Fake News Detection: Hackathon Report

1. Abstract

The objective of this project is to detect fake news using machine learning techniques. We implemented a solution that uses a combination of VADER sentiment analysis for headlines, a Random Forest Classifier for article text, incorporating the various subjects as features for the model and various linguistic features such as average sentence length, Flesch-Kincaid readability, and keywords feature.

---

2. Introduction

In today's digital age, misinformation has become a major concern, particularly with the rapid spread of fake news. In this hackathon, we set out to build a machine learning model that can accurately classify news articles as "real" or "fake". By leveraging multiple features like text sentiment analysis, readability, and keywords, along with subject-based information, we aim to develop a solution capable of detecting fake news.

---

3. Dataset

The dataset used for training and testing contains news articles labeled as 1 (for real) or 0 (for fake). It consists of 5 columns:

- **id**: Unique identifier for each article.
- **title**: The headline of each news article.
- **text**: Full text of the article.

- **date**: The date of publication.
- **label**: The true label indicating whether the article is fake or real, using which our model accuracy will be determined
- **subject**: The category or domain of the article (e.g., politics, world news, etc.).

After processing the datasets through the ML model, features such as sentiment analysis scores, readability metrics, and average sentence length, are added.

---

4. Feature Engineering

To improve the model's accuracy, we employed several feature engineering techniques:

1. **VADER Sentiment Analysis**:
    - We used VADER (Valence Aware Dictionary and sEntiment Reasoner) to analyze the sentiment of the headlines. Its implementation was done using logistic regression, which is useful in cases of binary classification. VADER analyser provides a sentiment score including negative, neutral, positive, and compound scores evaluating the overall sentiment of the sentence after considering the words' context. These features were added to the dataset for use in classification in the form of: neg,neu,pos and compound.
2. **Text-Based Features**:
    - **Average Sentence Length**: A measure of sentence complexity.
    - **Flesch-Kincaid Readability Score**: A metric indicating the readability of the text.

      These features were calculated and added to the dataset as additional predictors for the model.

3. **Keyword Features**:
    - We created a list of keywords (e.g., "exclusive", "scandal", "hoax") which most frequently appeared in fake news and flagged articles containing these words in their headlines or text body. This feature was used as a binary flag (True or False).
4. **Subject Features**:
    - The subject of the article (e.g., politics, government) had been provided in the dataset and was used to inform the classification model (using one hot encoding). Certain subjects may correlate strongly with "real" or "fake" labels, and using these features improved model accuracy.

---

5. Model and Approach

We used a **Random Forest Classifier** to build the classification model for the articles' text. The Random Forest algorithm was chosen for its ability to handle high-dimensional data, its ability to implement feature importance and its robustness in the context of overfitting. We also incorporated the following features:

- Content sentiment scores from VADER.
- Average sentence length and Flesch-Kincaid readability scores.
- Keywords for fake news detection.
- Subject categories (politics, government, left-news, politics, world, news, Middle-east, US).

The training and testing data were split using an 80-20 ratio.

---

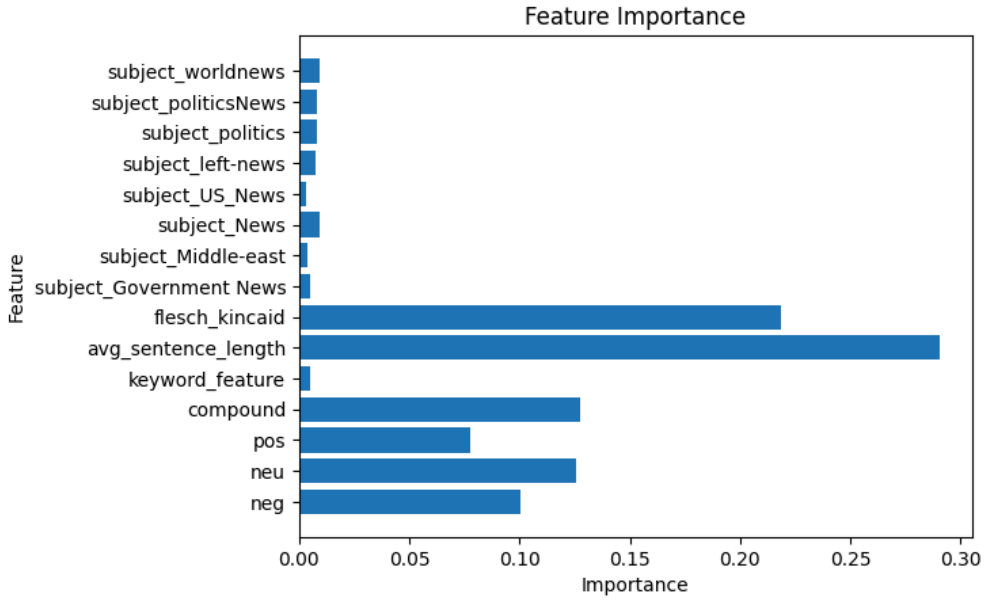
6. Results

After each feature implementation as shown in sections in the code notebook, the model accuracy climbed from initial 54% to 100% by the final stage, for both *train* and *test* datasets.

To ensure that the results were not just a fluke, the labels were shuffled and tested again, the accuracy dropped to **48%**, which is expected when the labels are randomized. This confirmed that the model was not overfitting and there was no data leakage due to overlapping between test and training data. This also implied that the accuracy resulted from the model observing patterns within the data.

Key evaluation metrics:

- **Accuracy**: 100% (on the original dataset before label shuffling).
- **Classification Report**: Included metrics such as precision, recall, and F1-score for both "fake" and "real" classes.
- This is the visual representation of feature importance. As shown, the model relies most on average sentence length, then Flesch-Kincaid, followed by VADER-generated scores, for classification of news articles.



Future Improvements:

- **Incorporate More Complex Features**: Exploring additional linguistic features, like word embeddings (e.g., DistilBERT or GPT), could help the model learn more subtle patterns used in Natural Language Processing.
- **Cross-Domain Validation**: To assess generalization, testing on a completely new dataset from a different domain (not just political or world news) would provide a better measure of model robustness.

---

8. Conclusion

This project demonstrated the ability to classify fake news using a combination of sentiment analysis, readability metrics, keyword detection, and subject-based features. The model showed strong performance on the original dataset, but the drop in accuracy after label shuffling indicates that the model might have relied too heavily on easily detectable patterns, such as subject categories. Future improvements should focus on diversifying the dataset, using advanced natural language processing techniques, and mitigating overfitting by regularizing or excluding overly dominant features.

---

9. Code and Results Files

- **Test Predictions**: The predictions on the test dataset are saved in the results.txt file, formatted as ["headline", “predicted_label”].

  test_results.csv file, formatted as ["article_id", “true_label”, “predicted_label”]

- **Metrics Report**: The classification report is saved in the classification_report.txt file.

- **Code**: The code, including feature engineering, model training, and evaluation, is available in the form of a Jupyter notebook showing the pre-run outputs alongside the code used.