem when a labelled dataset contains news examples and their corresponding categories. A machine-learning model learns statistical relationships between the text and the labels, then predicts a category for previously unseen input. In this project, the system is organized around a preprocessing stage, text-feature representation, a trained classifier, and an application interface. The prediction is an estimate based on patterns learned from the dataset; it is not independent proof that a claim is true or false.

The main objective is to develop a usable workflow that accepts news text, processes it consistently, generates a model prediction, and presents the outcome clearly. The project also emphasizes evaluation beyond accuracy, because a model may perform unevenly across fake and real classes. The paper follows an IEEE-style structure and documents the assumptions and measurements needed for a credible technical report.

1. Objectives

The objectives are to: (1) prepare labelled news text for supervised learning; (2) apply consistent text normalization; (3) transform text into numerical features using TF-IDF; (4) train and evaluate a machine-learning classifier, with Logistic Regression as a baseline; (5) connect model inference to a user-facing application; (6) report class-wise evaluation metrics and inspect classification errors; and (7) communicate predictions responsibly to users.

2. Scope and Contributions

The system focuses on textual input and a binary fake/real classification task, provided those are the labels used by the final dataset. It does not independently verify article sources, inspect image or video manipulation, or establish the factual accuracy of every statement in an article. The contribution of this project is an end-to-end application workflow that combines text preprocessing, feature extraction, model inference, and result presentation. The work does not claim a new learning algorithm. Its effectiveness must be established using measured results from the final implementation.

II. RELATED WORK

Research in fake-news detection explores content, source credibility, user behaviour, and the way information propagates through social networks. Content-based methods are useful when the available input is the article text itself. They can be implemented without collecting personal user-network information, although they may miss contextual signals such as publisher history or evidence linked outside the article.

© 2026 IJSRET

Traditional machine-learning models remain useful baselines for text classification. Naive Bayes is computationally efficient and often used with word-count features. Logistic Regression estimates class scores from a feature vector and is a common baseline for sparse text representations. Support Vector Machines can also perform well on high-dimensional text features. These methods are relatively lightweight and provide useful comparisons before adopting more complex neural architectures.

Deep-learning methods, including recurrent networks and transformer-based language models, can represent context in more complex ways. However, they may require additional computational resources, training data, and careful tuning. More complex models are not automatically more reliable: performance can be inflated by duplicated records, overlapping sources, or leakage between training and test sets. For a student project, a well-evaluated TF-IDF baseline provides a transparent starting point and helps establish whether a more advanced model offers a meaningful improvement.

A key challenge in this research area is dataset bias. A classifier may learn publisher names, recurring topics, writing style, or time-specific vocabulary rather than robust indicators of misinformation. Consequently, evaluation should include a held-out test set, error analysis, and—where feasible—testing on content from different sources or a later time period.

III. SYSTEM REQUIREMENTS AND DESIGN

The system consists of four main stages: user input, text preprocessing, model inference, and result presentation. Training and evaluation form an offline development workflow. The application should validate input, handle errors, and show the model output in a way that does not imply certainty beyond the evidence provided by the classifier.

1. Functional Requirements

The user should be able to enter or paste a news headline or article excerpt and request a prediction. Empty or invalid text should be rejected with a clear message. The model layer should apply the same preprocessing and feature transformation used during training. The result view should display the predicted label and, only if supported by the implementation, a model confidence or score. Model errors and connectivity failures should be reported rather than silently converted into a prediction.

2. Non-Functional Requirements

The application should be easy to use, maintainable, and consistent across repeated requests. Model files and preprocessing configuration should be versioned together to prevent mismatches. If the app communicates with a local or remote prediction service, the service should validate requests and avoid logging unnecessary personal content. Performance should be measured on the actual target device or server. The interface should explain that the prediction is an automated screening result and can be incorrect.

3. System Workflow

First, the user submits a text sample. Second, the input is checked for empty content and processed using the training-time cleaning rules. Third, the fitted TF-IDF vectorizer converts the text into a numerical representation. Fourth, the trained classifier returns a predicted label. Finally, the application displays the result and a brief caution about independent verification. During development, the same sequence is tested with known examples to ensure that the UI, backend, vectorizer, and classifier agree on the expected input format and label mapping.

IV. METHODOLOGY

1. Training and Artifact Management

The training pipeline should be implemented as a repeatable sequence rather than a set of manually disconnected notebook cells. The cleaned dataset, split configuration, fitted vectorizer, classifier, and evaluation report should be saved with clear version labels. The model artifact should be accompanied by a short description of the training data and the expected input format. This prevents a common deployment error in which the interface loads a classifier but uses a vectorizer fitted with a different vocabulary.

A simple experiment log should record the experiment identifier, date, dataset version, number of records per class, split method, vectorizer settings, classifier parameters, and measured metrics. If a model is retrained after correcting labels or changing preprocessing, its artifact and results should be saved separately. This makes it possible to trace a prediction workflow to the exact model configuration and reproduce the reported evaluation.

2. Prediction Pipeline and Label Mapping

The mapping between numeric labels and readable class names must be verified explicitly. For example, one dataset may encode fake as 0 and real as 1, while another may use a different convention. The application must not assume a label mapping without checking the training code and saved model. A unit test should pass representative text through the complete pipeline and verify that the returned label is translated correctly for display.

© 2026 IJSRET

Input length also needs to be handled deliberately. Extremely short text may not contain enough evidence for a useful classification, while very long content may exceed interface or service limits. The application can set a reasonable maximum length, inform users when input is truncated, and avoid silently classifying only a fragment without explaining it. These checks improve reliability even though they do not eliminate model errors.

3. Dataset Preparation

A labelled dataset is required for supervised classification. Each record should include a text field and a target label, such as fake or real. Before the final report is submitted, the team should identify the exact dataset name, source, download date where available, number of records, language, label definitions, and permitted use. These details are necessary for reproducibility and are not established by the supplied Nunnarri reference paper.

Data inspection should check missing text, empty records, duplicate articles, inconsistent labels, encoding issues, and class distribution. Exact duplicates should be identified before splitting the dataset because copies in both training and test sets can make performance appear higher than it is. The team should preserve a clean copy of the original data and document each transformation. If multiple datasets are combined, their label definitions should be checked for compatibility before merging.

4. Text Preprocessing

Text preprocessing prepares raw news content for feature extraction. A typical pipeline includes converting input to a consistent text type, removing leading and trailing whitespace, normalizing repeated spaces, and handling missing values. Depending on the dataset and model, the pipeline may also normalize case, remove selected punctuation, or tokenize text. These steps should be selected carefully: punctuation, capitalization, names, dates, and negation can carry useful meaning. Aggressive cleaning may remove signals needed to interpret a statement correctly.

The preprocessing rules used at inference time must match those used during training. If the model is trained on full article text, a short headline may provide insufficient information and should be treated as a different input condition. The system should not silently substitute empty text or unrelated content when preprocessing fails.

5. Tf-Idf Feature Extraction

Machine-learning algorithms require numerical input. TF-IDF represents a document using terms weighted by their frequency in that document and their relative rarity across the collection. A term that appears frequently in one article but is uncommon across the dataset can receive a higher weight than a term that appears in almost every article. This representation is suitable for sparse text data and can be combined with linear classifiers.

The vectorizer must be fitted only on the training data. Fitting it on the entire dataset before splitting allows information from the test set to influence the feature vocabulary and weighting, causing data leakage. After training, the fitted vectorizer must be saved and reused for validation, testing, and application inference. The final paper should report whether word unigrams, bigrams, or other n-gram ranges were used, along with any vocabulary limits and normalization settings.

6. Logistic Regression Classifier

Logistic Regression is a supervised classification method that estimates a class score from a weighted combination of input features. In a text-classification pipeline, the TF-IDF feature vector is passed to the fitted classifier, which returns the predicted class. The model is attractive as a baseline because training and inference are generally efficient, and its learned coefficients can be inspected to understand which terms are associated with a class. These associations should not be interpreted as proof of causation or factuality.
The implementation should record the solver, regularization setting, class-weight strategy, random seed where applicable, and any hyperparameter search. If the final application uses a different model, this subsection must be updated to match the actual source code. Model and vectorizer files should be loaded as a compatible pair, and the label mapping must be checked before deployment.

7. Application Integration

The project context includes a Flutter-based user interface and a prediction component. In the application flow, the user enters news text and initiates detection; the interface passes the input to the inference layer and receives the predicted label. The team should document whether the model runs locally on the device or through a backend service. For a service-based design, request validation, error handling, and connection status are required. For local inference, the model runtime and compatible model artifacts must be packaged with the app.

The interface should avoid presenting a result as an absolute verdict. Wording such as “Predicted: Fake” or “Predicted: Real” is more accurate than “This news is definitely fake/real.” A short note should encourage users to check the publisher, date, original source, and independent reporting. If a confidence score is shown, it should be described as a model score and not as the probability that the article is factually true.

© 2026 IJSRET

V. EXPERIMENTAL EVALUATION

1. Recommended Implementation Protocol

The following protocol can be used to produce the final experimental record. First, load the selected dataset and print its dimensions, column names, class counts, and missing-value summary. Confirm that the text and label columns are correct and that labels have been normalized to the intended categories. Remove rows with missing target labels, investigate empty text, and inspect duplicate records. Keep a record of how many rows are removed at each step so that the final dataset size can be explained.

Second, separate the data into training and test partitions using a fixed random seed and stratification where appropriate.

The split should occur before fitting the TF-IDF vectorizer. Fit the vectorizer only on the training text, transform the training and test text with the fitted vectorizer, and train the classifier using training features and labels. If a validation set is used to choose parameters, keep it separate from the final test set. Save the fitted vectorizer and classifier together with a short configuration file describing the expected input and output labels.

Third, run the final test predictions once the model choices are fixed. Generate a classification report and confusion matrix from the saved test labels and predictions. Check that precision, recall, and F1-score are reported for both fake and real categories, and state which category is treated as the positive class. Record the dataset split, class counts, software versions, and model parameters alongside the results. If the experiment is repeated with different random seeds, report the procedure and variation rather than selecting only the most favourable run.

Fourth, test application inference independently from model training. Load the saved artifacts in the same way the deployed app loads them, then submit a small, preselected set of test texts through the complete interface.

Confirm that preprocessing is not performed twice, the correct class names are displayed, and failures produce understandable messages. Measure inference latency over multiple runs after a warm-up, and report the median and 95th percentile if latency is included in the paper. These steps separate model quality from interface integration and make the results easier to reproduce.

2. Reproducibility Checklist

For a reproducible evaluation, the team should preserve the dataset version or download reference, preprocessing code, train-test split seed, vectorizer settings, classifier parameters, package versions, and final evaluation script. The reported metrics should be generated from the saved test labels and predictions, not copied from an earlier experiment or estimated from training performance. If the dataset licence does not permit redistribution, the paper can provide the source and preparation steps rather than republishing the data.

The final experiment should be run after all model and preprocessing decisions are complete. The report should state the number of test examples and class balance. A confusion matrix should show true positives, true negatives, false positives, and false negatives according to the defined positive class. These values help readers understand the type of error made by the model, not only the overall proportion of correct predictions.

3. Baseline Comparison

Where time permits, compare Logistic Regression with at least one simple baseline such as Multinomial Naive Bayes or a majority-class predictor. A majority-class predictor is not a useful deployed system, but it provides a minimum reference that exposes misleadingly high accuracy when the dataset is imbalanced. When comparing models, use identical data splits and evaluation metrics. Report training time and inference time only when they are measured under comparable conditions. A complex model should be preferred only when it provides a defensible improvement in performance, robustness, or usability relative to its additional cost.

4. Train-Test Strategy

The dataset should be separated into training and test subsets before fitting preprocessing transformations. A validation subset or cross-validation can be used for model selection, while the test set is reserved for the final evaluation. The split ratio, stratification method, random seed, and record counts must be reported. A stratified split helps preserve class proportions, but it does not prevent leakage from duplicate articles or related stories. Where possible, a source-separated or time-based test can provide a stronger test of generalization.

A baseline should be established before tuning. If multiple algorithms are evaluated, all models should use the same data split and comparable preprocessing. Hyperparameters should be selected using training/validation data only. The test set should not be repeatedly used to choose settings, because that turns it into part of the development process.

5. Evaluation Metrics

Accuracy is the proportion of all predictions that are correct, but it can conceal poor performance on one class. Precision measures the fraction of predicted positive examples that

© 2026 IJSRET

are correct; recall measures the fraction of actual positive examples identified; and F1-score is the harmonic mean of precision and recall. For a fake/real classifier, the positive class must be explicitly defined. Report precision, recall, and F1-score for both classes, macro-F1, and a confusion matrix. If model scores are available, additional threshold-based metrics may be considered, provided their use is explained.

Inference latency should also be measured if responsiveness is important. Record the hardware and software configuration, warm-up procedure, number of repeated runs, median latency, and 95th-percentile latency. These measurements allow the reader to understand both predictive performance and practical usability.

6. Error and Robustness Analysis

Review false positives and false negatives rather than relying only on aggregate metrics. Potentially difficult examples include satire, opinion pieces, short headlines, breaking news, copied articles, and claims whose meaning depends on external context. Investigate whether publisher names, repeated phrases, or topic-specific words dominate the model's predictions. Check for duplicates and near-duplicates across splits, and test examples from sources not strongly represented in training.

A model trained on a particular historical dataset may not generalize to newly emerging events or changes in writing style. If the system is intended for multiple languages, it requires appropriate multilingual data and evaluation; translating a model's output alone does not establish multilingual performance. Robustness claims should be limited to the conditions actually tested.

VI. RESULTS AND DISCUSSION

The project's final results must be populated from the actual test run. No accuracy or F1-score is reported here because verified measurements were not provided with the supplied reference document. The table below is a reporting template, not a claim of achieved performance.

Table I. Final Test Results to be Inserted

Test-set records: [insert measured count] Accuracy: [insert measured value]

Fake class—precision / recall / F1: [insert values] Real class—precision / recall / F1: [insert values] Macro-F1: [insert measured value]

Median / P95 inference latency: [insert values and units] Hardware, software, and dataset split: [insert configuration]

The discussion should explain what the measured values indicate, identify the stronger and weaker class, and summarize the confusion matrix. If one class has lower recall, describe the practical consequence of that error. Compare models only when the experiments were performed under the same data split and evaluation protocol. Do not use approximate figures from unrelated papers as if they were produced by this implementation.

A useful application can help users prioritize content for verification, but it has limitations. It can misclassify legitimate articles that resemble patterns in fake examples and can fail to detect fabricated claims written in a style similar to reliable reporting. A strong score on a single dataset is not sufficient evidence that the system works reliably across all publishers, topics, and future events. The user interface and accompanying documentation therefore form part of responsible system design.

VII. FUNCTIONAL AND INTEGRATION TESTING

The application should be tested at three levels. Unit tests verify text cleaning, label conversion, and the model wrapper independently. Integration tests verify that the user interface sends the intended text, the inference layer loads the correct model artifacts, and the response is parsed and displayed correctly. End-to-end tests exercise the complete workflow from entering news text to viewing the result. The test suite should include normal articles, a headline-only input, empty text, whitespace-only input, unusual punctuation, non-English characters if supported, and text near the maximum allowed length.

Expected outputs for test examples should be recorded before testing. When a test fails, the team should distinguish between an interface issue, preprocessing mismatch, model prediction error, and service or file-loading error. A successful app launch is not by itself evidence of classification quality; predictive performance must be assessed separately on labelled examples. Likewise, a correct prediction for a few hand-picked examples cannot replace a held-out evaluation.

1. Interpretation of Results

After the team runs the evaluation, the results should be interpreted in relation to the project objective. If recall for the fake class is low, many fake examples are being missed; if precision for the fake class is low, many items predicted as fake are actually labelled real in the test data. Both error types matter. False accusations can undermine trust in legitimate reporting, while missed misinformation can leave users with a misleading sense of safety. The appropriate balance depends on the intended use, and any decision threshold should be justified with evidence.

© 2026 IJSRET

The model's confidence score, if available, must be interpreted carefully. A classifier can return a high score while still being wrong, especially on text unlike the training examples. Confidence calibration and out-of-distribution detection are possible future improvements, but should not be claimed unless evaluated. The final discussion should state which observations are directly supported by test results and which are proposed explanations requiring further experiments.

2. Ethical and Privacy Considerations

News classification can affect how readers perceive people, organizations, and public events. The system should avoid implying that a single automated label settles a disputed factual question. The app should encourage checking original sources and independent reporting, and should make clear that the prediction is based on patterns learned from a dataset. The training data may itself contain annotation errors, political or geographic imbalance, outdated labels, or systematic differences between sources.

The team should also minimize the collection of personal information. If submitted news text is sent to a server, the interface should describe this behaviour and the service should avoid storing content unless it is needed and appropriately protected. Test logs should use synthetic or public examples where possible. Any deployment that allows users to submit private messages or unpublished documents requires stronger privacy safeguards than a demonstration using public news articles.

Limitations and Future Work

The current manuscript describes a text-based classification pipeline and evaluation plan. The exact dataset version, final training settings, measured performance, and final deployment arrangement must be confirmed from the team's code and experiment logs. If the implemented model differs from the TF-IDF and Logistic Regression baseline described here, the methodology must be corrected before submission.

Future work may compare Logistic Regression with Naive Bayes, Support Vector Machines, or a suitable transformer-based model; add a time-based external test set; calibrate model scores; and improve explanations of why a prediction was made.

Further improvements include duplicate detection, robustness checks, clearer uncertainty communication, privacy-aware logging, and usability testing. Source credibility and external fact-checking evidence could be added as complementary signals, but these features require their own data collection, evaluation, and privacy review.

VIII. DEPLOYMENT AND MAINTENANCE

The trained classifier and vectorizer should be treated as a single versioned artifact. The application should check that both files are present and compatible at startup, and show a useful error if they cannot be loaded. If the model is served through an API, the endpoint should validate the request schema, limit input size, return structured errors, and avoid exposing local file paths or internal stack traces. The team should document the supported environment and the steps required to reproduce the development setup.

If the system is used beyond a classroom demonstration, its predictions should be monitored for changes in input distribution and user expectations. News topics, terminology, and publisher styles change over time, so a model trained once may become less reliable. Retraining should be based on reviewed and appropriately labelled data, followed by the same evaluation process used for the original model. The previous model and its results should be retained so that changes in performance can be compared rather than assumed.

1. Threats to Validity

Several factors can limit the validity of the evaluation. First, labels in public datasets may be noisy or depend on the dataset creator’s definition of fake news. Second, records may be duplicated or related articles may appear in different splits. Third, the model may exploit shortcuts such as source names or topic vocabulary. Fourth, the test set may not represent current news or the intended users’ language. Finally, results from one dataset and one split do not establish universal performance. The final paper should acknowledge these threats and avoid broad claims that exceed the evidence.

A stronger future evaluation would use an independent dataset with documented label criteria, test articles from a later period, and examine performance across topics and sources. Human review can help determine whether errors arise from missing context, ambiguous labels, satire, or model limitations. Any human evaluation should define the review procedure and report disagreements rather than treating subjective judgments as perfect ground truth.

2. Reporting Checklist for the Final Manuscript

Before submission, the team should verify that the manuscript identifies the dataset and its source; defines the fake and real labels; reports the number of records before and after cleaning; describes duplicate handling and missing-data treatment; specifies the train-test split; names the exact classifier and vectorizer configuration; and provides measured metrics from the held-out test set. The paper should include a confusion matrix or a clear summary

© 2026 IJSRET

of its four outcomes, explain at least a few important errors, and distinguish measured results from planned future work. The team should also verify that every reference is complete and relevant, that any dataset licence requirements are followed, and that the paper does not report results from a different implementation as if they were produced by this project. The system description should match the actual code: if the final app uses a different model, a different dataset, or a different inference arrangement, update the corresponding sections. A careful report with modest but traceable results is academically stronger than a polished paper containing unsupported performance claims.

IX. CONCLUSION

This paper presents an IEEE-style design and evaluation framework for an AI-based fake news detection application. The proposed workflow accepts news text, applies consistent preprocessing, represents text with TF-IDF features, uses a supervised classifier such as Logistic Regression, and displays a predicted category through an application interface. The paper also outlines dataset checks, leakage prevention, class-wise evaluation, latency measurement, and error analysis. The system should be treated as a screening aid rather than an authority on truth. To finalize the paper for academic submission, the team must confirm the actual dataset and implementation, insert measured results, add the confusion matrix, and ensure that every technical claim matches the code and experiments performed.

Acknowledgment
The authors acknowledge the guidance and support provided by their institution and project mentors. Additional acknowledgments should be included only where applicable.

REFERENCES

1. W. Y. Wang, “Liar, Liar Pants on Fire: A New Benchmark Dataset for Fake News Detection,” in Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, 2017, pp. 422–426, doi: 10.18653/v1/P17-2067.
2. K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu, “Fake News Detection on Social Media: A Data Mining Perspective,” ACM SIGKDD Explorations Newsletter, vol. 19, no. 1, pp. 22–36, 2017, doi: 10.1145/3137597.3137600.
3. T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed Representations of Words and Phrases and their Compositionality,” in Advances in Neural Information Processing Systems 26, 2013, pp. 3111–3119.
4. scikit-learn developers, “scikit-learn User Guide.” [Online]. Available: https://scikit-learn.org/stable/user_guide.html
5. Google, “Flutter Documentation.” [Online]. Available: https://docs.flutter.dev/
6. A. Vaswani et al., “Attention Is All You Need,” in Advances in Neural Information Processing Systems 30, 2017, pp. 5998–6008.

© 2026 IJSRET