Fake News Detection Using Machine Learning

Report by Ritik Yadav

Introduction

Fake news has become a worldwide issue aided by the exponential rate at which digital platforms are being developed (e.g. social media platforms such as Facebook and Twitter). Users are able to post anything they would like while sharing that information, thus spreading fake news, and content verification is so effortless. Manual detection cannot be feasible due to the vast amount of data to analyze. Machine learning (ML) then becomes useful. ML-based methods are used to automate the detection process more effectively by deriving patterns in the text, language characteristics and metadata.

This literature paper looks into multiple supervised machine learning classifiers that were used for fake news detection, their comparative performance, and research areas of improvement.

Why Machine Learning for Fake News Detection?

Fake news detection has become a necessity according to Ahmed et al., (2017) and Zhou et al., (2019), because it negatively influences both social and political systems. Machine learning models, can absorb and digest large sections of relevant textual data, and learn to differentiate between real and fake.

As Ahmed et al., in the reviewed paper, review that human verification, or rule-based verification will no longer scale or be adequate. Automating this with intelligent classifiers such as SVM, Naïve Bayes, and Logistic Regression yields an efficient and reasonably accurate way to phone fake news, fake news.

Supervised Machine Learning Models for Fake News Detection

Many of the studies reviewed experimented with different ML algorithms using benchmark datasets to classify pieces of news meant as 'real' or 'fake.'

Naïve Bayes (NB): One of the simplest classifiers used for fake news detection. Performs well when the assumption of independence of word occurrence is valid. Accuracy: ~92–96% (Aphiwongsophon et al., 2018). Limitation: Assumes independence between features.

Support Vector Machine (SVM): Effective working with high-dimensional data, such as TF-IDF matrix. Accuracy: ~94–95% (Singh et al., 2017). Limitation: Can be slow and computationally intensive.

Logistic Regression (LR): Used for binary classification activities. Accuracy: ~91–94% (Kaur et al., 2020). Interpretable and computationally efficient.

Random Forest (RF): An ensemble method, generally uses the majority rules and effectively controls for overfitting. Accuracy: ~93% (Ni et al., 2020).

Recurrent Neural Networks (RNN): Useful for sequential data. These models require additional data and computational effort.

K-Nearest Neighbor (KNN): The model makes a classification based on resampling similar data. KNN networks perform well with short-text datasets (Kesarwani et al., 2020).

Feature Extraction Techniques

Common feature extraction techniques include:
- Bag of Words (BoW)
- Term Frequency-Inverse Document Frequency (TF-IDF)
- N-Grams
- Word Embeddings (Word2Vec, GloVe)

However, these techniques do not capture the context of words — a limitation addressed by models like BERT.

Key Research Gaps Identified

- There is a heavy reliance on using traditional ML models.

- Few works utilize BERT or transformer-style models.

- Imbalanced datasets have impacted model performances.

- There is no cross-dataset generalization.

- There are a few hybrid approaches using ML with deep learning.

Opportunity for Improvement

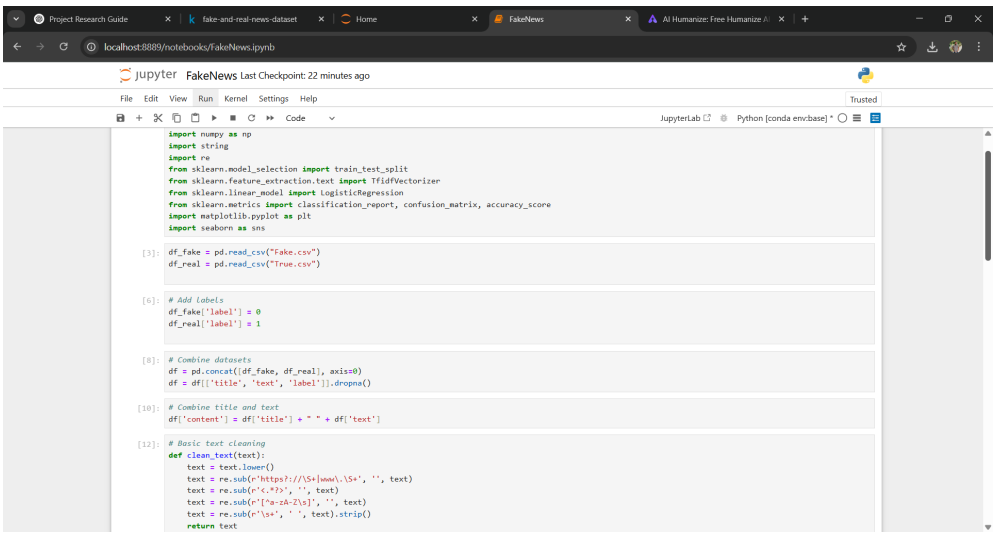
- A hybrid model: TF-IDF + Logistic Regression with BERT.

- Handling imbalanced data using SMOTE or class weighting.
- Using SHAP for model explainability.

Cited Papers

- Shu et al. (2017)
- Ahmed et al. (2017, 2018)
- Zhou et al. (2019)
- Kaur et al. (2020)
- Granik & Mesyura (2017)
- Reis et al. (2019)
- Kaliyar et al. (2020)
- Aphiwongsophon & Chongstitvatana (2018)

Python Code for Improvement



```
import numpy as np
import string
import re
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
```

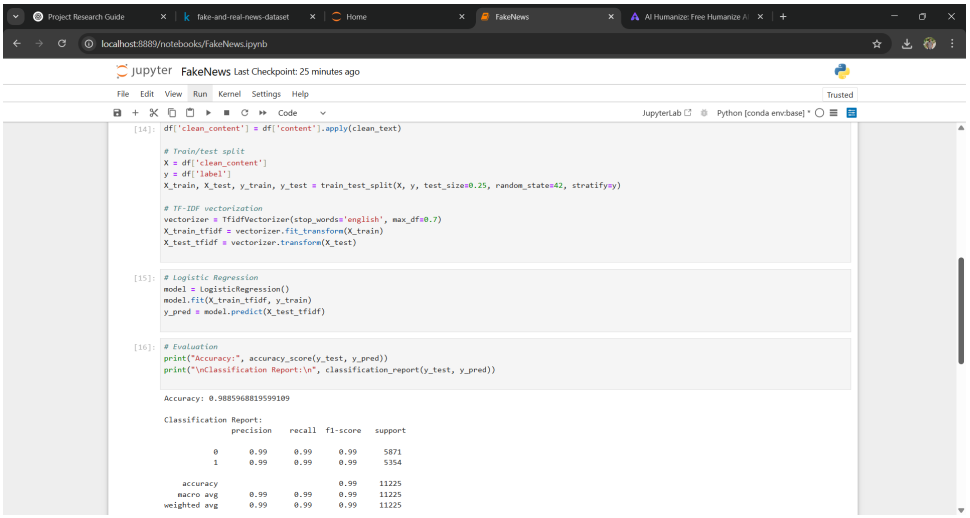
```
[3]: df_fake = pd.read_csv("Fake.csv")
     df_real = pd.read_csv("True.csv")
```

```
[6]: # Add labels
     df_fake['label'] = 0
     df_real['label'] = 1
```

```
[8]: # Combine datasets
     df = pd.concat([df_fake, df_real], axis=0)
     df = df[['title', 'text', 'label']].dropna()
```

```
[10]: # Combine title and text
      df['content'] = df['title'] + " " + df['text']
```

```
[12]: # Basic text cleaning
      def clean_text(text):
          text = text.lower()
          text = re.sub(r'https?://\S+|www\.\S+', '', text)
          text = re.sub(r'<.*?>', '', text)
          text = re.sub(r'[^a-zA-Z\s]', '', text)
          text = re.sub(r'\s+', ' ', text).strip()
          return text
```



```
[14]: df['clean_content'] = df['content'].apply(clean_text)

      # Train/test split
      X = df['clean_content']
      y = df['label']
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42, stratify=y)

      # TF-IDF vectorization
      vectorizer = TfidfVectorizer(stop_words='english', max_df=0.7)
      X_train_tfidf = vectorizer.fit_transform(X_train)
      X_test_tfidf = vectorizer.transform(X_test)
```

```
[15]: # Logistic Regression
      model = LogisticRegression()
      model.fit(X_train_tfidf, y_train)
      y_pred = model.predict(X_test_tfidf)
```

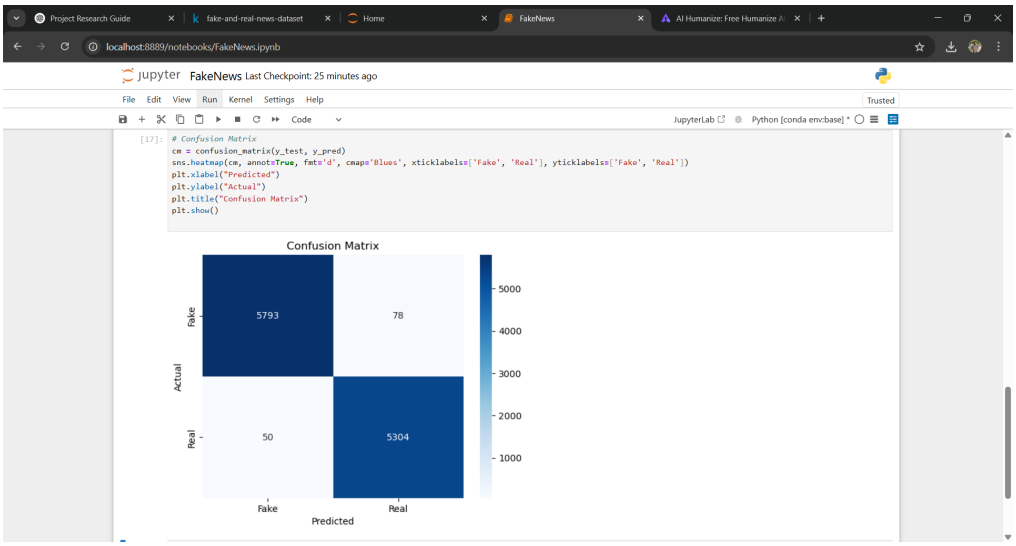
```
[16]: # Evaluation
      print("Accuracy:", accuracy_score(y_test, y_pred))
      print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9885968819599109

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      5871
           1       0.99      0.99      0.99      5354

    accuracy                           0.99     11225
   macro avg       0.99      0.99      0.99     11225
weighted avg       0.99      0.99      0.99     11225
```



```
[17]: # Confusion Matrix
      cm = confusion_matrix(y_test, y_pred)
      sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Fake', 'Real'], yticklabels=['Fake', 'Real'])
      plt.xlabel("Predicted")
      plt.ylabel("Actual")
      plt.title("Confusion Matrix")
      plt.show()
```