CS579 - Online Social Network Analysis
Project II - Fake News Classification
Report

Ganya Janardhan
A20517083

Joseph Rithvik Reddy Mettu
A20505019

Department of Computer Science
Illinois Institute of Technology
Chicago, IL - 60616

1. Introduction

In recent years, social media has become a major platform for obtaining news and information. However, the ease and speed of disseminating news through social media have led to the production of a large volume of fake news or misinformation. This phenomenon has serious negative consequences, including breaking the authenticity balance of the news ecosystem, intentionally persuading consumers to accept biased or false beliefs, and changing the way people interpret and respond to real news and information. As a result, it has become increasingly important to detect fake news and misinformation in social media. In this report, we aim to tackle this problem by proposing a machine learning-based approach that can classify news pairs as agreed, disagreed, or unrelated based on their titles. We use a dataset of news pairs with labeled relationships, train a classifier on this data, and evaluate its performance on a validation set. Finally, we use the trained classifier to predict the relationship labels for a test set, which can be used to identify and filter out fake news and misinformation in social media.

2. Problem Statement

A general fake new classification task can be broken down into making a decision whether an incoming news is real or not. Here we are given two news articles A and B, where A is a known fake news article and B is a coming news that we need to classify on. A and B will be compared based on this, we will classify B into the below mentioned categories.
Three labels are used for comparing the two articles A and B: agreed, disagreed, and unrelated.

- Agreed: B talks about the same fake news as A
- Disagreed: B refutes the fake news in A
- Unrelated: B is unrelated to A

3. Proposed Solution

We developed a machine learning model to classify news articles into three categories: agreed, disagreed, and unrelated. To create the model, we used natural language processing techniques and machine learning algorithms/neural networks. The following steps were taken:

1. Data preprocessing using NLP techniques
2. Transformation of data into bag of words and TF-IDF vectors
3. Model training using machine learning algorithms
4. Model testing to assess its accuracy and performance
5. Model evaluation using various metrics to gauge its effectiveness.
6. Predicting the labels for test data and attaching them to it.

This process enabled us to create a reliable and efficient model that can accurately classify news articles into the desired categories.

4. Data preprocessing using NLP techniques

Software Used:- R
Package Used:- tidyverse, tm

To prepare our train and test datasets for analysis, we applied NLP preprocessing techniques to extract only relevant information while making the datasets more manageable. Using the NLTK library, the following techniques were applied:

1) Converting strings to lowercase: This helped to interpret the text better, as lowercase and uppercase words are treated differently by machines.

2) Removal of stopwords: Words that are frequently used in language but not significant for text analysis (stopwords) such as "the," "an," "they," "there," were removed to focus on the more important words.

3) Removal of punctuation marks: Punctuation marks like commas, exclamation marks, and question marks were removed as they did not add any value to the text.

4) Removal of Null Values: Rows which contained empty or null values were removed.

5) Lemmatization: Words were brought to their root form to simplify the dataset, making it easier for the machine to identify relevant patterns and extract useful insights.

5. Bag of words and TF-IDF transformer

Software Used:- Python
Package Used:- Pandas, Sklearn(TfidfVectorizer)

We utilized two techniques to represent our textual data in a format that machine learning algorithms can process effectively: Bag of Words and TF-IDF.

Bag of Words: This technique generates a set of vectors that contain the frequency count of each word occurrence in the text/document. To implement this, we used the CountVectorizer function from the scikit-learn library.

TF-IDF: This approach determines the importance of a word in a document by considering its frequency and rarity across the entire corpus. We applied this method using the TfidfTransformer function from the scikit-learn library. This enabled us to obtain a more refined representation of our textual data and ensured that the most significant words were given more weight in our model's predictions.

$$TF(t,d) = \frac{number\ of\ times\ t\ appears\ in\ d}{total\ number\ of\ terms\ in\ d}$$

$$IDF(t) = log\frac{N}{1+df}$$

$$TF-IDF(t,d) = TF(t,d) * IDF(t)$$

Figure 1: Tf-idf formula

6. Model training using machine learning algorithms

Software Used:- Python
Package Used:- Pandas, Numpy, Sklearn(train_test_split, LogisticRegression, MultinomialNB), Tensorflow(Tokenizer, pad_sequences)

To train our model, we utilized various machine learning algorithms, including:

1. **Logistic Regression**: This is a statistical algorithm that estimates the probability of a binary outcome based on one or more predictor variables.

2. **Multinomial Naive Bayes**: This is a probabilistic algorithm that calculates the probability of a data point belonging to a specific class, based on the frequency of words in the document.

3. **Long Short-Term Memory (LSTM)**: This is a type of neural network that can process and analyze sequential data. It is often used for natural language processing tasks like text classification.

By utilizing these algorithms, our model was able to identify patterns and relationships within the data, allowing it to make accurate predictions about news article classification.

6.1 Logistic Regression for Multi-Class Text Classification

Logistic Regression is a statistical learning algorithm that is commonly used for binary classification tasks, where the goal is to predict whether an input belongs to one of two possible classes.

In the context of text classification, we can use Logistic Regression to predict the probability of a given text belonging to each possible class. This is achieved by representing the text as a set of features, such as word frequencies or TF-IDF values, and training the model to predict the probability of each class based on these features.

To train the model, we first need a labeled dataset that consists of text samples and their corresponding labels. This dataset is split into training and testing sets, and the training set is used to train the Logistic Regression model. The model is then evaluated on the testing set to assess its performance.

In Python, we can use scikit-learn, a popular machine learning library, to implement Logistic Regression for multi-class text classification. We start by vectorizing the text data using a technique such as TF-IDF, and then we train the Logistic Regression model on the training data. Once the model is trained, we can use it to make predictions on the testing data and evaluate its performance using metrics such as accuracy, precision, recall, and f1-score.

7. Model testing

We tested our trained models on validation data to check the performance of the models and use the best model to predict labels for our test dataset.

8. Model evaluation using different evaluation metrics

Software Used:- Python
Package Used:- Pandas, Numpy, Sklearn(classification_report, confusion_matrix, accuracy_score)

As we saw earlier, there is class imbalance in our dataset. The metrics which we used to evaluate our model are:-

- Accuracy
- Precision
- Recall
- F-1 score

9. Results

9.1 Logistic Regression:

In [13]:
```
# Evaluate the model on the validation data and print the classification report
valid_preds = clf.predict(valid_features)
print(classification_report(valid_labels, valid_preds))
```

```
              precision    recall  f1-score   support

      agreed       0.73      0.61      0.66     14868
   disagreed       0.80      0.21      0.34      1333
   unrelated       0.82      0.90      0.86     35088

    accuracy                           0.80     51289
   macro avg       0.79      0.58      0.62     51289
weighted avg       0.80      0.80      0.79     51289
```

In [14]:
```
from sklearn.metrics import confusion_matrix, accuracy_score

# Calculate the confusion matrix on the validation data
cm = confusion_matrix(valid_labels, valid_preds)

# Calculate the accuracy score on the validation data
acc = accuracy_score(valid_labels, valid_preds)

print('Confusion Matrix:\n', cm)
print('Accuracy Score:', acc)
```

```
Confusion Matrix:
 [[ 9023     9  5836]
 [   45   286  1002]
 [ 3279    61 31748]]
Accuracy Score: 0.8005030318391858
```

9.2 Multinomial Naive Bayes:

In [49]:
```
# Evaluate the model on the validation data and print the classification report
from sklearn.metrics import classification_report

# Use classification_report function here

# Print precision, recall, and f1-score
print(classification_report(y_val, y_pred_nb))
```

```
              precision    recall  f1-score   support

      agreed       0.59      0.39      0.47     14813
   disagreed       0.51      0.20      0.29      1321
   unrelated       0.75      0.88      0.81     35154

    accuracy                           0.72     51288
   macro avg       0.62      0.49      0.52     51288
weighted avg       0.70      0.72      0.70     51288
```

In [50]:
```
from sklearn.metrics import confusion_matrix, accuracy_score

# Calculate the confusion matrix on the validation data
cm = confusion_matrix(y_val, y_pred_nb)

# Calculate the accuracy score on the validation data
acc = accuracy_score(y_val, y_pred_nb)

print('Confusion Matrix:\n', cm)
print('Accuracy Score:', acc)
```

```
Confusion Matrix:
 [[ 5724    46  9043]
 [   16   264  1041]
 [ 3949   212 30993]]
Accuracy Score: 0.7210458586803931
```

9.3 Long short term memory:

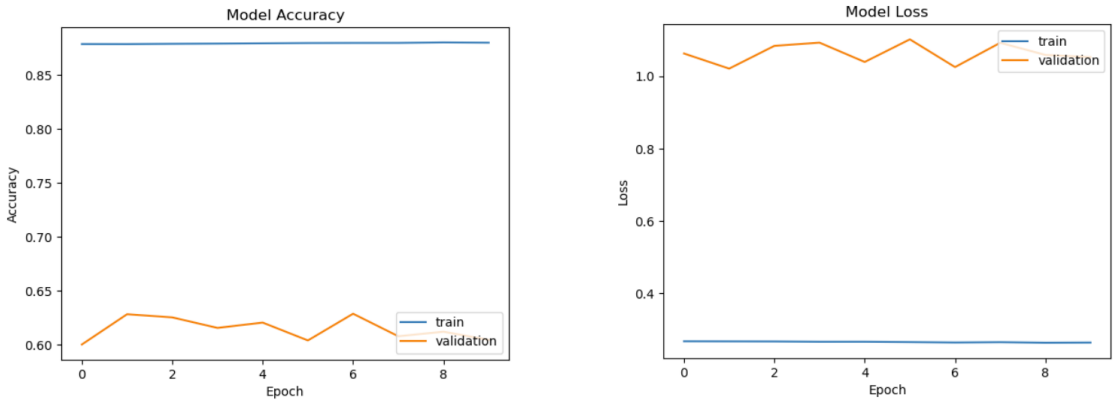

Figure 4: As we can see,there was no significant improvement in our loss and accuracy after the training, which may be because of class imbalance.

```
In [37]: # Train the model
         model.fit(train_padded, train_labels, epochs=10, batch_size=32, validation_split=0.2)

Epoch 1/10
6412/6412 [==============================] - 148s 23ms/step - loss: 0.2766 - accuracy: 0.8744 - val_loss: 1.0689 - val_accur
acy: 0.6151
Epoch 2/10
6412/6412 [==============================] - 153s 24ms/step - loss: 0.2749 - accuracy: 0.8757 - val_loss: 1.0250 - val_accur
acy: 0.6208
Epoch 3/10
6412/6412 [==============================] - 152s 24ms/step - loss: 0.2744 - accuracy: 0.8755 - val_loss: 1.0460 - val_accur
acy: 0.6269
Epoch 4/10
6412/6412 [==============================] - 163s 25ms/step - loss: 0.2737 - accuracy: 0.8763 - val_loss: 1.0541 - val_accur
acy: 0.6379
Epoch 5/10
6412/6412 [==============================] - 163s 25ms/step - loss: 0.2721 - accuracy: 0.8764 - val_loss: 0.9883 - val_accur
acy: 0.6270
Epoch 6/10
6412/6412 [==============================] - 160s 25ms/step - loss: 0.2709 - accuracy: 0.8772 - val_loss: 1.0612 - val_accur
acy: 0.5979
Epoch 7/10
6412/6412 [==============================] - 154s 24ms/step - loss: 0.2709 - accuracy: 0.8774 - val_loss: 1.0232 - val_accur
acy: 0.6132
Epoch 8/10
6412/6412 [==============================] - 156s 24ms/step - loss: 0.2698 - accuracy: 0.8777 - val_loss: 1.0169 - val_accur
acy: 0.6345
Epoch 9/10
6412/6412 [==============================] - 158s 25ms/step - loss: 0.2686 - accuracy: 0.8774 - val_loss: 1.0523 - val_accur
acy: 0.6043
Epoch 10/10
6412/6412 [==============================] - 152s 24ms/step - loss: 0.2686 - accuracy: 0.8783 - val_loss: 1.0755 - val_accur
acy: 0.6145

Out[37]: <keras.callbacks.History at 0x191f7a89ff0>
```

Figure 5: Classification report for LSTM

10. Final Model

```
In [63]: # Train a logistic regression model on the training data
         clf = LogisticRegression(max_iter=10000)
         clf.fit(train_features, train_labels)

Out[63]: LogisticRegression(max_iter=10000)
```

```
In [64]: # Evaluate the model on the validation data and print the classification report
         valid_preds = clf.predict(valid_features)
         print(classification_report(valid_labels, valid_preds))

                precision    recall  f1-score   support

       agreed       0.74      0.61      0.67     14975
    disagreed       0.73      0.18      0.29      1349
    unrelated       0.82      0.91      0.86     34965

     accuracy                           0.80     51289
    macro avg       0.76      0.57      0.61     51289
 weighted avg       0.80      0.80      0.79     51289
```

```
In [65]: from sklearn.metrics import confusion_matrix, accuracy_score

         # Calculate the confusion matrix on the validation data
         cm = confusion_matrix(valid_labels, valid_preds)

         # Calculate the accuracy score on the validation data
         acc = accuracy_score(valid_labels, valid_preds)

         print('Confusion Matrix:\n', cm)
         print('Accuracy Score:', acc)

Confusion Matrix:
 [[ 9127    11  5837]
 [   49   244  1056]
 [ 3139    79 31747]]
Accuracy Score: 0.8016923706837723
```

Our Logistic Regression model is giving better accuracy of ***80.05%*** compared to the other machine learning models Multinomial Naive Bayes algorithm ***(72%)*** and LSTM***(61.45%)***.

11. Test Data Prediction

We predicted the labels for the test data, attached those labels to the test data and created a csv file name submission.csv

12. Conclusion

In conclusion, we developed a machine learning model for the classification of news articles into three categories - agreed, disagreed, and unrelated. We utilized various NLP techniques to preprocess the data and applied machine learning algorithms such as Multinomial Naive Bayes, Logistic Regression, and LSTM for model training.

After evaluating the performance of the different models, we found that Logistic Regression achieved the highest accuracy in classifying the news articles. Thus, we recommend using Logistic Regression for future applications of this model.

Our findings demonstrate the effectiveness of combining NLP techniques with machine learning algorithms for text classification tasks. This model can be useful for various applications, such as content filtering and moderation in social media platforms, detecting fake news and propaganda, and identifying relevant news articles for personalized recommendations.

13. References

[1] Python libraries:
pytorch https://pytorch.org/docs/stable/index.html ,
Keras https://faroit.com/keras-docs/1.2.0/
Numpy https://numpy.org/doc/stable/ ,
Pandas https://pandas.pydata.org/pandas-docs/stable/,
[2] R,
tidyverse https://cran.r-project.org/web/packages/tidyverse/index.html
tm https://cran.r-project.org/web/packages/tm/vignettes/tm.pdf

[3] Shu, K., Sliva, A., Wang, S., Tang, J., & Liu, H. (2017). Fake news detection on social media: A data mining perspective. ACM SIGKDD Explorations Newsletter, 19(1), 22-36