Fake News Classification

CS579: Online Social Network Analysis

Final Report (Project 2)

Team Members:

- **Jinit Parikh (A20517770)**
- **Ruchit Mer (A20516697)**

Abstract

For this project, we are required to classify a given news article B into one of three categories based on its title and the title of a fake news article A.

- agreed: B talks about the same fake news as A.
- disagreed: B refutes the fake news in A.
- unrelated: B is unrelated to A.

Introduction

The prevalence of fake news and misinformation on social media can have a serious negative impact on individuals and society. Given the importance of detecting fake news, the task is to classify a coming news article into one of the three categories - agreed, disagreed, or unrelated - based on whether it talks about the same fake news as a given fake news article, refutes it, or is unrelated to it, respectively.

Project Outline

1. Data pre-processing
2. Model creation and model training.
3. Model result analysis.

Data Pre-processing:

First, we started with Data pre-processing using NLP techniques. The goal of pre-processing is to remove noise. By removing unnecessary features from our text, we can reduce complexity and increase predictability. This doesn't affect the meaning of text.

Data pre-processing consists of several methods which are performed on data. We have applied the following

1. Converting strings from uppercase to lower
   a. To make dataset consistent and normalize we convert words to lowercase. Words 'there' and 'There' are same but it may seems two different word to program. So to avoid duplication we converted it into lowercase. Screenshot of the output is attached below.

```
#converting into lower case
train_data["title1_en"] = train_data["title1_en"].apply(lambda train_word: train_word.lower())
train_data["title2_en"] = train_data["title2_en"].apply(lambda train_word: train_word.lower())
train_data
```

id	tid1	tid2	title1_en	title2_en	label
195611	0	1	there are two new old-age insurance benefits f...	police disprove "bird's nest congress each per...	unrelated
191474	2	3	"if you do not come to shenzhen, sooner or lat...	shenzhen's gdp outstrips hong kong? shenzhen s...	unrelated
25300	2	4	"if you do not come to shenzhen, sooner or lat...	the gdp overtopped hong kong? shenzhen clarifi...	unrelated
123757	2	8	"if you do not come to shenzhen, sooner or lat...	shenzhen's gdp overtakes hong kong? bureau of ...	unrelated
141761	2	11	"if you do not come to shenzhen, sooner or lat...	shenzhen's gdp outpaces hong kong? defending r...	unrelated
...	...	...	...	...	...
113364	167562	48447	egypt 's presidential election failed to win m...	salah is retiring? football association offici...	unrelated
49407	167562	49795	egypt 's presidential election failed to win m...	liverpool's bid for little germany? the echo's...	unrelated
130134	167562	114783	egypt 's presidential election failed to win m...	west media exposing tallahlach has been recomm...	unrelated
101494	167562	137705	egypt 's presidential election failed to win m...	rumor has it that egypt is very united and the...	unrelated
89356	167563	66480	will the united states wage war on iraq withou...	saddam's daughter refutes rumors: no. 2 of sad...	unrelated

256442 rows × 5 columns

Figure 1.1

2. Removing stopwords
   a. 'stopwords' are the set of most commonly used words in English dictionary. It consists of words like "the", "and", "a", "an", "in", "to", etc. These words carries little or no importance for text analysis and understanding the meaning of sentence. Screenshot of the output is attached below.

```
#removing stopwords from columns
train_data["title1_en"] = train_data["title1_en"].apply(lambda train_word: ' '.join(train_word for train_word in train_word.split() if train_wor
train_data["title2_en"] = train_data["title2_en"].apply(lambda train_word: ' '.join(train_word for train_word in train_word.split() if train_wor
train_data
```

id	tid1	tid2	title1_en	title2_en	label
195611	0	1	two new old-age insurance benefits old people ...	police disprove "bird's nest congress person g...	unrelated
191474	2	3	"if come shenzhen, sooner later son also come....	shenzhen's gdp outstrips hong kong? shenzhen s...	unrelated
25300	2	4	"if come shenzhen, sooner later son also come....	gdp overtopped hong kong? shenzhen clarified: ...	unrelated
123757	2	8	"if come shenzhen, sooner later son also come....	shenzhen's gdp overtakes hong kong? bureau sta...	unrelated
141761	2	11	"if come shenzhen, sooner later son also come....	shenzhen's gdp outpaces hong kong? defending r...	unrelated
...	...	...	...	...	...
113364	167562	48447	egypt 's presidential election failed win mill...	salah retiring? football association officials...	unrelated
49407	167562	49795	egypt 's presidential election failed win mill...	liverpool's bid little germany? echo's disclai...	unrelated
130134	167562	114783	egypt 's presidential election failed win mill...	west media exposing tallahlach recommended bar...	unrelated
101494	167562	137705	egypt 's presidential election failed win mill...	rumor egypt united differences us.	unrelated
89356	167563	66480	united states wage war iraq without destructio...	saddam's daughter refutes rumors: no. 2 saddam...	unrelated

256442 rows × 5 columns

Figure 1.2

3. Removing punctuation
    a. Punctuation marks like comma, period, exclamation mark etc are removed in order to reduce noise in data. It also helps in applying algorithms which are based on tokenization. Screenshot of the output is attached below.

```
#removing punctuation
train_data["title1_en"] = train_data["title1_en"].apply(lambda train_word: ''.join(train_word for train_word in str(train_word) if train_word nc
train_data["title2_en"] = train_data["title2_en"].apply(lambda train_word: ''.join(train_word for train_word in str(train_word) if train_word nc
train_data
```

id	tid1	tid2	title1_en	title2_en	label
195611	0	1	two new oldage insurance benefits old people r...	police disprove birds nest congress person get...	unrelated
191474	2	3	if come shenzhen sooner later son also come le...	shenzhens gdp outstrips hong kong shenzhen sta...	unrelated
25300	2	4	if come shenzhen sooner later son also come le...	gdp overtopped hong kong shenzhen clarified li...	unrelated
123757	2	8	if come shenzhen sooner later son also come le...	shenzhens gdp overtakes hong kong bureau stati...	unrelated
141761	2	11	if come shenzhen sooner later son also come le...	shenzhens gdp outpaces hong kong defending rum...	unrelated
...	...	...	...	...	...
113364	167562	48447	egypt s presidential election failed win milli...	salah retiring football association officials ...	unrelated
49407	167562	49795	egypt s presidential election failed win milli...	liverpools bid little germany echos disclaimer...	unrelated
130134	167562	114783	egypt s presidential election failed win milli...	west media exposing tallahlach recommended bar...	unrelated
101494	167562	137705	egypt s presidential election failed win milli...	rumor egypt united differences us	unrelated
89356	167563	66480	united states wage war iraq without destructio...	saddams daughter refutes rumors no 2 saddams r...	unrelated

256442 rows × 5 columns

Figure 1.3

4. Lemmatization
    a. It means to reduce the words to their base form. Lemmatization can help improve the accuracy of NLP models by grouping together words with similar meanings. Screenshot of this step is given below.

```
#lemmatizing training data
w_tokenizer = nltk.tokenize.WhitespaceTokenizer()
porter = nltk.stem.PorterStemmer()
lemmatizer = nltk.stem.WordNetLemmatizer()
def lemmatize_text(text):
    lemma_words = [lemmatizer.lemmatize(word) for word in w_tokenizer.tokenize(text)]
    return " ".join(lemma_words)

train_data["title1_en"] = train_data["title1_en"].apply(lambda data: lemmatize_text(data))
train_data["title2_en"] = train_data["title2_en"].apply(lambda data: lemmatize_text(data))
train_data
```

id	tid1	tid2	title1_en	title2_en	label
195611	0	1	two new oldage insurance benefit old people ru...	police disprove bird nest congress person get ...	unrelated
191474	2	3	if come shenzhen sooner later son also come le...	shenzhens gdp outstrips hong kong shenzhen sta...	unrelated
25300	2	4	if come shenzhen sooner later son also come le...	gdp overtopped hong kong shenzhen clarified li...	unrelated
123757	2	8	if come shenzhen sooner later son also come le...	shenzhens gdp overtakes hong kong bureau stati...	unrelated
141761	2	11	if come shenzhen sooner later son also come le...	shenzhens gdp outpaces hong kong defending rum...	unrelated
...	...	...	...	...	...
113364	167562	48447	egypt s presidential election failed win milli...	salah retiring football association official l...	unrelated
49407	167562	49795	egypt s presidential election failed win milli...	liverpool bid little germany echo disclaimer s...	unrelated
130134	167562	114783	egypt s presidential election failed win milli...	west medium exposing tallahlach recommended ba...	unrelated
101494	167562	137705	egypt s presidential election failed win milli...	rumor egypt united difference u	unrelated

Figure 1.4

Model Creation:

a. Loading and cleaning data:

- The code reads preprocessed data from a CSV file using the pandas library's read.csv() function and sets the 'id' column as the index for the dataframe.
- The shape of the dataframe is printed to show the number of rows and columns in the dataframe.
- The code drops any rows with missing values from the dataframe using the dropna() method.

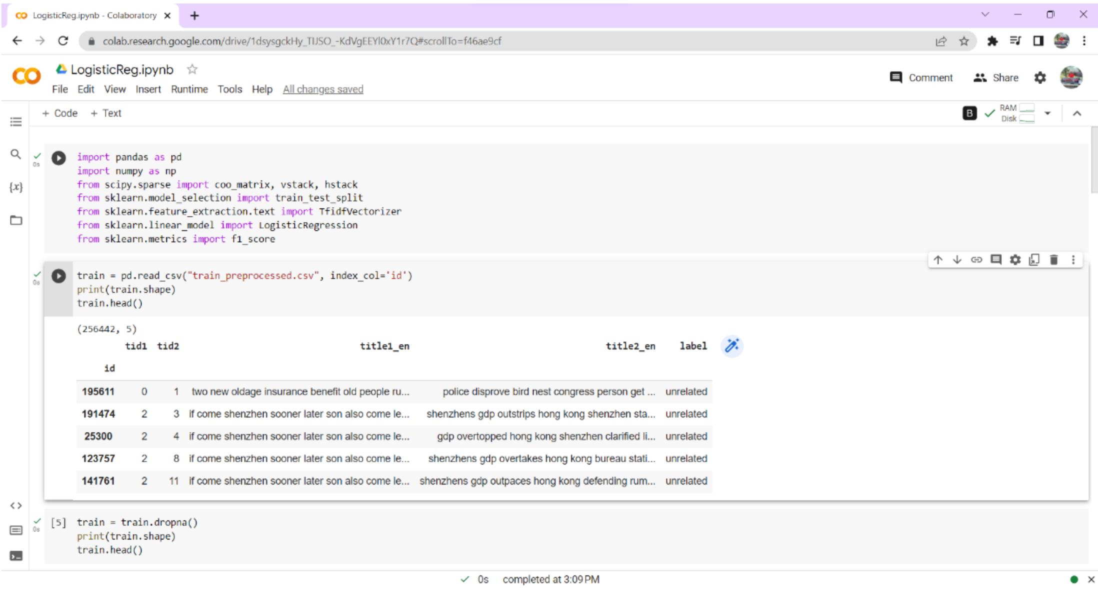


Figure 2.1

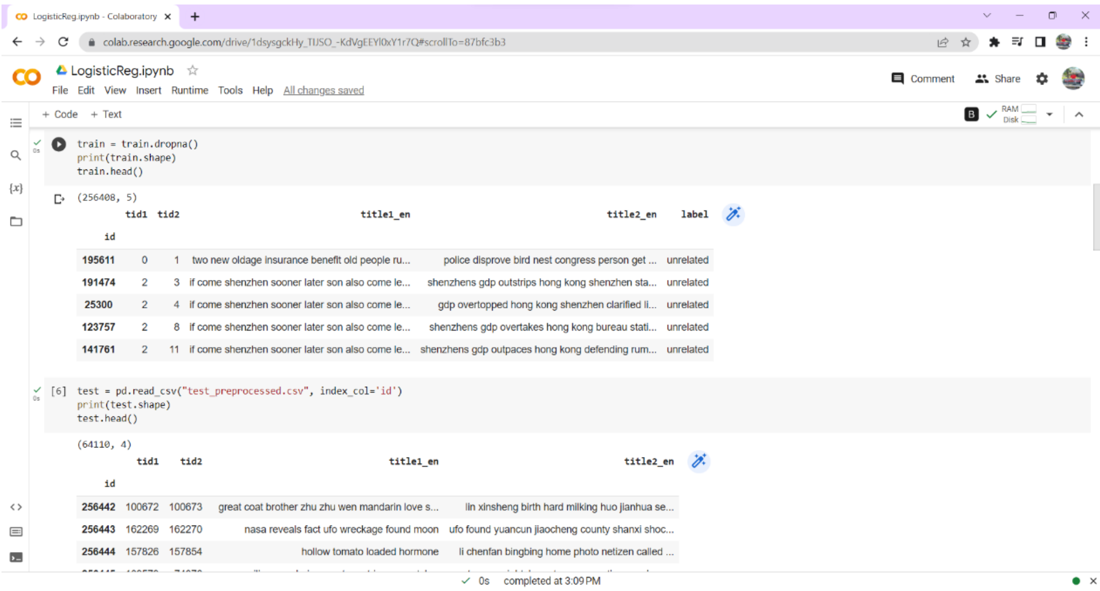
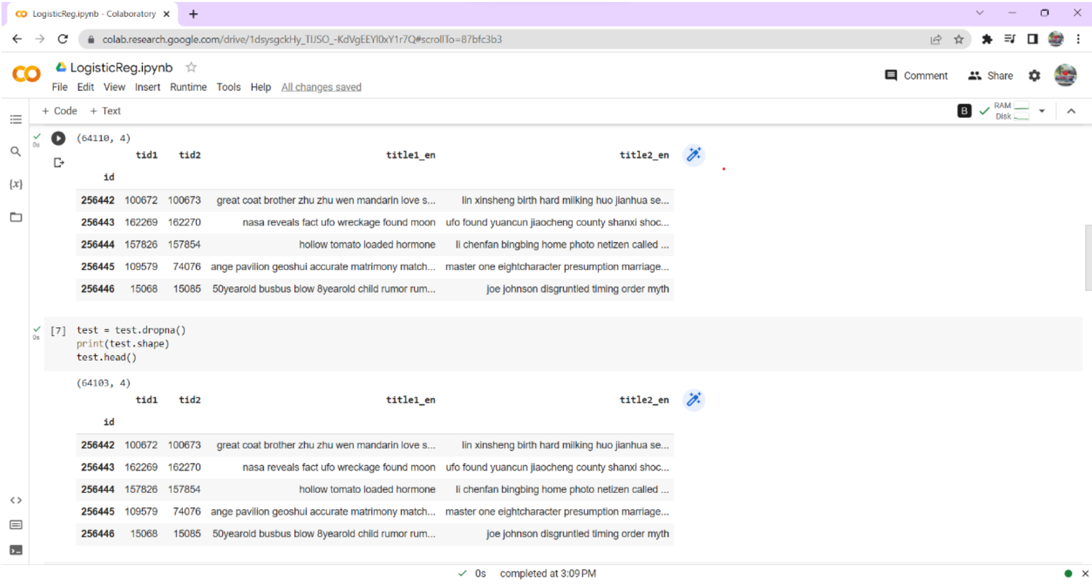


Figure 2.2



```
(64110, 4)
```

id	tid1	tid2	title1_en	title2_en
256442	100672	100673	great coat brother zhu zhu wen mandarin love s...	lin xinsheng birth hard milking huo jianhua se...
256443	162269	162270	nasa reveals fact ufo wreckage found moon	ufo found yuancun jiaocheng county shanxi shoc...
256444	157826	157854	hollow tomato loaded hormone	li chenfan bingbing home photo netizen called ...
256445	109579	74076	ange pavilion geoshui accurate matrimony match...	master one eightcharacter presumption marriage...
256446	15068	15085	50yearold busbus blow 8yearold child rumor rum...	joe johnson disgruntled timing order myth

```
test = test.dropna()
print(test.shape)
test.head()
```

```
(64103, 4)
```

id	tid1	tid2	title1_en	title2_en
256442	100672	100673	great coat brother zhu zhu wen mandarin love s...	lin xinsheng birth hard milking huo jianhua se...
256443	162269	162270	nasa reveals fact ufo wreckage found moon	ufo found yuancun jiaocheng county shanxi shoc...
256444	157826	157854	hollow tomato loaded hormone	li chenfan bingbing home photo netizen called ...
256445	109579	74076	ange pavilion geoshui accurate matrimony match...	master one eightcharacter presumption marriage...
256446	15068	15085	50yearold busbus blow 8yearold child rumor rum...	joe johnson disgruntled timing order myth

Figure 2.3

b. Tokenization and feature engineering:

- The dataframe is vectorized using the TfidfVectorizer class from the scikit-learn library to convert the text data into a numerical representation that can be used by machine learning algorithms.
- The title1_en column of the dataframe is vectorized using the TfidfVectorizer class and the resulting sparse matrix is stored in the title1_tfidf_vector variable.
- The title2_en column of the dataframe is also vectorized using the same TfidfVectorizer object as title1_en, and the resulting sparse matrix is stored in the title2_tfidf_vector variable.

```
[8] title1 = TfidfVectorizer(analyzer='word',stop_words= 'english').fit(train['title1_en'])
    title1_tfidf = title1.transform(train['title1_en'])

[9] title1_tfidf.shape

(256408, 28629)

    title2_tfidf = title1.transform(train['title2_en'])

[11] title2_tfidf.shape

(256408, 28629)
```

Figure 2.4

c. Train-test split:

- The two sparse matrices (title1_tfidf_vector and title2_tfidf_vector) are horizontally stacked together using the hstack() function from the scipy.sparse library to create a new sparse matrix with a shape of (205126, 57258).
- The resulting sparse matrix is stored in the title_stack variable and can be used as input to machine learning models.
- The sparse matrix formed by using hstack along with the label was split into training and validation sets using the train_test_split function from sklearn.model_selection.

```
[12] title_stack = hstack([title1_tfidf, title2_tfidf])

[13] title_stack

<256408x57258 sparse matrix of type '<class 'numpy.float64'>'
	with 4435812 stored elements in Compressed Sparse Row format>

[14] x_train, x_test, y_train, y_test = train_test_split(title_stack, train['label'], test_size=0.2)
```

Figure 2.5

Model Training:

1. Logistic Regression:

a. Model fitting:

- The first step in logistic regression is to fit the training data to a logistic regression model. In this project, the training data was fitted to a logistic regression model using the LogisticRegression function from the sklearn.linear_model library.
- The input features to the model were the stacked title vectors, and the output was the label indicating whether the news article was agreed, disagreed or unrelated.

```
#training model
model = LogisticRegression(max_iter=500).fit(x_train, y_train)
```

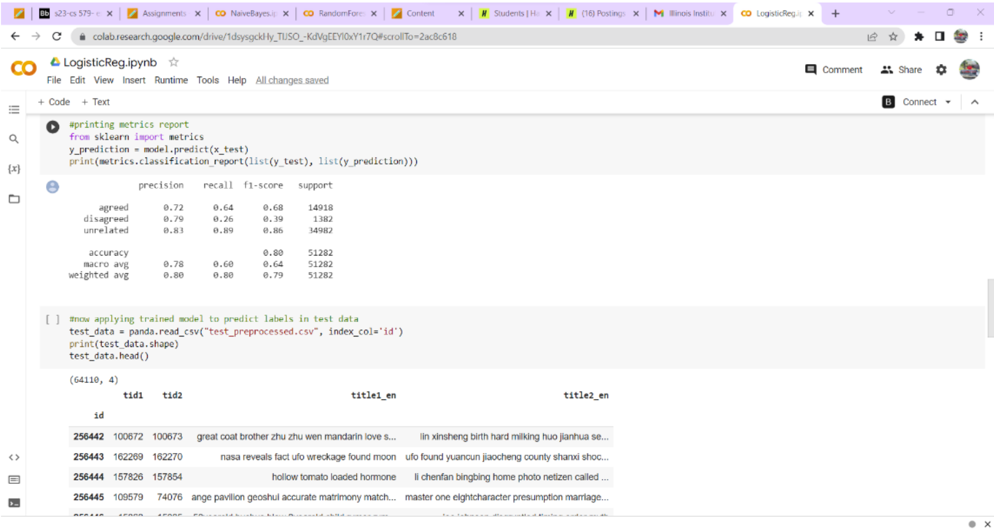
```
[ ] #getting accuracy
accuracy_score = model.score(x_test, y_test)
accuracy_score

0.8023478023478023
```

Figure 3.1

b. Model evaluation:

- After the model is trained, it is evaluated using validation data to assess its performance. The accuracy and F1 score were used as evaluation metrics in this project.
- The accuracy score gives the ratio of correct predictions made by the model, while the F1 score is a weighted average of precision and recall.



```
#printing metrics report
from sklearn import metrics
y_prediction = model.predict(x_test)
print(metrics.classification_report(list(y_test), list(y_prediction)))

              precision    recall  f1-score   support

      agreed       0.72      0.64      0.68     14918
   disagreed       0.79      0.26      0.39      1382
   unrelated       0.83      0.89      0.86     34982

    accuracy                           0.80     51282
   macro avg       0.78      0.60      0.64     51282
weighted avg       0.80      0.80      0.79     51282
```

```
[ ] #now applying trained model to predict labels in test data
test_data = panda.read_csv("test_preprocessed.csv", index_col='id')
print(test_data.shape)
test_data.head()

(64110, 4)
```

id	tid1	tid2	title1_en	title2_en
256442	100672	100673	great coat brother zhu zhu wen mandarin love s...	lin xinsheng birth hard milking huo jianhua se...
256443	162269	162270	nasa reveals fact ufo wreckage found moon	ufo found yuancun jiaocheng county shanxi shoc...
256444	157826	157854	hollow tomato loaded hormone	li chenfan bingbing home photo netizen called ...
256445	109579	74076	ange pavilion geoshui accurate matrimony match...	master one eightcharacter presumption marriage...

Figure 3.2

c. Prediction and storage:

- After evaluating the model, it was used to predict the labels for the test data. The predicted labels were stored in a pandas DataFrame along with the corresponding IDs, and the results were saved in a CSV file.

```
[ ] #vectorizing columns in test data
test_title1_tfidf_vector = title1_vector.transform(test_data['title1_en'])
test_title2_tfidf_vector = title1_vector.transform(test_data['title2_en'])
#stacking to form matric with incresed columns
test_title_stack = hstack([test_title1_tfidf_vector, test_title2_tfidf_vector])
```

```
[ ] #prediciting labels with the trained model
test_predict_data_labels = model.predict(test_title_stack)
```

Figure 3.3

```
[ ] #extracting label column
    test_label_column = test_data['label']
    print(test_label_column)

    id
    256442    unrelated
    256443    unrelated
    256444    unrelated
    256445    unrelated
    256446    unrelated
                ...
    320547    unrelated
    320548    unrelated
    320549      agreed
    320550    unrelated
    320551      agreed
    Name: label, Length: 64103, dtype: object

[ ] #saving output file
    from google.colab import files
    test_label_column.to_csv('test_logistic_Reg_predicted.csv')
    files.download('test_logistic_Reg_predicted.csv')
```

Figure 3.4

2. Naïve Bayes:

a. Model Fitting:

- The training data is fitted to a Naive Bayes model from sklearn.naive_bayes.
- In particular, we can use the MultinomialNB class for text classification tasks where the features are discrete values such as word counts.

```
# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(title_stack, train_data['label'], test_size=0.2)

# Create a Multinomial Naive Bayes model and train it on the training data
nb = MultinomialNB()
nb.fit(x_train, y_train)

▸ MultinomialNB
MultinomialNB()
```

Figure 3.5

b. Model evaluation:

- The accuracy and F1 score of the model are evaluated on the validation data.
- We can use the accuracy_score and classification_report functions from sklearn.metrics to evaluate the performance of the model.

```
# Predict the labels of the test data
y_pred = nb.predict(x_test)

# Evaluate the accuracy of the model on the test data
accuracy = nb.score(x_test, y_test)
print("Accuracy:", accuracy)

Accuracy: 0.7682227682227682
```

Figure 3.6

c. Prediction and storage:

- The model is used to predict the labels for the test data, and the results are stored in a pandas DataFrame along with the corresponding IDs. The predictions are saved in a CSV file.

```
[ ] #vectorizing columns in test data
    test_title1_tfidf_vector = title1_vector.transform(test_data['title1_en'])
    test_title2_tfidf_vector = title1_vector.transform(test_data['title2_en'])
    #stacking to form matric with incresed columns
    test_title_stack = hstack([test_title1_tfidf_vector, test_title2_tfidf_vector])
```

```
[ ] #prediciting labels with the trained model
    test_predict_data_labels = nb.predict(test_title_stack)
```

```
[ ] #adding labels to test data
    test_data['label'] = test_predict_data_labels
```

Figure 3.7

```
#extracting label column
test_label_column = test_data['label']
print(test_label_column)
```

```
id
256442      agreed
256443   unrelated
256444   unrelated
256445   unrelated
256446   unrelated
           ...
320547   unrelated
320548      agreed
320549      agreed
320550   unrelated
320551      agreed
Name: label, Length: 64103, dtype: object
```

```
[ ] #saving output file
    from google.colab import files
    test_label_column.to_csv('test_Naive_Bais_predicted.csv')
    files.download('test_Naive_Bais_predicted.csv')
```

Figure 3.8

3. Random Forest:

a. Model Fitting:

- The training data was fitted to a RandomForestClassifier model from sklearn.ensemble. RandomForest is an ensemble learning method that constructs a multitude of decision trees at training time and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.
- The model was trained using the title_stack and train_data['label'] generated earlier.

```
[ ] # Split the data into training and testing sets
    x_train, x_test, y_train, y_test = train_test_split(title_stack, train_data['label'], test_size=0.2)

    # Train a Random Forest model on the training set
    rf = RandomForestClassifier(n_estimators=100)
    rf.fit(x_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

Figure 3.9

b. Model evaluation:

- The accuracy and F1 score of the model were evaluated on the validation data using cross-validation with 5 folds. Cross-validation is a technique that divides the training dataset into k subsets and uses each subset in turn to evaluate the model, while using the remaining subsets as training data.

- The model was evaluated on the validation data to ensure that it is not overfitting the training data. The accuracy and F1 score were calculated to determine the performance of the mode.

```
# Predict the labels of the test data
y_pred = rf.predict(x_test)

# Evaluate the accuracy of the model on the test data
accuracy = rf.score(x_test, y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8588198588198588
```

Figure 3.10

c. Prediction and storage:

- The model was used to predict the labels for the test data. The test data was vectorized using the same TfidfVectorizer as used on the training data. The test data was then stacked horizontally using the hstack function from scipy.sparse. The resulting test_title_stack was used as input to the Random Forest model to generate the predicted labels for the test data. The predicted labels were stored in a pandas DataFrame along with the corresponding IDs. The predictions were then saved in a CSV file for further analysis.

```
#vectorizing columns in test data
test_title1_tfidf_vector = title1_vector.transform(test_data['title1_en'])
test_title2_tfidf_vector = title1_vector.transform(test_data['title2_en'])
#stacking to form matric with incresed columns
test_title_stack = hstack([test_title1_tfidf_vector, test_title2_tfidf_vector])
```

```
#prediciting labels with the trained model
test_predict_data_labels = rf.predict(test_title_stack)
```

```
#adding labels to test data
test_data['label'] = test_predict_data_labels
```

Figure 3.11

```
#extracting label column
test_label_column = test_data['label']
print(test_label_column)
```

```
id
256442    unrelated
256443    unrelated
256444    unrelated
256445    unrelated
256446    unrelated
            ...
320547      agreed
320548      agreed
320549      agreed
320550    unrelated
320551    unrelated
Name: label, Length: 64103, dtype: object
```

```
#saving output file
from google.colab import files
test_label_column.to_csv('submission.csv')
files.download('submission.csv')
```

Figure 3.12

4. Linear SVC

a. Model Fitting:

- The training data is fitted to a Linear SVC model from sklearn.svm.
- In particular, we can use the LinearSVC class for text classification tasks where the features are discrete values such as word counts.

```
#splitting data into train and test for input in model training
print(title_stack.shape)
x_train, x_test, y_train, y_test = train_test_split(title_stack, train['label'], test_size=0.2)

(256408, 57258)
```

```
[ ] #training model
    model = LinearSVC().fit(x_train, y_train)
```

Figure 3.13

b. Model evaluation:

- The accuracy and F1 score of the model are evaluated on the validation data.
- We can use the accuracy_score and classification_report functions from sklearn.metrics to evaluate the performance of the model.

```
[ ] #getting accuracy
    score = model.score(x_test, y_test)
    score

    0.8111033111033111
```

```
[ ] y_pred = model.predict(x_test)
    score_f1 = f1_score(y_pred, y_test, average=None)
    score_f1

    array([0.7011205 , 0.46021287, 0.8653368 ])
```

Figure 3.14

c. Prediction and storage:

- The model is used to predict the labels for the test data, and the results are stored in a pandas DataFrame along with the corresponding IDs. The predictions are saved in a CSV file.

```
#vectorizing columns in test data
test_title1_tfidf = title1.transform(test_preprocessed_data['title1_en'])
test_title2_tfidf = title1.transform(test_preprocessed_data['title2_en'])

#stacking to form matric with incresed columns
test_title_stack = hstack([test_title1_tfidf, test_title2_tfidf])
#prediciting labels with the trained model
test_predict = model.predict(test_title_stack)
```

```
[ ] #adding labels to test data
    test_preprocessed_data['label'] = test_predict
    test_preprocessed_data.head()
```

Figure 3.15

```
#extracting label column
test_labels = test_preprocessed_data['label']
test_labels.head()
```

```
id
256442    unrelated
256443    unrelated
256444    unrelated
256445    unrelated
256446    unrelated
Name: label, dtype: object
```

```
#saving output file
from google.colab import files
test_labels.to_csv('test_linearSVM_predicted.csv')
files.download('test_linearSVM_predicted.csv')
```

Figure3.16

Results

This section contains the accuracy got after implementing algorithms. Screenshots of each algorithm's accuracy report is given below.

Logistic Regression

- Logistic Regression was applied after using TFidf to convert text data into a sparse matrix.
- The sparse matrix had a shape of (256408, 57258).
- A test train split was performed with a test size of 0.2.
- The Logistic Regression model was trained on this training set.
- Accuracy report is given below.

```
[ ] #getting accuracy
    accuracy_score = model.score(x_test, y_test)
    accuracy_score

    0.8023478023478023

[ ] #printing metrics report
    from sklearn import metrics
    y_prediction = model.predict(x_test)
    print(metrics.classification_report(list(y_test), list(y_prediction)))

                  precision    recall  f1-score   support

         agreed       0.72      0.64      0.68     14918
      disagreed       0.79      0.26      0.39      1382
      unrelated       0.83      0.89      0.86     34982

       accuracy                           0.80     51282
      macro avg       0.78      0.60      0.64     51282
   weighted avg       0.80      0.80      0.79     51282
```

Figure 4.1 Logistic Regression accuracy

- Accuracy obtained on test data: 0.80234
- F1-score for agreed is 0.68, disagreed is 0.39 and unrelated is 0.86.

Linear SVC

- LinearSVC was applied after using TFidf to convert text data into a sparse matrix.
- The sparse matrix had a shape of (256408, 57258).
- A test train split was performed with a test size of 0.2.
- The Linear SVC model was trained on this training set.
- Accuracy report is given below.

```
[ ] #getting accuracy
    score = model.score(x_test, y_test)
    score

    0.8111033111033111

[ ] y_pred = model.predict(x_test)
    score_f1 = f1_score(y_pred, y_test, average=None)
    score_f1

    array([0.7011205 , 0.46021287, 0.8653368 ])

[ ] #printing metrics report
    from sklearn import metrics
    print(metrics.classification_report(list(y_test), list(y_pred)))

                  precision    recall  f1-score   support

         agreed       0.72      0.68      0.70     14864
      disagreed       0.70      0.34      0.46      1320
      unrelated       0.85      0.88      0.87     35098

       accuracy                           0.81     51282
      macro avg       0.75      0.64      0.68     51282
   weighted avg       0.81      0.81      0.81     51282
```

Figure 4.2 LinearSVC accuracy

- Accuracy obtained on test data: 0.8111033
- F1-score for agreed is 0.70, disagreed is 0.46 and unrelated is 0.87.

Naïve Bayes

- Naïve Bayes was applied after using TFidf to convert text data into a sparse matrix. MultinomialNB was used from sklearn library.
- The sparse matrix had a shape of (256408, 57258).
- A test train split was performed with a test size of 0.2.
- The Naïve Bayes (MultinomialNB) model was trained on this training set.
- Accuracy report is given below.

```
[ ] # Predict the labels of the test data
    y_pred = nb.predict(x_test)

    # Evaluate the accuracy of the model on the test data
    accuracy = nb.score(x_test, y_test)
    print("Accuracy:", accuracy)
```

```
    Accuracy: 0.7682227682227682
```

```
from sklearn import metrics

# Make predictions on the test data
y_pred = nb.predict(x_test)

# Print a classification report
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

      agreed       0.66      0.57      0.62     14975
   disagreed       0.73      0.13      0.22      1296
   unrelated       0.80      0.87      0.84     35011

    accuracy                           0.77     51282
   macro avg       0.73      0.53      0.56     51282
weighted avg       0.76      0.77      0.76     51282
```

Figure 4.3 Naïve Bayes accuracy

- Accuracy obtained on test data: 0.76822
- F1-score for agreed is 0.62, disagreed is 0.22 and unrelated is 0.84.

Random Forest

- Random Forest was applied after using TFidf to convert text data into a sparse matrix. RandomForestClassifier was used from sklearn library.
- The sparse matrix had a shape of (256408, 57258).
- A test train split was performed with a test size of 0.2.
- The Random Forest model was trained on this training set.
- Accuracy report is given below.

```
# Predict the labels of the test data
y_pred = rf.predict(x_test)

# Evaluate the accuracy of the model on the test data
accuracy = rf.score(x_test, y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8588198588198588
```

```
from sklearn import metrics

# Make predictions on the test data
y_pred = rf.predict(x_test)

# Print a classification report
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

     agreed        0.85      0.71      0.77     14766
  disagreed        0.80      0.33      0.47      1377
  unrelated        0.86      0.94      0.90     35139

   accuracy                            0.86     51282
  macro avg        0.84      0.66      0.71     51282
weighted avg       0.86      0.86      0.85     51282
```

Figure 4.4 Random Forest accuracy

- Accuracy obtained on test data: 0.85881
- F1-score for agreed is 0.77, disagreed is 0.47 and unrelated is 0.90.

As per above discussions and accuracy obtained, Random forest gave best performance. Logistic regression and LinearSVC also gave good performance. Naïve Bayes performed poorly for the given task as it produced accuracy only around 0.75. Hence, we decided to run Random forest to predict class labels for the test data. Snippet of the generated labels by Random Forest is given below.

	A	B
1	id	label
2	256442	unrelated
3	256443	unrelated
4	256444	unrelated
5	256445	unrelated
6	256446	unrelated
7	256447	unrelated
8	256448	unrelated
9	256449	unrelated
10	256450	unrelated
11	256451	unrelated
12	256452	agreed
13	256453	agreed
14	256454	agreed
15	256455	unrelated
16	256456	unrelated
17	256457	unrelated
18	256458	agreed
19	256459	unrelated
20	256460	unrelated
21	256461	unrelated
22	256462	unrelated
23	256463	unrelated
24	256464	unrelated
25	256465	unrelated
26	256466	unrelated

Figure 4.5 Generated labels for test data.

Team Effort

We worked on the project together so the below table is just an estimate. We always sat together and worked on the project task.

Task	**Jinit Parikh**	**Ruchit Mer**
Data Pre-processing	60	40
Logistic Regression	50	50
Linear SVC	60	40
Naïve Bayes	40	60
Random Forest	40	60
Project Report	50	50
Project Presentation	50	50

References

- Pandas library documentation: https://pandas.pydata.org/docs/user_guide/index.html
- Matplotlib documentation: https://matplotlib.org/stable/index.html#
- Scipy documentation: https://docs.scipy.org/doc/scipy/tutorial/index.html#user-guide
- Linear SVM: https://scikit-learn.org/stable/modules/svm.html#svm
- sklearn: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model
- Logistic regression: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression
- Random Forest: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier
- Naïve Bayes: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB

- TD-IDF: https://medium.com/@cmukesh8688/tf-idf-vectorizer-scikit-learn-dbc0244a911a
- Text Preprocessing: https://jon-dagdagan.medium.com/fake-news-detection-pre-processing-text-d9648a2854e5
- NLTK library: https://www.nltk.org/api/nltk.html