Phase-2 Submission Template

Student Name: B.RISHI KUMAR

Register Number: 510623104084

Institution: [C.ABDUL HAKEEM COLLEGE OF

ENGINEERING AND TECHNOLOGY]

Department:

[COMPUTER SCIENCE OF ENGINEERING]

Date of Submission: [08/05/2025]

GithubRepositorLink: https://github.com/rishikumar287/Exposing-the-truth-with-advanced-fake-news-detection-powered-by-natural-language-.git

1. Problem Statement

This project tackles the spread of fake news onlinebuilding upon Phase-1's goal to aid users in discerning truth. It refines the approach by focusing on automated URL analysis and advanced NLP. The core task is binary classification: labeling articles as reliable or unreliable . Solving this empowers critical evaluation, combats misinformation, and fosters a more informed society . Ultimately, it aims to contribute to tools that automatically detect misleading content [6]

2. Project Objectives

- **Develop a robust backend API using Flask to receive news article URLs and serve predictions.**
- **Implement advanced NLP techniques (spaCy, Transformers) for feature extraction beyond basic analysis (e.g., part-of-speech tagging, named entity recognition, semantic embeddings).**
- **Integrate and train a high-performing classification model (scikit-learn, Transformers) to categorize articles as "reliable" or "unreliable/fake".**
- **Achieve a target model performance with high accuracy and precision/recall, demonstrating real-world applicability.**
- **Maintain a degree of interpretability in the model to understand the linguistic features driving predictions (e.g., feature importance analysis).**
- **Evolve from Phase 1 by incorporating sophisticated NLP and focusing on a robust backend for practical implementation.**

3. Flowchart of the Project Workflow

User Input:

The user enters the URL of a news article they want to analyze into the Web application's input field.

URL Submission:

The user submits the URL, triggering an event that sends the URL to the backend.

Backend Receives URL:

The Flask backend API receives the submitted URL.

Article Content Fetching:

- **The backend uses the requests library to fetch the HTML content of the article from the provided URL.**
- **If the fetch is successful, the process continues to the next step.**
- **If the fetch fails (e.g., due to an invalid URL or network error), an error message is generated and sent back to the user.**

Data Extraction:

The backend uses Beautiful Soup to parse the HTML content and extract the relevant article text and headline.

NLP Feature Extraction:

- **The extracted text and headline are then processed using Natural Language Processing (NLP) techniques.**
- **This involves using libraries like NLTK for basic NLP tasks (e.g., tokenization, stop word removal) and spaCy/Transformers for more advanced feature extraction (e.g., part-of-speech tagging, named entity recognition, sentiment analysis, and potentially semantic embeddings).**

Model Loading:

The trained fake news detection model is loaded into memory. This model could be a traditional machine learning model (e.g., Naive Bayes, SVM, Random Forest from scikit-learn) or a deep learning model (e.g., RNN, Transformer).

Prediction Generation:

The extracted NLP features are fed into the loaded model to generate a prediction. The model classifies the article as either "reliable" or "unreliable/fake".

Result Display:

- **The prediction result is sent back to the frontend.**
- **The frontend displays the prediction to the user.**
- **If the article is classified as "unreliable/fake", the system highlights the key linguistic indicators or reasons that contributed to the classification (e.g., clickbait headlines,**

emotional language).

User Interaction:

The user views the prediction and the accompanying explanation, enabling them to better understand the potential misinformation.

4. Data Description

❖ Dataset name and origin:

The primary data source for real-time analysis is the content of news articles obtained by scraping user-submitted URLs.

❖ Type of data:

- Text data: **This is the core of your project, consisting of the article text and headlines that you'll be analyzing to detect fake news.**
- Structured data: If you include metadata about the articles (e.g., source name, publication date), that could be considered structured data, often organized in tables or dataframes.

❖ Number of records and features.

Model training utilizes a dataset of [Number] articles with [Number] extracted features (e.g., sentiment scores, clickbait markers). In real-time operation, each user-submitted article is processed to generate those same [Number] features for classification.

❖ Static or dynamic dataset.

The dataset used for training the fake news detection model is **static**, while the data processed by the application real-time (from user-submitted URLs) is **dynamic**."

❖ ● Target variable

The model predicts the reliability of a news article. The target variable is categorical, specifically binary, indicating whether an article belongs to the "reliable" or "unreliable/fake" class.

5. Data Preprocessing

❖ Handle missing values

Missing values in the training dataset will be handled by removing articles with a high proportion of missing data or imputing missing numerical features with the mean/median. During the scraping process, if critical article content is missing, the scrape will be discarded.

❖ Remove or justify duplicate records.

Exact duplicate articles within the training dataset will be identified and removed using pandas. To optimize performance, the application will handle duplicate URL submissions through caching or prevention mechanisms.

❖ Detect and treat outliers

ion.

❖ Convert data types and

ensure consistency.

Data types will be converted as needed (e.g., dates to datetime, text to numerical vectors). Consistency will be enforced by standardizing text encoding, case, whitespace, and categorical representations.

❖ Encode categorical variables

The target variable ("reliable"/"unreliable") will be label-encoded. Nominal categorical features, such as article source, will be one-hot encoded to create binary indicators. Python's scikit-learn and pandas libraries will facilitate these encoding processes.

❖ Normalize or standardize features where required.

Numerical features will be normalized or standardized if necessary to ensure consistent scaling. This will improve model performance and prevent features with larger ranges from dominating.

6. Exploratory Data Analysis (EDA)

1. **Choose a feature.**
2. **Pick a plot:**
    - Histograms (numerical distribution).
    - Boxplots (numerical distribution, outliers)

5. **Repeat** for other

features.

7.FeatureEngineering

Create New Features:

- Generate features from domain knowledge (e.g., clickbait score, source credibility score).
- Develop features based on EDA insights (e.g., frequency of specific words).

Combine/Split Columns:

If applicable, combine or split columns (e.g., extract date parts from a publication date).

Use Feature Engineering Techniques:

- Apply binning to numerical features (e.g., categorize article length).
- Consider polynomial features (e.g., article length²) for non-linear relationships.
- Calculate ratios between features (e.g., exclamation marks to punctuation ratio).

Apply Dimensionality Reduction (Optional):

- If necessary, use techniques like PCA to reduce the number of features.

Justify All Changes:

- Provide clear justification for each feature added, modified, or removed.

Sources and related content

8. Model Building

Select and Implement at Least 2 Machine Learning Models:

- You need to choose at least two different machine learning algorithms to train on your fake news detection task.
- The document suggests examples like Logistic Regression, Decision Tree, Random Forest, and KNN.
- Given the nature of your problem (binary classification), these are indeed suitable starting points.
- However, consider also including models known for strong performance in NLP tasks, such as:
  - Naive Bayes (especially Multinomial Naive Bayes, which works well with text data)
  - Support Vector Machines (SVM)
  - Gradient Boosting algorithms (e.g., XGBoost, LightGBM)

- If you're using Transformers, you would include a Transformer-based model (e.g., a fine-tuned BERT or ROBERTA).

- So, a good selection might be:
  - Logistic Regression (for a linear baseline)
  - Random Forest (for non-linear relationships and feature importance)
  - XGBoost (for high performance)
  - A Transformer-based model (for advanced NLP, if feasible)

Justify Why These Models Were Selected:

- You need to provide a clear rationale for choosing each model.
- Considerations for justification:
  - **Problem Type:** You have a binary classification problem.
  - **Data Characteristics:**
    - You're dealing with text data, which often has high dimensionality.
    - You might have non-linear relationships between linguistic features and fake news.
  - **Model Strengths:**
    - Logistic Regression: Simple, interpretable, good baseline.
    - Decision Tree: Interpretable, can capture non-linearities.
    - Random Forest: Ensemble method, robust, handles non-linearities, provides feature importance.
    - Naive Bayes: Efficient for text classification.
    - SVM: Effective in high-dimensional spaces.
    - XGBoost: High performance, handles complex relationships.
    - Transformers: State-of-the-art for NLP, captures contextual information.
  - **Computational Cost:** Some models (like Transformers) are more computationally expensive.
  - **Interpretability:** Some models (like Decision Trees, Logistic Regression) are easier to interpret than others (like Random Forests, XGBoost).
- Example Justification:

- "Logistic Regression was chosen as a baseline model due to its simplicity and interpretability. Random Forest was selected for its ability to capture non-linear relationships and provide feature importance. XGBoost was included for its high predictive accuracy. A Transformer-based model was chosen to leverage advanced NLP and capture contextual nuances in the text data."

Split Data into Training and Testing Sets:

- You need to divide your dataset into two parts:
  - **Training set:** Used to train the machine learning models.
  - **Testing set:** Used to evaluate the performance of the trained models on unseen data.

- A common split ratio is 80% for training and 20% for testing, but you can adjust this. •

Stratification (If Needed):

- If your target variable (reliable/unreliable) is imbalanced (i.e., one class has significantly more samples than the other), use stratified sampling.
- Stratification ensures that the class distribution in the training and testing sets is similar to the original dataset.
- scikit-learn's train_test_split function has a stratify parameter to help with this.

Train Models and Evaluate Initial Performance:

- Train each of your selected models on the training set.
- Use appropriate evaluation metrics to assess the performance of the trained models on the testing set.
- **For Classification:**
  - Accuracy: Overall correctness.
  - Precision: Ability to correctly identify "unreliable" articles (minimizing false

Visualization of Results & Model Insights

- Confusion matrix
- ROC curve
- Feature importance plot
- Residual plots

The document also emphasizes the importance of including visual comparisons of model performance, interpreting the top features influencing the outcome, and clearly explaining what each plot shows and how it supports conclusions.

10. Tools and Technologies Used

- ***Programming Language:***
    - *Python*
- ***Web Framework (Backend):***
    - *Flask*
- ***Frontend Technologies:***
    - *HTML*
    - *CSS*
    - *JavaScript*
- ***IDE/Notebook:***
    - *VS Code*
    - *Jupyter Notebook*
    - *Google Colab*
- ***Data Handling Libraries:***

- *pandas*
- *NumPy*

- ***Web Scraping Libraries:***
  - *requests*
  - *Beautiful Soup*

- ***NLP Libraries:***
  - *NLTK*
  - *spaCy*
  - *Transformers (Hugging Face)*
  - *TextBlob*

- ***Machine Learning Library:***
  - *scikit-learn*

- ***Visualization Libraries:***
  - *matplotlib*
  - *seaborn*
  - *wordcloud*

- ***Optional Tools (for Deployment):***
  - *Docker*
  - *Cloud hosting platforms (AWS, GCP, Heroku)*
  - *WSGI servers (Gunicorn/uWSGI)*

11. Team Members and Contributions

Mohammed Sakhee.B -Model development

Mohammed Sharuk.I-Feature Engineering

Mubarak Basha.S-EDA

Naseerudin-Data Cleaning

Rishi Kumar Baskar-Documentation and Reporting