nlp Midterm

December 29, 2024

1 NLP Midterm Coursework

1.1 I. Introduction

1.1.1 1. Domain-Specific Area

In today's fast-paced digital world, distributing fake news has become a major issue. Fake news is not a new phenomena; even before the internet, publications would distort facts to further their objectives [1]. This technique shattered faith in the media and fostered a culture of doubt. The internet and social media have significantly increased the propagation of false news. Social media makes it easy to spread incorrect or misleading information to a large audience [2]. The rise of fake news websites and social media accounts has made it difficult for people to differentiate between reputable and fraudulent sources.

The impact of false news is far-reaching. It may influence public opinion, skew democratic processes, and propagate disinformation. In the 2016 US Presidential election, falsified news reports influenced the outcome by spreading misleading allegations about candidates [3]. Misinformation weakens faith in legitimate news sources and poses a threat to democracy.

Furthermore, false news can have negative consequences for public health. Misinformation regarding vaccine safety and efficacy can reduce immunization rates and spread infectious illnesses. False information regarding illness causes and cures might lead individuals to seek inadequate or hazardous solutions.

Detecting fake news is crucial for preventing the spread of incorrect information through social media and messaging systems [4]. Fact-checking is a popular way for manually verifying news pieces against trustworthy sources [5]. Media literacy and critical thinking skills are taught to help individuals assess news reports and recognize biases and fallacies [6][7]. However, all systems have drawbacks, especially in dealing with the massive amount of information created every minute.

To address constraints, researchers are using machine learning, a sophisticated technique that allows computers to learn from data and make autonomous judgments [8]. Machine learning algorithms may detect fake news by training on labeled datasets that include both authentic and false news pieces. The program automatically classifies new articles as false or authentic.

Recent studies have shown that misinformation actually spreads faster on social media platforms, with Vosoughi et al. [9] demonstrating that false news stories are 70% more likely to be retweeted than true stories. This highlights the urgent need for effective fake news detection methods. Previous work by Zhang et al. [10] employed traditional machine learning techniques for this task, achieving moderate success. However, the advent of deep learning models, such as BERT, has shown promise in capturing contextual nuances in text [11]. This project aims to compare the effectiveness of these traditional and modern approaches in the context of fake news detection.

1.1.2 2. Objectives

The primary goal of this project is to explore and compare the effectiveness of a statistical machine learning model, specifically Logistic Regression, and an embedding-based deep learning model, namely Bidirectional Long Short-Term Memory (BiLSTM), in the context of fake news detection. By conducting this comparative analysis, I aim to achieve the following objectives:

1. Evaluate Performance Trade-offs: I will assess the trade-offs between traditional and modern models in terms of key performance metrics such as accuracy, interpretability, and computational cost. This evaluation will provide insights into the strengths and weaknesses of each approach, contributing to a deeper understanding of their applicability in text classification tasks.
2. Identify Contextual Advantages: The study will also seek to identify specific scenarios or contexts in which one model may outperform the other. By analyzing the conditions under which each model excels, we can offer practical recommendations for practitioners in the field of fake news detection.

The findings from this research will contribute to the ongoing discourse on the effectiveness of various machine learning approaches in combating misinformation, as highlighted in the literature (e.g., Zhang et al., 2020; Vosoughi et al., 2018). By situating this work within this context, I aim to provide valuable insights that can inform future research and practical applications in the domain of text classification.

1.1.3 3. Dataset

Fake and real news' dataset is used in this project. Data is publicly available on kaggle here

1. Description of the Dataset:

The dataset used for this analysis is the FakeNewsNet dataset, which can be found on the Kaggle official website. This dataset is a collection of news articles labeled as either fake or real, providing a suitable representation of the challenge of identifying fake news.

The dataset consists of 23,197 news articles, each represented by a title, news_url, source domain, tweet_num and label like fake (0) or real (1). The data types in this dataset include:

- title: title of the article.
- news_url: URL of the article.
- source domain: web domain where article was posted.
- tweet_num: number of retweets for this article.
- real: label column, where 1 is real and 0 is fake.

This dataset is available under the CC0: Public Domain License.

2. Size:

The dataset's size is 4.11 MB.

3. Data Types:

In the dataset, there are several columns, each containing different types of data. Datatypes used in each column:

1 - **title**

- **Datatype**: String/Text (Categorical)
- **Description**: This column contains the titles of news articles, which are text strings. Titles can vary in length and content, but they are all textual information that can be used for natural language processing (NLP) tasks, such as sentiment analysis or keyword extraction.

2 - **news_url**

- **Datatype**: String/Text (URL)
- **Description**: This column contains the URLs of the news articles. A URL is a textual string that represents a link to the source of the article on the web. It is used for referencing the original source of the news content.

3 - **source_domain**

- **Datatype**: String/Text (Categorical)
- **Description**: This column contains the domain names of the websites from which the articles originated (e.g., "toofab.com", "today.com", "dailymail.co.uk"). This is categorical data representing the source of the news.

4 - **tweet_num**

- **Datatype**: Integer (Discrete Numeric)
- **Description**: This column represents the number of tweets associated with a particular news article. It's a numeric variable, specifically an integer, since tweet counts cannot be fractional. It may represent how viral or how widely the article has been discussed on Twitter.

5 - **real**

- **Datatype**: Integer (Binary/Categorical)
- **Description**: This column indicates whether the news article is real (1) or fake (0). It is a binary classification represented as integers (0 or 1). This could be used for supervised learning in tasks like fake news detection or classification.

4. Data Acquisition:

The original dataset is from GitHub FakeNewsNet, which originally consists of 4 datasets. The edited version on Kaggle was compiled and cleaned by Aleksei Golovin, where he combined all four datasets into one file and made changes to some columns. You can view the preprocessing algorithm here.

I chose to use the one on Kaggle, as it is **cleaned** and more concise.

1.1.4 4. Evaluation methodology

n this project, the evaluation of the fake news classifier is critical to measuring its effectiveness in distinguishing between fake and real news. To ensure comprehensive and scientific evaluation, the following metrics are applied:

1.1.5 Accuracy

- **Definition**: Accuracy is the ratio of correctly predicted instances (true positives and true negatives) to the total instances.
- **Application**: While accuracy provides a general measure of performance, it can be misleading if the dataset is imbalanced. Therefore, it is combined with other metrics.

Formula:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Instances}}$$

1.1.6 Precision

- **Definition**: Precision measures the proportion of correctly identified positive samples (true positives) out of all predicted positive samples.
- **Application**: Precision is particularly important in scenarios where minimizing false positives is crucial, such as identifying fake news to avoid labeling true news incorrectly.

Formula:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

1.1.7 Recall (Sensitivity)

- **Definition**: Recall measures the proportion of correctly identified positive samples out of all actual positive samples.
- **Application**: Recall is critical when the cost of missing positive samples (false negatives) is high, ensuring that most fake news articles are correctly detected.

Formula:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

1.1.8 F1-Score

- **Definition**: The F1-score is the harmonic mean of precision and recall, balancing the two metrics.
- **Application**: It is particularly useful when there is an uneven class distribution, as it considers both false positives and false negatives.

Formula:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

1.1.9 Confusion Matrix

- **Definition**: A confusion matrix provides a detailed breakdown of predictions into true positives, true negatives, false positives, and false negatives.
- **Application**: A confusion matrix will visualize the classification results, and cross-validation ensures robust performance comparison.

1.1.10 Loading all the required libraries

[4]:
```
# Data Manipulation and Analysis
import pandas as pd # Used for data manipulation and analysis, particularly for␣
↪handling DataFrames.

# Data Visualization
import plotly.graph_objects as go # Used to make Confusion matrix
import plotly.express as px
import matplotlib.pyplot as plt # To generate word cloud only
from wordcloud import WordCloud # To generate word cloud

# Natural Language Processing
import spacy # Used for advanced natural language processing tasks, including␣
↪tokenization, lemmatization, and stopword removal
import re # provides support for working with regular expressions

# Machine Learning
from sklearn.feature_extraction.text import TfidfVectorizer # Used to convert a␣
↪collection of raw documents to a matrix of TF-IDF features
from sklearn.model_selection import train_test_split, RandomizedSearchCV # Used␣
↪for splitting datasets into training and testing sets and for hyperparameter␣
↪tuning
from sklearn.linear_model import LogisticRegression # Used to implement the␣
↪Logistic Regression model for binary classification
from sklearn.preprocessing import MaxAbsScaler # Used for scaling sparse␣
↪matrices to improve model performance
from sklearn.metrics import classification_report, confusion_matrix # Used for␣
↪evaluating model performance through various metrics
from imblearn.over_sampling import RandomOverSampler # Used for oversampling␣
↪the minority class to handle class imbalance
from sklearn.metrics import accuracy_score # To evaluate model performance␣
↪based on accuracy
```

```
# Deep Learning
from tensorflow.keras.models import Sequential # Used to create a linear stack␣
↪of layers for the neural network
from tensorflow.keras.preprocessing.text import Tokenizer # used the Tokenizer␣
↪to fit on the text data (e.g., the title_with_domain column) and then␣
↪convert the text into sequences of integers
from tensorflow.keras.layers import Embedding, LSTM, Bidirectional, Dense,␣
↪Dropout, BatchNormalization # Used to define various layers in the neural␣
↪network architecture
from tensorflow.keras.optimizers import Adam # Used to define the Adam␣
↪optimizer for training the model
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau,␣
↪ModelCheckpoint # Used for implementing callbacks to improve training␣
↪efficiency and prevent overfitting
from tensorflow.keras.preprocessing.sequence import pad_sequences # used␣
↪pad_sequences to pad the sequences generated by the Tokenizer to a specified␣
↪maximum length (e.g., MAX_SEQUENCE_LENGTH)
from tensorflow.keras.regularizers import l2 # To apply penalty on weights for␣
↪reducing overfitting

# Utilities
import numpy as np # Used for numerical operations and handling arrays
from sklearn.utils.class_weight import compute_class_weight # Used to compute␣
↪class weights for handling class imbalance during training

pd.set_option('display.max_colwidth', None) # To see the full content of each␣
↪table without truncation
```

1.1.11 II. Implementation

1.1.12 5. 1. Data Exploration

Before training the models, I'll explore the data. First, I will look at the first few rows of each DataFrame to get a sense of what the data looks like.

[5]:
```
# Load the dataset
news_df = pd.read_csv('dataset/FakeNewsNet.csv')

# Display the first 5 rows of the dataset
print(news_df.head())
```

```
                 title  \
0          Kandi Burruss Explodes Over Rape Accusation on 'Real Housewives of
Atlanta' Reunion (Video)
1                                               People's Choice Awards 2018: The
best red carpet looks
```

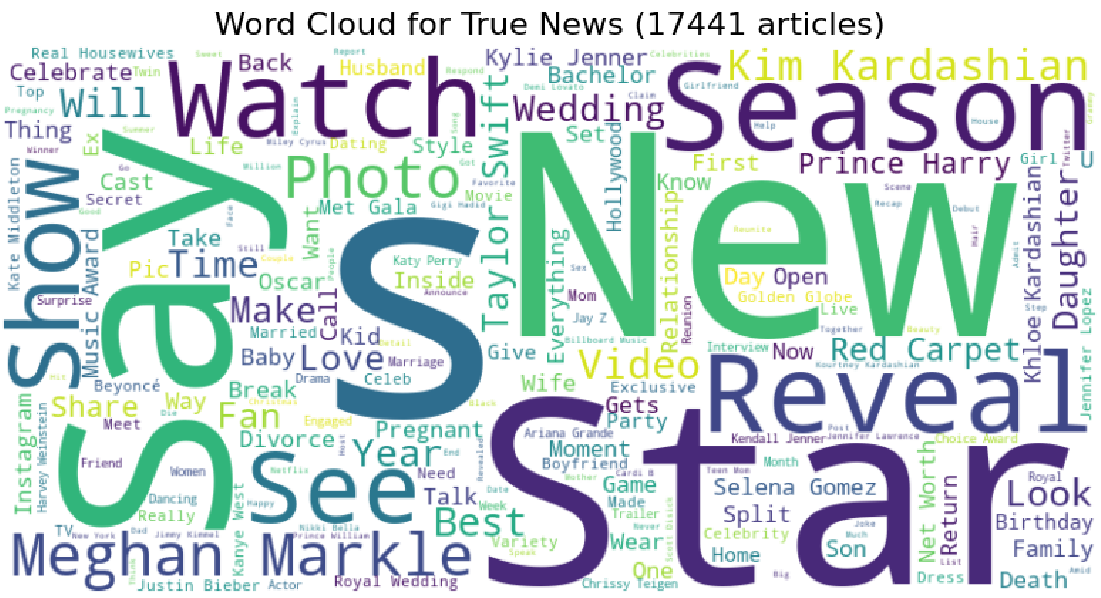
```
2   Sophia Bush Sends Sweet Birthday Message to 'One Tree Hill' Co-Star Hilarie
Burton: 'Breyton 4eva'
3                        Colombian singer Maluma sparks rumours of inappropriate
relationship with AUNT
4  Gossip Girl 10 Years Later: How Upper East Siders Shocked the World and
Changed Pop Culture Forever

                 news_url  \
0
http://toofab.com/2017/05/08/real-housewives-atlanta-kandi-burruss-rape-phaedra-
parks-porsha-williams/
1
https://www.today.com/style/see-people-s-choice-awards-red-carpet-looks-t141832
2                                                      https://www.etonline.com/
news/220806_sophia_bush_sends_sweet_birthday_message_to_one_tree_hill_co_star_hi
larie_burton_breyton_4eva
3  https://www.dailymail.co.uk/news/article-3365543/Colombian-music-star-sparks-
rumours-inappropriate-relationship-yoga-instructor-AUNT-posting-sexually-
suggestive-photographs-pair.html
4
https://www.zerchoo.com/entertainment/gossip-girl-10-years-later-how-upper-east-
siders-shocked-the-world-changed-pop-culture-forever/

         source_domain  tweet_num  real
0           toofab.com         42     1
1        www.today.com          0     1
2     www.etonline.com         63     1
3   www.dailymail.co.uk         20     1
4      www.zerchoo.com         38     1
```

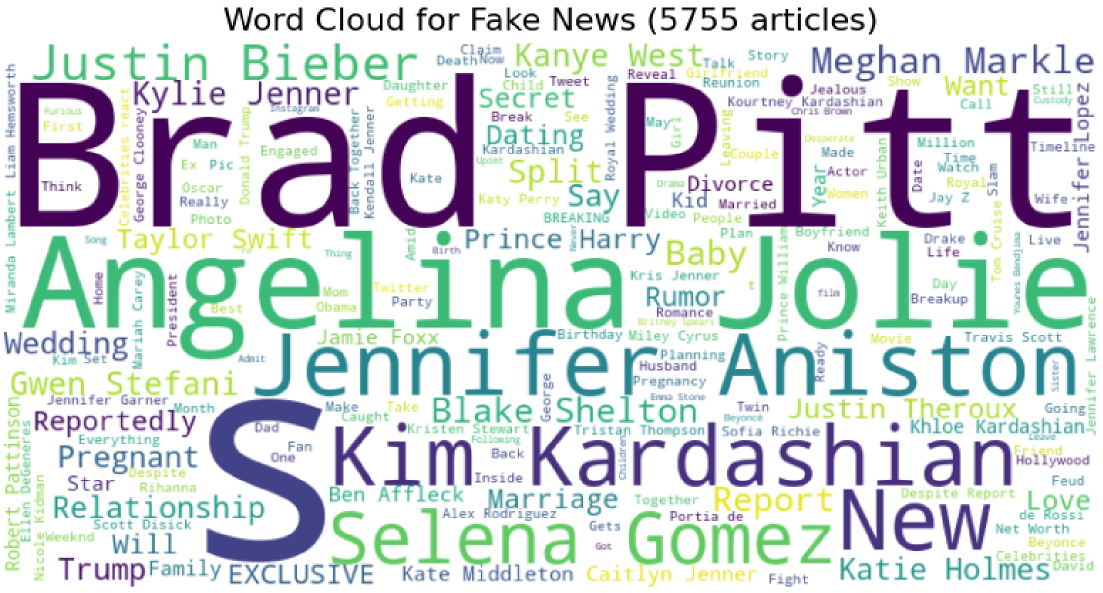
As seen from above, the data comprises multiple columns: the article's title, news_url, source_domain, tweet_num and real. Throughout this project, I will be using and relying on the **title**, **source_domain** and **real** columns to explore and preprocess the data and to train the models.

Now I'll do some exploratory data analysis to gain a better understanding of the data. For example, I will find out the **number of real and fake articles and generate their word clouds**

[6]:
```python
# Count the number of true articles (where 'real' column is 1)
true_articles = news_df[news_df['real'] == 1].shape[0]

# Count the number of fake articles (where 'real' column is 0)
fake_articles = news_df[news_df['real'] == 0].shape[0]

# Split the dataset into titles of true news articles (where 'real' is 1)
true_news = news_df[news_df['real'] == 1]['title']

# Split the dataset into titles of fake news articles (where 'real' is 0)
```

```python
fake_news = news_df[news_df['real'] == 0]['title']

# Combine all titles of true news into a single string, separated by a space
true_text = " ".join(true_news)

# Combine all titles of fake news into a single string, separated by a space
fake_text = " ".join(fake_news)

# Function to generate a word cloud from a given text & display it with a title
def generate_word_cloud(text, title, count):
    # Generate a word cloud with specific configurations (size, background␣
↪color, color map, max words)
    wordcloud = WordCloud(
        width=800, height=400,
        background_color='white', # background color of the word cloud
        colormap='viridis',
        max_words=200 # Limit the word cloud to show a maximum of 200 words
    ).generate(text) # Generate the word cloud using the combined text (either␣
↪true or fake news titles)

    # Create a plot for displaying the word cloud
    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation='bilinear') # Display the word cloud␣
↪image, using bilinear interpolation for smoothing
    plt.axis('off') # Turn off the axes to make the word cloud image clean
    plt.title(f"{title} ({count} articles)", fontsize=16)
    plt.show()

# Generate the word cloud for true & fake news articles, passing the text,␣
↪title & the count of true articles
generate_word_cloud(true_text, "Word Cloud for True News", true_articles)
generate_word_cloud(fake_text, "Word Cloud for Fake News", fake_articles)
```





The above image compares word clouds for true news articles and fake news articles, highlighting the differences in their content.

For true news articles, prominent words include “Stars,” “News,” “Show,” “See,” “Season,” “Watch,” “Reveal,” “Say,” “Red Carpet,” “Photo,” “Kim Kardashian,” “Meghan Markle,” and “Prince Harry.” These words suggest that true news articles often focus on celebrities, events, and visual media.

In contrast, fake news articles feature prominent words such as “Jennifer Aniston,” “Brad Pitt,” “Angelina Jolie,” “New,” “Split,” “Report,” “Justin Bieber,” “Kim Kardashian,” “Selena Gomez,” “Meghan Markle,” “Kylie Jenner,” and “Blake Shelton.” This indicates that fake news articles frequently mention celebrity names and sensational topics like relationships and rumors.

The analysis reveals that true news emphasizes actual events, shows, and notable public figures, presenting them in a more informative manner. On the other hand, fake news focuses on sensationalism and gossip, often featuring famous personalities and dramatic topics. This comparison highlights how true and fake news articles differ in their approach, particularly in the context of celebrity news, with true news tending to be more factual while fake news leans towards sensational content.

Also, **number of true news is more than fake news** which could indicates **imbalance**. When dataset is imbalanced, such as having more true news articles than fake news articles, it can lead to **biased** model performance. The model may become biased towards the **majority class** (in this case, true news), which can result in poor generalization and accuracy when predicting the minority class (true news). So keeping this in mind, I will check and confirm if the dataset is indeed imbalanced or it is just a **subtle difference**.

[7]:
```
# If the difference between the counts of true and fake articles is less than␣
↪10% of the larger count:
if abs(true_articles - fake_articles) < 0.1 * max(true_articles, fake_articles):
    print('The dataset is balanced.')
else:  # If the difference is greater than 10% of the larger count (i.e., the␣
↪dataset is heavily skewed):
    print('The dataset is imbalanced.')
```

```
The dataset is imbalanced.
```

As seen above, this difference in number of true and fake news is **major** and the dataset is imbalanced. Therefore I need to balance it for the models, which I will do it later in **7. Comparative Classification approach**.

1.1.13 5.2. Preprocessing

Before training, I will make sure there are no missing values and then apply several preprocessing steps to prepare the textual data:

- **Tokenization**: The title text is split into individual words or tokens. This allows us to analyze the frequency and distribution of words, which is important for both statistical and deep learning models.
- **Stopword Removal**: Common, meaningless words (e.g., “the,” “is”) are removed to reduce noise and focus on more meaningful words.
- **Lemmatization**: Words are reduced to their root form (e.g., “running” becomes “run”), ensuring that different word forms are treated as the same entity, which improves model accuracy.

These preprocessing steps prepare the data for analysis by cleaning the text and applying different text representations.

[8]:
```
# Drop missing values
news_df = news_df.dropna()
print(news_df.isnull().sum())  # Confirm no missing values remain
```

```
title            0
news_url         0
source_domain    0
tweet_num        0
real             0
dtype: int64
```

[9]:
```
# Load a pre-trained spaCy model for NLP, disabling parser & NER for efficiency
nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])

# Optimized text preprocessing function
def preprocess_text(text):
    text = re.sub(r'\d+', '', text) # Remove all numbers from the text using␣
 ↪regex

    text = re.sub(' +', ' ', text).strip() # Replace multiple spaces with a␣
 ↪single space & strip leading/trailing spaces

    doc = nlp(text.lower()) # Process the text using spaCy & convert it to␣
 ↪lowercase

    # Return the lemmatized tokens that are not stopwords or punctuation
    return " ".join([token.lemma_ for token in doc if not token.is_stop and not␣
 ↪token.is_punct])

# Optimized data preparation function
def prepare_data(data):
    # Apply the preprocess_text function to the title column
    data['title'] = data['title'].apply(preprocess_text)

    # Filter out rows where the title column is empty after preprocessing
    data = data[data['title'].str.strip() != '']

    # Fill missing or empty source_domain values with a placeholder
    data['source_domain'] = data['source_domain'].fillna('unknown').str.strip()

    # Create the new column title_with_domain, combining title and source_domain
    data['title_with_domain'] = data.apply(lambda row: f"{row['title']}␣
 ↪{row['source_domain']}".strip(), axis=1)

    # Filter out rows where the combined title_with_domain is empty
    data = data[data['title_with_domain'].str.strip() != '']
```

```python
    return data

# After preprocessing
processed_df = prepare_data(news_df)

print("Preprocessed Titles:")
print(processed_df['title_with_domain'].head())
```

```
C:\Users\karat\AppData\Local\Temp\ipykernel_9360\3158277440.py:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  data['source_domain'] = data['source_domain'].fillna('unknown').str.strip()
```

```
Preprocessed Titles:
0                    kandi burruss explode rape accusation real housewife atlanta reunion video toofab.com
1                                                 people choice award good red carpet look www.today.com
2    sophia bush send sweet birthday message tree hill co star hilarie burton breyton eva www.etonline.com
3                 colombian singer maluma spark rumour inappropriate relationship aunt www.dailymail.co.uk
4           gossip girl year later upper east sider shock world change pop culture forever www.zerchoo.com
Name: title_with_domain, dtype: object
```

```
C:\Users\karat\AppData\Local\Temp\ipykernel_9360\3158277440.py:27: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  data['title_with_domain'] = data.apply(lambda row: f"{row['title']} {row['source_domain']}".strip(), axis=1)
```

The above code processes a DataFrame of news articles by first cleaning & normalizing the text in the **title** column. It removes numbers, excess whitespace & converts the text to lowercase. Using spaCy, it tokenizes & lemmatizes the text while filtering out stopwords & punctuation. The **prepare_data** function applies this preprocessing to the titles, removes rows with empty titles & fills missing values in the **source_domain** column with **unknown**. It then creates a new column, **title_with_domain**, by combining the cleaned titles with their corresponding source domains & filters out rows where this combined column is empty. The result is a cleaned, ready-to-use DataFrame for further analysis.

Combining the cleaned **title** with the **source_domain** adds valuable context, because some titles may be ambiguous or identical across different sources, and the source domain helps differentiate them. The combined feature provides richer information, allowing machine learning models to better distinguish between articles from different sources and learn associations between the content and its origin. This improves the model's ability to make more accurate predictions by capturing both the topic and the source-specific nuances.

Differences in data preparation for statistical models versus embedding models

The data preprocessing steps I've implemented are suitable for both Logistic Regression and Bidirectional LSTM models. However, there are some differences in how the preprocessed data is used for each model and I will implement these differences in **7. Comparative Classification approach**. Below are the difference addressed.

- **Feature Representation**: Logistic Regression uses TF-IDF vectorization to create a sparse matrix, while the BiLSTM model uses padded sequences of integers converted to dense vectors through embeddings.
- **Input Format**: Logistic Regression requires a fixed-size feature vector, whereas BiLSTM requires sequences of varying lengths that are padded to a uniform size.
- **Class Balancing**: Both models address class imbalance, but Logistic Regression uses oversampling, while BiLSTM uses class weights during training.
- **Complexity**: The preprocessing for BiLSTM is more complex due to the need for sequence handling and embedding representation, while Logistic Regression focuses on simpler text feature extraction.

1.1.14 6. Baseline Performance

To establish a baseline for the fake news detection task, I have chosen to implement a Logistic Regression model. Logistic Regression is a widely used statistical method for binary classification tasks, particularly in text classification scenarios. It is favored for its simplicity, interpretability, and effectiveness, making it a suitable choice for this project. Justification for Choosing Logistic Regression

1. **Efficiency**: Logistic Regression is quick to train, even with large datasets, compared to more complex models like deep learning. It is scalable & computationally efficient [12] [13]
2. **Simplicity and Interpretability**: Logistic Regression is easy to use and understand. It shows which features (words or phrases) influence the prediction of fake or real news. [13]
3. **Handling Imbalanced Data**: It can be adjusted to manage imbalanced datasets using techniques like class weighting or resampling.

Implementation of Logistic Regression:

1. **Data Preparation & Preprocessing**: The dataset is preprocessed to create TF-IDF features from the titles (combined with domain) of the articles. This representation captures the importance of words in the context of the entire dataset.

2. **Train-Test Split**: The dataset is split into training & testing sets, ensuring that the distribution of real & fake articles is preserved.

3. **Class Balancing**: To address the class imbalance, I will use RandomOverSampler to oversample the minority class (fake news) in the training set.

4. **Model Training**: The Logistic Regression model is trained on the TF-IDF features of the training set.
5. **Evaluation**: The model's performance is evaluated using metrics such as accuracy, precision, recall & F1-score, providing a comprehensive view of its effectiveness in classifying fake & real news articles.

The results from the Logistic Regression model will serve as a baseline against which the performance of the more complex deep learning model (Bidirectional LSTM) will be compared.

1.1.15 7. Comparative Classification Approach

In this section, I have implemented both a traditional statistical model (Logistic Regression) & a modern deep learning model (Bidirectional Long Short-Term Memory, BiLSTM) to compare their performance in the context of fake news detection.

7.1 Logistic Regression Model

1. Architecture: The Logistic Regression model is a linear model that predicts the probability of a binary outcome based on one or more predictor variables. [13] In this case, the predictor variables are the TF-IDF features derived from the article titles combined with domains.

2. Training Process:

- **Data Preparation & Preprocessing**: The dataset is preprocessed to create TF-IDF features from the titles (combined with domain) of the articles. This representation captures the importance of words in the context of the entire dataset.
- **Train-Test Split**: The dataset is split into training and testing sets, ensuring that the distribution of real and fake articles is preserved.
- **Class Balancing**: To address the class imbalance, RandomOverSampler is used to oversample the minority class (fake news) in the training set.
- **Model Training**: The Logistic Regression model is trained on the TF-IDF features of the training set.

3. Optimization

- **Hyperparameter Tuning**: Hyperparameter tuning is performed using RandomizedSearchCV to optimize the regularization strength (C) and the solver. This process involves searching through a predefined set of hyperparameters to find the combination that yields the best performance on the validation set.
- **Evaluation**: The model's performance is evaluated using metrics such as accuracy, precision, recall, and F1-score, providing a comprehensive view of its effectiveness in classifying fake and real news articles.

4. Strengths & Weaknesses

Strengths:

- Fast training and inference times. [13]

- Easy to interpret results & understand feature importance, which is crucial for stakeholders. [13]

Weaknesses: May struggle with complex relationships in the data, as it assumes a linear relationship between features & the log-odds of the outcome.

7.2 Bidirectional LSTM Model

1. **Architecture [14]**: The BiLSTM model consists of:
   - An embedding layer that converts input sequences into dense vectors.
   - A Bidirectional LSTM layer that processes the input sequences in both forward & backward directions, capturing contextual information from both ends.
   - A dropout layer for regularization to prevent overfitting.
   - A dense output layer with a sigmoid activation function for binary classification.
2. **Training Process**:
   - **Data Preparation & Preprocessing**: The model is trained on padded sequences of the article titles, using GloVe embeddings for word representation. The sequences are padded to ensure uniform input lengths.
   - **Class Balancing**: Class weights are computed to address class imbalance during training, ensuring that the model pays more attention to the minority class (fake news).
   - **Model Training**: The model is trained using the training dataset, with early stopping and learning rate reduction callbacks to optimize training.
3. **Optimization Process**:
   - **Early Stopping**: This technique monitors the validation loss during training and stops the training process if the loss does not improve for a specified number of epochs (patience). This helps prevent overfitting and saves computational resources.
   - **Learning Rate Reduction**: The learning rate is reduced when the validation loss plateaus, allowing the model to converge more effectively. This is particularly useful in deep learning, where a high learning rate can lead to overshooting the optimal solution.
   - **Model Checkpointing**: The best model weights are saved during training based on validation performance, ensuring that the best-performing model is retained for evaluation.
4. **Strengths & Weaknesses**

Strengths:

- Capable of capturing complex patterns and dependencies in the data due to its recurrent architecture.
- Bidirectional processing allows the model to consider context from both directions, improving understanding of the text.

Weaknesses:

- Requires more computational resources & time to train compared to Logistic Regression [14].
- May be prone to overfitting, especially with limited data.

Preprocessing Comparison

Different data processing techniques are applied for the Logistic Regression & BiLSTM models, reflecting their distinct requirements & capabilities.

For the Logistic Regression model:

- **TF-IDF Vectorization:** The processed titles are transformed into a TF-IDF (Term Frequency-Inverse Document Frequency) representation. This technique captures the importance of words in the context of the entire dataset, converting the text data into a sparse matrix format suitable for Logistic Regression.
- **Train-Test Split:** The dataset is split into training and testing sets, ensuring that the distribution of real and fake articles is preserved. Stratified sampling is used to maintain the class distribution.
- **Class Balancing:** To address class imbalance, RandomOverSampler is applied to oversample the minority class (fake news) in the training set.
- **Feature Scaling:** The TF-IDF features are scaled using MaxAbsScaler to ensure that the model performs optimally, especially since Logistic Regression is sensitive to the scale of input features.

For the BiLSTM model

- **Padding Sequences:** Since LSTM models require input sequences to be of uniform length, the tokenized sequences are padded to a maximum length (e.g., 300). This ensures that all input sequences have the same shape, which is necessary for batch processing in deep learning.
- **Class Balancing:** Class weights are computed to address class imbalance during training. This ensures that the model pays more attention to the minority class (fake news) during the learning process.
- **Embedding Layer:** Pre-trained GloVe (Global Vectors for Word Representation) embeddings are used to convert the integer sequences into dense vector representations. This captures semantic relationships and contextual information, allowing the model to leverage the sequential nature of the data.
- **Train-Test Split:** The dataset is split into training, validation, and test sets, ensuring that the distribution of classes is maintained across all sets.

Performance Comparison

After training both models, their performance were evaluated using the same metrics: accuracy, precision, recall, and F1-score. The results are presented in a comparative format, highlighting the strengths and weaknesses of each approach in the context of fake news detection.

1.1.16 Logistic Regression

[10]:
```
# Initialize TF-IDF vectorizer (preprocessing: TF-IDF Vectorization)
tfidf = TfidfVectorizer(max_features=1000)  # Limit to 1000 features for better␣
 ↪performance

# Convert the 'title_with_domain' column from the DataFrame into a sparse␣
 ↪matrix of TF-IDF features
title_vectors = tfidf.fit_transform(processed_df['title_with_domain'])
```

```
# Convert the sparse matrix of TF-IDF features into a dense array for easier␣
↪manipulation
title_vectors_array = title_vectors.toarray()

# print the first few vectors
print(title_vectors_array[:5])
```

```
[[0.         0.         0.         … 0.         0.         0.        ]
 [0.         0.         0.         … 0.         0.         0.        ]
 [0.         0.         0.         … 0.         0.         0.        ]
 [0.         0.         0.         … 0.         0.         0.        ]
 [0.         0.         0.         … 0.         0.         0.37460543]]
```

[11]:
```
# Split the data into training and test sets (Preprocessing: Data Splitting)
X_train, X_test, y_train, y_test = train_test_split(
    title_vectors, # The feature vectors (e.g., TF-IDF of titles) as input␣
↪features for the model
    processed_df['real'], # The target variable (real/fake labels) as the␣
↪output labels for the model
    test_size=0.2, # 20% of the data will be used for testing, the remaining␣
↪80% for training
    # Stratified split ensures that the distribution of real/fake labels is␣
↪preserved in both training & testing sets
    stratify=processed_df['real'],
    random_state=42 # Set a random seed for reproducibility
)

# Random state in oversampler
ros = RandomOverSampler(random_state=42)
X_train_resampled, y_train_resampled = ros.fit_resample(X_train, y_train)

# Random state in TF-IDF Vectorizer
tfidf = TfidfVectorizer(max_features=1000, stop_words='english', lowercase=True)

# Verify & display the resampled class distribution
print("Training class distribution after resampling:")
print(y_train_resampled.value_counts())

# Check & display the class distribution in the test set (without resampling)
print("Test class distribution (without resampling):")
print(y_test.value_counts())
```

```
Training class distribution after resampling:
1    13894
0    13894
Name: real, dtype: int64
```

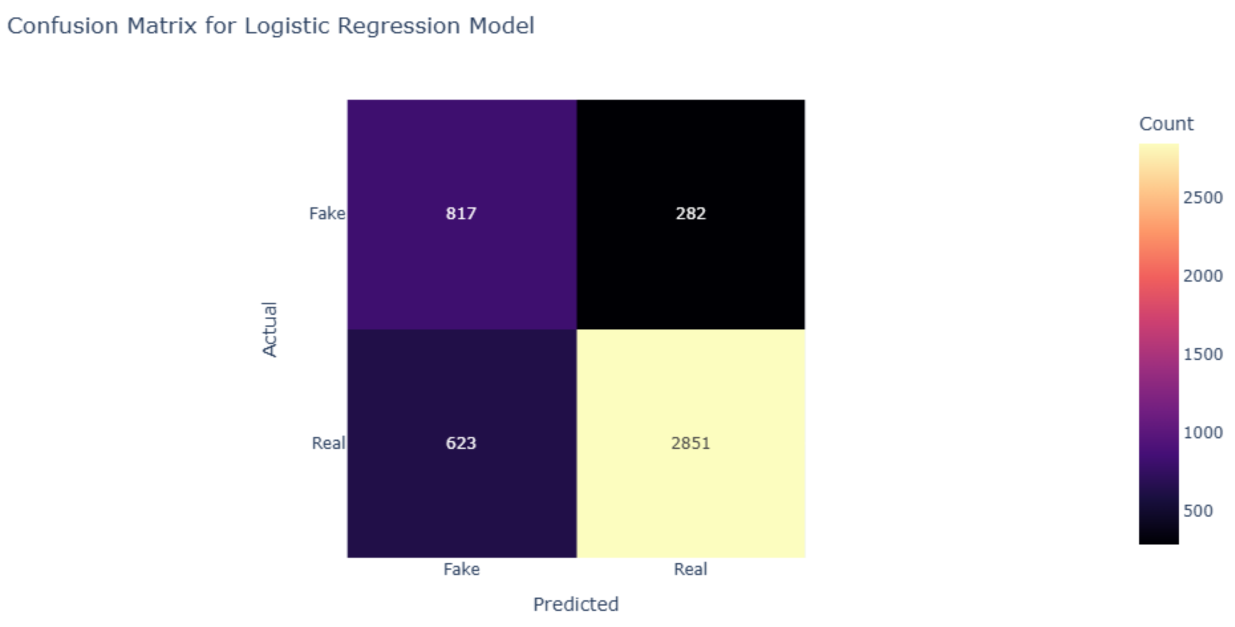
```
Test class distribution (without resampling):
1    3474
0    1099
Name: real, dtype: int64
```

```
C:\Users\karat\anaconda3\Lib\site-packages\sklearn\utils\deprecation.py:151: FutureWarning: 'force_all_finite' was renamed to
'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
C:\Users\karat\anaconda3\Lib\site-packages\sklearn\base.py:484: FutureWarning: `BaseEstimator._check_n_features` is deprecated
in 1.6 and will be removed in 1.7. Use `sklearn.utils.validation._check_n_features` instead.
  warnings.warn(
C:\Users\karat\anaconda3\Lib\site-packages\sklearn\base.py:493: FutureWarning: `BaseEstimator._check_feature_names` is deprecat
ed in 1.6 and will be removed in 1.7. Use `sklearn.utils.validation._check_feature_names` instead.
  warnings.warn(
```

[12]:
```
# (Preprocessing: Feature Scaling using MaxAbsScaler)
# Scale the features (TF-IDF values) using MaxAbsScaler for sparse matrices,␣
↪computationally
# efficient, especially with large datasets
scaler = MaxAbsScaler() # Initialize the scaler
# Apply scaling to the resampled training data (scales each feature to [0, 1]␣
↪without changing sparsity)
X_train_scaled = scaler.fit_transform(X_train_resampled)
X_test_scaled = scaler.transform(X_test) # Apply the same scaling to the test␣
↪data

# Define hyperparameter search space for Logistic Regression (Preprocessing:␣
↪Hyperparameter Tuning for Logistic Regression)
param_dist = {
    'C': np.logspace(-2, 2, 5), # Regularization strength (range: 0.01 to 100)
    'penalty': ['l2'], # L2 penalty (used for regularization)
    'solver': ['lbfgs', 'liblinear'] # Solvers to consider (both efficient for␣
↪large datasets)
}

# Initialize & perform RandomizedSearchCV with LogisticRegression, makes it␣
↪much faster to find a good model
random_search = RandomizedSearchCV(
    estimator=LogisticRegression(max_iter=1000), # Logistic Regression with␣
↪1000 iterations to ensure faster training.
    param_distributions=param_dist, # Hyperparameter grid
    cv=3, # Cross-validation with 3 folds
    scoring='accuracy', # Use accuracy as the metric for evaluation
    n_iter=3, # Number of random combinations to try
    n_jobs=1, # Disable parallelism to avoid issues with multiprocessing in␣
 ↪some environments
    random_state=42 # For reproducibility
)

# Fit the model on the resampled & scaled training data
random_search.fit(X_train_scaled, y_train_resampled) # Perform the randomized␣
 ↪search to find the best hyperparameters

# Retrieve the best logistic regression model based on the randomized search
best_lr_model = random_search.best_estimator_

# Predict & evaluate labels for the test data using the best model
y_pred_lr = best_lr_model.predict(X_test_scaled)

# Classification report
print("Logistic Regression Results")
print(classification_report(y_test, y_pred_lr))

# confusion_matrix function computes the confusion matrix for the actual labels␣
 ↪(y_test)
# & the predicted labels (y_pred_lr), which helps us evaluate the model's␣
 ↪performance
conf_matrix_lr = confusion_matrix(y_test, y_pred_lr)

# Create a DataFrame for the confusion matrix
conf_matrix_df = pd.DataFrame(conf_matrix_lr, index=['Fake', 'Real'],␣
 ↪columns=['Fake', 'Real'])

# Plotting the confusion matrix using Plotly
fig = px.imshow(conf_matrix_df,
                text_auto=True, # Automatically display the numerical values in␣
 ↪the matrix cells.
                color_continuous_scale='Magma',
                labels=dict(x="Predicted", y="Actual", color="Count"),
                title="Confusion Matrix for Logistic Regression Model")

# Show the plot
fig.show()
```

```
Logistic Regression Results
              precision    recall  f1-score   support

           0       0.57      0.74      0.64      1099
           1       0.91      0.82      0.86      3474

    accuracy                           0.80      4573
```

```
   macro avg       0.74      0.78      0.75      4573
weighted avg       0.83      0.80      0.81      4573
```



1.1.17 Deep Learning Model

[13]:
```python
# Constants
MAX_VOCAB_SIZE = 10000 # max. number of words to be used by the tokenizer.

# max. length of each text sequence. Texts longer than this will be truncated &␣
↪shorter ones will be padded to match this length
MAX_SEQUENCE_LENGTH = 300

EMBEDDING_DIM = 100 # dimension of the word embeddings (i.e., how many numbers␣
↪to represent each word)
RANDOM_STATE = 42  # Consistent random state for reproducibility

# Tokenizer: Convert text (title_with_domain) to sequences of integers␣
↪(Preprocessing: Tokenization)
tokenizer = Tokenizer(num_words=MAX_VOCAB_SIZE)
```

```
# Fit the tokenizer on the 'title_with_domain' column in the processed_df␣
↪DataFrame
tokenizer.fit_on_texts(processed_df['title_with_domain'])

# Convert each text in (title_with_domain) to  a sequence of integers & pad␣
↪them (Preprocessing: Sequence Conversion)
X_seq = tokenizer.texts_to_sequences(processed_df['title_with_domain'])

# Pad the sequences to a uniform length of `MAX_SEQUENCE_LENGTH` (300 in this␣
↪case) (Preprocessing: Padding Sequences)
X_pad = pad_sequences(X_seq, maxlen=MAX_SEQUENCE_LENGTH, padding='post')

# View the first few tokenized sequences
print("Tokenized Sequences (first 5):")
for i in range(5):
    print(f"Sequence {i+1}: {X_seq[i]}")

# View the first few padded sequences
print("Padded Sequences (first 5):")
print(X_pad[:5])
```

```
Tokenized Sequences (first 5):
Sequence 1: [4064, 4065, 2362, 791, 1565, 94, 258, 842, 209, 43, 676, 1]
Sequence 2: [3, 294, 26, 36, 108, 135, 37, 2, 87, 1]
Sequence 3: [1520, 996, 729, 382, 83, 409, 2363, 570, 4, 9, 4066, 4067, 9335,
677, 2, 22, 1]
Sequence 4: [5915, 426, 2986, 714, 1099, 2634, 81, 2987, 2, 10, 4, 5]
Sequence 5: [799, 109, 23, 1620, 1805, 1889, 9336, 800, 243, 334, 934, 2485,
1521, 2, 616, 1]
Padded Sequences (first 5):
[[4064 4065 2362 …    0    0    0]
 [   3  294   26 …    0    0    0]
 [1520  996  729 …    0    0    0]
 [5915  426 2986 …    0    0    0]
 [ 799  109   23 …    0    0    0]]
```

[14]:
```
# Extract labels (real/fake) from the real column
y = processed_df['real'].values

# Split the padded input sequences (X_pad) & the labels (y) into training &␣
↪temporary datasets
X_train_pad, X_temp_pad, y_train, y_temp = train_test_split(X_pad, y,␣
↪test_size=0.3, random_state=42)

# Split the remaining 30% of the data (X_temp_pad, y_temp) into validation &␣
↪test sets
```

```
X_val_pad, X_test_pad, y_val, y_test = train_test_split(X_temp_pad, y_temp,␣
↪test_size=0.5, random_state=42)

# Calculate class weights using compute_class_weight from scikit-learn␣
↪(Preprocessing: Calculate class weights)
class_weights = compute_class_weight('balanced', classes=np.unique(y_train),␣
↪y=y_train)

# Convert the calculated class weights into a dictionary where each class index␣
↪is mapped to its corresponding weight
class_weight_dict = dict(enumerate(class_weights))

# Verify the distribution in each dataset
print("\nTraining set distribution:")
print(pd.Series(y_train).value_counts())

# Print the distribution of classes in the validation set (y_val) for a similar␣
↪verification
print("\nValidation set distribution:")
print(pd.Series(y_val).value_counts())

# Print the distribution of classes in the test set (y_test) to check the␣
↪balance or imbalance in the test data
print("\nTest set distribution:")
print(pd.Series(y_test).value_counts())

# Print the class weights calculated earlier to ensure the model will account␣
↪for class imbalance during training
print("\nClass Weights:")
print(class_weight_dict)
```

```
Training set distribution:
1    12133
0     3870
dtype: int64

Validation set distribution:
1    2612
0     817
dtype: int64

Test set distribution:
1    2623
0     807
dtype: int64
```

```
Class Weights:
{0: 2.0675710594315246, 1: 0.6594824033627298}
```

[15]:
```
# GloVe embeddings, which are pre-trained word vectors (For Faster training)
# Initialize an empty dictionary embedding_index where each word will be mapped␣
↪to its corresponding embedding vector
embedding_index = {}

# Open the GloVe file ('glove.6B.100d.txt') in read mode with UTF-8 encoding
with open('glove.6B.100d.txt', 'r', encoding='utf-8') as f:
    # Iterate through each line in the file. Each line contains a word followed␣
↪by its embedding vector
    for line in f:
        # Split the line into a list of values (the first element is the word,␣
↪& the remaining are the embedding values)
        values = line.split()
        word = values[0] # The first element of values is the word (e.g., king,␣
↪queen)

        # Convert the embedding values (the rest of the values list) into a␣
↪NumPy array of type float32
        coefs = np.asarray(values[1:], dtype='float32')

        # Add the word & its corresponding embedding vector to the␣
↪embedding_index dictionary
        embedding_index[word] = coefs

print("GloVe embeddings loaded successfully!")

# Initialize a matrix embedding_matrix of zeros with dimensions␣
↪(MAX_VOCAB_SIZE, EMBEDDING_DIM)
embedding_matrix = np.zeros((MAX_VOCAB_SIZE, EMBEDDING_DIM))

# Iterate over each word and its index in the tokenizer's word index
for word, i in tokenizer.word_index.items():
    # Only consider words that have an index less than `MAX_VOCAB_SIZE` (to␣
↪avoid indexing beyond the maximum vocab size)
    if i < MAX_VOCAB_SIZE:
        # Look up the word in the `embedding_index` dictionary to get its␣
↪corresponding GloVe embedding vector
        embedding_vector = embedding_index.get(word)

        # If the word exists in the GloVe embeddings (embedding_vector is not␣
↪None),
        if embedding_vector is not None:
            #assign its embedding vector to the corresponding row in the␣
↪embedding matrix
```

```
            embedding_matrix[i] = embedding_vector

print("Embedding matrix created successfully!")
```

```
GloVe embeddings loaded successfully!
Embedding matrix created successfully!
```

I chose GloVe embeddings to start simple, establish a baseline & for faster training, HOWEVER;

When using a **Tokenizer**, some words in my dataset may not exist in the GloVe embeddings (embedding_index). These words are represented as zero vectors in the **embedding matrix**. If a significant portion of the dataset's vocabulary is not found in GloVe, this could degrade model performance. So therefore before training the model, I will log in the percentage of OOV words to see if OOV words are too high and if it's high then I will consider consider using a different pre-trained embedding with a broader vocabulary.

```
[16]: # Get the total number of unique words in the tokenizer's vocabulary
vocab_size = len(tokenizer.word_index)
# Count the number of out-of-vocabulary (OOV) words by checking if the word's␣
 ↪embedding is missing in embedding_index
oov_count = sum(1 for word, i in tokenizer.word_index.items() if i <␣
 ↪MAX_VOCAB_SIZE and embedding_index.get(word) is None)

print(f"Percentage of OOV words: {(oov_count / vocab_size) * 100:.2f}%")
```

```
Percentage of OOV words: 7.20%
```

A 7.20% percentage of Out-Of-Vocabulary (OOV) words is relatively **low** and generally **acceptable**, so I may not necessarily need to switch to a different pre-trained embedding. However, whether this percentage is acceptable **depends** on the following considerations such as whether the dataset contains **specialized** or **domain-specific** language, **frequency** of OOV words, etc. So next I will examine the OOV words to see if they are meaningful or domain-specific.

```
[17]: # Create a list of out-of-vocabulary (OOV) words by checking if their␣
 ↪embeddings are missing in the embedding_index
oov_words = [word for word, i in tokenizer.word_index.items() if i <␣
 ↪MAX_VOCAB_SIZE and embedding_index.get(word) is None]
print(oov_words[:10])  # Print the first 10 OOV wordas
```

```
['dailymail', 'usmagazine', 'etonline', 'longroom', 'hollywoodlife',
'hollywoodreporter', 'thewrap', 'harpersbazaar', 'disick', 'eonline']
```

Many of the OOV words in the list (dailymail, usmagazine, etonline, etc.) appear to be names of **news or entertainment outlets**. In a **fake vs. real news context**, the source of the article can play a significant role, like certain sources such as **dailymail** might have a reputation for publishing accurate information (real news) & others such as **hollywoodlife** might be known for sensationalism, misinformation, or fake news. So therefore, I can proceed with GloVe embeddings in training the model.

[18]:
```
# Define the model
model = Sequential([

    # Embedding layer with pre-trained weights for faster convergence
    Embedding(input_dim=MAX_VOCAB_SIZE, output_dim=EMBEDDING_DIM,␣
 ↪weights=[embedding_matrix], trainable=True),

    # Bidirectional LSTM for capturing context in both directions
    Bidirectional(LSTM(128, return_sequences=False)), # Faster than returning␣
 ↪sequences

    # Dropout for regularization to avoid overfitting
    Dropout(0.5),

    # Fully connected layer with 128 units, ReLU activation & L2 regularization␣
 ↪for weight penalty
    Dense(128, activation='relu', kernel_regularizer=l2(0.01)),

    Dropout(0.5), # Additional Dropout Layer

    # BatchNormalization to stabilize training and speed up convergence
    BatchNormalization(),

    # Output layer for binary classification
    Dense(1, activation='sigmoid')
])

# Compile the model with Adam optimizer for efficient training
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

# Callbacks to improve training speed
# Stop early if no improvement
early_stopping = EarlyStopping(monitor='val_loss', patience=5,␣
 ↪restore_best_weights=True, verbose=1)
# Reduce learning rate if loss plateaus
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3,␣
 ↪min_lr=1e-6, verbose=1)
# Save the best model
model_checkpoint = ModelCheckpoint('best_model_fast.keras',␣
 ↪save_best_only=True, monitor='val_loss', mode='min', verbose=1)

# Train the model with efficient batch & and early stopping
history = model.fit(
    X_train_pad, y_train,
    batch_size=64, epochs=50, # Batch size of 64 for faster training
    validation_data=(X_val_pad, y_val),
    class_weight=class_weight_dict, # Account for class imbalance
    callbacks=[early_stopping, reduce_lr, model_checkpoint], # Callbacks to␣
 ↪improve speed
    verbose=1
)

# Evaluate the model on the test data
y_pred_dl = (model.predict(X_test_pad) > 0.5).astype(int).flatten()
```

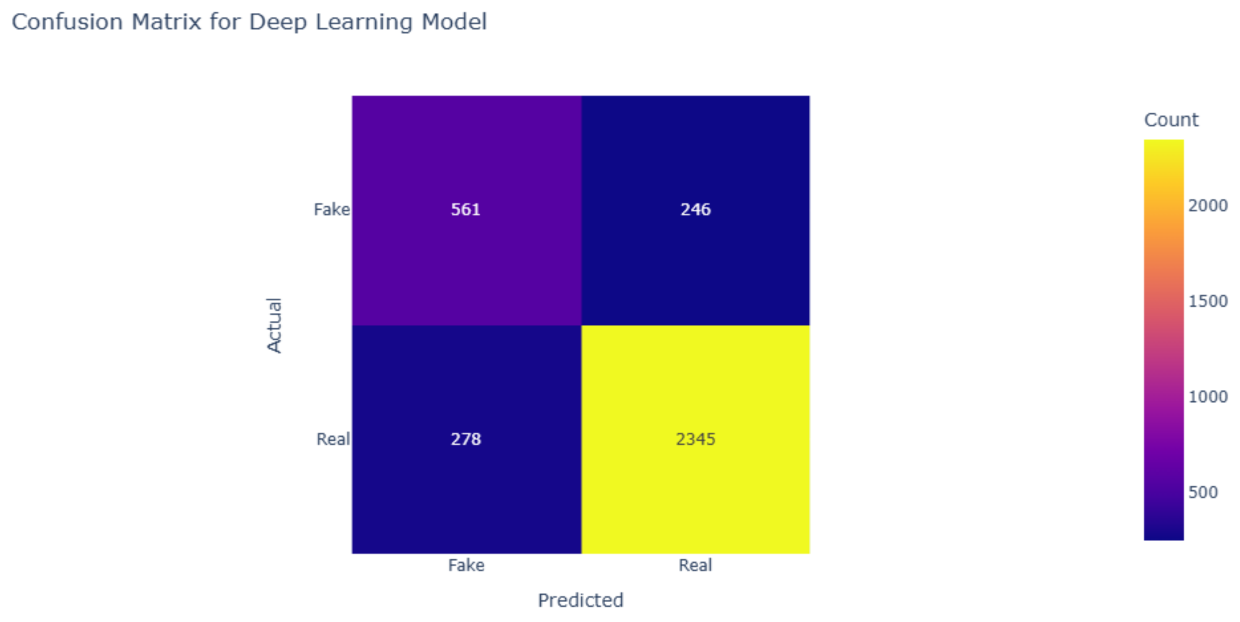
```
Epoch 1/50
251/251              0s 1s/step -
accuracy: 0.5971 - loss: 1.7257
Epoch 1: val_loss improved from inf to 0.75336, saving model to
best_model_fast.keras
251/251              348s 1s/step -
accuracy: 0.5974 - loss: 1.7240 - val_accuracy: 0.8344 - val_loss: 0.7534 -
learning_rate: 0.0010
Epoch 2/50
251/251              0s 1s/step -
accuracy: 0.7842 - loss: 0.7085
Epoch 2: val_loss improved from 0.75336 to 0.48542, saving model to
best_model_fast.keras
251/251              358s 1s/step -
accuracy: 0.7842 - loss: 0.7082 - val_accuracy: 0.8384 - val_loss: 0.4854 -
learning_rate: 0.0010
Epoch 3/50
251/251              0s 1s/step -
accuracy: 0.8474 - loss: 0.4452
Epoch 3: val_loss did not improve from 0.48542
251/251              388s 2s/step -
accuracy: 0.8474 - loss: 0.4452 - val_accuracy: 0.7892 - val_loss: 0.5022 -
learning_rate: 0.0010
Epoch 4/50
251/251              0s 1s/step -
accuracy: 0.8783 - loss: 0.3377
Epoch 4: val_loss improved from 0.48542 to 0.38244, saving model to
best_model_fast.keras
251/251              377s 2s/step -
accuracy: 0.8783 - loss: 0.3377 - val_accuracy: 0.8492 - val_loss: 0.3824 -
learning_rate: 0.0010
Epoch 5/50
251/251              0s 1s/step -
accuracy: 0.9094 - loss: 0.2519
Epoch 5: val_loss did not improve from 0.38244
251/251              377s 2s/step -
accuracy: 0.9093 - loss: 0.2520 - val_accuracy: 0.8288 - val_loss: 0.4211 -
```

```
learning_rate: 0.0010
Epoch 6/50
251/251              0s 918ms/step -
accuracy: 0.9160 - loss: 0.2201
Epoch 6: val_loss did not improve from 0.38244
251/251              243s 967ms/step -
accuracy: 0.9160 - loss: 0.2201 - val_accuracy: 0.8160 - val_loss: 0.4644 -
learning_rate: 0.0010
Epoch 7/50
251/251              0s 911ms/step -
accuracy: 0.9347 - loss: 0.1773
Epoch 7: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.

Epoch 7: val_loss did not improve from 0.38244
251/251              241s 958ms/step -
accuracy: 0.9347 - loss: 0.1774 - val_accuracy: 0.8049 - val_loss: 0.5372 -
learning_rate: 0.0010
Epoch 8/50
251/251              0s 969ms/step -
accuracy: 0.9460 - loss: 0.1374
Epoch 8: val_loss did not improve from 0.38244
251/251              256s 1s/step -
accuracy: 0.9460 - loss: 0.1374 - val_accuracy: 0.8381 - val_loss: 0.4975 -
learning_rate: 2.0000e-04
Epoch 9/50
251/251              0s 955ms/step -
accuracy: 0.9599 - loss: 0.1173
Epoch 9: val_loss did not improve from 0.38244
251/251              251s 1s/step -
accuracy: 0.9599 - loss: 0.1173 - val_accuracy: 0.8253 - val_loss: 0.5487 -
learning_rate: 2.0000e-04
Epoch 9: early stopping
Restoring model weights from the end of the best epoch: 4.
108/108              15s 132ms/step
```

[19]:
```
# Compute the confusion matrix
conf_matrix_dl = confusion_matrix(y_test, y_pred_dl)

# Print the classification report
print("Deep Learning Model Results")
print(classification_report(y_test, y_pred_dl))

# Create a DataFrame for the confusion matrix
conf_matrix_df_dl = pd.DataFrame(conf_matrix_dl, index=['Fake', 'Real'],␣
 ↪columns=['Fake', 'Real'])

# Plotting the confusion matrix using Plotly
```

```python
fig_dl = px.imshow(conf_matrix_df_dl,
                   text_auto=True,
                   color_continuous_scale='Plasma',
                   labels=dict(x="Predicted", y="Actual", color="Count"),
                   title="Confusion Matrix for Deep Learning Model")

# Show the plot
fig_dl.show()
```

```
Deep Learning Model Results
              precision    recall  f1-score   support

           0       0.67      0.70      0.68       807
           1       0.91      0.89      0.90      2623

    accuracy                           0.85      3430
   macro avg       0.79      0.79      0.79      3430
weighted avg       0.85      0.85      0.85      3430
```



The above display the results & visualization for BiLSTM (a deep learning model). Before making analysing & comparing the 2 models, I will run the BiLTM model **multiple (5)** times. **Running it**

multiple times is essential for assessing the robustness, consistency & statistical validity of deep learning models. This method is important because neural networks are inherently **stochastic**, as various factors such as **weight initialization, dropout & batch shuffling** can lead to **slight variations** in performance. A deep learning model might show 92% accuracy in one run but 89% in another. Without multiple evaluations, it's **difficult** to accurately gauge the true performance of a model, as a single run can lead to overestimation or underestimation. Running the model multiple times ensures that the reported performance metrics are not **skewed** by random factors.

Multiple runs also allows us to assess the stability of our model. **High variance** in performance across runs may indicate that the model is sensitive to initialization or training conditions. This could signal **underlying issues** such as overfitting, instability in the training dynamics (possibly due to an overly high learning rate), or inadequate data preprocessing. If, on the other hand, the model consistently shows **similar performance** in each run, it suggests a more reliable & stable model. A model that performs consistently across multiple runs is more likely to be robust and ready for real-world applications.

By performing multiple runs, we can also calculate **confidence intervals**, which offer a statistically robust way of evaluating the model's performance. Instead of relying on a single accuracy score from one run, we can calculate the mean & standard deviation of key metrics like accuracy or F1-score. The mean gives an **estimate** of the expected performance, while the standard deviation provides insight into the **variability** of the model's performance. **Lower variability** in performance indicates a more stable model, making it more reliable for production environments.

Finally, Logistic regression produces the **same results** every time for the same dataset & hyperparameters. **Without** multiple runs for deep learning model, the comparison might **unfairly favor** logistic regression because its results are stable, while the deep learning model's **results could fluctuate**. Multiple runs provide a clearer picture, allowing us to determine whether the deep learning model **consistently outperforms** traditional methods & whether the added computational cost of using a deep learning model is justified by its **performance gains**. Multiple runs ensure that the **trade-off** analysis between complexity and performance is based on reliable deep learning results.

[21]:
```
# Run the model multiple times and record accuracy
accuracies = []
for _ in range(5): # Number of runs
    # Re-train the model
    history = model.fit(
        X_train_pad, y_train,
        batch_size=64, epochs=20, # Fewer epochs for quicker experimentation
        validation_data=(X_val_pad, y_val),
        class_weight=class_weight_dict,
        callbacks=[early_stopping, reduce_lr], # Use early stopping and␣
↪reduce_lr callbacks
        verbose=0 # Suppress verbose output for repeated runs
    )

    # Predict on test data
    y_pred_dl = (model.predict(X_test_pad) > 0.5).astype(int).flatten()
    # Record accuracy
    accuracies.append(accuracy_score(y_test, y_pred_dl))

# Report the mean and standard deviation of accuracies
print("Deep Learning Model Results Over Multiple Runs")
print("Mean Accuracy:", np.mean(accuracies))
print("Standard Deviation:", np.std(accuracies))
```

```
Epoch 4: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.
Epoch 5: early stopping
Restoring model weights from the end of the best epoch: 1.
108/108              18s 168ms/step
Epoch 5: early stopping
Restoring model weights from the end of the best epoch: 1.
108/108              19s 176ms/step

Epoch 5: ReduceLROnPlateau reducing learning rate to 8.000000525498762e-06.
Epoch 5: early stopping
Restoring model weights from the end of the best epoch: 1.
108/108              20s 184ms/step

Epoch 4: ReduceLROnPlateau reducing learning rate to 1.6000001778593287e-06.
Epoch 5: early stopping
Restoring model weights from the end of the best epoch: 1.
108/108              20s 185ms/step
Epoch 5: early stopping
Restoring model weights from the end of the best epoch: 1.
108/108              19s 174ms/step
Deep Learning Model Results Over Multiple Runs
Mean Accuracy: 0.8351020408163266
Standard Deviation: 0.0015493096507972666
```

1.1.18 III Conclusions

1.1.19 9. Evaluation

In this project, I implemented two models for fake news detection: a traditional statistical model (Logistic Regression) and a modern deep learning model (Bidirectional Long Short-Term Memory, BiLSTM). The performance of both models was evaluated using key metrics such as accuracy, precision, recall, and F1-score.

1. Results:

Logistic Regression

- Accuracy: 80%
- Precision (Fake): 57%

- Precision (Real): 91%
- Recall (Fake): 74%
- Recall (Real): 82%
- F1-Score (Fake): 64%
- F1-Score (Real): 86%

BiLSTM

- Accuracy: 85%
- Precision (Fake): 67%
- Precision (Real): 91%
- Recall (Fake): 70%
- Recall (Real): 89%
- F1-Score (Fake): 68%
- F1-Score (Real): 90%

2. Visualizations: Using the confusion matrices:

Confusion Matrix for Logistic Regression Model:

- **True Positives (Real predicted as Real)**: 2851
- **True Negatives (Fake predicted as Fake)**: 817
- **False Positives (Fake predicted as Real)**: 282
- **False Negatives (Real predicted as Fake)**: 623

Confusion Matrix for Deep Learning Model:

- **True Positives (Real predicted as Real)**: 2345
- **True Negatives (Fake predicted as Fake)**: 561
- **False Positives (Fake predicted as Real)**: 246
- **False Negatives (Real predicted as Fake)**: 278

3. Significant Findings:

- Fake news (minority class):
  - Logistic regression struggles more, with lower precision (57%) & recall (74%).
  - The deep learning model performs better, with improved precision (67%) & balanced recall (0.70%).
- Real news (majority class):
  - Both models perform well, but the deep learning model achieves slightly better recall & F1-score.

4. Advantages and Disadvantages:

1. **Logistic Regression**:

- **Advantages**: Simplicity, interpretability, requires less computational power & training time. [13]
- **Disadvantages**: Limited ability to capture complex relationships in the data, which may lead to suboptimal performance in nuanced tasks like fake news detection.

2. **BiLSTM**:

- **Advantages**: Generally outperforms Logistic Regression in terms of precision, recall, and F1-score, especially for the minority class (Fake). is able to capture complex patterns &

contextual information, leading to improved performance in classification tasks [14].
- **Disadvantages**: Requires more computational resources & time to train & may be prone to overfitting if not properly regularized [14].

5. **Preferred Scenarios:**

- Logistic Regression may be preferred in scenarios where interpretability is crucial, such as in regulatory environments or when stakeholders need to understand model decisions. logistic regression can be used to find answers to questions that have two or more finite outcomes. [13]
- BiLSTM is more suitable for applications where accuracy is paramount, and the complexity of the data necessitates a more sophisticated approach.

6. **Hypothesis for Performance Disparities:**

- **Data Complexity**: The Deep Learning model might be better at capturing complex relationships & nuances in the data compared to Logistic Regression.
- **Class Imbalance**: The use of oversampling & class weights might have affected the performance differently in both models.
- **Model Architecture**: The architecture of the Deep Learning model, including layers like LSTM and regularization techniques, contributes to its higher performance.

7. **Trade-Off Analysis**

Complexity vs. Performance (based on results):

- **Logistic Regression**:
  - Simpler, faster & easier to interpret.
  - Performance is acceptable for some use cases but struggles with minority class (fake news).
- **Deep Learning**:
  - Computationally intensive & harder to interpret.
  - Shows consistent superiority in accuracy and minority class detection, making it worth the complexity if detecting fake news is critical.

Interpretability: - Logistic regression offers better interpretability, as it provides clear coefficients for each feature. - Deep learning requires additional tools like **SHAP** or **LIME** to explain predictions.

8. **Fairness Insights From Multiple Runs**

Mean Performance:

- **Logistic Regression Accuracy**: 80%.
- **Deep Learning Mean Accuracy**: 83.51%.
  - The deep learning model consistently outperforms logistic regression in terms of accuracy.

Stability

- **Logistic Regression**: No variability because it's deterministic.

- **Deep Learning**: Standard deviation of **0.0015** across 5 runs indicates very stable performance. This confirms that the deep learning model's superiority is not due to random chance but reflects consistent improvements over logistic regression.

1.1.20 10. Summary and Conclusions

This project gave valuable insights into how different machine learning methods detect fake news. Comparing Logistic Regression and BiLSTM showed each model's strengths and weaknesses, helping us understand their real-world use.

1. Learning Experience: Throughout the project, I gained hands-on experience in various aspects of natural language processing (NLP), including data preprocessing, feature extraction, model training & evaluation. The process of cleaning & preparing the dataset was particularly enlightening, as it underscored the importance of data quality in machine learning. I learned how different preprocessing techniques can significantly impact model performance, especially in the context of imbalanced datasets. Additionally, implementing both a traditional statistical model & a modern deep learning model allowed me to understand the strengths & weaknesses inherent in each approach.

2. Practicality of each model type: Logistic Regression proved to be a robust and interpretable model, suitable for scenarios where quick insights and explanations are necessary. Its simplicity and efficiency make it an excellent choice for applications where computational resources are limited or where model interpretability is paramount, such as in regulatory environments or when stakeholders require clear justifications for model predictions.

On the other hand, the BiLSTM model demonstrated superior performance in terms of accuracy, precision, and recall, particularly for the minority class (fake news). Its ability to capture complex patterns and contextual information makes it well-suited for applications where accuracy is critical, such as in automated news verification systems or social media monitoring tools. However, the increased computational requirements and complexity of deep learning models may limit their applicability in resource-constrained environments.

3. Contributions to the Problem Area: This project contributes to the ongoing discourse on fake news detection by providing a comparative analysis of traditional and modern machine learning approaches. The findings highlight the importance of selecting the appropriate model based on the specific requirements of the task at hand, such as the need for interpretability versus the need for accuracy. By situating this work within the broader context of misinformation research, it offers valuable insights for practitioners and researchers alike.

4. Transferability to Other Domain-Specific Areas: The methodologies and insights gained from this project are transferable to various other domains that involve text classification, such as sentiment analysis, spam detection, and document tagging. The preprocessing techniques, evaluation metrics, and model comparison strategies can be adapted to suit different datasets and classification tasks. For instance, the approaches used in fake news detection can be applied to identify fraudulent emails or analyze customer sentiment in product reviews.

5. Future Research Directions: Future work could combine statistical and deep learning models to leverage their strengths. Exploring transformer-based models like BERT could improve performance. Also, using a larger and more diverse dataset could give a better evaluation of the models. Data augmentation techniques, such as paraphrasing or back-translation, could help balance the dataset, especially for the minority class. Feature engineering can include embedding source_domain as a categorical feature. Domain-specific embeddings, trained on news from particular sectors like politics or finance, can capture specialized terms. To enhance explainability, tools like SHAP or LIME can be used to understand model decisions

1.2 Resources used

- **pandas** - pandas documentation
- **plotly.graph_objects** - Plotly Graph Objects documentation
- **plotly.express** - Plotly Express documentation
- **spacy** - spaCy documentation
- **re** - Python `re` module documentation
- **sklearn.feature_extraction.text.TfidfVectorizer** - TfidfVectorizer documentation
- **sklearn.model_selection** - train_test_split documentation
- **RandomizedSearchCV** - RandomizedSearchCV documentation
- **sklearn.linear_model.LogisticRegression** - LogisticRegression documentation
- **sklearn.preprocessing.MaxAbsScaler** - MaxAbsScaler documentation
- **sklearn.metrics** - classification_report documentation
- **confusion_matrix** - confusion_matrix documentation
- **imblearn.over_sampling.RandomOverSampler** - RandomOverSampler documentation
- **tensorflow.keras.models.Sequential** - Sequential documentation
- **tensorflow.keras.preprocessing.text.Tokenizer** - Tokenizer documentation
- **tensorflow.keras.layers** - Embedding documentation
- **LSTM** - LSTM documentation
- **Bidirectional** - Bidirectional documentation
- **Dense** - Dense documentation
- **Dropout** - Dropout documentation
- **BatchNormalization** - BatchNormalization documentation
- **tensorflow.keras.optimizers.Adam** - Adam documentation
- **tensorflow.keras.callbacks** - EarlyStopping documentation
- **ReduceLROnPlateau** - ReduceLROnPlateau documentation
- **ModelCheckpoint** - ModelCheckpoint documentation
- **tensorflow.keras.preprocessing.sequence.pad_sequences** - pad_sequences documentation
- **numpy** - NumPy documentation
- **sklearn.utils.class_weight.compute_class_weight** - compute_class_weight documentation

glove.6B.100d.txt - https://nlp.stanford.edu/projects/glove/

1.3 References

[1] P. Goyal, S. Taterh, and A. Saxena, "Fake News Detection using Machine Learning: A Review," International Journal of Advanced Engineering, Management and Science (IJAEMS), vol. 7, no. 3, pp. 2454–1311, 2021, doi: 10.22161/ijaems.

[2] S. Hakak, W. Z. Khan, S. Bhattacharya, G. T. Reddy, and K. K. R. Choo, "Propagation of Fake News on Social Media: Challenges and Opportunities," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 12575 LNCS, pp. 345–353, 2020, doi: 10.1007/978-3-030-66046-8_28.

[3] H. Allcott and M. Gentzkow, "Social Media and Fake News in the 2016 Election," Journal of Economic Perspectives, vol. 31, no. 2, pp. 211–36, Mar. 2017, doi: 10.1257/JEP.31.2.211.

[4] P. Kulkarni, S. Karwande, R. Keskar, P. Kale, and S. Iyer, "Fake News Detection using Machine Learning," ITM Web of Conferences, vol. 40, p. 03003, 2021, doi: 10.1051/itmconf/20214003003.

[5] H. Allcott, M. Gentzkow, and C. Yu, "Trends in the diffusion of misinformation on social media," Research and Politics, vol. 6, no. 2, Apr. 2019, doi: 10.1177/2053168019848554.

[6] S. M. Jones-Jang, T. Mortensen, and J. Liu, "Does Media Literacy Help Identification of Fake News? Information Literacy Helps, but Other Literacies Don't," American Behavioral Scientist, vol. 65, no. 2, pp. 371–388, Feb. 2021, doi: 10.1177/0002764219869406

[7] P. Machete and M. Turpin, "The Use of Critical Thinking to Identify Fake News: A Systematic Literature Review," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 12067 LNCS, pp. 235–246, 2020, doi: 10.1007/978-3-030-45002-1_20.

[8] A. Raj, "A Review on Machine Learning Algorithms," Int J Res Appl Sci Eng Technol, vol. 7, no. 6, pp. 792–796, Jun. 2019, doi: 10.22214/IJRASET.2019.6138.

[9] Vosoughi, S., Roy, D., & Aral, S. (2018). The spread of true and false news online. Science, 359(6380), 1146-1151.

[10] Zhang, X., et al. (2020). A survey on fake news detection. ACM Computing Surveys, 54(2), 1-35.

[11] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[12] IJNRD. (2023). Fake News Detection Using Logistic Regression. International Journal of Novel Research and Development.

[13] AWS. Logistic Regression. Amazon Web Services.

[14] Nama, A. (2023). Understanding Bidirectional LSTM for Sequential Data Processing. Medium.

rrrrrrrrrrrrrrrrrrrrrrfffffffffffffffffffffffffffffffffffffffffrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr

```
[ ]:
```