Fake News Detection

A production-oriented **Fake News Detection** system for automatic classification of news articles based on textual content.

The project implements a complete, reproducible **machine learning pipeline** for NLP tasks, including data preprocessing, feature extraction, model training, evaluation, and result visualization. It is designed with clean architecture, detailed logging, and scalability in mind.

🚀Features

- End-to-end ML pipeline: data loading → training → evaluation
- Text vectorization using **TF-IDF**
- Linear text classification with **SGDClassifier (Passive-Aggressive mode)**
- Robust evaluation with accuracy and classification report
- Visual analytics:
- Confusion Matrix
- Class distribution plots
- Automatic saving of figures to `results/figures/`
- Modular, production-ready project structure
- Extensive logging for monitoring and debugging

Project Structure

```
fake_news_detection/
├── data/                    # Input datasets
│   └── fake_news.csv
├── results/
│   └── figures/             # Generated visualizations
├── src/
│   ├── preprocessing.py     # Data loading & cleaning
│   ├── feature_extraction.py # Train/test split & TF-IDF
│   ├── model.py              # Model training
│   ├── evaluation.py         # Metrics & visualizations
│   └── __init__.py
├── main.py                  # Application entry point
├── config.py                # Project configuration
├── requirements.txt
├── pyproject.toml
└── README.md
```

Tech Stack

- Python 3.10+
- scikit-learn
- pandas, numpy
- scipy
- matplotlib, seaborn

Getting Started

Install dependencies

```
pip install -r requirements.txt
```

or with Poetry:

```
poetry install
```

Run the pipeline

```
from main import run_pipeline

run_pipeline("data/fake_news.csv")
```

The pipeline performs:

1. Dataset loading and validation
2. Stratified train/test split
3. TF-IDF feature extraction
4. Model training
5. Model evaluation
6. Visualization and artifact saving

🚡 ML Pipeline Overview

```
def run_pipeline(data_path: str):
    df = create_dataframe_from_dataset(data_path)

    x_train, x_test, y_train, y_test = split_dataset(
        df, target_column="label", test_size=0.2
    )
```

```
    vectorizer = create_tfidf_vectorizer(max_df=0.7)
    tfidf_train, tfidf_test, _ = vectorize_data(
        x_train, x_test, vectorizer
    )

    model = train_sgd_classifier(tfidf_train, y_train)

    y_pred = model.predict(tfidf_test)
    accuracy = evaluate_model(model, tfidf_test, y_test)

    plot_confusion_matrix(y_test, y_pred)
    plot_class_distribution(y_train)

    return model, vectorizer
```

🏫Model Performance

Typical results on the provided dataset:

- **Accuracy:** ~94%
- Balanced precision and recall across classes
- Stable convergence with early stopping

Generated artifacts:

- `results/figures/confusion_matrix_fake_news.png`
- `results/figures/class_distribution_train.png`

Model Choice

The project uses **SGDClassifier** configured to emulate the **Passive-Aggressive (PA-I)** algorithm:

```
SGDClassifier(
    loss="hinge",
    learning_rate="pa1",
    penalty=None,
    early_stopping=True
)
```

This approach provides:

- High efficiency on large, sparse text data
- Fast convergence
- Strong performance for binary text classification
- Compatibility with modern versions of scikit-learn

🚀 Roadmap & Extensions

Potential extensions for production or research use:

- Hyperparameter optimization (GridSearchCV)
- n-gram feature enrichment
- Full sklearn Pipeline integration
- Model persistence (`joblib`)
- REST API (FastAPI)
- Real-time inference service
- Monitoring and drift detection

License

MIT License