ILLINOIS INSTITUTE OF TECHNOLOGY

# *CS579 Online Social Network Analysis*
# *Project 2*

## ***Fake News Classification***


**By:**

**Group 62**

*Sanjitha Reddy Pathuri [A20524383]*
*Mohan Babu Kunchala [A20524765]*

# Table of Contents:

## Introduction

In today's world, Social media has become one of the major resources for people to obtain news and information. For example, it is found that social media now outperforms television as the major news source. However, because it is cheap to provide news online and much faster and easier to disseminate through social media, large volumes of fake news or misinformation are produced online for a variety of purposes, such as financial and political gain.

The extensive spread of fake news/misinformation can have a serious negative impact on individuals and society like below:

(i) breaking the authenticity balance of the news ecosystem;

(ii) intentionally persuading consumers to accept biased or false beliefs;

(iii) changing the way people interpret and respond to real news and information.

Therefore, it is important to detect fake news and misinformation in social media.

In this project, we built a machine learning classifier that detects fake news and misinformation based on the titles of a fake news article A and coming news article B

## Problem Statement

Given the title of a fake news article A and the title of a coming news article B, participants are asked to classify B into one of the three categories:

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

In the doc - Input folder, there are 3 CSV files:

- train.csv: Training data
- test.csv: Test data
- sample submission.csv: Expected submission format

The training data includes the “label” of each news pair, while the test data doesn’t. Validation data can be split from train.csv.

Using the training data to train a classifier and evaluate the model’s performance with the validation data.

Finally, by using the trained model, It is required to predict results for test data.

The format of the output file named as submission.csv should be the same as “sample submission.csv” with the prediction replaced in “label” column.

The columns in train and test data are as follows:

- id: the id of each news pair.
- tid1: the id of fake news title 1.
- tid2: the id of news title 2.
- title1 en: the fake news title 1 in English.
- title2 en: the news title 2 in English.
- label: indicates the relation between the news pair: agreed/disagreed/unrelated.

## Project Organization

```
└── data                                  <- Data files of the project
    ├── Input                             <- Contains all the input csv files
    └── Output                            <- Contains all the output submission/csv files
└── doc
    └── Project_Report.pdf                <- Documentation/Detailed Report of the project
└── src                                   <- Source code of the project
    └── Fake_News_Classification_Final_Submission.ipynb    <- Script to build and run
                                                              different classifier
```

## Tools Used

We can build a machine learning classification model in different programming languages.

But here, we are focusing on classification in Python Programming Language because of its versatility, ease of use and availability of various libraries.

With these libraries, we just have to write a few lines of code and the task will be done.

We used the "pandas", "numpy" and “sklearn” Python libraries to create a simple classification model that detects fake news/ misinformation based on the titles of a fake news article A and coming news article B

## Proposed Solution

Below are the steps followed to build a Machine learning classifier for the problem at hand:

**Step 1: Data pre-processing:**

```
#Import the necessary libraries to build a classifier: pandas,
numpy, sklearn, and nltk
import pandas as pd
import numpy as np
import scipy.sparse as sp
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix, classification_report
import string
import nltk
nltk.download('punkt')
nltk.download('stopwords')

# Load the training data
train_data = pd.read_csv('train.csv')

#Remove any duplicate rows, missing values, or irrelevant columns:
train_data.drop_duplicates(inplace=True)
train_data.dropna(inplace=True)
train_data.drop(columns=['id', 'tid1', 'tid2'], inplace=True)

# Define a function to clean the text: Convert the text data into
lowercase, remove any punctuations, and apply stemming
stopwords = set(nltk.corpus.stopwords.words('english'))
def clean_text(text):
    text = text.lower()
    tokens = nltk.word_tokenize(text)
    tokens=[token for token in tokens if token.isalpha() and token
not in stopwords]
    text = ' '.join(tokens)
    text = ''.join(c for c in text if c.isalnum() or c.isspace())
    return text
```

```
# Preprocess the data by applying the clean_text function
train_data['title1_en_clean']=train_data['title1_en'].apply(clean_t
ext)
train_data['title2_en_clean']=train_data['title2_en'].apply(clean_t
ext)
```

**Step 2: Feature engineering:**

Next, extract relevant features from the pre-processed text data. We'll use the bag-of-words model and TF-IDF vectorizer for this.

```
# Vectorize the text using TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()
X1 = tfidf_vectorizer.fit_transform(train_data['title1_en_clean'])
X2 = tfidf_vectorizer.transform(train_data['title2_en_clean'])
X = sp.hstack((X1, X2))
y = train_data['label']

# Split the preprocessed data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y,
test_size=0.2, random_state=42)
```

**Step 3: Model Training & Evaluation:**

We'll use the different classifier algorithms and Train the model on the training data

```
# Use different classifier algorithms and Train the model on the
train data and then evaluate different metrics for each classifier
on the test data

### Logical Regression

# Train the classifier
lr_classifier = LogisticRegression(random_state=42, max_iter=1000)
lr_classifier.fit(X_train, y_train)

# Evaluate the model
y_pred_lr = lr_classifier.predict(X_val)
accuracy_lr = accuracy_score(y_val, y_pred_lr)
precision_lr = precision_score(y_val, y_pred_lr, average='micro')
recall_lr = recall_score(y_val, y_pred_lr)
f1_score_lr = f1_score(y_val, y_pred_lr)
print('## Logical Regression ##')
```

```
print('Accuracy:', accuracy_lr)
print('Precision:', precision_lr)
print('Recall:', recall_lr)
print('F-1 Score:', f1_score_lr)
print('Confusion Matrix:')
print(confusion_matrix(y_val,y_pred_lr))
print('Classification Report:')
print(classification_report(y_val,y_pred_lr))


### Naive Bayes Classifier

# Train the classifier
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)

# Evaluate the model
y_pred_nb = nb_classifier.predict(X_val)
accuracy_nb = accuracy_score(y_val, y_pred_nb)
precision_nb = precision_score(y_val, y_pred_nb, average='micro')
recall_nb = recall_score(y_val, y_pred_nb)
f1_score_nb = f1_score(y_val, y_pred_nb)
print('## Naive Bayes classifier ##')
print('Accuracy:', accuracy_nb)
print('Precision:', precision_nb)
print('Recall:', recall_nb)
print('F-1 Score:', f1_score_nb)
print('Confusion Matrix:')
print(confusion_matrix(y_val,y_pred_nb))
print('Classification Report:')
print(classification_report(y_val,y_pred_nb))


### Random forest

# Train the classifier
rf_classifier = RandomForestClassifier(n_estimators=25,
random_state=42)
rf_classifier.fit(X_train, y_train)

# Evaluate the model
y_pred_rf = rf_classifier.predict(X_val)
accuracy_rf = accuracy_score(y_val, y_pred_rf)
precision_rf = precision_score(y_val, y_pred_rf, average='micro')
recall_rf = recall_score(y_val, y_pred_rf)
f1_score_rf = f1_score(y_val, y_pred_rf)
```

```
print('## Random Forest ##')
print('Accuracy:', accuracy_rf)
print('Precision:', precision_rf)
print('Recall:', recall_rf)
print('F-1 Score:', f1_score_rf)
print('Confusion Matrix:')
print(confusion_matrix(y_val,y_pred_rf))
print('Classification Report:')
print(classification_report(y_val,y_pred_rf))


### Linear Support Vector Classifier (Linear SVC)

# Train the classifier
svc_classifier = LinearSVC(random_state=42)
svc_classifier(X_train, y_train)

# Evaluate the model
y_pred_svc = svc_classifier.predict(X_val)
accuracy_svc = accuracy_score(y_val, y_pred_svc)
precision_svc = precision_score(y_val, y_pred_svc, average='micro')
recall_svc = recall_score(y_val, y_pred_svc)
f1_score_svc = f1_score(y_val, y_pred_svc)
print('## Support Vector classifier ##')
print('Accuracy:', accuracy_svc)
print('Precision:', precision_svc)
print('Recall:', recall_svc)
print('F-1 Score:', f1_score_svc)
print('Confusion Matrix:')
print(confusion_matrix(y_val,y_pred_svc))
print('Classification Report:')
print(classification_report(y_val,y_pred_svc))


### Stochastic Gradient Descent (SGD) Classifier

# Train the classifier
sgd_classifier = SGDClassifier(random_state=42, max_iter=1000)
sgd_classifier.fit(X_train, y_train)

# Evaluate the model
y_pred_sgd = sgd_classifier.predict(X_val)
accuracy_sgd = accuracy_score(y_val, y_pred_sgd)
precision_sgd = precision_score(y_val, y_pred_sgd, average='micro')
recall_sgd = recall_score(y_val, y_pred_sgd)
f1_score_sgd = f1_score(y_val, y_pred_sgd)
```

```
print('## SGD classifier ##')
print('Accuracy:', accuracy_sgd)
print('Precision:', precision_sgd)
print('Recall:', recall_sgd)
print('F-1 Score:', f1_score_sgd)
print('Confusion Matrix:')
print(confusion_matrix(y_val,y_pred_sgd))
print('Classification Report:')
print(classification_report(y_val,y_pred_sgd))
```

**Step 4: Prediction:**

Load the test data into a Pandas dataframe and pre-process/clean it.

```
# Loads test data, preprocess it using the same steps as train data
test_data = pd.read_csv('test.csv')
test_data.drop(columns=['id', 'tid1', 'tid2'], inplace=True)
test_data['title1_en_clean']=
test_data['title1_en'].apply(clean_text)
test_data['title2_en_clean']=
test_data['title2_en'].apply(clean_text)

# Extract features from the pre-processed test data:
X1_test = tfidf_vectorizer.transform(test_data['title1_en_clean'])
X2_test = tfidf_vectorizer.transform(test_data['title2_en_clean'])
X_test = sp.hstack((X1_test, X2_test))

# Make predictions on the test data using each classifier
y_pred_test_lr = lr_classifier.predict(X_test)
y_pred_test_nb = nb_classifier.predict(X_test)
y_pred_test_rf = rf_classifier.predict(X_test)
y_pred_test_svc = svc_classifier.predict(X_test)
y_pred_test_sgd = sgd_classifier.predict(X_test)
```

**Step 5: Submission:**

```
# Create a submission file for each classifier by saving the
predicted labels for test data in the same format as
sample_submission file

submission = pd.read_csv('sample_submission.csv')

submission['label'] = y_pred_test_lf
submission.to_csv('submission_lf.csv', index=False)
```

```
submission['label'] = y_pred_test_nb
submission.to_csv('submission_nb.csv', index=False)

submission['label'] = y_pred_test_svc
submission.to_csv('submission_svc.csv', index=False)

submission['label'] = y_pred_test_sgd
submission.to_csv('submission_sgd.csv', index=False)

#Saving the Random forest classifier as the actual submission file as
the accuracy is more than the other classifiers used

submission['label'] = y_pred_test_rf
submission.to_csv('submission_rf.csv', index=False)
```

## Methodology

We devised a model that employs NLP methods and machine learning algorithms to categorize news article B into one of three categories: agreed, disagreed and unrelated. We trained the model with a training data set and then tested it on given test data and also evaluated the performance of the model on various metrics.

The steps we followed to create the model are listed below:

1. Data preprocessing using NLP techniques
2. Bag of words and TF-IDF transformer
3. Model training using machine learning algorithms
4. Model testing
5. Model evaluation using different evaluation metrics

1. *Data preprocessing using NLP techniques*

In this phase, we used NLP preprocessing techniques on both the train and test datasets to make them lighter. Using such techniques, we kept only the information that is helpful and has some significance and removed the data that is irrelevant to prediction. The approaches used with the NLTK library functions are listed below:

- Remove any duplicate rows, missing values, or irrelevant columns
- Converting strings to lowercase: Because lowercase and uppercase words are handled differently by machines, changing the cases of the words will enable the machine to better interpret the text.
- Removal of stop words: Stop words are words which are not important for your text and occur frequently like the, an, they, there, etc.
- Removal of punctuation marks: Punctuation marks like commas, exclamation marks, question marks, etc are removed as they are not going to add any value to the text
- Lemmatization: The words are reduced to their root form during this procedure.
- Define a function to "clean_text" for the above preprocess steps and Preprocess the data by applying the clean_text function

*2. Bag of words and TF-IDF transformer*

Extract relevant features from the pre-processed text data using the bag-of-words model and TF-IDF vectorizer, Vectorize the text using TfidfVectorizer.

Bag of words generates a collection of vectors containing the number of word occurrences in the tex/document. We used scikit-learn's CountVectorizer for this. The term frequency - inverse document frequency (TF-IDF) is used to determine the importance of a word in an assortment of documents/texts. We used scikit-learn's TfidfTransformer to accomplish this.

$$TF(t,d) = \frac{number\ of\ times\ t\ appears\ in\ d}{total\ number\ of\ terms\ in\ d}$$
$$IDF(t) = log\frac{N}{1+df}$$
$$TF-IDF(t,d) = TF(t,d) * IDF(t)$$

*Tf-idf formula*

*3. Model training using multiple machine learning algorithms*

Split the preprocessed training data into training(80%) and validation(20%) sets. Using the initial training dataset, train the model using different classification models.

## Algorithms Used

We used 5 different machine learning algorithms listed below:

- **Logistic Regression:** A statistical model that predicts the probability of a binary or categorical outcome based on one or more predictor variables. It uses a logistic function to transform a linear regression output into a probability value
- **Naive Bayes:** A probabilistic model that uses Bayes' theorem to calculate the probability of a hypothesis given observed evidence. It assumes independence between the features and can be trained quickly on large datasets.
- **Random Forest:** A decision tree-based ensemble model that combines multiple decision trees to improve accuracy and reduce overfitting. It randomly selects subsets of features and data samples to create each tree
- **Linear Support Vector Machine(Linear SVC):** A binary classification model that separates two classes using a linear hyperplane. It finds the hyperplane that maximizes the margin between the closest points of each class.
- **Stochastic Gradient Descent(SGD):** An optimization algorithm commonly used for training machine learning models. It iteratively adjusts the model's parameters to minimize a loss function by computing the gradient of lthe oss with respect to the parameters.

## Importance of the Algorithms Used

The algorithms mentioned above are some of the most popular and widely used classifier models in machine learning. Each of these models has its own unique characteristics and assumptions about the data, making them useful in different situations. It's important to note, however, that there is no one-size-fits-all approach in machine learning, and the choice of a classifier model depends on the specific problem at hand and the characteristics of the data.

## Comparison of the Algorithms Used

Each model has its own strengths and weaknesses, and the best choice depends on the specific problem and the characteristics of the data. Logistic Regression and Linear SVM are good choices for linearly separable data, Naive Bayes is a good choice for high-dimensional data, Random Forest can handle non-linear relationships, and SGD can handle large datasets and sparse data.

Below is a quick comparison of the 5 Machine Learning algorithms used:

| Algorithm | Pros | Cons |
|---|---|---|
| Logistic Regression | Simple and fast to train and predict. Works well with linearly separable data. | Assumes a linear relationship between the features and the outcome. |
| Naive Bayes | Very fast to train and predict. Performs well with high-dimensional data and small training sets. | Assumes that features are independent of each other, which may not be true. |
| Random Forest | Can handle complex, non-linear relationships between features and outcome, reduce overfitting by using multiple trees. | Can be slow to train and predict, especially with large datasets. |
| Linear Support Vector Machine (SVC) | Can handle high-dimensional data. Works well with linearly separable data. | Can be sensitive to outliers. |
| Stochastic Gradient Descent (SGD) | Can handle large datasets and high-dimensional data, converge quickly, especially with sparse data. | Can be sensitive to the learning rate and the initial weights. |

*4. Model testing*

We used our trained model using the training data (80% of train.csv data)to predict results for the validation data(20% of train.csv data) to evaluate their performance and select the best model to predict labels for our test dataset(test.csv).

*5. Model evaluation using different evaluation metrics*

We then evaluated the models using the validation data. Compare the performance of different models using various metrics like:

- Accuracy
- Precision
- Recall
- F-1 score
- Confusion matrix and Classification Report

## Results

```
## Logical Regression ##
Accuracy: 0.8051823977851001
Precision: 0.8006534791283743
Recall: 0.8051823977851001
F-1 Score: 0.797999093990401
Confusion Matrix:
[[ 9394     2  5417]
 [   26   367   928]
 [ 3526    93 31536]]
Classification Report:
              precision    recall  f1-score   support

      agreed       0.73      0.63      0.68     14813
   disagreed       0.79      0.28      0.41      1321
   unrelated       0.83      0.90      0.86     35155

    accuracy                           0.81     51289
   macro avg       0.78      0.60      0.65     51289
weighted avg       0.80      0.81      0.80     51289
```

```
## Naive Bayes classifier ##
Accuracy: 0.7679034490826493
Precision: 0.7602843815177308
Recall: 0.7679034490826493
F-1 Score: 0.7581086317566167
Confusion Matrix:
[[ 8661    27  6125]
 [   12   174  1135]
 [ 4546    59 30550]]
Classification Report:
              precision    recall  f1-score   support

      agreed       0.66      0.58      0.62     14813
   disagreed       0.67      0.13      0.22      1321
   unrelated       0.81      0.87      0.84     35155

    accuracy                           0.77     51289
   macro avg       0.71      0.53      0.56     51289
weighted avg       0.76      0.77      0.76     51289
```

```
## Random Forest ##
Accuracy: 0.8484275380685917
Precision: 0.8459677925732622
Recall: 0.8484275380685917
F-1 Score: 0.8415312044227989
Confusion Matrix:
[[10140    12  4661]
 [   10   431   880]
 [ 2107   104 32944]]
Classification Report:
              precision    recall  f1-score   support

      agreed       0.83      0.68      0.75     14813
   disagreed       0.79      0.33      0.46      1321
   unrelated       0.86      0.94      0.89     35155

    accuracy                           0.85     51289
   macro avg       0.82      0.65      0.70     51289
weighted avg       0.85      0.85      0.84     51289
```

```
## Support Vector classifier ##
Accuracy: 0.8098227690147985
Precision: 0.8055554635157197
Recall: 0.8098227690147985
F-1 Score: 0.8053846585835107
Confusion Matrix:
[[ 9962    11  4840]
 [   29   438   854]
 [ 3850   170 31135]]
Classification Report:
              precision    recall  f1-score   support

      agreed       0.72      0.67      0.70     14813
   disagreed       0.71      0.33      0.45      1321
   unrelated       0.85      0.89      0.87     35155

    accuracy                           0.81     51289
   macro avg       0.76      0.63      0.67     51289
weighted avg       0.81      0.81      0.81     51289
```

```
## SGD classifier ##
Accuracy: 0.7571604047651543
Precision: 0.7715733532607736
Recall: 0.7571604047651543
F-1 Score: 0.713641545190657
Confusion Matrix:
[[ 4665     6 10142]
 [    7    77  1237]
 [ 1058     5 34092]]
Classification Report:
              precision    recall  f1-score   support

      agreed       0.81      0.31      0.45     14813
   disagreed       0.88      0.06      0.11      1321
   unrelated       0.75      0.97      0.85     35155

    accuracy                           0.76     51289
   macro avg       0.81      0.45      0.47     51289
weighted avg       0.77      0.76      0.71     51289
```

## Result Metrics

We can evaluate the Model's performance using different evaluation metrics like accuracy, precision, recall, F1-score

| Algorithm | Accuracy | Precision | Recall | F1-Score |
|---|---|---|---|---|
| Logistic Regression | 0.81 | 0.80 | 0.81 | 0.80 |
| Naive Bayes | 0.77 | 0.76 | 0.77 | 0.76 |
| Random Forest | 0.85 | 0.85 | 0.85 | 0.84 |
| Linear Support Vector Machine (SVC) | 0.81 | 0.81 | 0.81 | 0.81 |
| Stochastic Gradient Descent (SGD) | 0.76 | 0.77 | 0.76 | 0.71 |

Detailed Result Metrics for each data classification i.e. Agreed, Disagreed, Unrelated

| Algorithm | Data classification | Precision | Recall | F1-Score |
|---|---|---|---|---|
| Logistic Regression | Agreed<br>disagreed<br>unrelated | 0.73<br>0.79<br>0.83 | 0.63<br>0.28<br>0.90 | 0.68<br>0.41<br>0.86 |
| Naive Bayes | Agreed<br>disagreed<br>unrelated | 0.66<br>0.67<br>0.81 | 0.58<br>0.13<br>0.87 | 0.62<br>0.22<br>0.84 |
| Random Forest | Agreed<br>disagreed<br>unrelated | 0.83<br>0.79<br>0.86 | 0.68<br>0.33<br>0.94 | 0.75<br>0.46<br>0.89 |
| Linear Support Vector Machine (SVC) | Agreed<br>disagreed<br>unrelated | 0.72<br>0.71<br>0.85 | 0.67<br>0.33<br>0.89 | 0.70<br>0.45<br>0.87 |
| Stochastic Gradient Descent (SGD) | Agreed<br>disagreed<br>unrelated | 0.81<br>0.88<br>0.75 | 0.31<br>0.06<br>0.97 | 0.45<br>0.11<br>0.85 |

## Conclusion

We are able to build a model with 85% accuracy that can classify whether a given news B speaks about the same fake news as A or not by using various machine learning algorithms.

We generated the final Submission Report using the "Random Forest Algorithm" as it has the greatest accuracy compared to the other algorithms we used.

## References


https://blog.quantinsti.com/machine-learning-classification-strategy-python/

https://www.researchgate.net/publication/358015768_Fake_News_Classification_Based_on_Content_Level_Features

https://florence-egwu.medium.com/data-cleaning-and-preprocessing-with-python-7a9d54643a52

https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM

NLTK. Bird, Steven, Edward Loper and Ewan Klein (2009), Natural Language Processing with Python. O'Reilly Media Inc

Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.