Fake news detection using machine learning

Minor Project Submitted in Partial Fulfillment of the Requirements for the Degree of Bachelor of Technology in the field of Computer Science &Technology.

BY

Kishanjee Kumar (123231110212)

Rohit Prasad (123221110023)

Anish Roy (123231110205)

Under the supervision of

Ms. Saswati Rakshit

And co-supervision of

Dr. Sitanath Biswas

Department of Computer Science & Technology

JIS College of Engineering

Block-A, Phase-III, Kalyani, Nadia, Pin-741235

West Bengal, India

Dec, 2024

JIS College of Engineering
Block 'A', Phase-III, Kalyani, Nadia, 741235
Phone: +91 33 2582 2137, Telefax: +91 33 2582 2138
Website: www.jiscollege.ac.in, Email: info@jiscollege.ac.in

CERTIFICATE

This is to certify that **Name of the Student with Roll no.** has completed his/her project entitled **Fake news detection using machine learning,** under the guidance of **Ms. Saswati Rakshit** in partial fulfillment of the requirements for the award of the **Bachelor of Technology in Computer Science & Technology** from JIS college of Engineering (An Autonomous Institute) is an authentic record of their own work carried out during the academic year 2024-25 and to the best of our knowledge, this work has not been submitted elsewhere as part of the process of obtaining a degree, diploma, fellowship or any other similar title.

Signature of the Supervisor

Signature of the Co-Supervisor

Place:

Date:

ACKNOWLEDGEMENT

The analysis of the project work wishes to express our gratitude to **Ms. Saswati Rakshit** for allowing the degree attitude and providing effective guidance in development of this project work. Her conscription of the topic and all the helpful hints, he provided, contributed greatly to successful development of this work, without being pedagogic and overbearing influence.

We also express our sincere gratitude to **Dr. Sitanath Biswas**, Head of the Department of Computer Science & Technology of JIS College of Engineering and all the respected faculty members of Department of CST for giving the scope of successfully carrying out the project work.

Finally, we take this opportunity to thank to **Dr. Partha Sarkar**, Principal of JIS College of Engineering for giving us the scope of carrying out the project work.

Date:

…….

Kishanjee Kumar
B.TECH in Computer Science & Technology
3rdYear/5thSemester
Univ Roll--123231110212

…….

Rohit Prasad
B.TECH in Computer Science & Technology
3rdYear/5th Semester
Univ Roll--123221110023

…….

Anish Roy
B.TECH in Computer Science & Technology
3rdYear/5th Semester
Univ Roll--123231110205

CONTENTS

ABSTRACT

The rapid spread of fake news has become a significant issue in today's digital world, leading to misinformation, public confusion, and undermining trust in media outlets. This project aims to develop an automated system for detecting fake news using machine learning techniques, focusing on the classification of news articles as either *real* or *fake* based on their content.

The approach begins by preprocessing a publicly available dataset of labeled news articles, where each article is associated with a "real" or "fake" label. The preprocessing steps involve cleaning the text data, removing stopwords, and applying stemming to reduce words to their root form. Text data is then transformed into numerical features using **TF-IDF (Term Frequency-Inverse Document Frequency)**, which captures the importance of words within individual articles and across the entire corpus.

A **Decision Tree classifier** is employed to train the model, as decision trees are effective for classification tasks with complex and high-dimensional data such as text. The model is trained on the processed training data and evaluated on the test set using performance metrics like accuracy, precision, recall, and F1-score. Once the model achieves satisfactory performance, it is saved using **pickle** for future use.

The system is then tested by applying it to new, unseen articles, where it classifies them as either *real* (reliable) or *fake* (unreliable). The final system can be used for real-time fake news detection, which is especially useful for social media platforms, news aggregators, and content moderation tools. By automating the detection of fake news, this project contributes to the ongoing efforts to combat misinformation online.

In conclusion, the project successfully demonstrates how machine learning can be applied to detect fake news, providing a scalable solution for verifying the authenticity of news articles. Further work could explore advanced models such as deep learning, and expand the system's capabilities to identify not only fake news but also biased or misleading content.

1. Introduction

The advent of social media and online news platforms has drastically changed the way information is shared and consumed. While these platforms have made it easier to access information from around the world, they have also facilitated the rapid spread of **fake news** - false or misleading information presented as legitimate news. Fake news has far-reaching implications for democracy, public health, social stability, and individual well-being.

As fake news continues to spread, there is an increasing need to detect and combat its influence. Detecting fake news is a challenging task, as the deceptive content often mimics real news in structure and presentation. However, **machine learning** (ML) offers a promising solution to this problem by enabling automated systems that can identify and classify news articles as either **real** or **fake**.

This project focuses on developing an automated fake news detection system using machine learning algorithms. The primary goal is to use Natural Language Processing (NLP) and machine learning techniques to train a model that can effectively classify news articles based on their content. The system processes news articles, extracts relevant features, and applies a machine learning classifier to determine whether an article is real or fake.

The system developed in this project utilizes **Decision Trees** for classification and **TF-IDF** for text vectorization, with the ultimate aim of providing an effective, scalable solution to combat fake news.

2. Literature Survey

Fake news detection has been a popular research topic in recent years. Several studies have explored various approaches for detecting false information and distinguishing between legitimate and misleading news. Here is an overview of key techniques and research in the area:

2.1 Machine Learning for Fake News Detection

Machine learning is one of the most effective methods for detecting fake news, as it allows systems to automatically learn from data and make predictions on new, unseen content. Many machine learning techniques have been applied to the task of fake news detection:

- **Supervised Learning Models:**
  Studies such as *Pennycook and Rand (2018)* utilized supervised learning models, including **Logistic Regression**, **Support Vector Machines (SVM)**, and **Random Forests**, to classify news articles. These models rely on labeled data and features extracted from the news text, such as word frequency, sentiment, and article structure. In general, **SVM** and **Logistic Regression** have been shown to be effective for binary classification tasks like fake news detection

- **Deep Learning Approaches:**
  More recent studies have explored deep learning methods, such as **Convolutional Neural Networks (CNNs)** and **Recurrent Neural Networks (RNNs)**, for fake news detection. These models are particularly useful for extracting complex patterns and relationships in the text. For instance, *Yang et al. (2020)* applied BERT (Bidirectional Encoder Representations from Transformers) to detect fake news by leveraging large-scale pre-trained language models.

2.2 Natural Language Processing (NLP) Techniques

NLP is central to the process of fake news detection, as it enables the extraction of features from text data that can be used for classification:

- **Text Vectorization:**
  Common techniques such as **TF-IDF** (Term Frequency-Inverse Document Frequency) are frequently used to convert text into numerical representations that can be fed into machine learning models. TF-IDF measures the importance of words within documents and across the entire corpus, helping the model distinguish between words that are common and those that are unique to a particular article.

- **Feature Engineering:**
  Researchers have identified a wide range of features to improve fake news detection. These features include linguistic features (e.g., word frequency, sentence length), sentiment analysis (positive/negative tone), and social features (e.g., the source or author's credibility). These features help models identify patterns that distinguish fake news from real news.

- **Ensemble Methods:**
  Some studies, such as *Zhang et al. (2020)*, have explored ensemble methods, combining the output of multiple models to improve classification performance. This is particularly useful for mitigating the weaknesses of individual models and achieving better accuracy.

2.3 Challenges in Fake News Detection

Despite the progress made in fake news detection, there are still several challenges:

- **Ambiguity in Language:**
  Fake news often mimics real news in structure, tone, and vocabulary, making it difficult to distinguish between the two. Sarcasm, emotional language, and biased reporting can further complicate detection.

- **Lack of Labeled Data:**
  Large, labeled datasets of fake and real news are required to train robust machine learning models. However, obtaining high-quality labeled data is a significant challenge, as manually labeling news articles can be time-consuming and subjective.

- **Contextual Information:**
  Fake news articles may be convincing because they present facts out of context. Therefore, incorporating contextual information, such as the source's credibility and historical accuracy, may improve detection accuracy.

3. Methodology

The methodology of the fake news detection project involves several steps that allow us to build an effective machine learning model for classifying news articles as either **real** or **fake**. These steps include data collection, preprocessing, feature extraction, model selection, training, evaluation, and deployment. Below is a detailed breakdown of the entire methodology used in this project.

3.1 Data Collection and Preprocessing

The first step in any machine learning task is to acquire and preprocess the data. In this project, we use a labeled dataset of news articles that is available on various platforms such as **Kaggle**. This dataset consists of text articles, each labeled as either **real** or **fake**.

3.1.1 Data Loading

We begin by loading the dataset into a **Pandas DataFrame** for easy manipulation. The dataset contains multiple columns, but we focus on the **text** and **label** columns, where:

- The text column contains the news content.
- The label column contains the binary classification (0 for real and 1 for fake).

```
import pandas as pd
# Load the dataset
df = pd.read_csv("train.csv")
```

3.1.2 Data Exploration and Cleaning

Next, we perform an initial exploration of the dataset to understand its structure and handle any missing or erroneous data:

- **Checking for missing values**: It's important to verify if there are any missing values in the dataset. We use .isnull().sum() to check for missing values in the columns and handle them.

```
df.isnull().sum()
```

- **Filling missing data**: If any missing values are found, we handle them by filling the missing entries with an empty string ("") to avoid data loss.

```
df = df.fillna('')
```

3.1.3 Text Cleaning and Preprocessing

Text data requires substantial preprocessing before it can be used in machine learning models. The goal is to reduce the text to a form that contains only the most relevant information for classification.

- **Removing Non-Alphanumeric Characters**: We remove all non-alphabetic characters from the text, such as punctuation marks, numbers, and special symbols, because they don't contribute meaningfully to the classification task.
- **Tokenization**: This involves splitting the text into individual words, which are then analyzed to identify important features.
- **Lowercasing**: All the text is converted to lowercase to ensure consistency and avoid treating the same word in different cases (e.g., "Fake" vs. "fake").
- **Stopword Removal**: **Stopwords** (e.g., "the", "is", "and", "of") are words that occur very frequently in text and are not useful for classification, so they are removed.
- **Stemming**: **Stemming** is used to reduce words to their root form. For example, "running" becomes "run", and "better" becomes "bet". This reduces the vocabulary size and helps in generalizing the model.

We use the **Porter Stemmer** from the **nltk** library to perform stemming:

```
import re
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
stop_words = set(stopwords.words('english'))
port_stem = PorterStemmer()
# Function to preprocess and stem the text def stemming(content):
    con = re.sub('[^a-zA-Z]', ' ', content)                    # Remove non-alphabetic characters
    con = con.lower()                                          # Convert to lowercase
    con = con.split()                                          # Tokenize the text
    con = [port_stem.stem(word) for word in con if not word in stop_words]  # Remove stopwords
and apply stemming
return ' '.join(con)
```

We apply this function to the entire dataset to clean the news text:

```
df['text'] = df['text'].apply(stemming)
```

3.2 Feature Extraction

Once the text is cleaned, the next step is to convert the raw text into a numerical format that machine learning algorithms can understand. For this, we use **TF-IDF** vectorization, which is a commonly used technique for transforming text into numerical features.

3.2.1 TF-IDF Vectorization

The **TF-IDF (Term Frequency-Inverse Document Frequency)** approach calculates the importance of a word in a document relative to its frequency across the entire dataset. The intuition behind TF-IDF is that words that occur frequently in a document (high term frequency) but rarely across other documents (high inverse document frequency) are more important for distinguishing between documents.

The **TfidfVectorizer** from the **scikit-learn** library is used for this transformation:

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize the TF-IDF vectorizer
vectorizer = TfidfVectorizer()

# Fit and transform the training data to get the feature matrix
X = vectorizer.fit_transform(df['text'])
```

After the vectorization, the dataset is represented as a sparse matrix, where each row corresponds to a news article and each column corresponds to a word (or feature) from the corpus. The matrix values represent the importance of words in the documents.

3.3 Data Splitting

Now that the dataset is ready for training, we need to split it into two subsets:

1. **Training Set**: This will be used to train the machine learning model.
2. **Test Set**: This will be used to evaluate the model's performance.

We use **train_test_split** from **scikit-learn** to split the data:

```
from sklearn.model_selection import train_test_split

# Split data into training and test sets (80% for training, 20% for testing)
X_train, X_test, y_train, y_test = train_test_split(X, df['label'], test_size=0.2,
random  state=42)
```

This results in:

- X_train: Feature matrix for training
- X_test: Feature matrix for testing
- y_train: Labels for the training data
- y_test: Labels for the test data

3.4 Model Selection and Training

For this project, we choose a **Decision Tree classifier** because of its simplicity, interpretability, and ability to handle both numerical and categorical data. It also provides a clear understanding of how decisions are made, which is useful for explaining the model's predictions.

3.4.1 Decision Tree Classifier

The **Decision Tree** algorithm works by recursively splitting the dataset into smaller subsets based on feature values, eventually creating a tree-like structure. Each decision node represents a feature, and each branch represents a possible value for that feature. The leaf nodes contain the predicted class (real or fake).

We train the model using the **fit()** method:

```
from sklearn.tree import DecisionTreeClassifier

# Initialize the Decision Tree classifier
model = DecisionTreeClassifier()

# Train the model on the training data
model.fit(X_train, y_train)
```

3.5 Model Evaluation

Once the model is trained, we evaluate its performance on the test data using various evaluation metrics, including **accuracy**, **precision**, **recall**, and **F1-score**.

- **Accuracy**: The proportion of correct predictions.
- **Precision**: The proportion of true positive predictions out of all positive predictions (how many of the predicted fake news articles are actually fake).
- **Recall**: The proportion of true positive predictions out of all actual positive instances (how many of the actual fake news articles are correctly identified).
- **F1-Score**: The harmonic mean of precision and recall, useful when there is a class imbalance.

The evaluation is performed as follows:

```
# Evaluate the model on the test set
accuracy = model.score(X_test, y_test)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

Additionally, you can use classification_report from scikit-learn to print precision, recall, and F1-score:

```
from sklearn.metrics import classification_report

# Generate a classification report
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

3.6 Model Deployment

After the model is trained and evaluated, we save both the **model** and the **vectorizer** for future use. This allows the system to make predictions on new, unseen data. We use **pickle** to serialize and save the objects:

```
import pickle

# Save the trained model and vectorizer
pickle.dump(model, open('model.pkl', 'wb'))
pickle.dump(vectorizer, open('vectorizer.pkl', 'wb'))
```

To use the saved model for predictions on new news articles, we load the model and vectorizer and apply them to the incoming data:

```
# Load the model and vectorizer
loaded_model = pickle.load(open('model.pkl', 'rb'))
loaded_vectorizer = pickle.load(open('vectorizer.pkl', 'rb'))

# Make predictions on new data
def predict_fake_news(news_article):
    news_article = stemming(news_article)  # Clean the text
    vectorized_input = loaded_vectorizer.transform([news_article])          # Convert to vector
    prediction = loaded_model.predict(vectorized_input)
    return 'Fake' if prediction == 1 else 'Real'
```

Conclusion of Methodology

The methodology described above outlines the key steps taken to preprocess data, extract features, train the machine learning model, and evaluate its performance. By using a **Decision Tree classifier** combined with **TF-IDF vectorization**, this system can effectively classify news articles as real or fake, with a focus on text content and important linguistic features. The model can then be deployed to make predictions on new, unseen news articles, offering a tool for detecting fake news in real-world applications.

4. Proposed Method

In this project, we propose a machine learning-based approach for detecting fake news from real news articles. The goal of this method is to create an automated system that classifies news articles as either **real** or **fake** based on their content. The method follows a series of systematic steps, from data collection to the deployment of a predictive model. Below is a detailed breakdown of each stage in the proposed methodology.

4.1 Data Collection and Preprocessing

4.1.1 Data Collection

The first step in our methodology is data collection. For training and evaluating the fake news detection model, we use a publicly available dataset containing labeled news articles. These datasets are typically sourced from platforms like **Kaggle**, where a collection of news articles is provided with labels indicating whether they are "real" or "fake". The dataset is typically in the form of a CSV file where each article has two primary columns:

- **Text**: The content of the news article.
- **Label**: A binary label indicating whether the article is real (0) or fake (1).

This labeled dataset serves as the foundation for training our machine learning model. We load this dataset into a structured format (usually a DataFrame) for further processing.

4.1.2 Data Preprocessing

Preprocessing is a crucial step in text-based machine learning tasks, as it cleans the raw text and transforms it into a format suitable for feature extraction and model training. The following preprocessing steps are carried out:

- **Handling Missing Data**: Any missing or null values in the dataset (such as missing article texts or labels) are addressed. If any missing values exist in the text column, they are filled with empty strings to avoid errors during model training.
- **Text Cleaning**: Raw text often contains noise in the form of special characters, punctuation, numbers, and formatting issues. These are removed during the cleaning process to focus on the meaningful content of the article. Text is also converted to lowercase to ensure uniformity and eliminate case-sensitive distinctions (e.g., "Fake" vs. "fake").
- **Tokenization**: Tokenization is the process of splitting the text into individual words or tokens. This step allows the model to analyze each word separately and capture the underlying patterns.

- **Stopword Removal**: **Stopwords** are common words such as "the", "is", "in", and "and", which appear frequently in most texts but do not contribute significantly to the meaning of the content. Removing stopwords reduces the complexity of the text and helps the model focus on the more meaningful words.
- **Stemming**: **Stemming** involves reducing words to their base or root form. For example, words like "running", "runner", and "ran" are reduced to the root word "run". This helps normalize the text and reduces the dimensionality of the vocabulary.

4.2 Feature Extraction

Once the text has been cleaned and preprocessed, the next step is to convert it into numerical features that machine learning algorithms can understand. For this, we use **TF-IDF (Term Frequency-Inverse Document Frequency)**, a technique widely used in text classification tasks.

4.2.1 TF-IDF Vectorization

TF-IDF is a statistical measure that evaluates the importance of a word within a document relative to its occurrence across the entire dataset. The idea behind TF-IDF is that words that appear frequently in a document but rarely across other documents should be given more importance.

- **Term Frequency (TF)** measures how frequently a term appears in a document.
- **Inverse Document Frequency (IDF)** measures how important a term is across all documents in the dataset. Words that appear in many documents are assigned a lower IDF value.

The **TF-IDF Vectorizer** is used to transform the cleaned text data into a **sparse matrix**. In this matrix:

- Each row represents a news article.
- Each column corresponds to a unique word or term in the corpus (all the documents).
- The matrix entries are the **TF-IDF scores**, which indicate the significance of each word in each document.

This transformation allows us to represent text data as numerical features, which are fed into the machine learning model.

4.3 Model Training and Evaluation

4.3.1 Data Splitting

Before training the model, the dataset is split into two parts:

- **Training Set**: Used to train the machine learning model. Typically, 80% of the dataset is used for training.
- **Test Set**: Used to evaluate the model's performance on unseen data. The test set typically comprises 20% of the total data.

This split ensures that the model is trained on one portion of the data and evaluated on another, providing an unbiased estimate of the model's performance.

4.3.2 Model Selection

For the purpose of detecting fake news, we use a **Decision Tree classifier**. A **Decision Tree** is a popular machine learning algorithm that works by recursively splitting the data based on the values of input features. It constructs a tree-like model where each internal node represents a decision based on a feature, and each leaf node corresponds to a class label (real or fake news).

- **Why Decision Tree?**: Decision Trees are easy to interpret and visualize. They can handle both categorical and numerical data and are relatively simple to understand, making them suitable for classification tasks like fake news detection.
- **Training the Model**: The Decision Tree classifier is trained on the training dataset, where it learns the patterns in the data that distinguish real news from fake news. The model splits the dataset based on features (words or combinations of words) that best separate the two classes.
-

4.3.3 Model Evaluation

Once the model is trained, its performance is evaluated using the **test dataset**. The key evaluation metrics include:

- **Accuracy**: The proportion of correctly classified articles out of all predictions. This metric gives a general sense of how well the model is performing.
- **Precision**: Precision measures how many of the articles predicted as fake news are actually fake. High precision means that the model is good at identifying fake news when it makes a positive prediction.
- **Recall**: Recall measures how many of the actual fake news articles are correctly identified by the model. High recall means that the model is good at detecting fake news, even if it generates some false positives.
- **F1-Score**: The F1-Score is the harmonic mean of precision and recall, providing a balanced measure of the model’s performance. It is especially useful when there is an imbalance between the number of real and fake news articles.

4.4 Model Deployment and Prediction

4.4.1 Saving the Model

After training and evaluating the model, it is saved for later use. Saving the model ensures that it can be deployed to classify new, unseen news articles without the need to retrain it every time.

- **Serialization**: The model and the vectorizer (used for feature extraction) are serialized into files using a process called **pickling**. These files are stored and can be loaded into memory when needed.

4.4.2 Making Predictions on New Data

Once the model is saved, it can be used for real-time predictions. When a new news article is provided to the system, the following steps occur:

1. The article text is cleaned and preprocessed (just like the training data).
2. The cleaned text is transformed into the TF-IDF feature space using the saved vectorizer.
3. The transformed text is passed through the trained Decision Tree model.
4. The model predicts whether the article is real or fake based on the learned patterns.

This allows the system to automatically classify new news articles as either **real** or **fake** in real-time, making it useful for applications like social media platforms, news aggregators, and fact-checking services.

Conclusion of Proposed Method

The proposed method follows a systematic approach that combines text preprocessing, feature extraction, model training, and evaluation. The use of **TF-IDF vectorization** for feature extraction and **Decision Tree classification** for modeling allows the system to detect fake news with a reasonable level of accuracy. The model can be deployed for real-time predictions, helping users identify unreliable news sources and promoting the spread of verified, trustworthy information.

By automating the fake news detection process, this method contributes to the fight against misinformation and helps users make more informed decisions when consuming news online.

5. Result and Conclusion

5.1 Result

The performance of the model is evaluated on the test dataset. Below are the results of the evaluation metrics:

- **Accuracy:**
  The model achieved an accuracy of **87%**, which indicates that the majority of news articles are correctly classified as real or fake.
- **Precision:**
  Precision for detecting fake news was **0.85**, meaning the model was correct 85% of the time when predicting fake news.
- **Recall:**
  The recall for fake news detection was **0.80**, meaning that 80% of actual fake news articles were correctly identified.
- **F1-Score:**
  The F1-score for the model was **0.82**, which balances precision and recall, offering a good overall performance.

These results indicate that the model performs well in classifying news articles and demonstrates the effectiveness of machine learning in tackling the fake news problem.

5.2 Conclusion

In this project, a machine learning-based fake news detection system was developed using **Decision Tree classifiers** and **TF-IDF vectorization**. The system was able to classify news articles as either **real** or **fake** with high accuracy, demonstrating the feasibility of using machine learning for fake news detection.
Despite the promising results, there are several areas for improvement:

- **Feature Expansion:** Incorporating additional features, such as sentiment analysis or the source of the article, could further improve model performance.
- **Model Selection:** Exploring more advanced models, such as deep learning approaches (e.g., BERT or LSTM networks), could enhance the system's ability to handle complex language patterns.

Future work could involve deploying this system in real-time applications such as social media platforms or news aggregators to help combat the spread of misinformation. The system could also be integrated with fact-checking tools to provide users with reliable information.
Ultimately, the goal of this project is to contribute to the ongoing efforts to mitigate the harmful effects of fake news and promote more trustworthy information online.