Project Report: Full-Stack Fake News Detection System

Course: Machine Learning - mini project
Student: Md Faizan Khan
Roll no: 2301127

1. Abstract

The proliferation of social media has led to the rapid and damaging spread of "fake news," posing significant threats to societal discourse and public trust. This project presents a full-stack web application designed to detect fake news with high accuracy. The system utilizes a machine learning model, specifically a **Random Forest Classifier**, to analyze and classify news articles as either "Real" or "Fake." The backend is built with Python and Flask, serving a trained Scikit-learn model via a REST API. The frontend is a responsive web interface built with HTML, CSS, and JavaScript, allowing users to paste text and receive instant classification. By training the model on a large, balanced dataset of over 44,000 articles, the system achieved a final test accuracy of **95.91%**, demonstrating a highly effective and practical end-to-end data science solution.

2. Introduction

2.1 Problem Statement

In the digital age, misinformation can be created and disseminated at an unprecedented scale. Fake news articles, often designed to be sensational and emotionally charged, frequently spread faster than factual news. This phenomenon leads to public confusion, political polarization, and a dangerous erosion of trust in media and public institutions. The sheer volume of content generated daily makes manual verification impossible. Therefore, there is a critical and urgent need for automated, data-driven systems that can effectively identify and flag potential misinformation in real-time.

2.2 Objectives

The primary objectives of this project are:

- To collect and prepare a large, labeled dataset of real and fake news articles.
- To implement a robust text preprocessing pipeline to clean and normalize raw article text for machine learning.
- To train and evaluate a **Random Forest** classifier, optimizing for high accuracy and balanced performance between classes.

- To develop a full-stack application, including:
    - A **Flask backend** to host the trained model and vectorizer as a REST API.
    - A **web-based frontend** to provide a user-friendly interface for real-time article analysis.
    -

3. Literature Survey

Fake news detection is a well-studied subfield of Natural Language Processing (NLP). Initial approaches often relied on metadata (e.g., source, author) or simple linguistic features. More advanced methods utilize the content of the articles themselves.
Common models include:

- **Naive Bayes:** A probabilistic classifier that works well with text data, particularly with TF-IDF features. It is fast but can be overly simplistic.
- **Support Vector Machines (SVM):** A powerful classifier effective in high-dimensional spaces, making it highly suitable for text classification where the feature set (words) is very large.
- **Random Forest (RF):** The model chosen for this project. As an ensemble method, it builds multiple decision trees during training and merges their results. This makes it highly robust to overfitting, effective at handling high-dimensional data, and provides strong performance without the extensive tuning required by other models.

Given the project's goal of creating a high-performance and deployable system, Random Forest was selected as it provides an excellent balance of accuracy, training efficiency, and interpretability.

4. Methodology

4.1 Data Collection

The initial dataset, comprising 422 articles from BuzzFeed and PolitiFact, was found to be insufficient, resulting in a model with an accuracy of 43.53% (worse than random guessing). To build a robust model, a new, much larger dataset was sourced from Kaggle ("Fake and Real News Dataset"). This dataset consists of two files:

- True.csv: 21,417 legitimate news articles.
- Fake.csv: 23,481 fake news articles.

This provided a substantial and well-balanced corpus of **44,898 articles**, which is essential for training a reliable classifier.

4.2 Data Preprocessing

Before training, the raw article text was passed through a rigorous cleaning pipeline using the NLTK library to normalize the data and reduce noise:

1. **Lowercasing:** All text was converted to lowercase.

2. **Punctuation Removal:** All punctuation marks were removed.
3. **Number Removal:** All numerical digits were stripped from the text.
4. **Tokenization:** The cleaned text was split into individual words (tokens).
5. **Stopword Removal:** Common English stopwords (e.g., "the", "is", "in") were removed, as they provide little predictive value.
6. **Lemmatization:** Tokens were reduced to their root form (e.g., "running" -> "run") using the WordNetLemmatizer. This consolidates words with similar meanings, reducing the feature space.
7.

4.3 Feature Engineering (TF-IDF)

Machine learning models cannot understand raw text. To convert the processed text into a numerical format, the **Term Frequency-Inverse Document Frequency (TF-IDF)** vectorization technique was used.

- **Term Frequency (TF):** Measures how frequently a word appears in a single article.
- **Inverse Document Frequency (IDF):** Measures how important a word is by penalizing common words that appear across all articles.

This technique creates a matrix where each row is an article and each column is a word's TF-IDF score. To maintain efficiency and prevent memory issues, the vocabulary was limited to the top **5,000 most frequent features** (words) in the corpus.

4.4 Model Architecture

The chosen model is a **Random Forest Classifier** from Scikit-learn. This is an ensemble learning method that builds a "forest" of decision trees and aggregates their votes to make a final prediction.

- **Ensemble Power:** It corrects for the high-variance (overfitting) tendency of single decision trees.
- **Robustness:** It is less sensitive to noisy data and performs well on high-dimensional datasets like this one.
- **Hyperparameters:** The model was trained with n_estimators=100 (meaning 100 individual trees) and n_jobs=-1 to utilize all available CPU cores (essential on the M2 processor) for parallel training.
-

5. Implementation

5.1 Tools and Libraries

- **Python:** Core programming language.
- **Pandas:** For loading and manipulating the .csv datasets.
- **NLTK (Natural Language Toolkit):** For all text preprocessing tasks.
- **Scikit-learn:** For TF-IDF vectorization, the Random Forest model, and model evaluation metrics.

- **Flask:** For creating the backend web server and API.
- **Pickle:** For serializing (saving) the trained model and vectorizer to disk.
- **HTML/CSS/JavaScript:** For the frontend user interface.
-

5.2 System Design

The application follows a client-server architecture, allowing the compute-heavy ML model to run on a server while the user interacts with a lightweight web page.

1. **Frontend (Client):** The user visits the index.html page. When they paste text and click "Analyze," JavaScript captures the text.
2. **API Request:** The frontend JavaScript sends this text as a JSON payload to the /predict endpoint of the Flask server using an asynchronous fetch (POST) request.
3. Backend (Server): The Flask app.py script, which has pre-loaded the model.pkl and vectorizer.pkl files, receives the JSON data.
   a. It applies the exact same preprocessing (from utils.py) to the new text.
   b. It uses the loaded vectorizer to transform the processed text into a TF-IDF vector.
   c. It feeds this vector into the loaded model to get a prediction (0 or 1) and confidence probabilities.
4. **API Response:** The server formats this result into a new JSON object (e.g., {'prediction': 'Fake', 'confidence': '98.76%'}) and sends it back to the frontend.
5. **Frontend (Update):** The JavaScript receives the JSON response and updates the webpage's DOM to display the prediction and confidence, dynamically coloring the result box red for "Fake" or green for "Real."
6.

6. Results and Discussion

The model was trained on 80% of the 44,898-article dataset (35,918 articles) and evaluated on the remaining 20% (8,980 articles). The results were outstanding.

6.1 Model Performance

The Random Forest model achieved the following performance on the unseen test data:

- **Overall Accuracy: 95.91%**
-

6.2 Classification Report

The classification report provides a detailed breakdown of precision, recall, and F1-score for each class, confirming the model is highly balanced.

Class	Precision	Recall	F1-Score	Support
Fake (0)	0.96	0.96	0.96	4488
Real (1)	0.96	0.96	0.96	4455

Accuracy			**0.96**	**8943**
Macro Avg	0.96	0.96	0.96	8943
Weighted Avg	0.96	0.96	0.96	8943

(Note: Support values may differ slightly from screenshot due to data cleaning steps)

6.3 Confusion Matrix

The confusion matrix visualizes the model's predictions versus the true labels.

	Predicted: Fake	**Predicted: Real**
Actual: Fake	**4305** (True Negative)	**183** (False Positive)
Actual: Real	**181** (False Negative)	**4274** (True Positive)

6.4 Discussion

The results are excellent. An accuracy of 95.91% demonstrates a highly effective and reliable model.

- **High and Balanced Performance:** The key finding is in the classification report. Both the "Fake" and "Real" classes have identical Precision, Recall, and F1-scores of 0.96. This is an ideal outcome, as it shows the model is **not biased** and is equally strong at identifying *both* real and fake news. This is a direct result of using a large, balanced dataset.
- **Low Error Rate:** The confusion matrix confirms this. The number of incorrect predictions (False Positives and False Negatives) are exceptionally low (183 and 181, respectively) compared to the high number of correct predictions (4305 and 4274).
- **Project Iteration:** This success stands in stark contrast to the initial model trained on 422 articles, which had a 43.53% accuracy. This proves that for complex NLP tasks, the **quantity and quality of data** are the most critical factors for success.

7. Conclusion and Future Scope

This project successfully achieved its goal of building a full-stack fake news detection system. A Random Forest model was trained on a large corpus of over 44,000 news articles, achieving an outstanding accuracy of 95.91%. This model was then successfully deployed as an interactive web application using a Flask API and an HTML/CSS/JS frontend, demonstrating a complete and practical end-to-end data science solution.
Possible areas for **Future Scope** include:

- **Model Enhancement:** Experimenting with more advanced models like Gradient Boosting (XGBoost) or transformers (e.g., a fine-tuned DistilBERT) could potentially push the accuracy even higher.
- **Feature Expansion:** Incorporating metadata features, such as the source/publisher of the article (if available), author, or analyzing the sentiment of the headline.
- **Scalable Deployment:** Deploying the application to a cloud platform (like AWS, Google Cloud, or Heroku) to make it publicly accessible and handle a larger volume of requests.
- **Browser Extension:** Re-packaging the application as a browser extension that could automatically analyze articles as the user browses social media, providing real-time warnings.

8. References

- Scikit-learn: Machine Learning in Python. (https://scikit-learn.org)
- Flask Web Development. (https://flask.palletsprojects.com)
- Natural Language Toolkit (NLTK). (https://www.nltk.org)
- Pandas: Python Data Analysis Library. (https://pandas.pydata.org)
- Kaggle: Fake and Real.