Fake News Detection — Full Project Report

August 26, 2025

Abstract

This report documents a complete end-to-end Fake News Detection project built using a classic machine-learning approach: TF–IDF feature extraction combined with a Logistic Regression classifier implemented in scikit-learn. The pipeline includes dataset preparation (Kaggle Fake and True news CSVs), preprocessing, model training, evaluation, artifact saving, and a small Streamlit web app for interactive prediction. The model achieved strong validation results on the combined dataset: **Accuracy = 0.98965**, **Precision = 0.99252**, **Recall = 0.98766**, **F1 = 0.99008**, **ROC-AUC = 0.99913**. All code, commands, artifacts, and reproducible steps are provided.

Table of Contents

---

1. Introduction

Fake news detection is the task of classifying a piece of news (headline, body, or both) as real (truthful, legitimate reporting) or fake (fabricated, misleading). This project implements a baseline classical NLP pipeline that is fast to train, interpretable, and suitable for immediate experimentation. The goal was to produce a working detector and produce deliverables (model artifacts, README, demo app) that are reproducible and explainable.

2. Dataset

Source used: Kaggle “Fake and Real News” dataset (two CSV files: `Fake.csv` and `True.csv`) — merged into a single `combined.csv` with a `label` column.

Columns present originally: `title, text, subject, date`.

Transformed columns used in modeling: `title, text, label` where `label` ∈ {`fake`, `real`}.

Notes:

- The dataset contains tens of thousands of articles (combined). The training shown in this project used the full combined CSV. Labels are textual (`fake` / `real`) and were handled directly by the pipeline.
- Class distribution in validation set (from final run): real = 4,283; fake = 4,701 (total validation samples = 8,984).

3. Data preparation and preprocessing

3.1 Combining files

A small Python script (`prepare_data.py`) was used to:

- Load `Fake.csv` and `True.csv` with pandas.
- Add `label` column (`fake` or `real`).
- Select only `title`, `text`, `label` columns.
- Concatenate and save `combined.csv`.

3.2 Text concatenation

For each example, `title` and `text` were concatenated (with a space separator) to form a single textual field fed to the model. This often gives better performance because the title contains useful signals.

3.3 Cleaning & validation

- Basic normalization: collapse repeated whitespace, strip leading/trailing spaces.
- Rows with empty text or missing labels were dropped.
- The training script automatically inferred label mapping and handled both string and numeric labels.

4. Feature engineering

TF–IDF Vectorization

- Technique: `sklearn.feature_extraction.text.TfidfVectorizer`.
- Settings used (defaults or tuned in script):
    - `lowercase=True`
    - `stop_words='english'`
    - `strip_accents='unicode'`
    - `ngram_range=(1, 2)` (unigrams + bigrams)
    - `max_features=50000` (vocabulary cap)

Why TF–IDF?

- TF–IDF is fast, sparse, and effective for text classification problems where lexical patterns are discriminative.
- Works well with linear models like Logistic Regression/Linear SVM.

5. Model selection and training

5.1 Model type

- **Logistic Regression** (`sklearn.linear_model.LogisticRegression`) was chosen as the primary model. It is a linear classifier suitable for high-dimensional, sparse TF–IDF features.
- `class_weight='balanced'` was used to mitigate class imbalance.
- `max_iter=1000` ensures convergence on larger feature sets.

5.2 Training flow

- Train/validation split: 80/20 (stratified by label to keep class balance in both splits).
- If dataset is tiny, the script contains logic to avoid stratify errors (fallback to training/evaluating on the same data for tiny datasets).
- Fitting steps:
    1. Fit TF–IDF vectorizer on the training text.
    2. Transform validation text using the fitted vectorizer.
    3. Fit Logistic Regression on vectorized training set.
    4. Predict on validation set and compute metrics.

6. Evaluation results

Final validation metrics (from your run):

- Accuracy: **0.9896482635796973**

- Precision: **0.9925181701581872**
- Recall: **0.9876621995320145**
- F1 score: **0.9900842307282226**
- ROC-AUC: **0.9991272640437999**

Per-class (validation) report:

```
          precision    recall  f1-score   support

   real(0)      0.99      0.99      0.99      4283
   fake(1)      0.99      0.99      0.99      4701
```

Confusion matrix

- Saved to `artifacts/confusion_matrix.png`. The matrix shows very few off-diagonal errors: both classes are well-separated by the model.

Interpretation

- These high metrics indicate the model separates the two classes well on the held-out validation set of the same dataset.
- Caveat: these results measure in-dataset performance. Domain shift (new sources, time, topical differences) can reduce real-world performance.

7. Artifacts and reproducibility

After training the script created and saved these artifacts inside the `artifacts/` folder:

- `vectorizer.joblib` — TF–IDF vectorizer (vocabulary and settings)
- `model.joblib` — trained Logistic Regression classifier
- `label_map.json` — mapping of label strings to numeric classes (e.g., `{ "real": 0, "fake": 1 }`)
- `report.txt` — textual copy of the evaluation metrics
- `confusion_matrix.png` — visualization of the confusion matrix

These files allow inference without retraining: load the vectorizer and model, transform new text, and call `predict()`.

8. Streamlit demo

A small interactive demo `app_streamlit.py` was created. Functionality:

- Load artifacts (vectorizer + model + label map).
- Accept text input in a textbox (title + article body).
- Display predicted label (`real` or `fake`).

Run locally with:

```
streamlit run app_streamlit.py
```

Open the Local URL printed in the terminal (typically `http://localhost:8501`).

9. Limitations and risks

1. **Dataset bias and representativeness**: The Kaggle dataset may not reflect the full distribution of contemporary misinformation; it has domain and time biases.
2. **Surface-level signals**: TF–IDF + Logistic Regression relies on lexical patterns. Malicious actors can obfuscate writing or blend styles to evade detection.
3. **No external fact-checking**: The model classifies text patterns, not factual truth. It can't verify claims or check sources.
4. **Overfitting to dataset**: Very high accuracy may indicate dataset shortcuts (e.g., source-specific words) rather than true generalization.

10. Appendix A: Commands and scripts

Windows setup

```
cd C:\Users\SWARNAVA\Downloads\Fake_News_Detection
python -m venv .venv
.venv\Scripts\activate
pip install --upgrade pip
pip install pandas scikit-learn matplotlib joblib streamlit
```

Prepare combined CSV (script: `prepare_data.py`)

```python
import pandas as pd
fake = pd.read_csv('Fake.csv')
true = pd.read_csv('True.csv')
fake['label'] = 'fake'
true['label'] = 'real'
fake = fake[['title','text','label']]
true = true[['title','text','label']]
df = pd.concat([fake,true], ignore_index=True)
df.to_csv('combined.csv', index=False)
print('combined.csv created')
```

Train model (script: `train_fake_news.py`)

Key command used in this project:

```
python train_fake_news.py --data combined.csv --text-cols title text --label-
col label --val-size 0.2
```

Predict (script: `predict.py`)

Example:

```
python predict.py "This just in: scientists discover water is wet."
```

Batch test script

A small `batch_test.py` was used to predict multiple sample inputs at once using the saved artifacts.

11. Appendix B: Model interpretation snippets

Top features

We can print the top positive/negative coefficients for the logistic regression model:

```
from joblib import load
import numpy as np
vec = load('artifacts/vectorizer.joblib')
clf = load('artifacts/model.joblib')
feature_names = np.array(vec.get_feature_names_out())
coef = clf.coef_[0]
top_pos = feature_names[np.argsort(coef)[-30:]]
top_neg = feature_names[np.argsort(coef)[:30]]
print('Top features for class=1 (fake):', top_pos)
print('Top features for class=0 (real):', top_neg)
```

This helps to see which words push predictions toward `fake` or `real`.

12. References

- Kaggle: “Fake and Real News Dataset” (Fake.csv / True.csv).
- scikit-learn documentation: TfidfVectorizer, LogisticRegression, train_test_split.