Mid-Term Project Report: Fake News Detection System

Course Title: Machine Learning & Data Science

Instructor: Gulshan Yasmeen

Institution: NAVTAC AI Training Program

Student Name: Sheryar Sher

Project Title: Fake News Detection System

Submission Date: October 25, 2024

# Table of Contents

A. Project Proposal

B. Data Mining and Exploration

C. Data Preprocessing

D. Data Visualization

E. Model Development

F. Model Evaluation

G. Conclusion and Recommendations

H. Documentation

# A. Project Proposal

## Title

Fake News Detection System: A Machine Learning Approach to Identify Misinformation

## Problem Statement

In today's digital age, the rapid spread of misinformation and fake news has become a significant challenge affecting public opinion, political discourse, and social stability. With the exponential growth of social media platforms and online news consumption, distinguishing between authentic and fabricated news articles has become increasingly difficult for the average reader. This project addresses the critical need for automated systems that can accurately identify fake news articles, helping users make more informed decisions about the information they consume.

## Objectives

The primary objectives of this project are:

1. Develop an accurate machine learning model that can distinguish between real and fake news articles with high precision

2. Implement comprehensive text preprocessing techniques including stemming, stopword removal, and feature extraction

3. Compare multiple machine learning algorithms including Logistic Regression, Naive Bayes, Random Forest, and SVM to identify the best-performing approach

4. Create an interactive web application using Streamlit that allows users to input news text and receive real-time predictions

5. Achieve high accuracy and reliability in fake news detection to make the system practically useful

6. Develop comprehensive evaluation tools and visualization functions for model assessment

## Dataset Description

The project utilizes a comprehensive fake news dataset with the following characteristics:

• Source: Publicly available dataset for fake news detection research

• Size: 24,353 training samples, 8,117 test samples, and 8,117 evaluation samples

• Features: id (unique identifier), title (headline), text (article content), label (0=Real, 1=Fake)

• Class Distribution: Approximately balanced with 13,246 fake news articles and 11,107 real news articles

• Data Quality: Clean dataset with no missing values, ensuring reliable model training

#### [SCREENSHOT: Dataset Overview - First 5 rows of training data]

(Space reserved for screenshot - please insert image here)

# B. Data Mining and Exploration

## Initial Data Analysis

The dataset exploration revealed several key insights through comprehensive analysis:

### Dataset Structure

• Training Set: 24,353 articles (54.4% fake, 45.6% real)

• Test Set: 8,117 articles (53.8% fake, 46.2% real)

• Evaluation Set: 8,117 articles (53.1% fake, 46.9% real)

### Data Quality Assessment

• No missing values detected across all features

• Consistent data format across all three datasets

• Text content varies significantly in length and complexity

• Memory usage: 79.4 MB for text content, 3.7 MB for titles

#### [SCREENSHOT: Dataset Shape and Basic Statistics]

(Space reserved for screenshot - please insert image here)

#### [SCREENSHOT: Missing Values Analysis]

(Space reserved for screenshot - please insert image here)

#### [SCREENSHOT: Label Distribution Visualization]

(Space reserved for screenshot - please insert image here)

## Key Findings from Exploration

1. Balanced Dataset: The relatively balanced class distribution prevents bias toward either real or fake news classification

2. Rich Text Content: The combination of titles and article text provides comprehensive information for analysis

3. Diverse Topics: The dataset covers various domains, making the model more robust and generalizable

4. Clean Data: No preprocessing required for missing values, allowing focus on feature engineering

# C. Data Preprocessing

## Text Preprocessing Pipeline

A comprehensive text preprocessing pipeline was implemented to prepare the data for machine learning:

### 1. Data Cleaning

• Removed the id column as it's not relevant for classification

• Handled any potential null values (none found in this dataset)

• Combined title and text columns to create a comprehensive content field

### 2. Text Normalization

The following preprocessing steps were applied:

• Removed non-alphabetic characters using regular expressions

• Converted all text to lowercase for consistency

• Applied Porter Stemmer to reduce words to their root forms

• Removed common English stopwords using NLTK library

### Preprocessing Function Implementation

def preprocess\_text(text):  
 # Removing non-alphabetic characters  
 text = re.sub('[^a-zA-Z]', ' ', text)  
 text = text.lower()  
 words = text.split()  
 # Stemming and removing stopwords  
 processed\_words = [ps.stem(word) for word in words if word not in stop\_words]  
 return ' '.join(processed\_words)

#### [SCREENSHOT: Text Preprocessing Code Implementation]

(Space reserved for screenshot - please insert image here)

#### [SCREENSHOT: Before and After Text Preprocessing Example]

(Space reserved for screenshot - please insert image here)

### 3. Feature Engineering

• Stemming: Applied Porter Stemmer to reduce words to their root forms

• Stopword Removal: Eliminated common English stopwords using NLTK

• Text Vectorization: Used CountVectorizer with:

- Maximum features: 5,000

- N-gram range: (1, 3) to capture unigrams, bigrams, and trigrams

- This approach captures both individual words and word combinations

### 4. Data Splitting

• Training set: 80% of the data (19,482 samples)

• Test set: 20% of the data (4,871 samples)

• Random state: 2 (for reproducibility)

#### [SCREENSHOT: Feature Vectorization Results]

(Space reserved for screenshot - please insert image here)

#### [SCREENSHOT: Data Splitting Visualization]

(Space reserved for screenshot - please insert image here)

# D. Data Visualization

## Model Performance Visualization

The project includes comprehensive visualizations to analyze model performance:

1. Confusion Matrices: Generated for each model to visualize:

• True Positives (correctly identified fake news)

• True Negatives (correctly identified real news)

• False Positives (real news misclassified as fake)

• False Negatives (fake news misclassified as real)

2. Performance Comparison Charts:

• Bar charts comparing accuracy across different algorithms

• Detailed classification reports showing precision, recall, and F1-scores

• Visual representation of model strengths and weaknesses

3. Model Evaluation Metrics:

• Accuracy scores for each algorithm

• Precision and recall for both classes

• F1-scores for balanced performance assessment

## Evaluation Function Implementation

def evaluate\_model(model, model\_name, X\_train, y\_train, X\_test, y\_test):  
 """Train model, print metrics, and show confusion matrix"""  
 model.fit(X\_train, y\_train)  
 y\_pred = model.predict(X\_test)  
  
 acc = accuracy\_score(y\_test, y\_pred)  
 print(f"  
===== {model\_name} =====")  
 print(f"✅ Accuracy: {acc:.4f}")  
 print("  
🧾 Classification Report:")  
 print(classification\_report(y\_test, y\_pred))  
  
 # Confusion Matrix plot  
 cm = confusion\_matrix(y\_test, y\_pred)  
 plt.figure(figsize=(5, 4))  
 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')  
 plt.title(f"{model\_name} - Confusion Matrix")  
 plt.xlabel("Predicted Label")  
 plt.ylabel("True Label")  
 plt.show()  
  
 return acc

#### [SCREENSHOT: Confusion Matrix - Random Forest Model]

(Space reserved for screenshot - please insert image here)

#### [SCREENSHOT: Confusion Matrix - Logistic Regression Model]

(Space reserved for screenshot - please insert image here)

#### [SCREENSHOT: Confusion Matrix - Naive Bayes Model]

(Space reserved for screenshot - please insert image here)

#### [SCREENSHOT: Confusion Matrix - SVM Model]

(Space reserved for screenshot - please insert image here)

#### [SCREENSHOT: Model Performance Comparison Bar Chart]

(Space reserved for screenshot - please insert image here)

#### [SCREENSHOT: Classification Report - Random Forest]

(Space reserved for screenshot - please insert image here)

# E. Model Development

## Algorithm Selection and Implementation

Four different machine learning algorithms were implemented and compared using a systematic evaluation approach:

### 1. Logistic Regression

• Rationale: Simple, interpretable, and effective for binary classification

• Configuration: Maximum iterations set to 1000 for convergence

• Performance: 97.00% accuracy

### 2. Naive Bayes (Multinomial)

• Rationale: Excellent for text classification due to its probabilistic approach

• Configuration: Default parameters with multinomial distribution

• Performance: 94.85% accuracy

### 3. Random Forest

• Rationale: Robust ensemble method that handles overfitting well

• Configuration: 100 estimators, random state 42

• Performance: 97.95% accuracy (best performing)

### 4. Support Vector Machine (SVM)

• Rationale: Effective for high-dimensional text data

• Configuration: Linear kernel for efficiency

• Performance: 95.96% accuracy

## Model Training Implementation

models = [  
 (LogisticRegression(max\_iter=1000), "Logistic Regression"),  
 (MultinomialNB(), "Naive Bayes"),  
 (RandomForestClassifier(n\_estimators=100, random\_state=42), "Random Forest"),  
 (LinearSVC(), "SVM (Linear)")  
]  
  
accuracies = {}  
  
for model, name in models:  
 acc = evaluate\_model(model, name, X\_train, y\_train, X\_test, y\_test)  
 accuracies[name] = acc

#### [SCREENSHOT: Model Training Code Implementation]

(Space reserved for screenshot - please insert image here)

#### [SCREENSHOT: Model Training Progress and Results]

(Space reserved for screenshot - please insert image here)

#### [SCREENSHOT: Model Comparison Visualization]

(Space reserved for screenshot - please insert image here)

## Model Training Process

1. Feature Extraction: Converted preprocessed text to numerical features using CountVectorizer

2. Model Training: Each algorithm was trained on the training set using systematic evaluation

3. Performance Comparison: All models evaluated using consistent metrics and visualization

4. Model Selection: Random Forest was selected as the best-performing model

# F. Model Evaluation

## Performance Metrics Analysis

### Random Forest (Best Model) Results:

• Overall Accuracy: 97.95%

• Precision (Real News): 97%

• Recall (Real News): 99%

• F1-Score (Real News): 98%

• Precision (Fake News): 99%

• Recall (Fake News): 97%

• F1-Score (Fake News): 98%

### Model Comparison Summary:

1. Random Forest: 97.95% accuracy (Selected)

2. Logistic Regression: 97.00% accuracy

3. SVM (Linear): 95.96% accuracy

4. Naive Bayes: 94.85% accuracy

## Performance Comparison Implementation

acc\_df = pd.DataFrame(list(accuracies.items()), columns=['Model', 'Accuracy'])  
  
plt.figure(figsize=(8, 5))  
sns.barplot(x='Model', y='Accuracy', data=acc\_df, palette='Blues\_d')  
plt.title("📈 Model Performance Comparison", fontsize=14)  
plt.xlabel("Model", fontsize=12)  
plt.ylabel("Accuracy", fontsize=12)  
plt.ylim(0.9, 1.0)  
  
# Add accuracy labels on top of bars  
for index, value in enumerate(acc\_df['Accuracy']):  
 plt.text(index, value + 0.002, f"{value:.3f}", ha="center", fontsize=10)  
  
plt.show()

#### [SCREENSHOT: Detailed Classification Report - All Models]

(Space reserved for screenshot - please insert image here)

#### [SCREENSHOT: Model Performance Metrics Table]

(Space reserved for screenshot - please insert image here)

#### [SCREENSHOT: Performance Comparison Visualization]

(Space reserved for screenshot - please insert image here)

## Model Persistence

The best performing model and vectorizer were saved for deployment:

import joblib  
  
joblib.dump(rf\_model, "random\_forest\_model.pkl")  
joblib.dump(vectorizer, "vectorizer.pkl")

This allows for easy model loading and deployment in production environments.

# G. Conclusion and Recommendations

## Key Insights and Findings

1. Model Performance:

The Random Forest algorithm achieved the highest accuracy of 97.95%, demonstrating that ensemble methods are particularly effective for fake news detection. The model shows excellent balance between precision and recall for both real and fake news classification.

2. Feature Engineering Impact:

The combination of title and article text, along with comprehensive preprocessing (stemming, stopword removal, and n-gram features), significantly improved model performance. The 5,000-feature vectorization with 1-3 gram ranges captured important linguistic patterns.

3. Algorithm Comparison:

While all algorithms performed well (above 94% accuracy), Random Forest's ensemble approach provided the most robust and reliable classification, making it the optimal choice for this application.

4. Evaluation Tools:

The development of comprehensive evaluation functions and visualization tools enhanced the analysis process, providing clear insights into model performance and enabling systematic comparison across different algorithms.

## Practical Applications

1. Web Application:

The developed Streamlit application provides an intuitive interface for real-time fake news detection, making the technology accessible to end users.

2. Educational Tool:

The system can serve as an educational resource to help users understand the characteristics of fake news and improve their media literacy.

3. Content Moderation:

The model can be integrated into social media platforms and news websites to automatically flag potentially fake content for human review.

4. Research Platform:

The comprehensive evaluation tools and modular design make this system suitable for further research and development in fake news detection.

#### [SCREENSHOT: Streamlit Web Application Interface]

(Space reserved for screenshot - please insert image here)

#### [SCREENSHOT: Real-time Prediction Example]

(Space reserved for screenshot - please insert image here)

#### [SCREENSHOT: Web Application in Action]

(Space reserved for screenshot - please insert image here)

## Limitations and Future Improvements

1. Current Limitations:

• Language Dependency: Model trained only on English text

• Temporal Bias: Performance may degrade with evolving language patterns

• Context Understanding: Limited ability to understand nuanced context and satire

• Source Verification: Cannot verify factual accuracy, only linguistic patterns

2. Recommended Future Enhancements:

• Multilingual Support: Expand to other languages for global applicability

• Real-time Learning: Implement online learning to adapt to new patterns

• Fact-checking Integration: Combine with external fact-checking databases

• Ensemble with Other Models: Integrate with transformer-based models like BERT

• User Feedback Loop: Incorporate user corrections to improve accuracy over time

• Advanced Visualization: Develop interactive dashboards for model monitoring

3. Technical Improvements:

• Model Deployment: Deploy on cloud platforms for scalability

• API Development: Create REST API for integration with other applications

• Performance Monitoring: Implement continuous monitoring and alerting

• A/B Testing: Test different model versions with real users

# H. Documentation

## Project Structure

ML-Projects/

├── datasets/

│ ├── train.csv (24,353 samples)

│ ├── test (1).csv (8,117 samples)

│ └── evaluation.csv (8,117 samples)

├── fake-news-detection/

│ ├── Sheryar\_Sher\_fake-news-detection.ipynb (Complete implementation)

│ ├── streamlit.py (Web application)

│ ├── random\_forest\_model.pkl (Trained model)

│ └── vectorizer.pkl (Feature vectorizer)

└── document/

└── Sheryar\_Sher\_Fake\_News\_Detection\_Project\_Report.docx (This report)

## Technical Implementation

• Programming Language: Python 3.12

• Key Libraries: pandas, scikit-learn, nltk, streamlit, matplotlib, seaborn, joblib

• Model Persistence: joblib for saving trained models

• Web Framework: Streamlit for user interface

• Development Environment: Jupyter Notebook for development and experimentation

• Evaluation Tools: Custom functions for model assessment and visualization

## Code Quality and Documentation

• Comprehensive comments throughout the code

• Modular function design for reusability

• Error handling and user input validation

• Clean, readable code structure following Python best practices

• Systematic evaluation approach with consistent metrics

• Professional visualization with clear labeling and formatting

#### [SCREENSHOT: Jupyter Notebook Code Structure]

(Space reserved for screenshot - please insert image here)

#### [SCREENSHOT: Streamlit Application Code]

(Space reserved for screenshot - please insert image here)

#### [SCREENSHOT: Model Files and Project Structure]

(Space reserved for screenshot - please insert image here)

Project Completion Date: October 25, 2024

Total Development Time: Approximately 2 weeks

Model Training Time: ~30 minutes (including evaluation)

Web Application Status: Fully functional and ready for deployment

This project demonstrates a complete end-to-end machine learning solution, from data exploration to model deployment, showcasing practical skills in data science, machine learning, and web application development. The implementation includes comprehensive evaluation tools and professional documentation suitable for academic and professional environments.