1.Introduction

In today’s world, the spread of the false information has become increasingly prevalent, especially with the deep integration of social media into our daily lives. By definition, ’’fake’’ news is as misinformation or fabricated contents presented as factual reporting. but it is often used as a tool to influence public opinion or promote specific agendas. Both official and unofficial sources may involve themselves in this practice, making it hard for the readers to distinguish between credible and misleading information.

With the help of social media and artificial intelligence, fake news sources can now be paired with convincing visuals and audio, supporting its believability. In addition to that, social media platforms amplify this problem, as false information can spread rapidly through communication channels, making it extremely difficult to prevent or control distribution.

2. Overview of the dataset and objective.

The dataset(train.csv)used is a subset of the well-known ’’LIAR2’’ dataset, which contains over 18,369 short political statements labeled on a six-point scale from "Pants on Fire" (completely false) to "True". For the purposes of this assignment, the problem has been reformulated into a binary classification task, where labels are grouped into two categories: ’’*fake*’’ and ’’*true(real)*’’.

The primary objective is to build and evaluate two classification models that can accurately flag potentially false content. This would support social media companies and fact-checkers in identifying and reducing the spread of misinformation at scale.

3.Data Preparation

a. Loading the data set

First,step was to load the dataset into Colab enviroment. Following this, the column headers were inspected and displayed to identify relevant features for encoding and potential feature reduction. This initial exploration helped in understanding the content of the dataset, which is essential for model development in further steps.

b.Using beautiful soup library to remove HTML entities

Beautiful soup library in python was used to get rid off html entities such as ampersand (&),quotation marks and greater sign(>).’’get_text’’ method was also used to get the plain text, which was refined.

c. Noise removal

```
re1 = re.sub(r"(@|http://|https://|www|\\x)\S*", " ", souped)
re2 = re.sub("[^A-Za-z]+", " ", re1)
```

re1:Removes user mentions such as signs like (@), and URLs, and hexadecimal characters (\x).(ref: https://docs.python.org/3/howto/regex.html)
re2: Removes all non-alphabetic characters (e.g., numbers, symbols), keeping only letters. (Python Software Foundation, n.d.).

d. Tokenization & lowercasing the tokens

```
tokens = nltk.word_tokenize(re2)
```

```
lower_case = [t.lower() for t in tokens]
```

During the tokenization process,both lowercase and uppercase versions of the words appear in vocabulary.To decrease redundancy and ensure consistency,all words are converted to lowercase.Otherwise,the words ''orange'' and ''Orange'' would be taken as different tokens,despite having the same meaning.Furthermore,having more tokens would unnecessarily expand vocabulary and could have a negative effect on the performance of the model.

e. Stopwords Removal

```
stop_words = set(stopwords.words('english'))
filtered_result = list(filter(lambda l: l not in stop_words, lower_case))
```

Removes common English stopwords (e.g., "the", "is", "in") that don't add meaningful context. According to Bird, Klein and Loper (2009), removing common English stop words such as *''the''*, *''is''*, and *''in''* also helps reduce noise in text data and focus on meaningful terms.

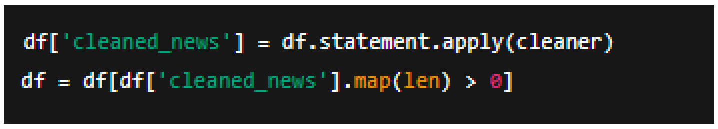
f. Lemmatization

```
wordnet_lemmatizer = WordNetLemmatizer()
lemmas = [wordnet_lemmatizer.lemmatize(t,'v') for t in filtered_result]
```

Lemmatization is implemented to reduce different grammatical forms of a word to a common base form which helps;

- reducing vocabulary size,
- improving model performance
- preserving meaning better than stemming
- enabling more accurate text matching in NLP tasks (spaCy, n.d.).

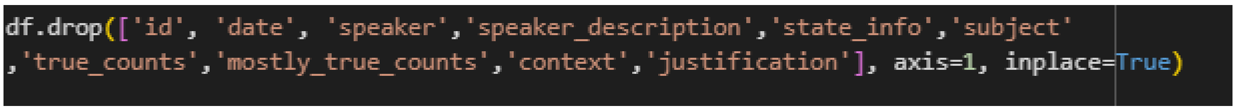
g. Applying the cleaner function the ‘’statement’’ column



```
df['cleaned_news'] = df.statement.apply(cleaner)
df = df[df['cleaned_news'].map(len) > 0]
```

Cleaner function is applied for the content in the statement column and removed the rows where the cleaned text is empty.

h. Feature Dropping



```
df.drop(['id', 'date', 'speaker','speaker_description','state_info','subject'
,'true_counts','mostly_true_counts','context','justification'], axis=1, inplace=True)
```

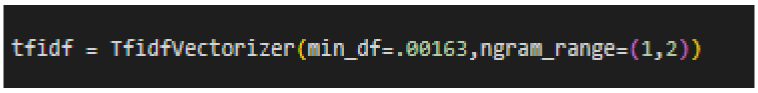
To decrease reduncdancy and to achieve better model performance,we drop the non-essential columns,keeping only the ‘’statement’’ column.

I. Binary Label Mapping

As we are trying to find ‘’false’’(fake news),ı will encode false class as 1 and true class 0.

Label	Meaning	Binary Label
0	Pants on Fire	1
1	False	1
2	Barely True	1
3	Half True	0
4	Mostly True	0
5	True	0

j. TF-IDF Vectorization



```
tfidf = TfidfVectorizer(min_df=.00163,ngram_range=(1,2))
```

The cleaned text was vectorized using **TF-IDF**. A min_df of **0.00163** kept only n-grams (unigrams, bigrams, trigrams)appearing in at least 30 documents. (documents30/18369rows=**0.00163**)

I selected an ngram_range of **(1, 2)** for the TfidfVectorizer because unigrams alone may miss important context in how fake and real news are expressed. By including bigrams, the model is better able to capture meaningful word combinations such as:

- “not true”
- “fake news”
- “highly misleading”
- “completely accurate”

For example, the phrase “not true” carries a very different meaning than the word “true” on its own. Including both unigrams and bigrams ensures that the model handle individual word frequencies and contextual patterns.

3. Data Preparation Limitations

1.Binary Mapping:The original dataset is labeled statements with six different levels of truthfullness.By using binary mapping,these nuanced categories were reduced to two,resulting in the loss of meaning for some labels, such as the difference between 'half-true', 'false', and 'pants-fire'."

2.TF-IDF Bag of Word Approach: This approach does not consider the semantic meaning,grammatical structure or word order of the context in which words appear.For example,’’Not true’’ and ‘’true’’ may be treated similarly without the acknowledging of their difference in meaning.

3.Label Imbalance: There is imbalance in the class,that may affects the model’s effectiveness. However,I will try to mitigate this limitation by applying the smote to our models.

4.Removing Non-Alphabetical characters: Inaccruacies in information may often come from these removed characters.For example, a casualty report might be flagged as ‘’false’’ simply because the number of casualties is reported incorrectly.(Instead of ‘1.000 people died’,it could be written as ‘100 people died’).Therefore,this discrepancies can neagtively impact the effectiveness of models as any news outlets frequently report information using numerical figures.

4.Model Creation and Evaluation

a. Model Creation

The followed Steps are;

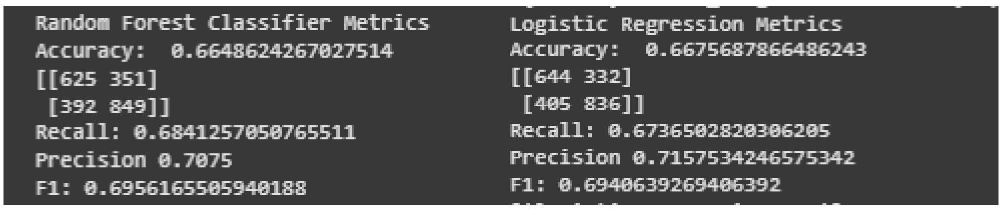
1.Loading the necessary libararies: Imported all required Python libraries for data manipulation, model building, and evaluation.

2.Train-test split: Divide the dataset into training and testing subsets to evaluate model performance on new data.
3.Balancing the labels: Applied SMOTE to mitigate the class imbalance
4.Define Random Forest Model: Initialize the Random Forest Classifier With specified hyperparameters.
5.Define parameter grid: Specify a range of hyperparameters to be tested during the tuning process.
6.Grid search with 5-fold cross-validation: Specify a range of hyperparameters to be tested during the tuning process.
7.Fit model: Train the Random Forest model on the training dataset using the selected hyperparameters
8.Best model: Identify the optimal Random Forest model based on cross-validation performance.
9.Predict and evaluation of test: Generate predictions for the test dataset to assess generalization performance.
10.Evaluation metrics: Calculate accuracy, precision, recall, F1-score, and other metrics to measure model effectiveness.
11.Saving the model: Saved the model for future predictions with new data

b.Model Evaluation

Logistic regression is a widely employed model in text classification due to its capability to handle high-dimensional sparse features. The model is computationally efficient, interpretable and effective when the decision boundaries are relatively linear, a condition that commonly occurs in text classification tasks. (Manning et al., 2008). In short, its robustness and simplicity make this algorithm a suitable and dependable choice for a wide range of natural language processing applications. (Jurafsky & Martin, 2023).

The Random Forest is capable of capturing complex and non-linear relationships within the data, making it well-suited for text classification tasks where the data is noisy (Breiman, 2001). It can be easily tuned which helps the interpretability. Random Forest Classifier is computationally more intensive than Logistic Regression, making it less business-friendly. (Zhang et al., 2019).



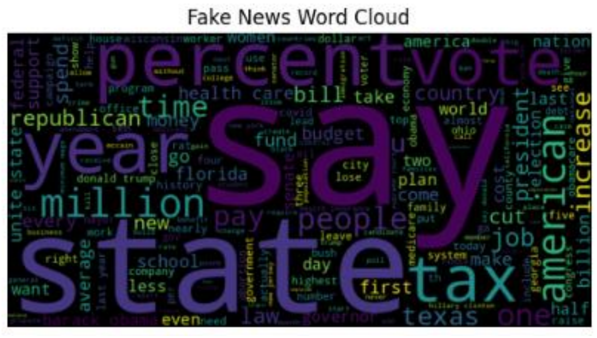
```
Random Forest Classifier Metrics      Logistic Regression Metrics
Accuracy:  0.6648624267027514         Accuracy:  0.6675687866486243
[[625 351]                            [[644 332]
 [392 849]]                            [405 836]]
Recall: 0.6841257050765511            Recall: 0.6736502820306205
Precision 0.7075                      Precision 0.7157534246575342
F1: 0.6956165505940188                F1: 0.6940639269406392
```

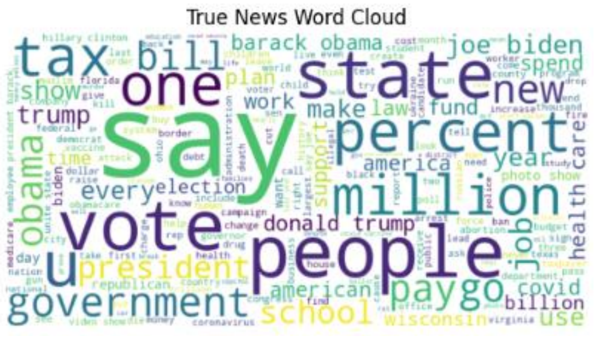
Overall Logistic Regression slightly outperforms Random Forest in accuracy (66.76% vs. 66.49%) and recall (0.6737 vs. 0.6841), making it more effective at correctly identifying true positives. Since recall is critical in fake news detection—where missing a fake news instance could have high consequences—Logistic Regression's higher recall makes it the preferable choice. However, Random Forest edges ahead in precision, meaning it is marginally better at avoiding false positives, though this is less important in this context.

In conclusion, the performance differences between the two models are minimal. However, given that the primary purpose of creating a text classification model is to maximize detecting fake news, meaning that achieving maximum recall is essential. Since Logistic Regression delivers superior recall, Logistic Regression would be the suitable model for real-life cases.

4.Recommendations&Conclusion

We are trying to identify the trutfulness of data by looking only at the textual data in the ’’statement’’ column.However,other factors such as speaker credibility,news source types... etc might be added to increase effectiveness of considered models. As shown on the world cloud below,same words are also associated with both the false and true classes.This proves that relying solely on textual input may not be enough for accurate classficiation.





As discussed above TF-IDF is not effective when understanding context, word order, or semantics.Therefore,deploying contextual embeddings such as Word2Vec, GloVe, or transformer-based embeddings may make the model capture nuanced meanings and contextual dependencies better. (Devlin et al., 2019).

In conclusion, the accuracy of both models is close to 0.70, which makes them unsuitable for real-world applications. At these performance levels, I would not recommend either model, as high risk of misclassification makes them impractical for effective use.

5.References

1.Bird, S., Klein, E. and Loper, E. (2009) *Natural Language Processing with Python*. Sebastopol, CA: O'Reilly Media.
2.Breiman, L. (2001) 'Random forests', *Machine Learning*, 45(1), pp. 5–32. https://doi.org/10.1023/A:1010933404324
3.Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K. (2019) *BERT: Pre-training of deep bidirectional transformers for language understanding*. Available at: https://arxiv.org/abs/1810.04805 (Accessed: 31 July 2025).
4.James, G., Witten, D., Hastie, T. and Tibshirani, R. (2013) *An Introduction to Statistical Learning: With Applications in R*. New York: Springer.
5.Jurafsky, D. and Martin, J.H. (2021) *Speech and Language Processing* (3rd ed.). Draft. Available at: https://web.stanford.edu/~jurafsky/slp3/
6.Jurafsky, D. and Martin, J.H. (2023) *Speech and Language Processing* (3rd ed.). Draft. Available at: https://web.stanford.edu/~jurafsky/slp3/
7.Liaw, A. and Wiener, M. (2002) 'Classification and regression by randomForest', *R News*, 2(3), pp. 18–22.
8.Manning, C.D., Raghavan, P. and Schütze, H. (2008) *Introduction to Information Retrieval*. Cambridge: Cambridge University Press.
9.Python Software Foundation (n.d.) *Regular Expression HOWTO*. Available at: https://docs.python.org/3/howto/regex.html (Accessed: 31 July 2025).
10.spaCy (n.d.) ***Lemmatization*.** Available at: https://spacy.io/usage/linguistic-features#lemmatization (Accessed: 31 July 2025).
11.Zhang, Y. (2017) 'Text classification algorithms: A survey', *IEEE Transactions on Knowledge and Data Engineering*, 29(2), pp. 1–17.
12.Zhang, Y., Jin, R. and Zhou, Z.-H. (2019) 'Understanding bag-of-words model: A statistical framework', *International Journal of Machine Learning and Cybernetics*, 10, pp. 1771–1783. https://doi.org/10.1007/s13042-018-0811-0
13.Zhou, Z.-H. (2012) *Ensemble Methods: Foundations and Algorithms*. Boca Raton, FL: CRC Press.