Fake News Classification Using LTSM, NLP and Transformers

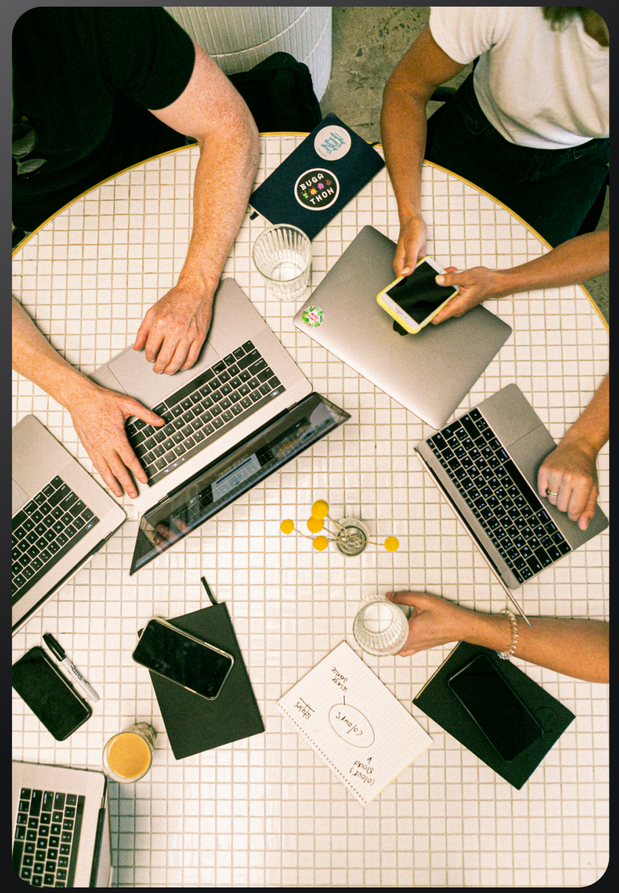
Leveraging Advanced Machine Learning to Combat Misinformation

Group 1: Terry Brown, Elijah Mercier, LaQuita Palmer, Ivanna Price

Subject Summary

In an era where misinformation spreads rapidly across digital platforms, the ability to detect and classify fake news is more critical than ever.

- Fake news is a pervasive issue that influences public opinion and decision-making.
- Our project aimed to develop a machine learning pipeline to predict whether a given news articles is based off of real or fake information.
- We used advanced machine learning techniques, including LDA, Vader Sentiment Analysis, and LSTM.
- Goals:
  - Create a model that can predict whether a given news articles is fake with high accuracy and confidence.
  - Analyze patterns and trends in fake news.



OVERVIEW

- Objective:
  - Develop a machine learning model to predict newly given news articles as either fake or real.
- Key Goals:
  - Improve detection of fake news using NLP and LSTM techniques.
  - Understand keywords and topics commonly associated with fake news.
  - Visualize trends such as frequency, timing, and targeted topics.
- Applications:
  - Curb misinformation in online platforms.
  - Build trust in online journalism.

PROJECT FEATURES

Data Cleaning and Conversion:

Function to convert and combine topic labels into a singular DataFrame, enhancing interpretability of the results.

Latent Dirichlet Allocation (LDA) Implementation:

LDA Model: Initialized with 10 topics.

Fit Model: Applied to the document-term matrix (DTM)

Topic Analysis:

Top Words: Extracted top 15 words for each topic to understand key themes.

Transform DTM: DTM transformed into topic distributions for each document.

Vader Sentiment Analysis:

Used to analyze the sentiment of news headlines and articles, providing insights into the emotional tone of the content

Topic Labels:

Assigned labels (e.g., Entertainment, Sports) to the most probable topic for each document.

LSTM Model Framework:

Usage of LSTM, Multiple LSTM, and Bidirectional LSTM models.

Application Interface:

Added for user-friendly design and model usage functionality.

DATASETS

- Collected datasets from Kaggle: WELFake, train.csv, test.csv, and evaluation.csv.
- News headlines, content, and labels (real/fake) were used.

Data cleaning steps:

  - Removed duplicates, null values, punctuation, and stop words.
  - Standardized data label formats and combined datasets

- Collaborated via Google Colab for unified analysis and processing.

```
[ ] # Load the datasets
    wel_fake_df = pd.read_csv('./saurabh_fake_news_classification/WELFake_Dataset.csv')
    train_df = pd.read_csv('./aadyasingh_fake_news_classification/train (2).csv', sep=';')
    test_df = pd.read_csv('./aadyasingh_fake_news_classification/test (1).csv', sep=';')
    eval_df = pd.read_csv('./aadyasingh_fake_news_classification/evaluation.csv', sep=';')
```

```
[ ] wel_fake_df.head()
```

	Unnamed: 0	title	text	label
0	0	LAW ENFORCEMENT ON HIGH ALERT Following Threat...	No comment is expected from Barack Obama Membe...	1
1	1	NaN	Did they post their votes for Hillary already?	1
2	2	UNBELIEVABLE! OBAMA'S ATTORNEY GENERAL SAYS MO...	Now, most of the demonstrators gathered last ...	1
3	3	Bobby Jindal, raised Hindu, uses story of Chri...	A dozen politically active pastors came here f...	0
4	4	SATAN 2: Russia unvelis an image of its terrif...	The RS-28 Sarmat missile, dubbed Satan 2, will...	1

```
[ ] train_df.head()
```

	Unnamed: 0	title	text	label
0	0	Palestinians switch off Christmas lights in Be...	RAMALLAH, West Bank (Reuters) - Palestinians s...	1
1	1	China says Trump call with Taiwan president wo...	BEIJING (Reuters) - U.S. President-elect Donal...	1
2	2	FAIL! The Trump Organization's Credit Score W...	While the controversy over Trump s personal ta...	0
3	3	Zimbabwe military chief's China trip was norma...	BEIJING (Reuters) - A trip to Beijing last wee...	1
4	4	THE MOST UNCOURAGEOUS PRESIDENT EVER Receives ...	There has never been a more UNCOURAGEOUS perso...	0

```
[ ] # Combine all datasets into a single DataFrame
    combined_df = pd.concat([wel_fake_df, train_df, test_df, eval_df], ignore_index=True)
    combined_df.shape
```

```
(112721, 4)
```

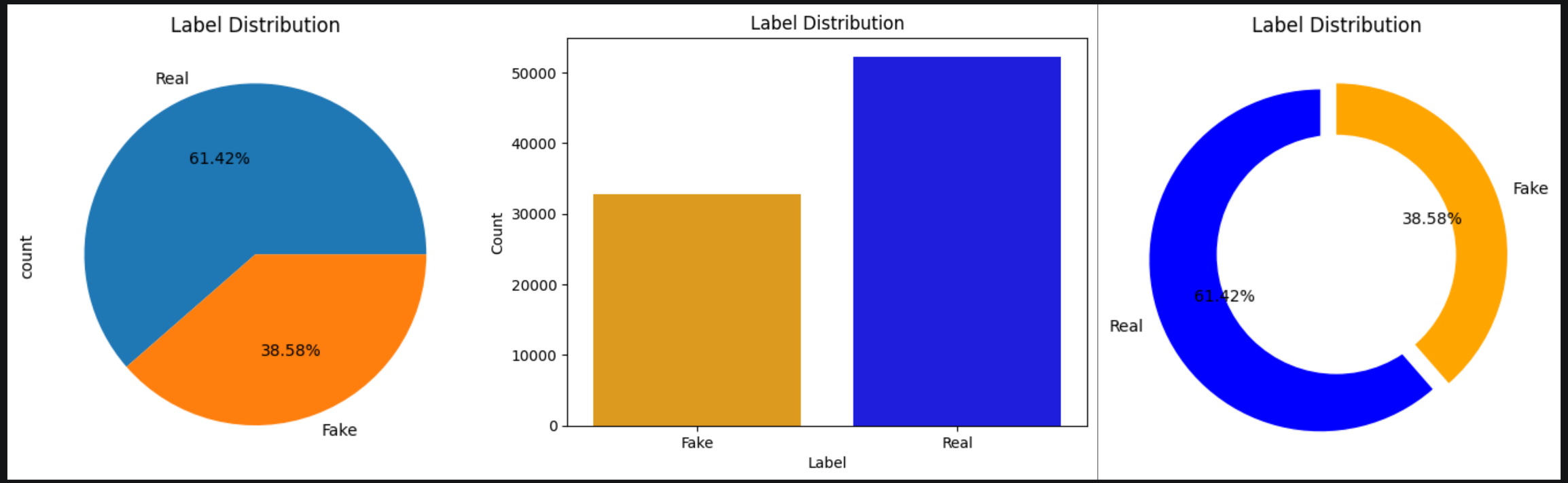
```
[ ] # Count duplicates based on "title" and "text" columns in the combined DataFrame
    total_duplicates = combined_df.duplicated(subset=["title", "text"]).sum()
    print("Total Duplicates Across All Datasets:", total_duplicates)
```

```
Total Duplicates Across All Datasets: 27054
```

```
[ ] deduplicated_df = combined_df.drop_duplicates(subset=["title", "text"], inplace=False)
    deduplicated_df.shape
```

```
(85667, 4)
```

REAL VS FAKE DISTRIBUTION



What’s in the Data?

The label distribution graphs for "Real" vs "Fake" show the following:

- The graphs provide a visual representation of how many articles are labeled as "Real" compared to how many are labeled as "Fake." The pie chart and bar graph reveal that **around 60% of the articles are labeled as "Real" and 40% are labeled as "Fake."** This is an important insight because class imbalance can affect model performance, especially if one class dominates the dataset.

```
# One-hot encode the corpus and add padding to standardize input lengths
# Convert the corpus to one-hot representations with a defined vocabulary size
vocab_size = 10000  # Define vocabulary size for one-hot encoding
onehot_repr = [one_hot(words, vocab_size) for words in corpus]

# Pad sequences to ensure uniform input size
sentence_length = 50
embedded_docs = pad_sequences(onehot_repr, padding='pre', maxlen=sentence_length)

# Print a preview of the padded documents
print("Sample of embedded documents:")
print(embedded_docs[:5])

# Concatenate topic results and sentiment scores into the final input array
# Combine topic results and sentiment scores with the embedded documents
X_final_with_topics = np.concatenate((embedded_docs, topic_results), axis=1)
X_final_with_sentiment = np.concatenate((X_final_with_topics, X_sentiment), axis=1)

# Convert to NumPy arrays for training
X_final = np.array(X_final_with_sentiment)
y_final = np.array(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_final, y_final, test_size=0.2, random_state=42)
print("Train-test split completed.")
```

```
Sample of embedded documents:
[[   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    0    0    0    0    0    0    0    0    0    0    0    0
  4627 8026 5055 1015 3498 9858 4350  825]
 [   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    0    0    0    0    0    0 7057 9443 4118 6824 5198 1204
  6967 2116 9247 7312  543 7407  265  825]
 [   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    0    0    0    0    0    0    0    0 9528 4863 2580 9342
  9193  506 5779 2109 2499 1061 7748 5932]
 [   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    0    0    0    0    0    0    0    0    0 9534 8569 7598
  2747 9789 6497 8579 6222  441  243  737]
 [   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    0    0    0    0    0    0    0    0    0    0    0 5077
  5779 6478 6311 7529 3457 6495 7328 6478]]
Train-test split completed.
```

Data Splitting & Baseline Model

- The dataset was split into training, validation, and testing sets using an 80-20 split to ensure proper evaluation.
- This process allowed us to train the models on a majority of the data while using the testing set to measure generalization.
- Split datasets into training, validation, and testing sets.
- Developed a logistic regression model using TF-IDF vectors as features to set a performance baseline for comparison with advanced models.
- The logistic regression model achieved an initial accuracy of approximately 75%, providing a benchmark for more advanced models.

One-Hot Encoding

One-Hot Encoding and Padding:

The corpus is converted to one-hot representations using a vocabulary size of 10,000, and sequences are padded to a uniform length of 50 to standardize input sizes.

Concatenation of Features:

Embedded documents are combined with topic results and sentiment scores to form a comprehensive input array for model training.

Data Preparation:

The final input array and labels are converted to NumPy arrays, followed by splitting the data into training and testing sets with a 20% test size.

```python
# One-hot encode the corpus and add padding to standardize input lengths
# Convert the corpus to one-hot representations with a defined vocabulary size
vocab_size = 10000  # Define vocabulary size for one-hot encoding
onehot_repr = [one_hot(words, vocab_size) for words in corpus]

# Pad sequences to ensure uniform input size
sentence_length = 50
embedded_docs = pad_sequences(onehot_repr, padding='pre', maxlen=sentence_length)

# Print a preview of the padded documents
print("Sample of embedded documents:")
print(embedded_docs[:5])

# Concatenate topic results and sentiment scores into the final input array
# Combine topic results and sentiment scores with the embedded documents
X_final_with_topics = np.concatenate((embedded_docs, topic_results), axis=1)
X_final_with_sentiment = np.concatenate((X_final_with_topics, X_sentiment), axis=1)

# Convert to NumPy arrays for training
X_final = np.array(X_final_with_sentiment)
y_final = np.array(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_final, y_final, test_size=0.2, random_state=42)
print("Train-test split completed.")
```

```
Sample of embedded documents:
[[   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    0    0    0    0    0    0    0    0 5754 8116 4832 8298
  4283 3197 7060  233 3634 9770 2708  819]
 [   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    0    0    0    0    0    0    0 1445 1876 4500 6707 5272  739
  6560 8422 6394 1757 5243 1699 2782  819]
 [   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    0    0    0    0    0    0    0    0 4420 1627 9595 1739
   178  520 3351 7596  904 6625 4870 8042]
 [   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    0    0    0    0    0    0    0    0    0 5220 2245 2027
  2700  773 9386  179 5060 1055 8178  970]
 [   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    0    0    0    0    0    0    0    0    0    0    0 6438
  3351 9442 9251 4018 6064 9778 7164 9442]]
Train-test split completed.
```

Model Implementations

We implemented three advanced models to improve the accuracy of fake news detection. Each model was designed to leverage different strengths in processing text data:

- LDA: Topic modeling to uncover common themes, allowing us to better understand the different subject matters that were being targeted by fake news.
- VADER Sentiment Analysis: VADER (Valence Aware Dictionary and Sentiment Reasoner) was used to analyze the sentiment of news headlines and articles, providing insights into the emotional tone of the content.
- LSTM-RNN: Captured sequential dependencies in text, optimized with batch size, learning rate, and dropout rate, which helped improve understanding of the temporal relationships in the content.

```python
# Create an instance of CountVectorizer and fit-transform the corpus
cv = CountVectorizer(max_df=0.95, min_df=5, stop_words='english')
dtm = cv.fit_transform(corpus)
print("Document-Term Matrix shape:", dtm.shape)


# Initialize and fit the LDA model
lda = LatentDirichletAllocation(n_components=10, random_state=42)
lda.fit(dtm)


# Display the top 15 words for each topic for analysis
for index, topic in enumerate(lda.components_):
    print(f'The Top 15 Words For Topic #{index + 1}')
    print([cv.get_feature_names_out()[i] for i in topic.argsort()[-15:]])
    print('\n')


# Transform the DTM into topic results
topic_results = lda.transform(dtm)

# Define topic labels and map them to the most probable topic for each document
topic_labels = {
    1: 'Entertainment',
    2: 'Sports',
    3: 'Business',
    4: 'Politics',
    5: 'Technology'

}

# Function to add topic labels to the DataFrame
def add_topic_labels(cleaned_df, topic_results, topic_labels):
    cleaned_df['topic'] = topic_results.argmax(axis=1) + 1
    cleaned_df['topic_label'] = cleaned_df['topic'].map(topic_labels)

# Apply the function to add topic labels
add_topic_labels(cleaned_df, topic_results, topic_labels)

# Display a preview of the labeled DataFrame
print(cleaned_df.head(20))
```

```
                                                text  label  topic  \
0   No comment is expected from Barack Obama Membe...      0     10
1    Now, most of the demonstrators gathered last ...      0     10
2   A dozen politically active pastors came here f...      1      1
3   The RS-28 Sarmat missile, dubbed Satan 2, will...      0      5
4   All we can say on this one is it s about time ...      0     10
5   DR. BEN CARSON TELLS THE STORY OF WHAT HAPPENE...      0      1
6                                                          0      2
7   The owner of the Ringling Bar, located south o...      0     10
8   FILE - In this Sept. 15, 2005 file photo, the ...      0     10
9   The most punchable Alt-Right Nazi on the inter...      0      2
10  BRUSSELS (Reuters) - British Prime Minister Th...      1      4
11  WASHINGTON (Reuters) - Charles Schumer, the to...      1      5
12  After watching this telling video, you ll wond...      0      2
13  As more and more sports fans turn off ESPN to ...      1      3
14  RIO DE JANEIRO/SAO PAULO (Reuters) - Billionai...      1      8
15  Europe is likely not going to be a top destina...      0     10
16  GENEVA (Reuters) - The United Nations called o...      1      5
17  The Atlantic, a publication that wouldn t know...      0      2
18  NEW YORK (Reuters) - A second federal judge ha...      1      7
19  Wednesday 9 November 2016 by Lucas Wilde Ameri...      0      2

      topic_label
0             NaN
1             NaN
2   Entertainment
3      Technology
4             NaN
5   Entertainment
6          Sports
7             NaN
8             NaN
9          Sports
10       Politics
11     Technology
12         Sports
13       Business
14            NaN
15            NaN
16     Technology
17         Sports
18            NaN
19         Sports
```

Latent Dirichlet Allocation (LDA)

- Headline vs. content matching analysis.
- Identified common keywords and phrases in fake news through frequency analysis and word clouds.
- Conducted topic modeling using LDA to identify targeted topics.
- Visualized posting frequency and time trends to understand distribution over time.

Sentiment Analysis with VADER

- In addition to the other models, we incorporated sentiment analysis using VADER to further enrich our understanding of the content.
- VADER (Valence Aware Dictionary and sEntiment Reasoner) was used to analyze the sentiment of news headlines and articles, providing insights into the emotional tone of the content.
- We implemented three advanced models to improve the accuracy of fake news detection.

Sentiment Score Calculation:

- Utilized VADER (Valence Aware Dictionary and sEntiment Reasoner) to compute sentiment scores for each title in the corpus, focusing on the compound score for a holistic sentiment measurement.

Data Reshaping:

- Reshaped the sentiment scores into a one-dimensional array for seamless concatenation with other features.

Sample Sentiment Scores:

- Displayed a sample of the calculated sentiment scores to validate the results and ensure correct processing.

```
# Sentiment Analysis using VADER
# Calculate sentiment scores for each title in the corpus
sentiments = [analyzer.polarity_scores(title)['compound'] for title in corpus]

# Reshape sentiment scores for concatenation
X_sentiment = np.array(sentiments).reshape(-1, 1)

# Display a sample of sentiment scores
print("Sample sentiment scores:")
print(X_sentiment[:5])
```

```
Sample sentiment scores:
[[-0.7845]
 [-0.25  ]
 [ 0.4767]
 [ 0.    ]
 [-0.4588]]
```

LSTM Model Selection and Training

Model Selection and Training:
Defined LSTM models with singular, multiple, and bidirectional layers, starting with an embedding layer (150 features), followed by three LSTM layers with 100, 50, and 20 units respectively.

Model Compilation and Training:
Compiled the model using the binary cross-entropy loss function and the Adam optimizer, and trained it for 5 epochs, using validation data to check performance.

Model Evaluation:
Predicted outcomes on the test data, converted predictions to binary labels (threshold of 0.5), and calculated accuracy and confusion matrix for performance evaluation.

Model Saving:
Saved the trained model in the '.keras' format for future use.

LSTM Model
(model)

Model Definition and Training:

- Defined an LSTM model with an embedding layer (125 features) and trained it using the binary cross-entropy loss function and the Adam optimizer. Input length was set based on the training data shape.

Model Compilation and Training:

- The model was compiled and trained for 5 epochs, with validation data used to monitor performance.

- Predicted the test data using the trained model, converting predictions to binary labels with a threshold of 0.5. Calculated accuracy and confusion matrix for performance evaluation.

```python
# Define and train LSTM models
vocab_size = 10000
embedding_vector_features = 125
input_length = X_train.shape[1]

# First Model
model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_vector_features))#input_length=input_length
model.add(LSTM(50))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())

model.fit(X_train, y_train, epochs=5)

# Evaluate First Model
y_log = model.predict(X_test)
y_pred = np.where(y_log > 0.5, 1, 0)

acc = accuracy_score(y_test, y_pred)
confusion_mat = confusion_matrix(y_test, y_pred)

model.save('model.keras')

print(acc)
print(confusion_mat)
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
lstm (LSTM)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)

```
Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)
None
Epoch 1/5
2128/2128 ━━━━━━━━━━━━━━━━━━━━ 20s 7ms/step - accuracy: 0.8342 - loss: 0.3546
Epoch 2/5
2128/2128 ━━━━━━━━━━━━━━━━━━━━ 15s 7ms/step - accuracy: 0.9255 - loss: 0.1870
Epoch 3/5
2128/2128 ━━━━━━━━━━━━━━━━━━━━ 20s 7ms/step - accuracy: 0.9437 - loss: 0.1468
Epoch 4/5
2128/2128 ━━━━━━━━━━━━━━━━━━━━ 21s 8ms/step - accuracy: 0.9565 - loss: 0.1114
Epoch 5/5
2128/2128 ━━━━━━━━━━━━━━━━━━━━ 22s 8ms/step - accuracy: 0.9672 - loss: 0.0853
532/532 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
0.9114675126307132
[[5693  891]
 [ 616 9822]]
```

Multiple LSTM Model
(model1)

Model Definition and Training:

Defined an LSTM model with multiple layers, starting with an embedding layer (150 features), followed by three LSTM layers with 100, 50, and 20 units respectively.

Model Compilation and Training:

Compiled the model using the binary cross-entropy loss function and the Adam optimizer, and trained it for 5 epochs, using validation data to check performance.

```
# Define and train LSTM models
vocab_size = 10000
embedding_vector_features = 150
input_length = X_train.shape[1]

# Second Model with Multiple LSTM Layers
model1 = Sequential()
model1.add(Embedding(input_dim=vocab_size, output_dim=embedding_vector_features))#input_length=input_length
model1.add(LSTM(100, return_sequences=True))
model1.add(LSTM(50, return_sequences=True))
model1.add(LSTM(20))
model1.add(Dense(1, activation='sigmoid'))
model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model1.summary())

model1.fit(X_train, y_train, epochs=5)

# Evaluate Second Model
y_log_1 = model1.predict(X_test)
y_pred_1 = np.where(y_log_1 > 0.5, 1, 0)

acc_1 = accuracy_score(y_test, y_pred_1)
confusion_mat_1 = confusion_matrix(y_test, y_pred_1)

model.save('model1.keras')

print(acc_1)
print(confusion_mat_1)
```

```
Model: "sequential_8"
```

Layer (type)	Output Shape	Param #
embedding_8 (Embedding)	?	0 (unbuilt)
lstm_14 (LSTM)	?	0 (unbuilt)
lstm_15 (LSTM)	?	0 (unbuilt)
lstm_16 (LSTM)	?	0 (unbuilt)
dense_8 (Dense)	?	0 (unbuilt)

```
 Total params: 0 (0.00 B)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 0 (0.00 B)
None
Epoch 1/5
2128/2128 ━━━━━━━━━━━━━━━━━━━━ 379s 175ms/step - accuracy: 0.8518 - loss: 0.3234
```

Bidirectional LSTM Model
(model2)

Model Definition and Training:

Defined an Bidirectional LSTM model starting with an embedding layer (150 features), followed by two LSTM layers with 100 and 200 units, respectively.

Model Compilation and Training:

Compiled the model using the binary cross-entropy loss function and the Adam optimizer, and trained it for 5 epochs, using validation data to check performance.

```python
# Define and train LSTM models
vocab_size = 10000
embedding_vector_features = 150
input_length = X_train.shape[1]

# Third Model with Bidirectional LSTM
model2 = Sequential()
model2.add(Embedding(input_dim=vocab_size, output_dim=embedding_vector_features))#input_length=input_length
model2.add(LSTM(100, return_sequences=True))
model2.add(Bidirectional(LSTM(200, return_sequences=False)))
model2.add(Dense(1, activation='sigmoid'))
model2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model2.summary())

model2.fit(X_train, y_train,  epochs=5, batch_size=100)

# Evaluate Third Model
y_log_2 = model2.predict(X_test)
y_pred_2 = np.where(y_log_2 > 0.5, 1, 0)

acc_2 = accuracy_score(y_test, y_pred_2)
confusion_mat_2 = confusion_matrix(y_test, y_pred_2)

model2.save('model2.keras')

print(acc_2)
print(confusion_mat_2)
```

```
Model: "sequential_10"
```

Layer (type)	Output Shape	Param #
embedding_10 (Embedding)	?	0 (unbuilt)
lstm_20 (LSTM)	?	0 (unbuilt)
bidirectional_2 (Bidirectional)	?	0 (unbuilt)
dense_10 (Dense)	?	0 (unbuilt)

```
 Total params: 0 (0.00 B)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 0 (0.00 B)
None
Epoch 1/5
681/681 ━━━━━━━━━━━━━━━━━━━━ 552s 800ms/step - accuracy: 0.8215 - loss: 0.3709
Epoch 2/5
681/681 ━━━━━━━━━━━━━━━━━━━━ 564s 803ms/step - accuracy: 0.9284 - loss: 0.1880
Epoch 3/5
681/681 ━━━━━━━━━━━━━━━━━━━━ 566s 810ms/step - accuracy: 0.9409 - loss: 0.1591
```

Model Evaluation & Optimization

- The evaluation of our models involved analyzing several important performance metrics to ensure reliability and accuracy in detecting fake news.
- Metrics such as accuracy, precision, recall, F1-score, and confusion matrix were used to provide a comprehensive view of model performance.
- Metrics used included accuracy, precision, recall, F1-score, and confusion matrix, which allowed us to understand both the correctness of predictions and the balance between precision and recall for both classes.
- We iteratively fine-tuned our models to enhance performance:
  - Adjusted hyperparameters like learning rate and dropout rate.
  - Used class balancing techniques.
- Visualized and documented performance metrics to effectively compare the models and assess which approach had the best balance of recall and precision.

```
# Ensure X_train and vocab_size are defined correctly
vocab_size = 10000
embedding_vector_features = 125
input_length = X_train.shape[1]

# Try to build the model
try:
    print("Building model...")
    model = Sequential()
    model.add(Embedding(input_dim=vocab_size, output_dim=embedding_vector_features))
    print("Added Embedding layer.")
    model.add(LSTM(50))
    print("Added LSTM layer.")
    model.add(Dense(1, activation='sigmoid'))
    print("Added Dense layer.")
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    print("Compiled the model.")

    # Create dummy data with the appropriate input shape
    dummy_data = np.zeros((1, input_length))

    # Perform a forward pass to build the model layers
    model.predict(dummy_data)

    # Print the model summary after the layers have been built
    print(model.summary())
except Exception as e:
    print(f"Error in building the model: {e}")

# Try to train the model
try:
    print("Starting training...")
    history = model.fit(X_train, y_train, epochs=5)
    print("Training completed.")
except Exception as e:
    print(f"Error in training the model: {e}")

# Try to make predictions and evaluate the model
try:
    print("Predicting...")
    y_log = model.predict(X_test)
    y_pred = np.where(y_log > 0.5, 1, 0)
    print("Predictions done.")

    print("Calculating accuracy...")
    acc = accuracy_score(y_test, y_pred)
    confusion_mat = confusion_matrix(y_test, y_pred)
    print("Accuracy:", acc)
    print("Confusion Matrix:\n", confusion_mat)
except Exception as e:
    print(f"Error in prediction or evaluation: {e}")

# Try to save the model
try:
    print("Saving model...")
    model.save('model_1.keras')
    print("Model saved.")
except Exception as e:
    print(f"Error in saving the model: {e}")
```

(variable) confusion_mat: Any

And now...

GRADIO Interface Demonstration

Attention Audience:

Please now turn your attention to the chat.
A link has been provided to test the results of our models.

https://6e3e248b6fe430e057.gradio.live/

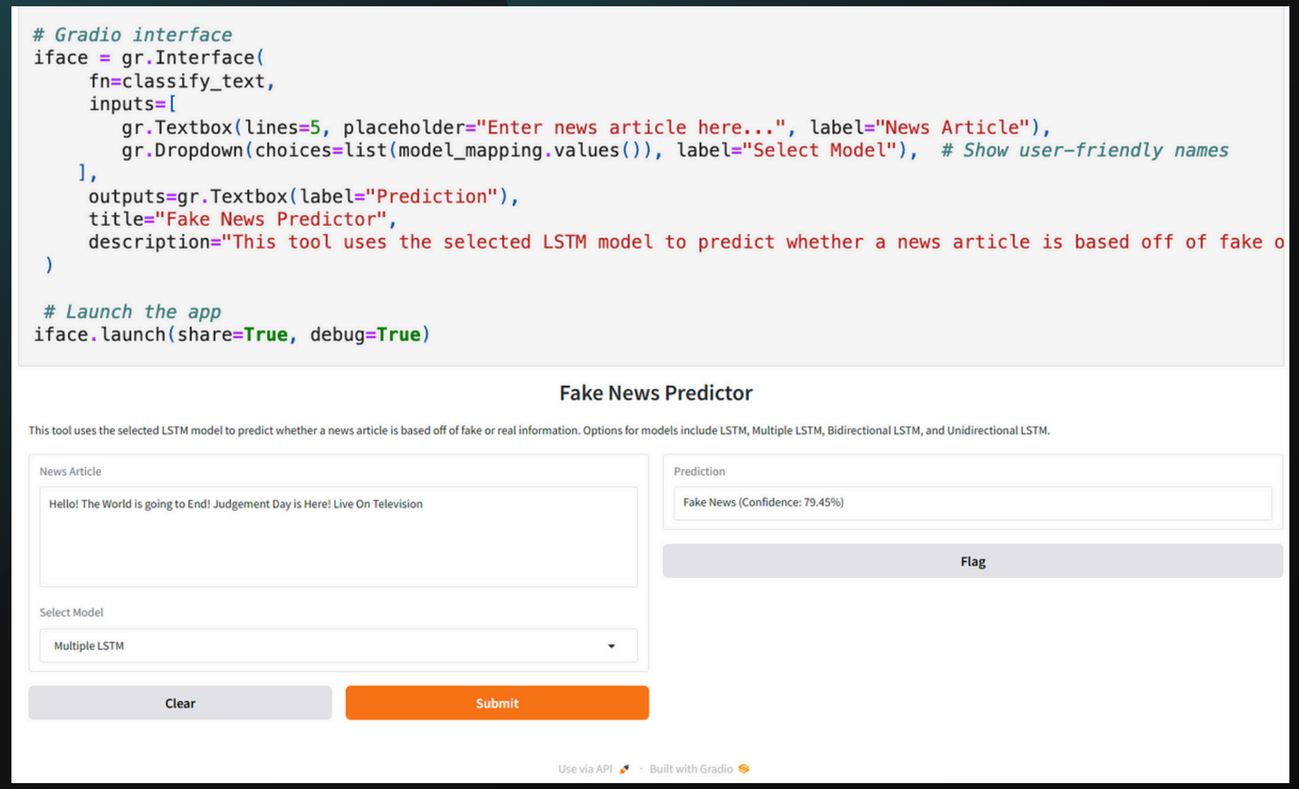
GRADIO Interface Implementation

The Gradio interface was implemented to make the model accessible to users without any programming knowledge. The key aim was to bridge the gap between advanced machine learning models and practical end-user applications.

- Functionality: The Gradio interface provides an easy-to-use web interface where users can:
- Input news articles or headlines to evaluate their credibility.
- Select different models (e.g., LSTM or Transformer) to see how predictions might vary.
- Get an immediate classification label (Real or Fake) along with the probability score that indicates the confidence of the prediction.

```python
# Gradio interface
iface = gr.Interface(
     fn=classify_text,
     inputs=[
        gr.Textbox(lines=5, placeholder="Enter news article here...", label="News Article"),
        gr.Dropdown(choices=list(model_mapping.values()), label="Select Model"),  # Show user-friendly names
    ],
     outputs=gr.Textbox(label="Prediction"),
     title="Fake News Predictor",
     description="This tool uses the selected LSTM model to predict whether a news article is based off of fake o
 )

 # Launch the app
iface.launch(share=True, debug=True)
```



GRADIO Interface Implementation

Why It Was Necessary:

- User Accessibility: Many potential users, such as journalists or educators, may not be familiar with coding or machine learning. The Gradio interface allows these users to benefit from the model without technical barriers.
- Practical Application: The real-time feedback feature supports journalists in quickly verifying the credibility of news articles, making it an effective tool for fighting misinformation.

Results:

- The Gradio interface successfully provided a platform for public use of our fake news classification model.
- It was tested with a variety of inputs, and the accuracy of the model could be demonstrated effectively to end-users through this interactive platform.
- Users could see the impact of choosing different models, helping in understanding the reliability of machine learning in fake news detection.

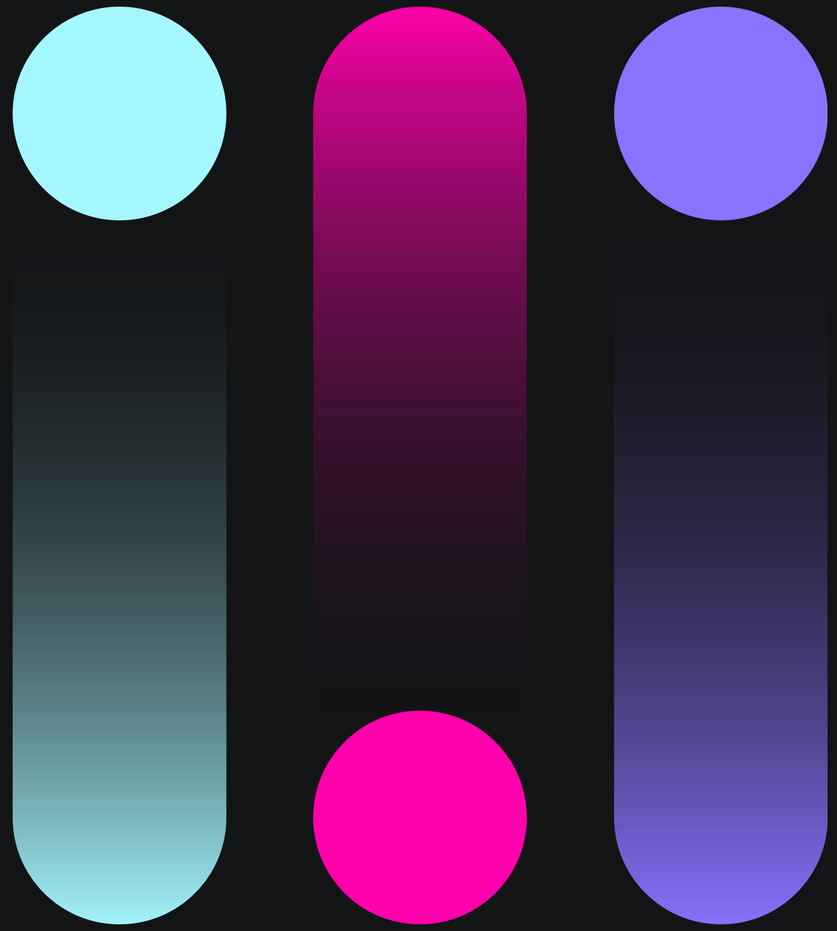
Results and Conclusions

Our results demonstrated significant improvements over the baseline model, particularly when utilizing advanced deep learning models.

- The logistic regression baseline achieved an accuracy of approximately 75%, but it struggled with nuanced linguistic patterns often found in fake news.
- The **LSTM and Multiple LSTM models achieved significantly better** performance compared to the logistic regression model, with the **Multiple LSTM model achieving the highest accuracy and recall.**
- The LSTM model was particularly confident at capturing results, which helped improve detection of fake news patterns over simpler models.
- Common keywords and topics targeted by fake news were identified, which helped reveal the strategies and narratives commonly employed by fake news publishers. These insights can be used to inform future mitigation strategies and content monitoring.
- Conclusion: The Multiple LSTM model provided superior accuracy and robustness for fake news detection. Its contextual understanding capabilities made it ideal for distinguishing between real and fake news, even when similar language was used.

Challenges and Solutions

- Challenges:
- 1. Initial models had no output. This was resolved by removing the dropout layers, adding try and except blocks, and using a dummy data forward pass.
- Removing the dropout layer contributed to resolving the issue by allowing the model to utilize its full learning capacity and providing a stable, uninterrupted learning process.
- The try-except blocks offer clear error messages, and the dummy data forward pass ensures the model layers are correctly built.
- These modifications prevented scenarios where the models failed silently, leading to no output.
- 2. Excessive run time. Solved by adjusting parameters such as # of units, # of epochs, batch size, etc. in each LSTM layer. Also changed runtime type to T4.
- 3. model_1.keras



Next Steps

- Analyze VADER results to explain emotional sentiment findings visually.
- F1-Score, Recall, and Precision parameters and analysis.
- Further epoch training.
- Visualize trends such as frequency, timing, and targeted topics.
- Explore additional transformer models.
- Enhance features using more advanced embeddings.
- Integrate the model into a real-time detection system.

Conclusion

- Model Conclusions:

LSTM:

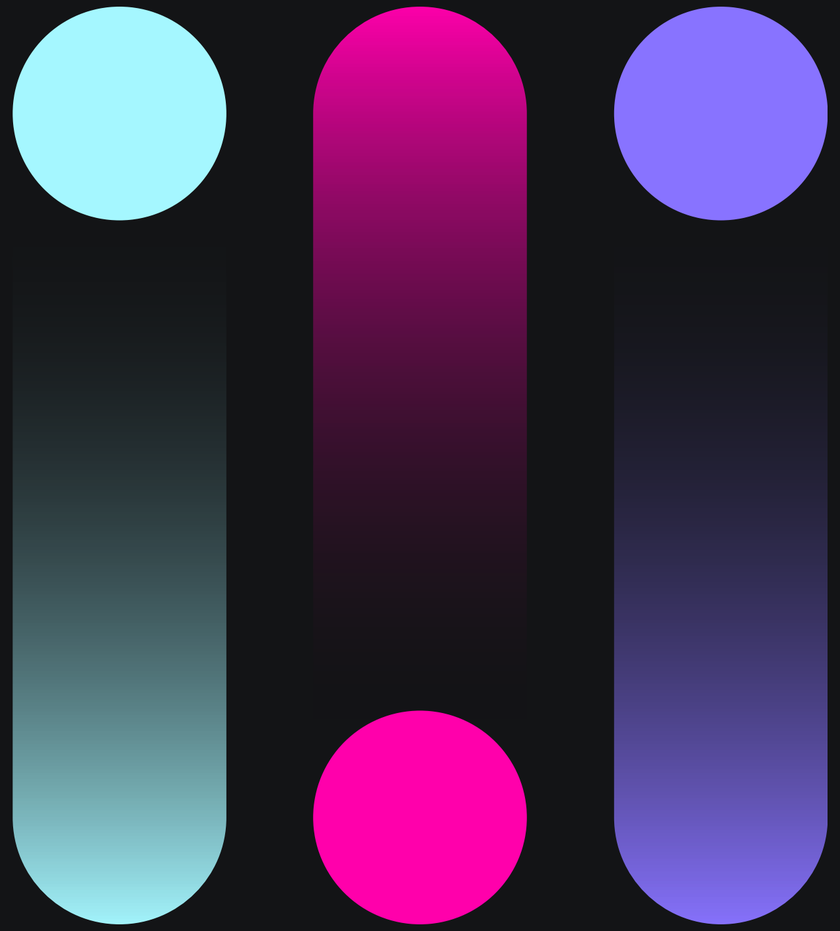
- model= Epoch 5/5
- 2128/2128 ━━━━━━━━━━━━━━━━━━━━ 21s 6ms/step - accuracy: 0.9663 - loss: 0.0858
- Accuracy: 0.9117612501468687

Multiple LSTM:

- model1= Epoch 5/5
- 2128/2128 ━━━━━━━━━━━━━━━━━━━━ 41s 12ms/step - accuracy: 0.9678 - loss: 0.0819
- Accuracy: 0.9148161203148866

Bidirectional LSTM:

- model2= Epoch 5/5
- 681/681 ━━━━━━━━━━━━━━━━━━━━ 23s 24ms/step - accuracy: 0.9611 - loss: 0.1000
- Accuracy: 0.910351310069322



Thank You