CSP 579 Online Social Network Analysis

Project Report

On

Fake News Classification

By

Name of Student	CWID
1. Henilkumar Hareshbhai Patel	A20513297
2. Harsh Gordhan Dungrani	A20514062

Under The Supervision
of
Prof. Kai Shu

College Of Computing

Illinois Institute of Technology
Chicago, Illinois.

April 2023

CONTENT:

Section 1: Project Details

1.1 Project Topic: In this Project, we need to define the classes based on the given piece of information. Here, there are three classes based on the fake titles, If the fake news article A and the title of a coming news article suppose is B, then based on the training and testing data sets, it is divided into the three categories,
The first is agreed: if B talks about the same fake news as A,
The second is disagreed: if B refutes (denies) the fake news in A,
The third is unrelated: if B is not related to the fake news in A.

1.2 Application Subject Area: Social Media

1.3 Data Set Source: For running the project, we need to use the dataset, which is provided by the professor Kai Shu.

- This data set has more than 2.5 lakhs data rows, and this data has six attributes like id, tid1, tid2, title1_n, title2_n, and label, the description of labels is,
    1) id : The id of each pair
    2) tid1 : The id of fake news title 1
    3) tid2 : The id of fake news title 2
    4) title1_en : The fake news title 1
    5) title2_en : The fake news title 2
    6) label : indicates the relation between the news pairs like agreed, disagreed, and unrelated.

Section 2: Methodology

2.1 Data Exploration:

Import the necessary libraries

1) Pandas: Pandas is a Python package used for data analysis. The panda's library has useful functions for analyzing, cleaning, exploring, and manipulating datasets.
2) Seaborn: Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
3) Scikit-learn (sklearn): Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms.
4) Imbalanced-learn (imblearn): Imbalanced-learn is an open source library that provides tools when dealing with classification with imbalanced classes. It provides various methods like undersampling, oversampling, and SMOTE to handle and removing the imbalance from the dataset.
5) Natural Language Toolkit (NLTK): NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries.
6) String: String module in Python contains some constants, utility function, and classes for string manipulation.

```
# import libraries
import pandas as pd
import seaborn as sns
import sklearn
import imblearn
import nltk
nltk.download('all')
import string
```

Read the train.csv file using pandas into a data frame ‘train’. The first five entries of the train data set can be seen below.

```
train=pd.read_csv('/content/train.csv')
```

```
train.head()
```

	id	tid1	tid2	title1_en	title2_en	label
0	195611	0	1	There are two new old-age insurance benefits f...	Police disprove "bird's nest congress each per...	unrelated
1	191474	2	3	"If you do not come to Shenzhen, sooner or lat...	Shenzhen's GDP outstrips Hong Kong? Shenzhen S...	unrelated
2	25300	2	4	"If you do not come to Shenzhen, sooner or lat...	The GDP overtopped Hong Kong? Shenzhen clarifi...	unrelated
3	123757	2	8	"If you do not come to Shenzhen, sooner or lat...	Shenzhen's GDP overtakes Hong Kong? Bureau of ...	unrelated
4	141761	2	11	"If you do not come to Shenzhen, sooner or lat...	Shenzhen's GDP outpaces Hong Kong? Defending R...	unrelated

The train dataset consists of 256442 rows and 6 columns.

```
train.shape
```

```
(256442, 6)
```

Out of the 6 columns, id, tid1 and tid2 are of int64 data type while title1_en, title2_en and label have object data type.

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 256442 entries, 0 to 256441
Data columns (total 6 columns):
 #   Column     Non-Null Count   Dtype 
---  ------     --------------   ----- 
 0   id         256442 non-null  int64 
 1   tid1       256442 non-null  int64 
 2   tid2       256442 non-null  int64 
 3   title1_en  256442 non-null  object
 4   title2_en  256442 non-null  object
 5   label      256442 non-null  object
dtypes: int64(3), object(3)
memory usage: 11.7+ MB
```

There are no null values in the train dataset.

```
train.isna().sum()
```

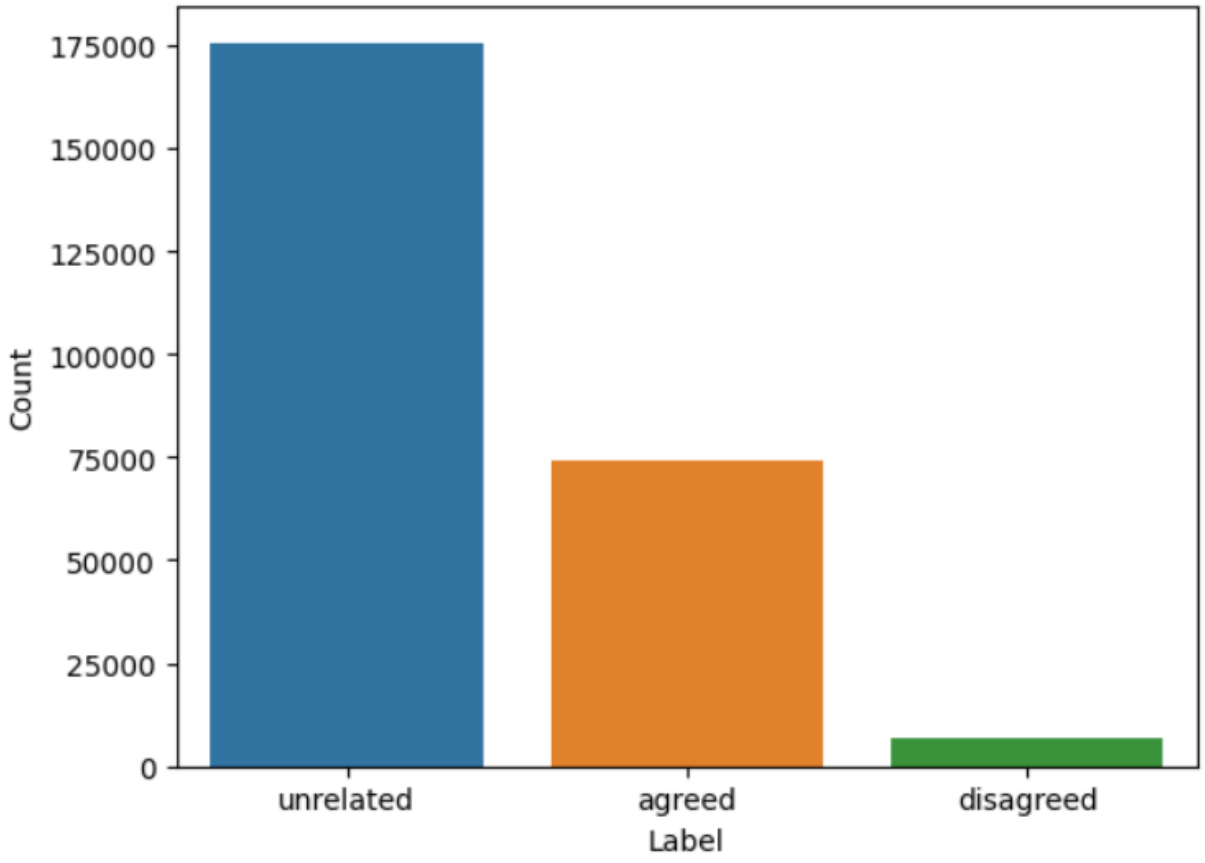
```
id           0
tid1         0
tid2         0
title1_en    0
title2_en    0
label        0
```

The target class ‘label’ has three categories namely agree, disagree and unrelated with the following size.

```
print(train.groupby('label').size())
```

```
label
agreed        74238
disagreed      6606
unrelated    175598
```

The count plot of target ‘label’ below depicts the number of entries in the three categories. Unrelated has the maximum while disagreed has the minimum number of samples in the ‘train.csv’ file.



2.2 Text Data Preprocessing:

The first entries of title1_en and title2_en for ‘train.csv’ file.

```
print(train['title1_en'].iloc[0])
print(train['title2_en'].iloc[0])
```

```
There are two new old-age insurance benefits for old people in rural areas. Have you got them?
Police disprove "bird's nest congress each person gets 50,000 yuan" still old people insist on going to beijing
```

1) Convert Lowercase: Now, as we observe from the above data, there are two data fields like title1_en and the title_2 which have entries in lower and upper case letter, so before using it for the next step, we need to convert them into the lower case.

```
#Convert Lowercase
train["title1_en"] = train["title1_en"].str.lower()
train["title2_en"] = train["title2_en"].str.lower()
print(train['title1_en'].iloc[0])
print(train['title2_en'].iloc[0])
```

```
there are two new old-age insurance benefits for old people in rural areas. have you got them?
police disprove "bird's nest congress each person gets 50,000 yuan" still old people insist on going to beijing
```

2) Remove stopwords: The main idea of removal of stop words (like a, an, the, etc.) is to get rid of noise in the data and extract more relevant information from it.

```
#Remove stopwords
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
train["title1_en"] = train["title1_en"].apply(lambda x: ' '.join([word for word in str(x).split() if word not in (stop_words)]))
train["title2_en"] = train["title2_en"].apply(lambda x: ' '.join([word for word in str(x).split() if word not in (stop_words)]))
print(train['title1_en'].iloc[0])
print(train['title2_en'].iloc[0])
```

```
two new old-age insurance benefits old people rural areas. got them?
police disprove "bird's nest congress person gets 50,000 yuan" still old people insist going beijing
```

3) Strip whitespace: Stripping the whitespaces is good as it doesn't store extra memory and the text becomes clearer.

```
#Strip whitespace
train["title1_en"] = train["title1_en"].str.strip()
train["title2_en"] = train["title2_en"].str.strip()
print(train['title1_en'].iloc[0])
print(train['title2_en'].iloc[0])
```

```
two new old-age insurance benefits old people rural areas. got them?
police disprove "bird's nest congress person gets 50,000 yuan" still old people insist going beijing
```

4) Remove Punctuation: The removal of punctuation will help to treat each sentence equally.

```
#Remove punctuation
import string
string.punctuation
train["title1_en"] = train["title1_en"].apply(lambda x: ''.join([word for word in str(x) if word not in (string.punctuation)]))
train["title2_en"] = train["title2_en"].apply(lambda x: ''.join([word for word in str(x) if word not in (string.punctuation)]))
print(train['title1_en'].iloc[0])
print(train['title2_en'].iloc[0])
```

```
two new oldage insurance benefits old people rural areas got them
police disprove birds nest congress person gets 50000 yuan still old people insist going beijing
```

5) Lemmatization: Lemmatization is the process of converting a word to its base form. Lemmatization considers the context and converts the word to its meaningful base form. Wordnet is a large, freely and publicly available lexical database for the English language aiming to establish structured semantic relationships between words.

```
#Lemmatization
from nltk.stem import WordNetLemmatizer
lemm = WordNetLemmatizer()
train["title1_en"] = train["title1_en"].apply(lambda x: ' '.join([lemm.lemmatize(word) for word in str(x).split()]))
train["title2_en"] = train["title2_en"].apply(lambda x: ' '.join([lemm.lemmatize(word) for word in str(x).split()]))
print(train['title1_en'].iloc[0])
print(train['title2_en'].iloc[0])
```

```
two new oldage insurance benefit old people rural area got them
police disprove bird nest congress person get 50000 yuan still old people insist going beijing
```

The first entries of title1_en and title2_en for 'train.csv' file after text preprocessing using nltk library.

```
print(train['title1_en'].iloc[0])
print(train['title2_en'].iloc[0])
```

```
two new oldage insurance benefit old people rural area got them
police disprove bird nest congress person get 50000 yuan still old people insist going beijing
```

2.3 TF-IDF Vectorizer: TF-IDF stands for Term Frequency Inverse Document (TF-IDF). It is used to convert a collection of raw documents to a matrix of TF-IDF features. It is an algorithm that is useful for transforming the text into numbers, which is useful for fitting the machine learning algorithms. The TF-IDF is calculated using the multifaction of term frequency (TF), which is the number of times the term appears in the document, and IDF represents the inverse of the frequency. It is based on the idea that words that appear in a document more often are more relevant to the document. Higher the TF-IDF value of a term, the more relevant the term is in that document.
TF-IDF(t) = TF(t) x IDF(t) where TF(t) = No. of times term 't' occurs in a document and IDF(t) =1 + $\log_e$[n/df(t)] where n = Total number of documents available and df(t) = Number of documents in which the term t appears.

```
vectorizer =TfidfVectorizer(stop_words= 'english')
```

We split the 'train.csv' data into 80/20 train and validation test and apply TF-IDF vectorizer to make the text data usable for applying different machine learning algorithms and compare them based on different performance parameters.

```
#Create 80% train test and 20% validation test
X_train, X_test, Y_train, Y_test = train_test_split(X, y, train_size=0.8, random_state=42)
```

```
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
```

```
print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
```

```
(205153, 171130) (51289, 171130) (205153,) (51289,)
```

2.4 Machine Learning Algorithms:

2.4.1 Logistic Regression:

- As the logistic regression, we can use the two dependent variables, also we are using the loss function, so for that reason to predict the three dependent variable, we need the multinomial logistic regression, also it predicts the probability of the distribution.
- We take the max_iter as the hyperparameter for the logistic regression model which is used to solve the maximum number of operations, take until it onverges.
- We evaluate the logistic regression model using the classification report, accuracy, recall score, precision score, and f1 score, and compared these values with those of the other machine learning models.

```
#Logistic Regression Model
lr = LogisticRegression(max_iter = 500)
lr.fit(X_train,Y_train)
Y_pred_lr = lr.predict(X_test)
print(classification_report(Y_test, Y_pred_lr))
#Accuracy Score
print("Accuracy Score: {}".format(accuracy_score(Y_test,Y_pred_lr)))
#Recall score
print("Recall Score: {}".format(recall_score(Y_test,Y_pred_lr,average='weighted')))
#Precision Score
print('Precision Score: {}'.format(precision_score(Y_test,Y_pred_lr,average='weighted')))
#F1 Score
print('F1 Score: {}'.format(f1_score(Y_test,Y_pred_lr,average='weighted')))
#Confusion Matrix
print(confusion_matrix(Y_test,Y_pred_lr))
```

```
              precision    recall  f1-score   support

      agreed       0.75      0.60      0.67     14813
   disagreed       0.77      0.19      0.30      1321
   unrelated       0.82      0.92      0.87     35155

    accuracy                           0.80     51289
   macro avg       0.78      0.57      0.61     51289
weighted avg       0.80      0.80      0.79     51289

Accuracy Score: 0.8047534559067246
Recall Score: 0.8047534559067246
Precision Score: 0.7994297392962509
F1 Score: 0.7933414033568648
[[ 8854    11  5948]
 [   45   245  1031]
 [ 2915    64 32176]]
```

2.4.2 Multinomial Naïve Bayes:

- Multinomial Naïve Bayes is used to do the analysis of text with the multiple cases, and the main advantage of using this algorithm is that it can be easily implemented as we need to calculate just the probability, but when we compared its accuracy with other model algorithm then its probability accuracy is less than the multinomial logistic regression algorithm, so when compared to logistic regression it shows less accuracy.
- We evaluate the multinomial naïve bayes model using the classification report, accuracy, recall score, precision score, and f1 score, and compared these values with those of the other machine learning models.

```
#Multinomial NaiveBayes Model
mnb = MultinomialNB()
mnb.fit(X_train,Y_train)
Y_pred_mnb = mnb.predict(X_test)
print(classification_report(Y_test, Y_pred_mnb))
#Accuracy Score
print("Accuracy Score: {}".format(accuracy_score(Y_test,Y_pred_mnb)))
#Recall score
print("Recall Score: {}".format(recall_score(Y_test,Y_pred_mnb,average='weighted')))
#Precision Score
print('Precision Score: {}'.format(precision_score(Y_test,Y_pred_mnb,average='weighted')))
#F1 Score
print('F1 Score: {}'.format(f1_score(Y_test,Y_pred_mnb,average='weighted')))
#Confusion Matrix
print(confusion_matrix(Y_test,Y_pred_mnb))
```

```
              precision    recall  f1-score   support

      agreed       0.87      0.22      0.35     14813
   disagreed       1.00      0.01      0.02      1321
   unrelated       0.73      0.99      0.84     35155

    accuracy                           0.74     51289
   macro avg       0.87      0.41      0.40     51289
weighted avg       0.78      0.74      0.68     51289

Accuracy Score: 0.7395932851098677
Recall Score: 0.7395932851098677
Precision Score: 0.7770855932526382
F1 Score: 0.6763328701079517
[[ 3240     0 11573]
 [    4    15  1302]
 [  477     0 34678]]
```

2.4.3 Linear Support Vector Classification:

- Linear SVC is similar to Support Vector Classifier but in this case the kernel is set to linear.
- Linear SVC implements the linear kernel in terms of liblinear instead of libsvm and has more flexibility in the choice of applied penalties and the calculated loss functions.
- It is useful for datasets having large numbers of samples and supports both dense and sparse input and has multiclass support based on the one-vs-the-rest technique.
- We evaluate the LinearSVC model using the classification report, accuracy, recall score, precision score, and f1 score, and compared these values with those of the other machine learning models.

```
#LinearSVC
svc = LinearSVC()
svc.fit(X_train,Y_train)
Y_pred_svc = svc.predict(X_test)
print(classification_report(Y_test, Y_pred_svc))
#Accuracy Score
print("Accuracy Score: {}".format(accuracy_score(Y_test,Y_pred_svc)))
#Recall score
print("Recall Score: {}".format(recall_score(Y_test,Y_pred_svc,average='weighted')))
#Precision Score
print('Precision Score: {}'.format(precision_score(Y_test,Y_pred_svc,average='weighted')))
#F1 Score
print('F1 Score: {}'.format(f1_score(Y_test,Y_pred_svc,average='weighted')))
#Confusion Matrix
print(confusion_matrix(Y_test,Y_pred_svc))
```

```
              precision    recall  f1-score   support

      agreed       0.76      0.66      0.70     14813
   disagreed       0.73      0.25      0.38      1321
   unrelated       0.84      0.91      0.87     35155

    accuracy                           0.82     51289
   macro avg       0.78      0.61      0.65     51289
weighted avg       0.81      0.82      0.81     51289

Accuracy Score: 0.8194739612782468
Recall Score: 0.8194739612782468
Precision Score: 0.814375593526316
F1 Score: 0.8120243238968857
[[ 9731    17  5065]
 [   52   334   935]
 [ 3085   105 31965]]
```

2.4.4 Decision Tree Classifier:

- Decision Tree is a supervised ML algorithm capable of both regression and classification.
- It predicts the target class based on the decision rules formed by training on the data features.
- It can create complex trees which might overfit the data they can be biased if some of the target classes are dominant.
- We evaluate the Decision Tree Classifier model using the classification report, accuracy, recall score, precision score, and f1 score, and compared these values with those of the other machine learning models.

```
#DecisionTreeClassifier Model
dt = DecisionTreeClassifier()
dt.fit(X_train,Y_train)
Y_pred_dt = dt.predict(X_test)
print(classification_report(Y_test, Y_pred_dt))
#Accuracy Score
print("Accuracy Score: {}".format(accuracy_score(Y_test,Y_pred_dt)))
#Recall score
print("Recall Score: {}".format(recall_score(Y_test,Y_pred_dt,average='weighted')))
#Precision Score
print('Precision Score: {}'.format(precision_score(Y_test,Y_pred_dt,average='weighted')))
#F1 Score
print('F1 Score: {}'.format(f1_score(Y_test,Y_pred_dt,average='weighted')))
#Confusion Matrix
print(confusion_matrix(Y_test,Y_pred_dt))
```

```
              precision    recall  f1-score   support

       agreed       0.65      0.66      0.65     14813
   disagreed       0.42      0.36      0.39      1321
   unrelated       0.83      0.83      0.83     35155

    accuracy                           0.77     51289
   macro avg       0.63      0.62      0.62     51289
weighted avg       0.77      0.77      0.77     51289

Accuracy Score: 0.7699701690420948
Recall Score: 0.7699701690420948
Precision Score: 0.7696022323865177
F1 Score: 0.7697212175624862
[[ 9753    51  5009]
 [   57   481   783]
 [ 5279   619 29257]]
```

2.4.5 Random Forest Classifier:

- Random Forest is an extension of the decision tree algorithm, as we are using multiple decision trees
- When we are using single decision tree, it has high variance, but here we combine multiple decision trees, so we can get low variance, and as a result the output is not only dependent on single decision tree, it depends on multiple decision trees, thereby we observe reduction in the overfitting issues and get higher accuracy compared to the other models.
- We evaluate the Random Forest Classifier model using the classification report, accuracy, recall score, precision score, and f1 score, and compared these values with those of the other machine learning models.

```
#RandomForestClassifier Model
rf = RandomForestClassifier()
rf.fit(X_train,Y_train)
Y_pred_rf = rf.predict(X_test)
print(classification_report(Y_test, Y_pred_rf))
#Accuracy Score
print("Accuracy Score: {}".format(accuracy_score(Y_test,Y_pred_rf)))
#Recall score
print("Recall Score: {}".format(recall_score(Y_test,Y_pred_rf,average='weighted')))
#Precision Score
print('Precision Score: {}'.format(precision_score(Y_test,Y_pred_rf,average='weighted')))
#F1 Score
print('F1 Score: {}'.format(f1_score(Y_test,Y_pred_rf,average='weighted')))
#Confusion Matrix
print(confusion_matrix(Y_test,Y_pred_rf))
```

```
              precision    recall  f1-score   support

      agreed       0.83      0.69      0.75     14813
   disagreed       0.83      0.26      0.39      1321
   unrelated       0.86      0.94      0.90     35155

    accuracy                           0.85     51289
   macro avg       0.84      0.63      0.68     51289
weighted avg       0.85      0.85      0.84     51289

Accuracy Score: 0.8501823002983095
Recall Score: 0.8501823002983095
Precision Score: 0.8484542597100638
F1 Score: 0.8421384694873852
[[10203    13  4597]
 [   26   338   957]
 [ 2033    58 33064]]
```

2.5 Data Sampling:

2.5.1 Undersampling (RandomUnderSampler):

- Resampling methods are designed to change the composition of a training dataset for an imbalanced classification task.
- Undersampling refers to a technique designed to balance the target class distribution for a classification dataset that has an imbalanced class distribution by deleting data from the majority class.
- RandomUnderSampler() class is used to perform random under-sampling.
- It under samples the majority classes by randomly picking samples with or without replacement.
- It is prone to loss of information as data is being removed.
- Here, we can see that the data is undersampled and the three classes are balanced with each consisting 5285 samples.

```
#Undersampling data
rus = RandomUnderSampler()
X_rus, Y_rus = rus.fit_resample(X_train, Y_train)
Y_rus.value_counts()
```

```
agreed       5285
disagreed    5285
unrelated    5285
```

- We trained the above mentioned machine learning algorithms on the undersampled data and obtained the following performances.

- Logistic Regression:

```
              precision    recall  f1-score   support

      agreed       0.48      0.76      0.59     14813
   disagreed       0.13      0.81      0.22      1321
   unrelated       0.83      0.46      0.59     35155

    accuracy                           0.56     51289
   macro avg       0.48      0.68      0.47     51289
weighted avg       0.71      0.56      0.58     51289

Accuracy Score: 0.5564546004016456
Recall Score: 0.5564546004016456
Precision Score: 0.7141496229686666
F1 Score: 0.5824463999296245
[[11306   481  3026]
 [   78  1067   176]
 [12149  6839 16167]]
```

- Multinomial Naïve Bayes:

```
              precision    recall  f1-score   support

      agreed       0.49      0.69      0.57     14813
   disagreed       0.11      0.79      0.20      1321
   unrelated       0.83      0.49      0.62     35155

    accuracy                           0.56     51289
   macro avg       0.48      0.66      0.46     51289
weighted avg       0.71      0.56      0.59     51289

Accuracy Score: 0.5601006063678372
Recall Score: 0.5601006063678372
Precision Score: 0.7098474145492171
F1 Score: 0.594693119573099
[[10294  1078  3441]
 [   53  1046   222]
 [10778  6990 17387]]
```

- Linear Support Vector Classification:

```
              precision    recall  f1-score   support

      agreed       0.50      0.74      0.60     14813
   disagreed       0.13      0.83      0.22      1321
   unrelated       0.84      0.49      0.62     35155

    accuracy                           0.57     51289
   macro avg       0.49      0.69      0.48     51289
weighted avg       0.72      0.57      0.60     51289

Accuracy Score: 0.5715260582191113
Recall Score: 0.5715260582191113
Precision Score: 0.7193127868835444
F1 Score: 0.6004276574719225
[[11035   555  3223]
 [   61  1102   158]
 [11149  6830 17176]]
```

- Decision Tree Classifier:

```
              precision    recall  f1-score   support

      agreed       0.47      0.66      0.55     14813
   disagreed       0.10      0.75      0.18      1321
   unrelated       0.78      0.45      0.58     35155

    accuracy                           0.52     51289
   macro avg       0.45      0.62      0.43     51289
weighted avg       0.67      0.52      0.56     51289

Accuracy Score: 0.5221197527734992
Recall Score: 0.5221197527734992
Precision Score: 0.6742989455151536
F1 Score: 0.5573251788901403
[[ 9804   789  4220]
 [  105   989   227]
 [10990  8179 15986]]
```

- Random Forest Classifier:

```
              precision    recall  f1-score   support

      agreed       0.46      0.89      0.61     14813
   disagreed       0.10      0.90      0.19      1321
   unrelated       0.89      0.29      0.43     35155

    accuracy                           0.48     51289
   macro avg       0.49      0.69      0.41     51289
weighted avg       0.75      0.48      0.48     51289

Accuracy Score: 0.47653492951705045
Recall Score: 0.47653492951705045
Precision Score: 0.7485107762342317
F1 Score: 0.4777105536399271
[[13179   488  1146]
 [   89  1183    49]
 [15337  9739 10079]]
```

- Here, we observe that the performance for all the machine learning algorithms trained on undersampled data is very poor as compared to the performance for all the machine learning algorithms trained on original/unsampled data.
- This can be because of the loss in information caused by the random deletion of samples in the data.

2.5.2 Oversampling (Synthetic Minority Oversampling Technique):

- Synthetic Minority Oversampling Technique (SMOTE) refers to a technique designed to balance the target class distribution for a classification dataset that has an imbalanced class distribution by synthesizing new samples from the existing ones instead of just duplicating the samples in the minority class as they don't add any new information to the model.
- SMOTE() class is used to perform oversampling using SMOTE.
- Here, we can see that the data is oversampled using SMOTE with sampling_strategy = 'minority' to resample only the minority class which in our case is 'disagreed' with only 6606 samples in the train.csv file.
- SMOTE helps in getting 140443 samples of unrelated class, 140443 samples of disagreed class and 59425 samples of agreed class.

```
#Synthetic Minority Oversampling Technique(SMOTE)
s = SMOTE(sampling_strategy='minority')
X_s, Y_s = s.fit_resample(X_train, Y_train)
Y_s.value_counts()
```

```
unrelated    140443
disagreed    140443
agreed        59425
```

- We trained the above mentioned machine learning algorithms on the oversampled data and obtained the following performances.

- Logistic Regression:

```
              precision    recall  f1-score   support

      agreed       0.75      0.60      0.66     14813
   disagreed       0.29      0.68      0.40      1321
   unrelated       0.83      0.86      0.84     35155

    accuracy                           0.78     51289
   macro avg       0.62      0.71      0.64     51289
weighted avg       0.79      0.78      0.78     51289

Accuracy Score: 0.7780810700150129
Recall Score: 0.7780810700150129
Precision Score: 0.792047066032957
F1 Score: 0.7804441490837397
[[ 8829   213  5771]
 [   22   896   403]
 [ 2977  1996 30182]]
```

- Multinomial Naïve Bayes:

```
                precision    recall  f1-score   support

       agreed       0.87      0.22      0.35     14813
    disagreed       0.18      0.75      0.28      1321
    unrelated       0.73      0.88      0.80     35155

     accuracy                           0.68     51289
    macro avg       0.59      0.61      0.48     51289
 weighted avg       0.76      0.68      0.66     51289

 Accuracy Score: 0.6830704439548441
 Recall Score: 0.6830704439548441
 Precision Score: 0.7593947412906107
 F1 Score: 0.6553855442192174
 [[ 3198   798 10817]
  [    4   985   332]
  [  472  3832 30851]]
```

- Linear Support Vector Classification:

```
               precision    recall  f1-score   support

      agreed       0.76      0.65      0.70     14813
   disagreed       0.40      0.56      0.47      1321
   unrelated       0.85      0.89      0.87     35155

    accuracy                           0.81     51289
   macro avg       0.67      0.70      0.68     51289
weighted avg       0.81      0.81      0.81     51289

Accuracy Score: 0.808516446021564
Recall Score: 0.808516446021564
Precision Score: 0.8093676329317371
F1 Score: 0.807086502201345
[[ 9610   111  5092]
 [   30   737   554]
 [ 3040   994 31121]]
```

- Decision Tree Classifier:

```
              precision    recall  f1-score   support

      agreed       0.66      0.66      0.66     14813
   disagreed       0.31      0.40      0.35      1321
   unrelated       0.84      0.82      0.83     35155

    accuracy                           0.77     51289
   macro avg       0.60      0.63      0.61     51289
weighted avg       0.77      0.77      0.77     51289

Accuracy Score: 0.767357523055626
Recall Score: 0.767357523055626
Precision Score: 0.7716001853469177
F1 Score: 0.769307301489134
[[ 9836    56  4921]
 [   54   528   739]
 [ 5055  1107 28993]]
```

- Random Forest Classifier:

```
              precision    recall  f1-score   support

      agreed       0.85      0.68      0.75     14813
   disagreed       0.67      0.41      0.51      1321
   unrelated       0.86      0.94      0.90     35155

    accuracy                           0.85     51289
   macro avg       0.79      0.68      0.72     51289
weighted avg       0.85      0.85      0.85     51289

Accuracy Score: 0.852288014973971
Recall Score: 0.852288014973971
Precision Score: 0.8500328454577797
F1 Score: 0.8460225998445388
[[10007    28  4778]
 [   12   541   768]
 [ 1757   233 33165]]
```

- Here, we observe that the performance for all the machine learning algorithms trained on oversampled data is quite similar to the performance for all the machine learning algorithms trained on original/unsampled data.

2.6 Prediction on test.csv data:

We import the 'test.csv' file and run the steps mentioned in section 2.2 for text data preprocessing.

```
#Import test Dataset
test=pd.read_csv('/content/test.csv')
```

The first entries of title1_en and title2_en for 'test.csv' file.

```
#Before text processing test data
print(test['title1_en'].iloc[0])
print(test['title2_en'].iloc[0])
```

```
The great coat brother Zhu Zhu Wen, in the mandarin love song to sing the song is really the lanca- talent is very sweet!
Lin xinsheng after the birth of "hard milking," Huo jianhua is not seen, "forced marriage" is real?
```

The first entries of title1_en and title2_en for 'test.csv' file after text preprocessing using nltk library.

```
#After text processing test data
print(test['title1_en'].iloc[0])
print(test['title2_en'].iloc[0])
```

```
great coat brother zhu zhu wen mandarin love song sing song really lanca talent sweet
lin xinsheng birth hard milking huo jianhua seen forced marriage real
```

We combine title1_en and title2_en into titles and apply TfidVectorizer as mentioned in section 2.3

```
test_data = test['titles']
test_data = vectorizer.transform(test_data)
```

Then, we predicted the target classes for the 'test.csv' data using the Random Forest Classifier model trained with original/unsampled data having an accuracy score of 0.85 which is the highest in comparison to all the other models. It also has the most decent performance based on the precision, recall and F1 score compared to the other models.

```
#Predict the output on test data using RandomForestClassifier
y_sub = rf.predict(test_data)
```

We create lists of column 'id' and 'label' to create a dataframe using pandas.

```
#Create id list
id_list = list(test["id"])
```

```
#Create label list
label_list=list(y_sub)
```

Create a 'submission_df' dataframe from the id_list and 'label_list' with the column name as 'id' and 'label'.

```
#Creating submission dataframe
submission_df = pd.DataFrame(list(zip(id_list, label_list)), columns =['id', 'label'])
```

We can see that the ‘submission_df’ dataframe consists of two columns ‘id’ and ‘label’.

```
submission_df.head(10)
```

	id	label
0	256442	unrelated
1	256443	unrelated
2	256444	unrelated
3	256445	unrelated
4	256446	unrelated
5	256447	unrelated
6	256448	unrelated
7	256449	unrelated
8	256450	unrelated
9	256451	unrelated

Finally, we create a ‘submission.csv’ file from the ‘submission_df’ dataframe.

```
#Creating 'submission.csv' file
submission_df.to_csv('submission.csv', header=True, index=False)
```

Section 3: Conclusion

- We performed text preprocessing to extract meaningful information from the data and used Tf-idf Vectorizer to make it usable for different machine learning algorithms.
- We implemented sampling techniques to tackle imbalanced classes.
- We compared the performance of different machine algorithms trained on original, undersampled and oversampled data.
- The machine learning algorithms performed poorly for the undersampled data which could be due to the loss of information in the process of random deletion of samples.
- The machine learning algorithms performed quite similarly for the original and oversampled data.
- The random forest classifier model had an accuracy score of 0.85 which was the best among all the other models.
- Random forest classifier consists of multiple decision trees and as a result, it reduces the problem of overfitting or underfitting the data as compared to a single decision tree and hence, performs better than the decision tree classifier.
- Apart from that, it can efficiently handle a large dataset.
- Also, feature selection is done naturally in random forest, as multiple decision trees are made randomly with no dependency for the target variable on various sub-samples of the dataset.

Section 4: Team Contribution

Task	Done by
Data Exploration and Preprocessing	Harsh & Henil
TF-IDF Vectorizer	Henil
Selecting ML algorithms	Harsh & Henil
Implementing ML algorithms on original data	Harsh & Henil
Implementing undersampling and training ML models	Harsh
Implementing oversampling and training ML models	Henil
Prediction on test.csv and creating submission.csv file	Harsh
Project Report	Harsh & Henil
Project Presentation	Harsh & Henil

Section 5: References

[1] Pandas: https://pandas.pydata.org/
[2] Seaborn: https://seaborn.pydata.org/
[3] Scikit-learn: https://scikit-learn.org/stable/
[4] Imbalanced-learn: https://imbalanced-learn.org/stable/
[5] Natural Language Toolkit (nltk): https://www.nltk.org/
[6] String library: https://docs.python.org/3/library/string.html
[7] Text Preprocessing: NLP Text Preprocessing with NLTK | Towards Data Science
[8] Lemmatization: https://www.geeksforgeeks.org/python-lemmatization-with-nltk/
[9] TfidfVectorizer: sklearn.feature_extraction.text.TfidfVectorizer — scikit-learn 1.2.2 documentation
[10] Logistic Regression: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[11] Multinomial Naïve Bayes Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
[12] Linear Support Vector Classification: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
[13] Decision Tree Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
[14] Random Forest Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[15] RandomUnderSampler: https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html
[16] SMOTE: https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html