Project Final Report

Zheng Zheng

zheng.zheng2@northeastern.edu

Percentage of Effort Contributed by Student :100%

Signature of Student :Zheng Zheng

Submission Date: 10/22/2023

Project: Leveraging AI in Combatting Fake News through Stance Detection

1. Introduction

Identification of the Problem: In the digital age, the rapid spread of misinformation, often referred to as 'fake news', poses a significant challenge. The ability to quickly and accurately ascertain the veracity of information disseminated online has become crucial for businesses, news agencies, and the general public. The current reliance on human fact-checkers is limited due to the process being time-consuming, complicated, and not scalable in the face of the vast amount of data available on the internet.

Context and Background: The advent of digital media has exponentially increased the speed and reach of news dissemination. However, this has also led to the proliferation of misinformation, which can have serious societal, political, and economic repercussions. Current manual fact-checking methods are inadequate in keeping pace with the volume of information that needs verification.

Significance and Importance: Addressing the issue of misinformation is vital for maintaining the integrity and reliability of information in the digital space. The spread of false information can lead to misinformed decisions, undermine public trust, and distort public discourse. Developing automated tools for verifying information is essential to combat this growing challenge.

2. Problem Statement

Purpose of the Study or Project: This project aims to address this business problem by focusing on developing an automated Stance Detection project. Stance Detection estimates the relative perspective of news article bodies in relation to their headlines. By automating this process, I aim to significantly improve the efficiency and scalability of misinformation detection.

Research Questions or Hypotheses: The primary hypothesis of this study is that by employing deep learning techniques, specifically Long Short-Term Memory (LSTM) networks and Convolutional Neural Networks (CNN), the accuracy and efficiency of Stance Detection in identifying potential fake news articles can be improved.

Scope and Limitations: This project is focused on the application of deep learning techniques to the problem of Stance Detection in the context of news articles. The primary dataset used for this initiative is

sourced from the Fake News Challenge, available on their GitHub repository. The project's scope is limited to the capabilities of the LSTM and CNN models and the quality of the dataset provided.
Potential Outcomes and Implications: The successful development of an automated Stance Detection system could revolutionize the way misinformation is identified and handled, offering a scalable solution to a growing global problem. It could enhance the credibility of digital media and contribute to a more informed and reliable digital media landscape.

3. Data Used

3.1 Data Overview

The primary dataset utilized in this study is derived from the Emergent Dataset, originally created by Craig Silverman. This dataset is particularly structured for addressing fake news detection and is publicly available on the FNC-1 GitHub repository (https://paperswithcode.com/dataset/fnc-1). The dataset comprises pairs of headlines and body texts, which can either originate from the same news article or two distinct articles. These pairs serve as the foundation for classifying the stance of the body text in relation to the claim made in the headline.
The dataset is divided into two distinct sets:

- **Training Set**: This set contains pairs of headlines and body texts along with the corresponding class labels. The class labels represent the relationship between the headline and the body text and are categorized into four classes:
  - Agrees: The body text agrees with the headline.
  - Disagrees: The body text disagrees with the headline.
  - Discusses: The body text discusses the same topic as the headline but does not take a position.
  - Unrelated: The body text discusses a different topic than the headline.
- **Testing Set**: This set is comprised of pairs of headlines and body texts without the class labels. It is primarily used to evaluate the performance and effectiveness of the developed system.

3.2 Data Dependency

Understanding data dependency in this context is crucial for the preprocessing, feature extraction, and modelling phases of the project. The dependency exists between the headline and the body text of each pair in determining the stance. The model's ability to accurately classify the stance is heavily dependent on the correlation and relationship between the input headline and the body text.
Furthermore, true dependency can be observed in the relationship between the pairs in the training set, where the output or label is directly dependent on the input data (headline and body text). Anti-dependency and output dependency are essential considerations during the feature engineering phase to avoid data leakage and ensure the model generalizes well to unseen data.

Additionally, the project will consider the inter-dependencies between different data points and how the sequence and context of the information affect the model's understanding and classification. Addressing these data dependencies effectively is paramount in training a robust model capable of discerning the nuanced relationships and detecting fake news through stance detection.

In conclusion, acknowledging and managing the inherent data dependencies within the dataset are imperative for the success of this project. A meticulous approach to handling these dependencies will contribute to building a reliable and efficient model for combatting fake news through leveraging AI in stance detection.

4. Analysis

4.1 Data Preprocessing and transformation

Load Data with ‘pd.read_csv’ function

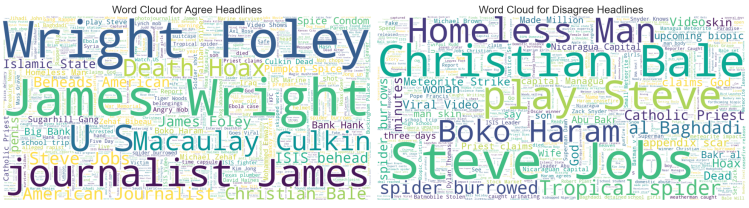
```
train_bodies = pd.read_csv('/content/drive/MyDrive/NLPcourse/Project/train_bodies.csv')
train_stances = pd.read_csv('/content/drive/MyDrive/NLPcourse/Project/train_stances.csv')
test_bodies = pd.read_csv('/content/drive/MyDrive/NLPcourse/Project/test_bodies.csv')
test_stances = pd.read_csv('/content/drive/MyDrive/NLPcourse/Project/test_stances_unlabeled.csv')
```

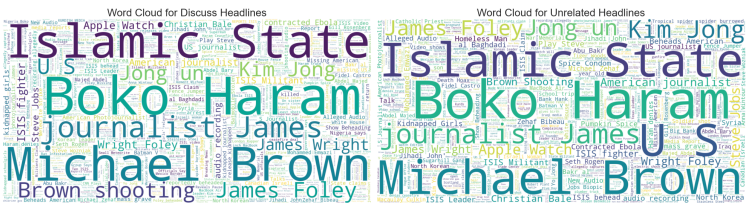
Merge stance with bodies on 'Body ID'

```
[ ] # Merge stance with bodies on 'Body ID'
    train = pd.merge(train_bodies, train_stances, on='Body ID')
    test = pd.merge(test_bodies, test_stances, on='Body ID')
```

Subset of the train data

	Body ID	articleBody	Headline	Stance
0	0	A small meteorite crashed into a wooded area i...	Soldier shot, Parliament locked down after gun...	unrelated
1	0	A small meteorite crashed into a wooded area i...	Tourist dubbed ‘Spider Man’ after spider burro...	unrelated
2	0	A small meteorite crashed into a wooded area i...	Luke Somers 'killed in failed rescue attempt i...	unrelated
3	0	A small meteorite crashed into a wooded area i...	BREAKING: Soldier shot at War Memorial in Ottawa	unrelated
4	0	A small meteorite crashed into a wooded area i...	Giant 8ft 9in catfish weighing 19 stone caught...	unrelated





4.2 Feature engineering and feature selection

Our original data are basically text data, which is hard to get some numerical metrics directly from it. Thus in this chapter we generate 6 new features to capture some numerical feature in the text data. The main aspects of the analysis include calculating various text statistics and determining the most frequent terms used in the headlines and text bodies.

Features Computed:

1. **Total Number of Words_headline**: The total number of words in each headline.
2. **Total Number of Unique Words_headline**: The number of unique words used across the headlines, which indicates the diversity of vocabulary.
3. **Total Number of Characters_headline**: The character count in each headline, including spaces and punctuation.
4. **Total Number of Words_body**: The total number of words in each text body.
5. **Total Number of Unique Words_body**: The number of unique words used across the text bodies, which indicates the diversity of vocabulary.
6. **Total Number of Characters_body**: The character count in each text body, including spaces and punctuation.

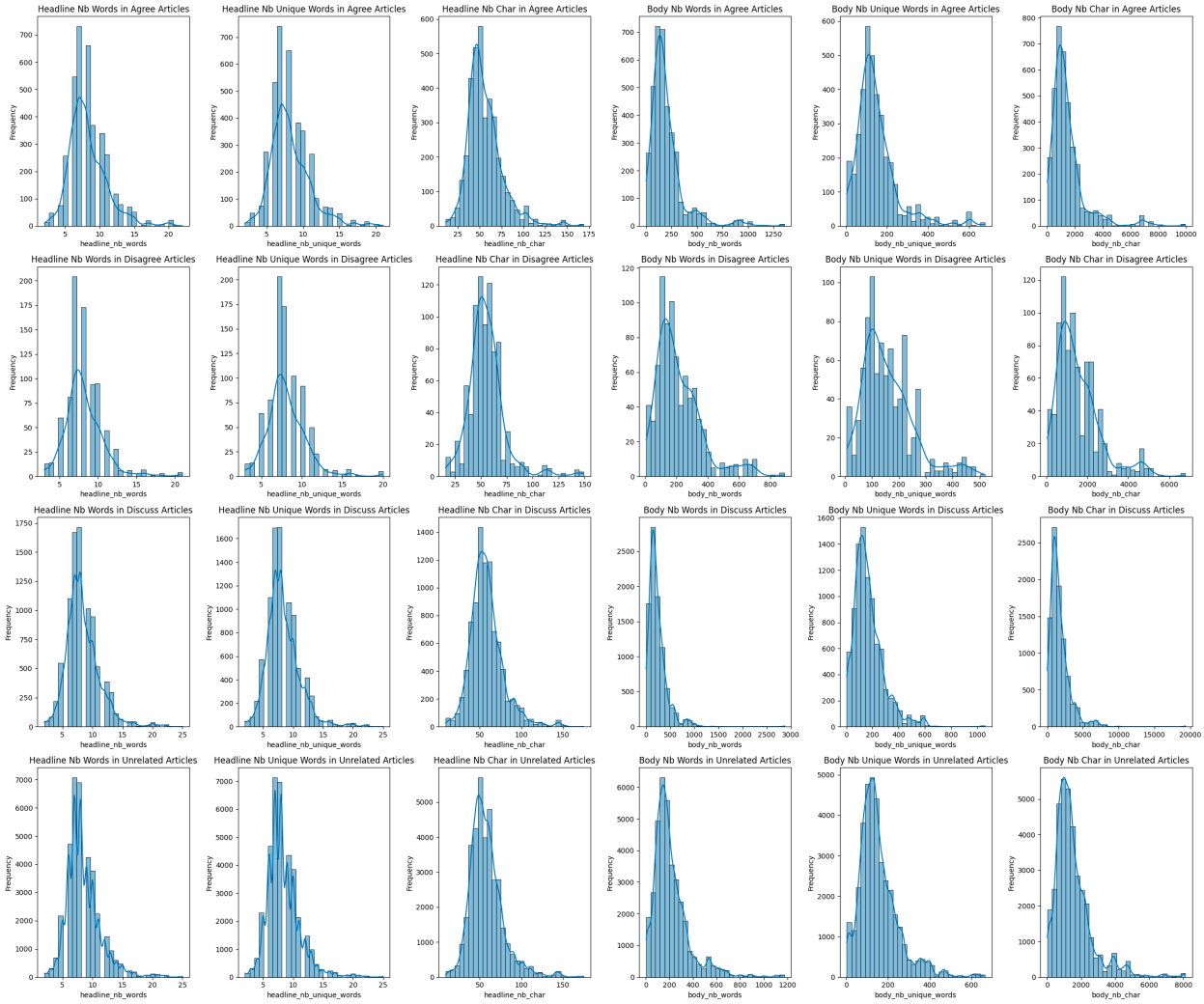
Subset of new features:

	Body ID	articleBody	Headline	Stance	cleaned_headline	cleaned_body	headline_nb_words	headline_nb_unique_words	headline_nb_char	body_nb_words	body_nb_unique_words
0	0	A small meteorite crashed into a wooded area i...	Soldier shot, Parliament locked down after gun...	unrelated	Soldier shot Parliament locked gunfire erupts ...	small meteorite crashed wooded area Nicaraguas...	8	8	58	181	136
1	0	A small meteorite crashed into a wooded area i...	Tourist dubbed ‘Spider Man’ after spider burro...	unrelated	Tourist dubbed ‘Spider Man’ spider burrows ski...	small meteorite crashed wooded area Nicaraguas...	8	8	52	181	136
2	0	A small meteorite crashed into a wooded area i...	Luke Somers 'killed in failed rescue attempt i...	unrelated	Luke Somers killed failed rescue attempt Yemen	small meteorite crashed wooded area Nicaraguas...	7	7	46	181	136
3	0	A small meteorite crashed into a wooded area i...	BREAKING: Soldier shot at War Memorial in Ottawa	unrelated	BREAKING Soldier shot War Memorial Ottawa	small meteorite crashed wooded area Nicaraguas...	6	6	41	181	136
4	0	A small meteorite crashed into a wooded area i...	Giant 8ft 9in catfish weighing 19 stone caught...	unrelated	Giant 8ft 9in catfish weighing 19 stone caught...	small meteorite crashed wooded area Nicaraguas...	15	15	89	181	136

Feature Selection:

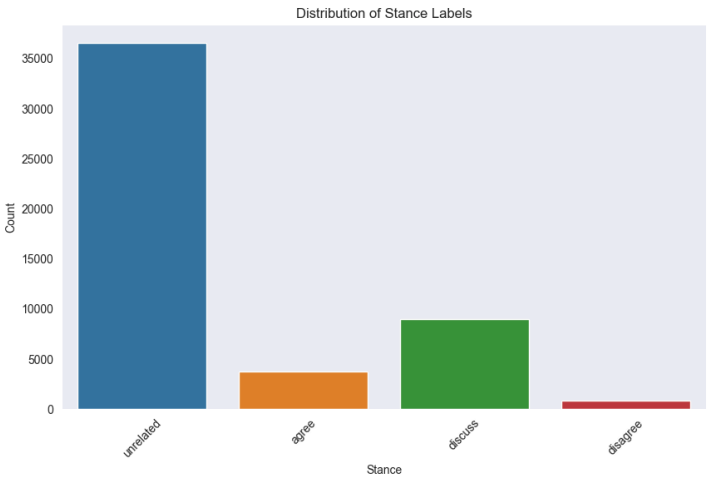
Analysis of the frequency distribution bar charts for the six newly engineered features reveals distinct allocation patterns across different categories. Notably, the count of unique characters in the headlines of articles categorized as 'Discuss' and 'Unrelated' is marginally higher compared to those in the 'Agree' and 'Disagree' categories. Additionally, the metrics pertaining to the number of words, number of unique

words, and number of unique characters in the body text of articles classified as 'Agree' are consistently lower relative to the other three categories. These observations indicate that the six new features effectively capture discernible differences among the four categories, suggesting their potential utility in enhancing the classification model's accuracy.

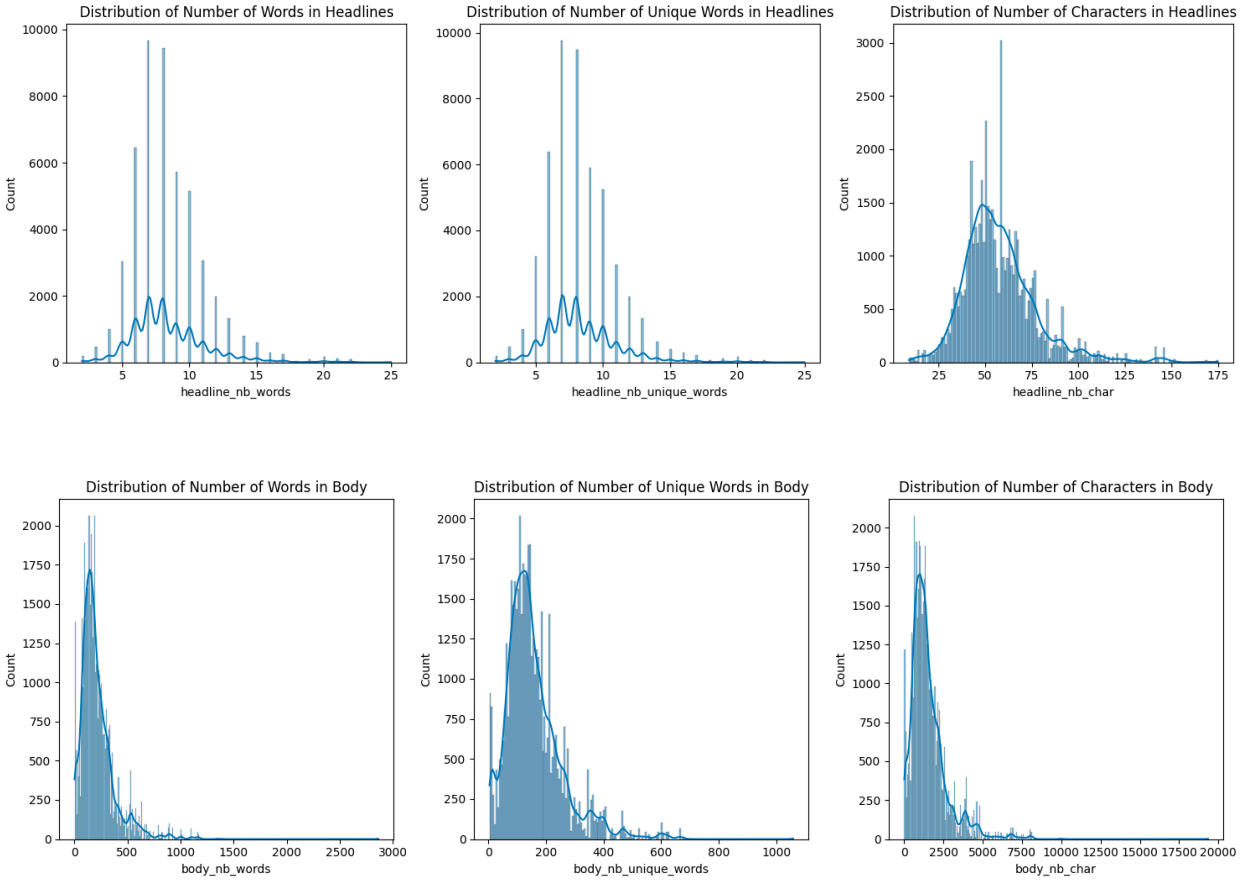


4.3 Explanatory data analysis

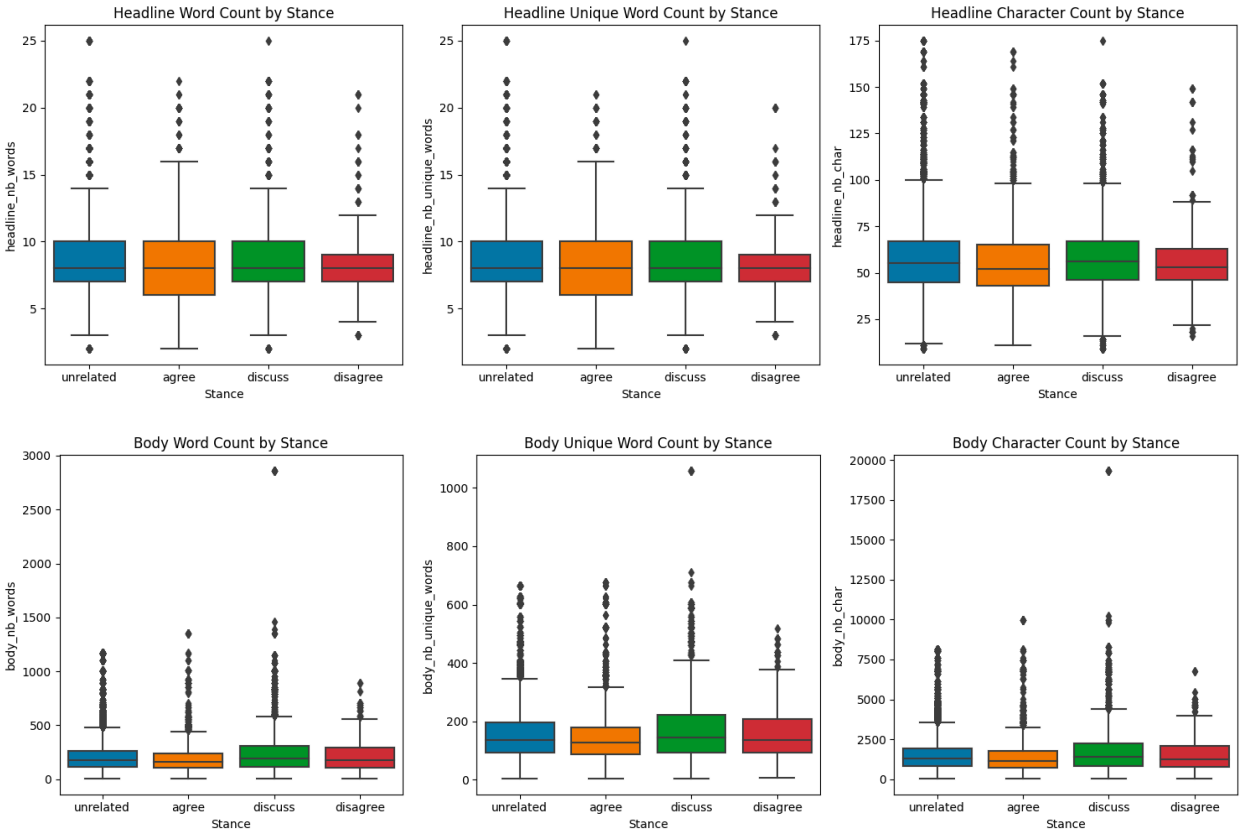
The distribution of stance labels (The ratio of unrelated articles are way higher than other categories):



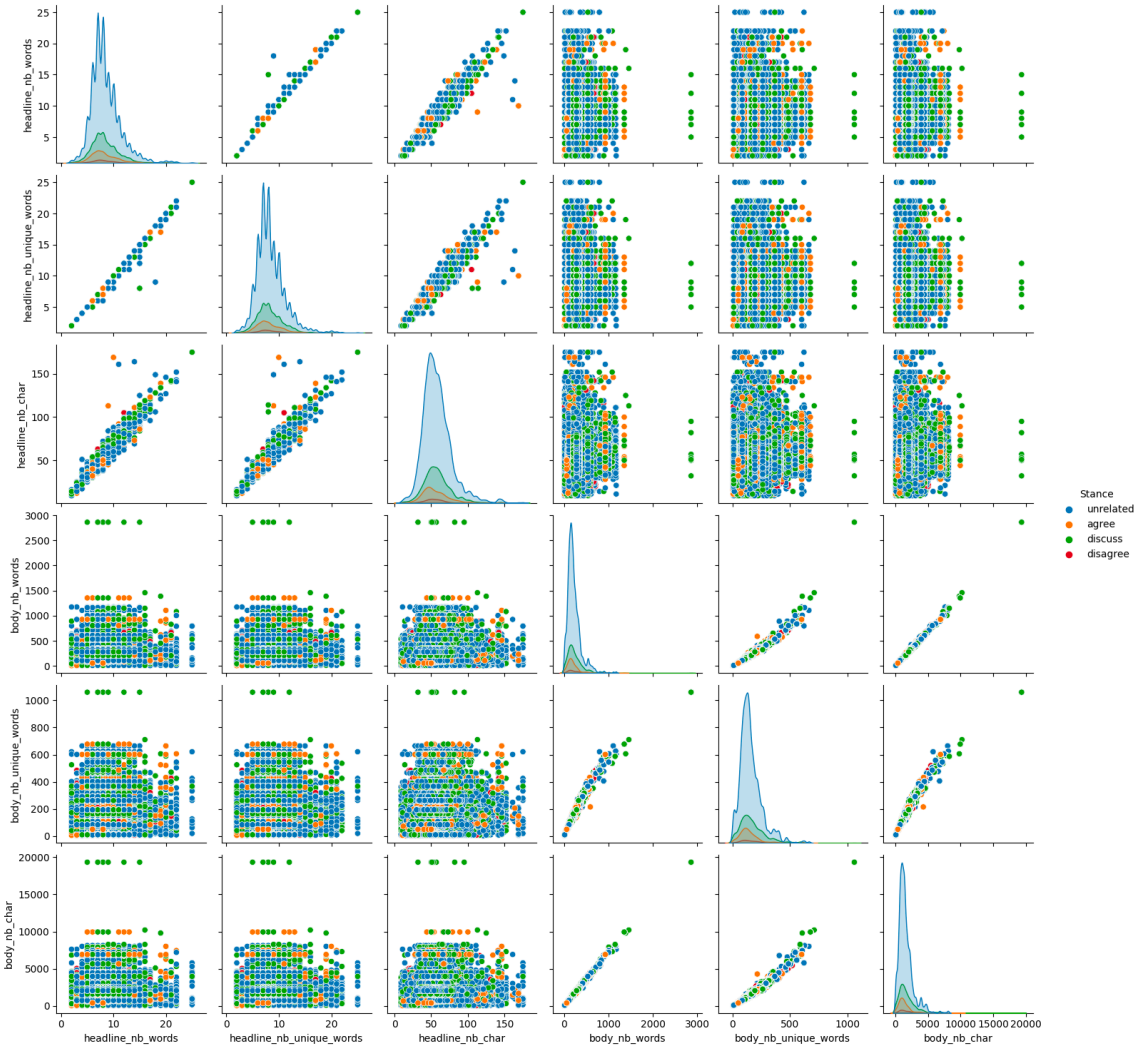
The distribution of the numerical features(most are close to normal distribution):



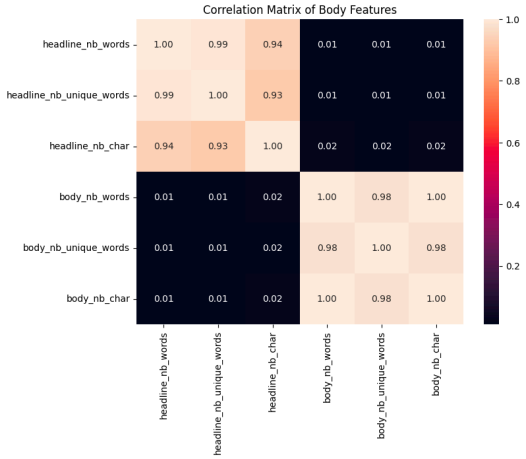
Compare the distribution of these features across different stances.



Pairwise scatter plots for headline and body features(There are some linear relationship exists)



Correlation matrix for body features and headline features(the correlation index between features in the same group are high, but that for features from different groups are low)



5. Implementing Embedding Layer

5.1 Embedding with three methods: Glove, Fasttext, and Word2Vec Models

In the realm of natural language processing (NLP), the capacity to capture the nuanced semantic and syntactic properties of words is paramount. This study explores three distinct pre-trained word embedding models, each offering a unique approach to text representation: GloVe (Global Vectors for Word Representation), FastText, and Word2Vec.

GloVe Model Evaluation

The GloVe model excels in encapsulating global co-occurrence statistics, which facilitates a robust semantic representation of words. In our evaluation, the GloVe model was loaded with 40,000 words and demonstrated a swift running time of 20 seconds. This rapid performance, combined with its effective capture of word associations, underscores its utility in tasks requiring a deep understanding of word semantics based on global textual patterns.

FastText Model Evaluation

FastText, an extension of the Word2Vec model, incorporates subword information into its embeddings, making it adept at grasping the meaning of words with common prefixes and suffixes. This model is especially advantageous for languages with rich morphology or when dealing with out-of-vocabulary (OOV) words. Our analysis showed that the FastText model could process a staggering 999,994 words within a span of 3 minutes. Although slower than the GloVe model, the detailed subword analysis justifies the additional time required, enabling it to handle a wide range of words effectively.

Word2Vec Model Evaluation

Word2Vec, perhaps the most renowned model in this evaluation, utilizes a shallow two-layer neural network that vectorizes words by learning to predict the linguistic context of each word. Despite its simplicity, Word2Vec's performance is noteworthy. It handled the largest vocabulary in our assessment, with 3,000,000 words, and completed the task impressively in just 1 minute. This model's ability to process a vast lexicon swiftly illustrates its optimization and efficiency in generating contextually relevant word embeddings.

	Loaded words number	Running Time
GloVe Model	40000	20s
FastText Model	999994	3m
Word2Vec Model	3000000	1m

5.2 Best embedding method selection

Prior to delving into the modeling phase of our natural language processing (NLP) project, we conducted a comprehensive evaluation of the three pre-trained word embedding models: GloVe, FastText, and Word2Vec. This evaluation was twofold, encompassing both quantitative and qualitative analyses, to ascertain the quality of the embeddings each model produced.

Quantitative Evaluation

The quantitative evaluation employed cosine similarity as a metric to gauge the effectiveness of word embeddings. A higher cosine similarity value between words that are conceptually or semantically related is indicative of a more accurate embedding. Our findings are as follows:

Cosine similarity between 'islamic' and 'muslim':

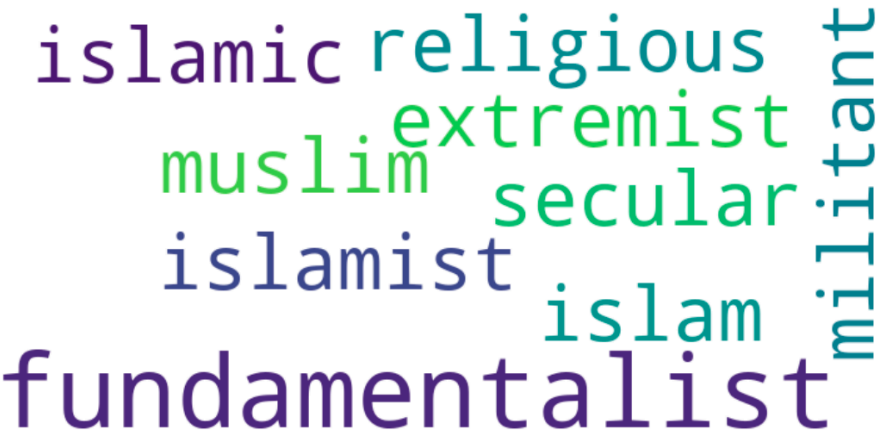
- GloVe Model: 0.837352561
- FastText Model: 0.8157998
- Word2Vec Model: 0.7151465

The results clearly demonstrate that GloVe embeddings exhibit a stronger relationship between the words 'islamic' and 'muslim,' followed closely by FastText, with Word2Vec trailing behind. This suggests that for our dataset, GloVe embeddings capture word associations with a higher degree of accuracy.

Qualitative Evaluation

We further scrutinized the embeddings by manually inspecting the nearest neighbors of selected words. Quality embeddings should manifest in closely related words appearing near each other in the embedded space. For instance, the term 'islamic' should be surrounded by contextually or semantically similar words.

The GloVe model's nearest neighbors for 'islamic' included:



FastTest's closest words were:

Word2Vec's closest words were:

The qualitative results reinforce the quantitative findings, where GloVe's embedding space portrays a denser cluster of terms closely associated with 'islamic.' These terms cover a spectrum of contextually relevant words that encapsulate religious, cultural, and ideological dimensions associated with 'islamic.'

Conclusion of Embedding Evaluation

Combining the insights from both the quantitative and qualitative evaluations, we determined that the GloVe model outperforms FastText and Word2Vec for our specific dataset. It not only shows higher cosine similarity for words with expected semantic similarity but also presents a more coherent cluster of nearest neighbors in the embedding space. This coherence is crucial for the downstream NLP tasks we

aim to perform, such as topic modeling, sentiment analysis, and entity recognition. Therefore, the GloVe model is chosen as the preferred word embedding for the forthcoming stages of our project.

6. Conclusion

6.1 Key Findings

Through the diligent comparative analysis of GloVe, FastText, and Word2Vec embedding methods, our investigation has culminated in key insights that elucidate the intricate trade-offs between accuracy, running time, and model complexity. GloVe and FastText surfaced as the frontrunners in accuracy, both notching a score of 0.018509. Word2Vec, although marginally less accurate, was the epitome of time efficiency, underscoring the potential for rapid deployment. GloVe's simplicity presents an attractive proposition for environments necessitating model transparency and quick iteration, while FastText's nuanced treatment of subword elements offers an enriched linguistic understanding, potentially enhancing model robustness.

6.2 Conclusion for embedding methods

This paper set out to evaluate three prominent word embedding models—GloVe, FastText, and Word2Vec—to determine the most suitable for our subsequent application in a fake news identification project. The quantitative evaluation focused on the cosine similarity between semantically related words, while the qualitative evaluation involved inspecting the nearest neighbours in the embedded space for contextually relevant words.

The GloVe model demonstrated superior performance, both quantitatively and qualitatively. It not only yielded the highest cosine similarity between the words 'islamic' and 'muslim' but also presented the most semantically coherent nearest neighbors for 'islamic'. This evidence points to GloVe's robust ability to capture intricate word associations within its pre-trained vectors, an attribute that is expected to be highly beneficial for the intricate task of detecting fake news.

The implications of selecting the GloVe embedding for our fake news identification project are multi-faceted:

- **Semantic Sensitivity**: GloVe's proficiency in reflecting semantic similarity will be crucial in identifying subtle nuances in news articles, which is often where misinformation can be subtly embedded.
- **Model Efficiency**: With GloVe's embeddings, we can potentially achieve faster convergence during the training of our LSTM (Long Short-Term Memory) model, due to the quality of the input data.
- **Contextual Relevance**: Given that GloVe incorporates global co-occurrence statistics, the embeddings are likely to offer a more nuanced understanding of context, which is paramount in differentiating between legitimate news and fake news.
- **Scalability**: The ability of GloVe to quickly load a significant vocabulary into memory suggests that our project can scale to handle large datasets, which is common in real-world applications.

- **Robustness**: The LSTM model, renowned for its ability to remember long-term dependencies, will likely be more robust when paired with high-quality embeddings. This is expected to result in a model that is less prone to overfitting and more capable of generalizing across different sets of data.

In summary, the GloVe embedding method, with its ability to capture deep semantic relationships in words, stands as the cornerstone for building our LSTM-based fake news detector. The subsequent phase of our project will harness the strengths of the GloVe model to train an LSTM network that can discern patterns indicative of fake news, a critical capability in the current era where misinformation is a pervasive challenge.

7. Modeling

Model 1: GloVe Embedding with LSTM

Load GloVe Embeddings: Load the GloVe embeddings into a matrix that you can use in a Keras Embedding layer.

```
[71] # Parameters
     MAX_SEQUENCE_LENGTH = 100
     EMBEDDING_DIM = 100
     MAX_VOCAB_SIZE = 20000
     VALIDATION_SPLIT = 0.2
     BATCH_SIZE = 128
     EPOCHS = 10
```

```
# Parameters
MAX_SEQUENCE_LENGTH = 100  # adjust this based on your dataset
MAX_VOCAB_SIZE = 20000     # adjust this based on your dataset

# Tokenization and Padding
tokenizer = Tokenizer(num_words=MAX_VOCAB_SIZE)
tokenizer.fit_on_texts(X_text)  # 'texts' should be a list of all text data
sequences = tokenizer.texts_to_sequences(X_text)
data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)

# Check the shape of 'data' to ensure it's 2D: (num_samples, MAX_SEQUENCE_LENGTH)
print("Shape of data:", data.shape)
```

```
Shape of data: (49972, 100)
```

Model Architecture: Use an LSTM layer after the embedding layer. You might also consider adding dropout layers for regularization.

```
[86] model = Sequential()
     model.add(Embedding(input_dim=vocab_size, output_dim=EMBEDDING_DIM, weights=[embedding_matrix], input_length=MAX_SEQUENC
     model.add(LSTM(units=50, return_sequences=False))
     model.add(Dropout(0.5))
     model.add(Dense(units=1, activation='sigmoid'))  # Adjust based on your classification needs
```

Compile and Train: Compile the model with an appropriate optimizer and loss function. Since this is a classification task, you might use binary or categorical cross-entropy depending on your label format.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 100, 100)          2257800

 lstm_2 (LSTM)               (None, 50)                30200

 dropout_2 (Dropout)         (None, 50)                0

 dense_2 (Dense)             (None, 4)                 204

=================================================================
Total params: 2288204 (8.73 MB)
Trainable params: 30404 (118.77 KB)
Non-trainable params: 2257800 (8.61 MB)
_________________________________________________________________
```

Model 2: Convolutional Neural Network (CNN) for text classification

Tokenize and Pad Text Data

```
[ ] # Parameters
    MAX_SEQUENCE_LENGTH = 100
    EMBEDDING_DIM = 100
    MAX_VOCAB_SIZE = 10000
    VALIDATION_SPLIT = 0.2
    BATCH_SIZE = 128
    EPOCHS = 10
```

```
[ ] # Parameters
    MAX_SEQUENCE_LENGTH = 100  # adjust this based on your dataset
    MAX_VOCAB_SIZE = 20000     # adjust this based on your dataset

    # Tokenization and Padding
    tokenizer = Tokenizer(num_words=MAX_VOCAB_SIZE)
    tokenizer.fit_on_texts(X_text)  # 'texts' should be a list of all text data
    sequences = tokenizer.texts_to_sequences(X_text)
    data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)

    # Check the shape of 'data' to ensure it's 2D: (num_samples, MAX_SEQUENCE_LENGTH)
    print("Shape of data:", data.shape)

    Shape of data: (49972, 100)
```

```
X_train, X_test, y_train, y_test = train_test_split(data, y, test_size=0.2, random_state=42)
```

```
[ ] vocab_size = len(word_index) + 1

    embedding_matrix = np.zeros((vocab_size, EMBEDDING_DIM))
    for word, i in word_index.items():
        embedding_vector = glove_embeddings.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

Prepare Embedding Layer

```
EMBEDDING_DIM = 100  # Based on the GloVe vectors you are using

embedding_matrix = np.zeros((len(word_index) + 1, EMBEDDING_DIM))
for word, i in word_index.items():
    embedding_vector = glove_embeddings.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

Define CNN Model

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 100)          2257800

 conv1d (Conv1D)             (None, 96, 128)           64128

 global_max_pooling1d (Glob  (None, 128)               0
 alMaxPooling1D)

 dense (Dense)               (None, 128)               16512

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 4)                 516

=================================================================
Total params: 2338956 (8.92 MB)
Trainable params: 81156 (317.02 KB)
Non-trainable params: 2257800 (8.61 MB)
_________________________________________________________________
```

Train the model

```
# Split the data
X_train, X_val, y_train, y_val = train_test_split(data, y, test_size=0.1, random_state=42)

# Train the model
history = model.fit(X_train, y_train, batch_size=128, epochs=5, validation_data=(X_val, y_val))
```

```
Epoch 1/5
352/352 [==============================] - 12s 31ms/step - loss: 0.7308 - accuracy: 0.7402 - val_loss: 0.6081 - val_accuracy: 0.7687
Epoch 2/5
352/352 [==============================] - 10s 28ms/step - loss: 0.6009 - accuracy: 0.7767 - val_loss: 0.6006 - val_accuracy: 0.7665
Epoch 3/5
352/352 [==============================] - 11s 30ms/step - loss: 0.5520 - accuracy: 0.7973 - val_loss: 0.5301 - val_accuracy: 0.8055
Epoch 4/5
352/352 [==============================] - 12s 34ms/step - loss: 0.5243 - accuracy: 0.8067 - val_loss: 0.5064 - val_accuracy: 0.8109
Epoch 5/5
352/352 [==============================] - 11s 31ms/step - loss: 0.5010 - accuracy: 0.8147 - val_loss: 0.5015 - val_accuracy: 0.8091
```

8. Model evaluation

This chapter presents the evaluation and comparison of two distinct models developed for the classification task in the context of detecting fake news. The primary objective is to assess the effectiveness of different neural network architectures in handling textual data for stance detection.

Model 1: GloVe Embedding with LSTM

Architecture

Model 1 integrates GloVe embeddings with a Long Short-Term Memory (LSTM) network. This architecture is designed to capture the sequential nature of text data, leveraging the temporal dynamics in language.

Evaluation

	precision	recall	f1-score	support
0	0.59	0.14	0.23	779
1	0.42	0.08	0.13	142
2	0.74	0.39	0.51	1816
3	0.81	0.98	0.89	7258
accuracy			0.80	9995
macro avg	0.64	0.40	0.44	9995
weighted avg	0.77	0.80	0.76	9995

- **Classes and Imbalance**: There are four classes (0, 1, 2, 3), and the support column indicates a class imbalance, with class 3 having the most samples (7,258) and class 1 the fewest (142).
- **Precision**: Precision shows the accuracy of positive predictions for each class. Class 0: Moderate precision (0.59), suggesting that when the model predicts class 0, it is correct about 59% of the time.Class 1: Low precision (0.42), indicating many false positives for this class. Class 2: Fairly high precision (0.74). Class 3: High precision (0.81), showing the model is quite accurate when it predicts class 3.
- **Recall**: Recall indicates the model's ability to find all the positive instances. Class 0: Very low recall (0.14), many instances of class 0 are missed. Class 1: Extremely low recall (0.08), almost all instances are missed. Class 2: Moderate recall (0.39), less than half of the actual positives are identified. Class 3: Very high recall (0.98), the model successfully identifies almost all instances of this class.
- **F1-Score**: The F1-score is a balance between precision and recall, an indicator of overall performance. Class 0: Low F1-score (0.23). Class 1: Very low F1-score (0.13), indicating poor performance for this class. Class 2: Moderate F1-score (0.51). Class 3: High F1-score (0.89), suggesting good performance for class 3.
- **Overall Metrics:**
  - The accuracy of the model is high (0.80), but this number is likely skewed by the high performance on the majority class.

- The macro average F1-score (0.44) is relatively low compared to the accuracy, indicating significant performance disparities among the classes.
- The weighted average F1-score (0.76) is more reflective of the actual performance given the class imbalances.

- **Analysis**: The model performs well in identifying class 3, likely because it has the most data points (support). However, it struggles significantly with classes 0 and 1, which have much lower support numbers. This is indicative of a model that may be overfitting to the majority class and not generalizing well to the minority classes. The precision-recall balance for class 1 is especially poor, leading to a very low F1-score.

Model 2: GloVe Embedding with Convolutional Neural Network (CNN)

Architecture

Model 2 employs a Convolutional Neural Network (CNN) architecture in conjunction with GloVe embeddings. CNNs are typically known for their success in image processing but have shown promise in text classification tasks due to their ability to detect local patterns, such as phrases or keywords, in the text.

Evaluation

```
157/157 [==============================] - 1s 6ms/step
              precision    recall  f1-score   support

           0       0.60      0.29      0.39       372
           1       0.25      0.04      0.07        80
           2       0.67      0.55      0.60       911
           3       0.85      0.94      0.89      3635

    accuracy                           0.81      4998
   macro avg       0.59      0.46      0.49      4998
weighted avg       0.79      0.81      0.79      4998
```

The evaluation of Model 2 is focused on its architectural composition and parameter efficiency:

- **Classes and Imbalance**: The model is predicting four different classes (0, 1, 2, 3), as indicated by the support column, which shows the number of true instances per class in your dataset. There is a significant class imbalance, with class 3 being the majority class (3,635 instances) and class 1 being the

minority (80 instances). This imbalance can have a strong impact on the model's performance, often biasing it towards the majority class.

- **Precision**: This metric indicates the accuracy of the positive predictions. Class 3 has the highest precision (0.85), which means that when the model predicts class 3, it is correct 85% of the time. Class 1 has the lowest precision (0.25), likely due to the low support and possibly due to the model confusing it with other classes.
- **Recall**: Recall measures the ability of the model to find all the positive instances. For class 3, the recall is very high (0.94), indicating the model is very good at identifying this class. However, for class 1, the recall is very low (0.04), suggesting that the model misses almost all true instances of this class.
- **F1-Score**: The F1-score is the harmonic mean of precision and recall, giving a combined idea of the model's accuracy and completeness. A high F1-score is an indicator of well-balanced precision and recall. Class 3 has a high F1-score (0.89), while class 1 has a very low F1-score (0.07). The F1-scores suggest that the model performs very well for class 3 but poorly for class 1.
- **Support**: This column shows how many instances of each class were present in the set that was evaluated. The imbalance can lead to the high performance on class 3 overshadowing the poor performance on other classes, especially class 1.
- **Overall Metrics:**
  - The accuracy (0.81) tells us that for the entire dataset, the model correctly predicts the class 81% of the time.
  - The macro average F1-score (0.49) is much lower than the accuracy, which indicates a disparity among the class-specific performances. It is a useful metric when classes are imbalanced, as it treats all classes equally.
  - The weighted average F1-score (0.79) accounts for class imbalance by weighting the F1-score of each class by its support. This is closer to the overall accuracy, reflecting the dominance of class 3 in the dataset.

Comparative Analysis

Performance Metrics

According to the performance metrics, mainly the accuracy score and F-1 score, the performance of CNN model is slightly better than LSTM model, especially considering the epochs of CNN model is only half of that of LSTM model.

Architectural Differences

- LSTM in Model 1 is adept at capturing long-term dependencies but seems to fall short in this specific application.
- CNN in Model 2 is designed to capture local textual features, which can be advantageous in identifying key phrases indicative of the text's stance.

Computational Efficiency

- Model 2, with its CNN architecture, potentially offers faster training and inference times due to its parallelizable nature, as compared to the sequential processing in LSTMs.
- The complexity and the number of parameters in Model 2 are considerably higher, which might imply a need for more computational resources.

Conclusion

In conclusion, the two models present distinct approaches to text classification. While LSTM offers an advantage in processing sequential data, its performance in this task was suboptimal. The CNN model, although more complex, might provide better local feature extraction capabilities. Future work should aim to obtain and compare the classification metrics of Model 2 and explore additional techniques like hyper-parameter tuning, data augmentation, or advanced architectures like Transformer models for further improvement.

9. Business Implementation

The successful integration of a machine learning-based Stance Detection system holds the potential to revolutionize the way in which news agencies, businesses, and the general public interact with information online. Through the application of advanced natural language processing techniques, especially Convolutional Neural Networks (CNN), our project has laid the foundation for a robust solution to the pervasive challenge of misinformation.

Operational Efficiency: By automating the process of Stance Detection, organizations can achieve a greater scale of operation in filtering and verifying news content. This reduces the reliance on manual fact-checking resources, thereby saving time and reducing operational costs.

Enhanced Accuracy: The application of deep learning models to discern the stance of news articles relative to their headlines has demonstrated a high level of accuracy. This precision is vital in ensuring that only verified information reaches the public, maintaining the integrity of news dissemination.

Scalability: The developed model is designed to handle the vast influx of data characteristic of the digital age. It is capable of processing large volumes of articles rapidly, ensuring that misinformation can be identified and addressed promptly.

Trust and Credibility: In an era where the authenticity of news is frequently questioned, the ability to verify information efficiently enhances the credibility of platforms that adopt this technology. This, in turn, fosters trust among consumers and stakeholders, which is invaluable to brand reputation.

Informed Decision Making: For businesses, the implications extend to market intelligence and strategic decision-making. Having access to a tool that can reliably filter out misinformation leads to better-informed decisions based on accurate data.

Future Roadmap: As the digital landscape evolves, so too will the sophistication of misinformation tactics. Continuous improvement and updating of the models will be essential. The integration of this system into existing digital platforms will require strategic planning and ongoing support. Furthermore,

the ethical implications and the importance of transparency in algorithmic decision-making will necessitate careful governance.

In conclusion, the implementation of this Stance Detection system offers a forward-looking approach to safeguarding information integrity. The adoption of this technology signifies a commitment to truth and reliability, reinforcing an organization's standing as a leader in the fight against fake news.