American International University-Bangladesh (AIUB)

Faculty of Science & Technology (FST)

Department of Computer Science

Natural Language Processing

Mid-Term Project Report

Fall 2025-2026

Section: B

Group:08

SL #	Student Name	Student ID	Contribution (%)
1.	MD. MARUF HASAN CHOWDHURY	22-48582-3	25%
2.	DIBAJIT ROY	22-48569-3	25%
3.	A.K.M SHAHRIAR EMTIAZ	22-48012-2	25%
4.	MD. OSMAN GONI	22-48535-3	25%

Dataset Description

The dataset selected for this study is a fake news detection dataset form Kaggle. The dataset consists of two separate CSV files-fake.csv and true.csv. Both files contain real examples of news articles published online. Since the dataset is completely raw, no cleaning, filtering or formatting has been performed. As a result, the articles are exactly as published. Hence, the dataset is suitable for analyzing the natural writing style of true and false news.

The fake.csv file contains a total of 23,481 fake articles. The true.csv file contains a total of 21,417 true articles. Both files have four main columns:

Title: Here are the headlines of the news.

Text: Here is the full text of the news, where information, explanations, opinions and various claims are presented in detail.

Subject: News topic or category-such as politics, world news, social issues, etc.

Date: Date of publication-the day the article was published.

These two files combine to create a comprehensive, diverse and realistic news dataset. Because it contains articles from different time periods, different categories, and natural writing styles, this dataset provides a solid foundation for building Naive Bayes models for detecting fake news or misleading information.

Dataset Source link: https://www.kaggle.com/datasets/bhavikjikadara/fake-news-detection

Project Implementation Detail

Project Task 1:

Title of the task1: similarity between two texts using the minimum edit distance algorithm

Description of the task1:

In this task, we used Minimum Edit Distance Algorithm to find Similarity between two texts.

Our main objective was-

- Take two input texts
- Find Minimum Edit Distance (MED)
- Find similarity score from Distance
- Show output on console

Minimum Edit Distance is an algorithm that calculates a cost by calculating three types of operations: Insert, Delete, or Substitute to make two strings similar to each other. The smaller this distance, the more similar the two texts are. We found MED using Dynamic Programming, because it can compare strings quickly and accurately.

Code for task 1:

```python
def min_edit_distance(s1, s2):
    m = len(s1)
    n = len(s2)
    dp = [[0 for _ in range(n+1)] for _ in range(m+1)]
    for i in range(m+1):
        dp[i][0] = i
    for j in range(n+1):
        dp[0][j] = j
    for i in range(1, m+1):
        for j in range(1, n+1):
            if s1[i-1] == s2[j-1]:
                cost = 0
            else:
                cost = 1
            dp[i][j] = min(
                dp[i-1][j] + 1,
                dp[i][j-1] + 1,
                dp[i-1][j-1] + cost
            )
    return dp[m][n]

text1 = input("Enter first text: ")
text2 = input("Enter second text: ")
distance = min_edit_distance(text1, text2)
similarity = 1 - (distance / max(len(text1), len(text2)))

print("Minimum Edit Distance:", distance)
print("Similarity:", round(similarity, 4))
```

Sample output of task 1:

```
Enter first text: nlpproject
Enter second text: nplprejact
Minimum Edit Distance: 4
Similarity: 0.6
```

Code description of task1:

1.Creating a Function — min_edit_distance (s1, s2):

Here we have two text inputs. Their lengths are created using m and n. Then we create a Dynamic Programming table (dp matrix) where:

- dp[i][j] = number of edits required to make the first i character of s1 and the first j character of s2 equal.

2. Initialization part:

The first row and first column are pre-filled because:

- First row = insertion only
- First column = deletion only

This part sets the base case of the DP matrix.

3. Filling the DP Table

We compare each character of the string. If the character is the same → cost = 0.If the character is different → cost = 1Then three operations are calculated:

- Deletion → dp[i-1][j] + 1
- Insertion → dp[i][j-1] + 1
- Substitution → dp[i-1][j-1] + cost

The minimum of these is placed in dp[i][j].

This part runs the entire Minimum Edit Distance algorithm.

4. Similarity Calculation:

After getting the distance, similarity is calculated using this formula:

$$\text{Similarity}=1-\frac{Distance}{\max\left(len(s1),len(s2)\right)}$$

That is, the less different the two texts are, the closer the similarity will be to 1. Finally, we have printed the similarity up to four decimal places.

Project Task 2:

Title of then task 1: Initial Setup & Library import

Description of the task1:

In this task, we have imported the necessary Python libraries for the Fake News Detection project. These libraries will be required to complete the next steps: data preprocessing, TF-IDF vectorization, Naïve Bayes model training,etc.This task is mainly about preparing the environment and preparing the NLP tools.

Code For task 1:

```
[3]
✓ 6s
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.tokenize import word_tokenize
import re
import string
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
```

Sample output of task 1:

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

Code description of task1:

1. Library Imports

Data Handling:

- We have imported pandas and numpy.
- pandas is used for dataset loading, manipulation and analysis.
- numpy is used to simplify numerical operations.

Text Preprocessing:

nltk library has been imported to perform text processing tasks.

We used the following modules:

- stopwords : to remove common, non-informative words (e.g. "the", "is", "in")
- word_tokenize : to split text into individual words
- PorterStemmer and WordNetLemmatizer : to reduce words to base form
- re and string : for punctuation removal, text cleaning, regex operations

Machine Learning:

- train_test_split : to split dataset into training and testing sets
- TfidfVectorizer :to convert text to numerical vectors
- MultinomialNB : to apply Naïve Bayes classification algorithm

Model Evaluation & Visualization:

- accuracy_score, confusion_matrix, classification_report, ConfusionMatrixDisplay : to evaluate and visualize model performance
- matplotlib.pyplot : to create confusion matrix and other plots

2. Downloading NLTK Resources

We used nltk.download() to download the necessary datasets:

- punkt: tokenization For
- stopwords: for removing common words
- wordnet: for lemmatization (e.g. "better" → "good")

Title of then task 2: Dataset Loading

Description of the task2:

In this step, we loaded the dataset for the Fake News Detection project. We used two CSV files:

fake.csv: Fake news dataset

true.csv: True news dataset

We loaded the dataset into the Colab environment using pandas.read_csv(). Next, we checked the shape of the dataset to make sure that the dataset was loaded correctly.

Code for task 2:

```
fake = pd.read_csv('/content/drive/MyDrive/Fake_news_detection/fake.csv')
true = pd.read_csv('/content/drive/MyDrive/Fake_news_detection/true.csv')

print("Fake shape:", fake.shape)
print("True shape:", true.shape)
```

Sample output of task 2:

```
Fake shape: (23481, 4)
True shape: (21417, 4)
```

Code description of task2:

- pd.read_csv() : Reads data from CSV file and loads it into pandas DataFrame.
- print(... .shape) :Displays the number of rows and columns in each DataFrame.
- Through this step, we have confirmed that the dataset has been successfully loaded and is ready for further preprocessing and model training.

Title of then task 3: Dataset Merge and Labeling

Description of the task 3:

In this step we:

1.Label – We gave each dataset a label to identify it:

- fake.csv = "fake"
- true.csv ="true"

2.Dataset Merge – We combined the two datasets vertically using pandas.concat().

3.Shuffling – We shuffled the datasets in random order using sample(frac=1).

4.Reset Index – We created a new index using reset_index(drop=True), which has a sequential index.

With this step, we have prepared a combined, labeled, shuffled dataset, which will be used for model training.

Code for 3:

```
[ ]
fake['label'] = 'fake'
true['label'] = 'true'

data = pd.concat([fake, true], axis=0).reset_index(drop=True)
data = data.sample(frac=1).reset_index(drop=True)
data.head()
```

Sample output of task 3:

	title	text	subject	date	label
0	China says it will make efforts on Syria recon...	BEIJING (Reuters) - China hopes Syria can show...	worldnews	November 24, 2017	true
1	Trump Tries To Throw Michelle Obama Under The...	You would think that after President Barack Ob...	News	October 21, 2016	fake
2	In push for Yemen aid, U.S. warned Saudis of t...	WASHINGTON (Reuters) - The United States has w...	worldnews	December 8, 2017	true
3	Boiler Room EP #119 – Zombie Disneyland & The ...	Tune in to the Alternate Current Radio Network...	Middle-east	July 29, 2017	fake
4	The Weather Channel Just SLAMMED Breitbart Fo...	Right-wingers everywhere have been trying to d...	News	December 6, 2016	fake

Code description of task3:

1. **fake['label'] = 'fake'**:
   - Here we give each row of the fake news dataset a label called "fake", so that later the model can easily understand which one is fake.
2. **true['label'] = 'true':**
   - Similarly, we give the rows of the real news dataset a label "true", so that the two datasets are clearly separated.
3. **pd.concat([fake, true], axis=0):**
   - With this line, we are joining the fake and true datasets together, bottom to bottom. As a result, a large combined dataset is created.
4. **.sample(frac=1):**
   - We are mixing the entire dataset in a completely random order. In this, the model sees mixed data without any bias.
5. **reset_index(drop=True):**
   - After shuffling, the values of the indexes are random. So we are creating a new clean index by dropping the old index.
6. **data.head():**
   - Then we look at the first few rows to make sure that the dataset has been merged, shuffled and labeled correctly.

Title of then task 4: Dataset Cleaning and Preprocessing

Description of the task4:

In this step, we preprocessed the collected fake and true news data for the Fake News Detection project to make it clean, accurate, and model-ready. Machine Learning models can learn well-so duplicate text, missing values, URLs-only rows, spam-like text, and very short news content were removed from the dataset. After that, we applied basic text cleaning (URL remove, HTML tag remove, extra space remove) functions and created a clean clean_text column. This will serve as the basis for future Text Preprocessing (tokenization, stopwords removal, stemming/lemmatization, TF-IDF) and model training.

Code for 4:

```
df = pd.read_csv("/content/drive/MyDrive/Fake_news_detection/merged_fake_true_news.csv")
df.drop_duplicates(subset=['text'], inplace=True)
df.dropna(subset=['text'], inplace=True)
df = df[df['text'].str.len() > 20]
df = df[~df['text'].str.match(r'^[0-9\W]+$', na=False)]
df = df[~df['text'].str.contains(r'^http', na=False)]
df = df[~df['text'].str.contains("subscribe|click here|read more|follow us", case=False)]

def clean_basic(text):
    text = re.sub(r"http\S+|www.\S+", " ", text)
    text = re.sub(r"<.*?>", " ", text)
    text = re.sub(r"\s+", " ", text)
    return text.strip()

df["clean_text"] = df["text"].apply(clean_basic)
df.reset_index(drop=True, inplace=True)
df.head()
```

Sample output of task 4:

	title	text	subject	date	label	clean_text
0	China says it will make efforts on Syria recon...	BEIJING (Reuters) - China hopes Syria can show...	worldnews	November 24, 2017	true	BEIJING (Reuters) - China hopes Syria can show...
1	Trump Tries To Throw Michelle Obama Under The...	You would think that after President Barack Ob...	News	October 21, 2016	fake	You would think that after President Barack Ob...
2	In push for Yemen aid, U.S. warned Saudis of t...	WASHINGTON (Reuters) - The United States has w...	worldnews	December 8, 2017	true	WASHINGTON (Reuters) - The United States has w...
3	Boiler Room EP #119 – Zombie Disneyland & The ...	Tune in to the Alternate Current Radio Network...	Middle-east	July 29, 2017	fake	Tune in to the Alternate Current Radio Network...
4	The Weather Channel Just SLAMMED Breitbart Fo...	Right-wingers everywhere have been trying to d...	News	December 6, 2016	fake	Right-wingers everywhere have been trying to d...

Code description of task 4:

1. **Dataset Load:**
   - We read the merged fake and true news dataset to start the analysis.
2. **Duplicate Removal:**
   - The model can be biased incorrectly if the same news is repeated.Therefore, duplicate rows are removed based on the text column.
3. **Remove Empty Text:**
   - Rows that do not contain text (NaN) are removed, as they are not meaningful in training.
4. **Short Text Remove:**
   - News with less than 20 characters are usually not meaningful, so they are filtered out.
5. **Non-Text Rows Remove:**
   - Rows that contain only symbols (!!??) or only numbers—they are inconsistent, so they are removed.
6. **URL-only Rows Remove:**
   - News that are only “http://...”-so not real news, and create noise—are removed.
7. **Spam-like News Remove:**
   - “subscribe”, “click here”, “read more”, “follow us” type promotional content is removed.
8. **Basic Cleaning Function:**
   - From the text using the clean_basic() function:URLs,HTML tags,Extra spaces.All removed and the clean version placed in the clean_text column.
9. **Reset Index:**
   - After dropping rows, gaps were created in the indexes of the dataset .a new index was created to fix them.
10. **Preview:**
    - The first 5 cleaned rows were viewed with df.head(),to ensure that the dataset was preprocessed correctly.

Title of then task 5: Case folding

Description of the task5:

In this step, we applied case folding as part of the text processing in the dataset. Case folding is the process of converting all the letters in the text to lowercase. Its main goals are:

- To treat different forms of the same word (e.g. “NEWS”, “News”, “news”) as one word
- To reduce vocabulary size
- To make it easier for machine learning models to learn patterns from data
- To maintain data normalization and preprocessing standards

Code for task 5:

```
[ ]
    df["clean_text_lower"] = df["clean_text"].str.lower()
    df[["clean_text", "clean_text_lower"]].head()
```

Sample output of task 5:

	clean_text	clean_text_lower
0	BEIJING (Reuters) - China hopes Syria can show...	beijing (reuters) - china hopes syria can show...
1	You would think that after President Barack Ob...	you would think that after president barack ob...
2	WASHINGTON (Reuters) - The United States has w...	washington (reuters) - the united states has w...
3	Tune in to the Alternate Current Radio Network...	tune in to the alternate current radio network...
4	Right-wingers everywhere have been trying to d...	right-wingers everywhere have been trying to d...

Code description of task5:

1. **df["clean_text_lower"] = df["clean_text"].str.lower()**
    - Here we convert each sentence in the clean_text column to lowercase..str.lower() is used as part of text normalization.We store the result in a new column called clean_text_lower. This reduces the vocabulary of the model and maintains consistency.
2. **df[["clean_text", "clean_text_lower"]].head()**
    - With this line we see the original text and lowercase text side by side. It is used to verify whether the case folding is done correctly.

Title of then task 6: Punctuation removal

Description of the task 6:

In this task, we remove punctuation from the text to leave only meaningful content. Punctuation marks such as commas (,), periods (.), exclamation points (!), question marks (?) etc. do not help to convey sentiment or news content. Removing them normalizes the text and simplifies subsequent preprocessing steps (such as tokenization, stopwords removal, TF-IDF).This is an important clean-up step in the NLP pipeline.

Code for task 6 :

```
punct_mapping = str.maketrans("", "", string.punctuation)
df["clean_text_no_punct"] = df["clean_text_lower"].apply(lambda x: x.translate(punct_mapping))
df["clean_text_no_punct"] = df["clean_text_no_punct"].str.strip()
df[["clean_text_lower", "clean_text_no_punct"]].head()
```

Sample output task 6:

	clean_text_lower	clean_text_no_punct
0	beijing (reuters) - china hopes syria can show...	beijing reuters china hopes syria can show fl...
1	you would think that after president barack ob...	you would think that after president barack ob...
2	washington (reuters) - the united states has w...	washington reuters the united states has warn...
3	tune in to the alternate current radio network...	tune in to the alternate current radio network...
4	right-wingers everywhere have been trying to d...	rightwingers everywhere have been trying to di...

Code description of task6:

1. **Removing Punctuation:**
   - A mapping to remove punctuation is created using str.maketrans("", "", string.punctuation).The string.punctuation constant contains all punctuation characters (., !, ?, etc.).
2. **Applying the Removal:**
   - The punctuation removal process is applied to each row of the clean_text_lower column using the apply() function.The result is stored in a new column clean_text_no_punct.
3. **Displaying the Output:**
   - The first 5 rows are viewed using the head() function to verify that the punctuation has been removed.

Title of then task 7: Tokenization

Description of the task 7:

In this task, we have divided the text into small tokens (individual words).Tokenization is done because:

- Machine Learning models can take word-based features or embeddings.It is convenient to do text analysis (e.g. TF-IDF, word counts).Important preprocessing step of NLP pipeline.

We used punctuation removed text (clean_text_no_punct) and converted each sentence/line into a word-list.

Code for task 7:

```
df["tokens"] = df["clean_text_no_punct"].apply(lambda x: x.split())
df[["clean_text_no_punct", "tokens"]].head()
```

Sample output of task 7:

	clean_text_no_punct	tokens
0	beijing reuters china hopes syria can show fl...	[beijing, reuters, china, hopes, syria, can, s...
1	you would think that after president barack ob...	[you, would, think, that, after, president, ba...
2	washington reuters the united states has warn...	[washington, reuters, the, united, states, has...
3	tune in to the alternate current radio network...	[tune, in, to, the, alternate, current, radio,...
4	rightwingers everywhere have been trying to di...	[rightwingers, everywhere, have, been, trying,...

Code description of task 7:

1. **Tokenization:**
   - Each sentence/line is created by separating individual words with a space using x.split().
2. **Applying the Tokenization:**
   - Apply(lambda x: x.split()) to each row of the clean_text_no_punct column.
3. **Storing the Result:**
   - The token-list is stored in the new column tokens.
4. **Displaying the Output:**
   - The first 5 rows are displayed using df[["clean_text_no_punct", "tokens"]].head() for verification.

Title of then task 8: Stop words removal

Description of the task 8:

In this task, we removed stopwords from the text.Stopwords Removal was done because:

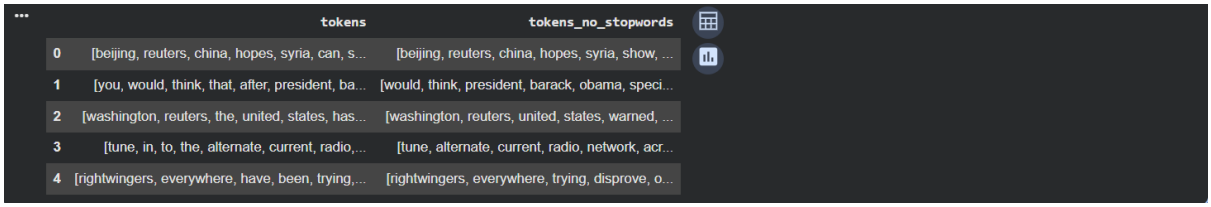
- Stopwords are very common, frequent words such as: the, is, in, a, of.They do not help to understand the main meaning or context of the text.Removing them improves the performance of the model and reduces the feature space.

We used tokenized text (tokens) and created a new column excluding stopwords.

Code for task 8:

```
stop_words = set(stopwords.words("english"))
df["tokens_no_stopwords"] = df["tokens"].apply(
    lambda words: [word for word in words if word not in stop_words]
)
df[["tokens", "tokens_no_stopwords"]].head()
```

Sample output of task 8:



	tokens	tokens_no_stopwords
0	[beijing, reuters, china, hopes, syria, can, s...	[beijing, reuters, china, hopes, syria, show, ...
1	[you, would, think, that, after, president, ba...	[would, think, president, barack, obama, speci...
2	[washington, reuters, the, united, states, has...	[washington, reuters, united, states, warned, ...
3	[tune, in, to, the, alternate, current, radio,...	[tune, alternate, current, radio, network, acr...
4	[rightwingers, everywhere, have, been, trying,...	[rightwingers, everywhere, trying, disprove, o...

Code description of task 8:

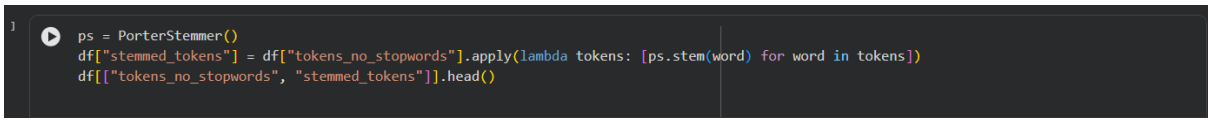
1. **Loading Stopwords:**
   - English stopwords are loaded using stopwords. Words("english"). Converted for quick lookup using set().
2. **Removing Stopwords:**
   - Process each row of the tokens column using the apply() function. Only non-stopwords words are retained using lambda.
3. **Storing the Result:**
   - Store the result in a new column tokens_no_stopwords.
4. **Displaying the Output:**
   - Display the first 5 rows with df[["tokens", "tokens_no_stopwords"]].head() to verify that the stopwords have been removed.

Title of the task 9: Stemming

Description of the task 9:

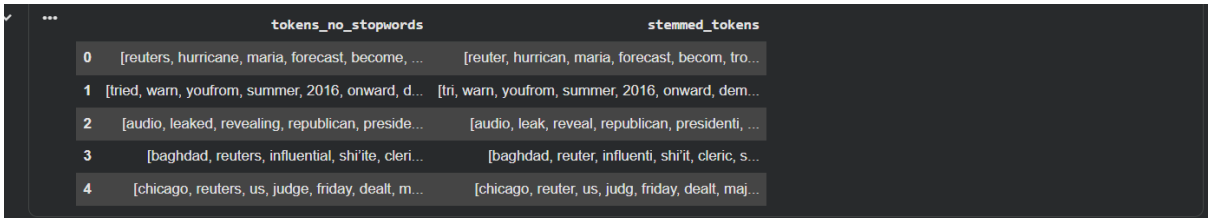
In this step, we apply stemming to the tokens obtained after removing stopwords. Stemming is the process of converting a word to its original or root form. For example, “playing”, “played”, “plays”-all are reduced to the root form “play”. For this, we used PorterStemmer from the nltk library. Stemming makes the text more uniform, reduces word variation, and makes the data cleaner and more useful for sentiment analysis or modeling. Stemmed words are stored in a new column called “stemmed_tokens”.

Code for task 9:



```
ps = PorterStemmer()
df["stemmed_tokens"] = df["tokens_no_stopwords"].apply(lambda tokens: [ps.stem(word) for word in tokens])
df[["tokens_no_stopwords", "stemmed_tokens"]].head()
```

Sample output of task 9:



	tokens_no_stopwords	stemmed_tokens
0	[reuters, hurricane, maria, forecast, become, ...	[reuter, hurrican, maria, forecast, becom, tro...
1	[tried, warn, youfrom, summer, 2016, onward, d...	[tri, warn, youfrom, summer, 2016, onward, dem...
2	[audio, leaked, revealing, republican, preside...	[audio, leak, reveal, republican, presidenti, ...
3	[baghdad, reuters, influential, shi'ite, cleri...	[baghdad, reuter, influenti, shi'it, cleric, s...
4	[chicago, reuters, us, judge, friday, dealt, m...	[chicago, reuter, us, judg, friday, dealt, maj...

Code description of task 9:

1. **PorterStemmer Object Creation:**
   - A stemmer object is created using ps = PorterStemmer().
2. **Applying Stemming:**
   - Each word in the token list is converted to its stem form using ps.stem(word).
3. **Storing the Results:**
   - The stemmed words are stored in a new column stemmed_tokens.
4. **Previewing Output:**
   - The first few rows are viewed using head().

Title of the task 10: Lemmatization

Description of the task 10:

In this step, we apply lemmatization to the stemmed tokens. Lemmatization is more accurate and linguistically meaningful than stemming. It converts words into their base form or lemma, such as nouns, verbs, adjectives. For example, “better” -“good”, which helps the model recognize different word forms as the same base word. Applying lemmatization makes the text more normalized and linguistically accurate.

Code for task 10:

```
[ ]
lemmatizer = WordNetLemmatizer()
df["lemmatized_tokens"] = df["stemmed_tokens"].apply(lambda tokens: [lemmatizer.lemmatize(word) for word in tokens])
df[["stemmed_tokens", "lemmatized_tokens"]].head()
```

Sample output of task 10:

	stemmed_tokens	lemmatized_tokens
0	[reuter, hurrican, maria, forecast, becom, tro...	[reuter, hurrican, maria, forecast, becom, tro...
1	[tri, warn, youfrom, summer, 2016, onward, dem...	[tri, warn, youfrom, summer, 2016, onward, dem...
2	[audio, leak, reveal, republican, presidenti, ...	[audio, leak, reveal, republican, presidenti, ...
3	[baghdad, reuter, influenti, shi'it, cleric, s...	[baghdad, reuter, influenti, shi'it, cleric, s...
4	[chicago, reuter, us, judg, friday, dealt, maj...	[chicago, reuter, u, judg, friday, dealt, majo...

Code description of task 10:

1. **WordNet Lemmatizer Initialization:**
   - WordNetLemmatizer() is built from the nltk library. It converts words to their base form (lemma).
2. **Applying Lemmatization:.apply(lambda tokens:**
   - [lemmatizer.lemmatize(word) for word in tokens]) → The original form of each token is extracted.
3. **Storing the Result:**
   - The lemmatized tokens are stored in the "lemmatized_tokens" column.

4. **Displaying Output:**
    - The first few rows are displayed using .head().

Title of the task 11: Final Text Preparation

Description of the task11:

In this task, we have converted the lemmatized tokens back into sentence form (string). Final Text Preparation is done because:

- The ML model requires text in string format as input. TF-IDF or other vectorization techniques take string text as input. After all the preprocessing steps (case folding, punctuation removal, tokenization, stopwords removal, lemmatization) are completed, the text is ready for model training.

We have created a new column final_text by joining the lemmatized_tokens list with spaces.

Code for task11:

```
df["final_text"] = df["lemmatized_tokens"].apply(lambda x: " ".join(x))

df[["lemmatized_tokens", "final_text"]].head()
```

Sample output of task11:

	lemmatized_tokens	final_text
0	[beijing, reuters, china, hope, syria, show, f...	beijing reuters china hope syria show flexibil...
1	[would, think, president, barack, obama, speci...	would think president barack obama specificall...
2	[washington, reuters, united, state, warned, s...	washington reuters united state warned saudi a...
3	[tune, alternate, current, radio, network, acr...	tune alternate current radio network acr anoth...
4	[rightwingers, everywhere, trying, disprove, o...	rightwingers everywhere trying disprove overwh...

Code description of task11:

1. **Joining Tokens:**
    - All words in the lemmatized_tokens list are concatenated with spaces using apply(lambda x: " ".join(x)).
2. **Storing the Result:**
    - The result is stored in a new column final_text.
3. **Displaying the Output:**
    - The first 5 rows are displayed using df[["lemmatized_tokens", "final_text"]].head(). This helps verify that the tokens are converted correctly to the final text.

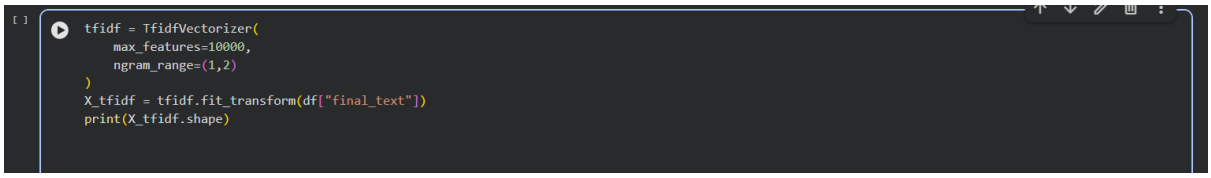
Title of the task 12:

Description of the task 12:

In this task, we use TF-IDF (Term Frequency–Inverse Document Frequency) to convert text into numerical feature vectors. TF-IDF Vectorization is done because:

- ML models only take numerical input, not raw text. TF-IDF measures the importance of words, reducing the weight of frequent but less informative words. Features of single word and two-word phrases are created using N-grams (1,2). With max_features=10000, only the top 10000 important features are kept, which makes model training fast and memory-efficient.

Code for task 12 :



```
tfidf = TfidfVectorizer(
    max_features=10000,
    ngram_range=(1,2)
)
X_tfidf = tfidf.fit_transform(df["final_text"])
print(X_tfidf.shape)
```

Sample output of task 12:

```
(36608, 10000)
```

Code description of task 12:

1. **InitializingTF-IDFVectorizer:**
   - TfidfVectorizer(max_features=10000, ngram_range=(1,2)) is used to create the vectorizer. With max_features=10000, the top 10000 important features are taken. With ngram_range=(1,2), unigram and bigram features are created.
2. **Fitting and Transforming Text:**
   - Numerical TF-IDF vectors are created from text data with fit_transform(df["final_text"]). The results are stored in X_tfidf.
3. **Checking Shape:**
   - The dimensions of the feature matrix are shown with X_tfidf.shape. rows = number of documents, columns = number of features.

Title of the task 13: Train-Test Split

Description of the task 13:

In this task, we have divided the dataset into training set and test set. Train-Test Split is done because:

- Training data is required for learning the model. Separate test data is kept for model evaluation. Using stratify=y, the label distribution is the same in training and test set. Test size=0.3 means 30% data is kept in test set and 70% data is kept in training set.

.

Code for task 13:

```
X = X_tfidf
y = df["label"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
print(X_train.shape, X_test.shape)
```

Sample output of task 13:

```
(25625, 10000) (10983, 10000)
```

Code description of task 13:

1. **Defining Features and Target:**
   - X = X_tfidf → TF-IDF vectors are used as features. y = df["label"] -Target label (fake/true) is taken.
2. **Splitting the Dataset:**
   - The dataset is randomly split using train_test_split(). test_size=0.3 → 30% of the data is kept in the test set. stratify=y -to preserve label distribution. random_state=42 - to make the result reproducible.
3. **Verifying Shapes:**
   - The dimensions of the train and test set are verified by showing X_train.shape, X_test.shape.

Title of the task 14: Multinomial Naïve Bayes Model Training

Description of the task14:

In this task, we trained the model with training data using Multinomial Naïve Bayes (MNB) algorithm. Reasons for choosing Multinomial Naïve Bayes:

- Very effective for text classification tasks. Works well with TF-IDF or word counts feature. Computationally fast and gives good performance even on small datasets.

Code for task 14:

```
model = MultinomialNB()
model.fit(X_train, y_train)
```

Sample output of task 14:

```
MultinomialNB
MultinomialNB()
```

Code description of task 14:

1. **Initializing the Model:**
   - MultinomialNB() creates a Multinomial Naïve Bayes classifier object.
2. **Training the Model:**
   - model.fit(X_train, y_train) trains the model using training data.X_train -TF-IDF features.y_train -Target labels (fake/true)
3. **Result:**
   - The model is now ready to make predictions. The next step is to evaluate the model with test data.

Title of the task 15: Model Prediction and Evaluation

Description of the task 15:

In this task, we used the trained Multinomial Naïve Bayes model to predict test data and evaluate the performance of the model. Evaluation was done because:

- The effectiveness of the model can be measured with the accuracy and classification report. It is understood how well the model is classifying fake and true news. Detailed error analysis can be done with the confusion matrix.

Code for task 15:

```
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test,y_pred)
print("Accuracy:", accuracy)
print(classification_report(y_test, y_pred))
```

Sample output of task 15:

```
Accuracy: 0.9562050441591551
              precision    recall  f1-score   support

        fake       0.95      0.95      0.95      4643
        true       0.96      0.96      0.96      6340

    accuracy                           0.96     10983
   macro avg       0.96      0.95      0.96     10983
weighted avg       0.96      0.96      0.96     10983
```

Code description of task15:

1. **Prediction on Test Data:**
    - labels for the test set are predicted using model.predict(X_test).
2. **Overall correctness of the model is measured using Calculating Accuracy:**
    - accuracy_score(y_test, y_pred).
3. **Precision, recall, f1-score and support are displayed using Generating Classification Report:**
    - classification_report(y_test, y_pred).
    - Precision - how accurate the model is at predicting positives.
    - Recall - how accurately the model detects positive instances.
    - F1-score -harmonic mean of precision and recall.

Title of the task 16:

Description of the task 16:

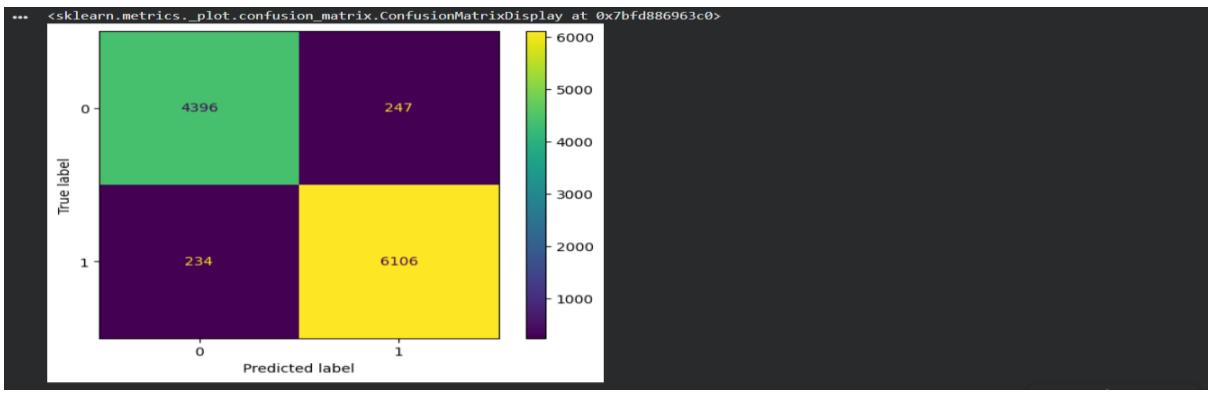
In this task, we used the Confusion Matrix to analyze the prediction quality of our trained Naïve Bayes model in more depth. The Confusion Matrix helps us understand:

- Where the model is predicting correctly Where the model is predicting incorrectly Whether the misclassification is Fake → True or True → Fake Detailed picture behind real-world accuracy Visualization makes this analysis easier to understand.

Code for task 16:

```
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
```

Sample output of task 16:



Code description of task 16:

1. **Generating the Confusion Matrix:**
    - confusion_matrix(y_test, y_pred)- This creates a matrix by comparing the test labels and predicted labels.
2. **Preparing the Display Object:**
    - ConfusionMatrixDisplay(confusion_matrix=cm). A display object is created for the confusion matrix visualization.
3. **Plotting the Matrix:**
    - disp.plot()-Visually plots the confusion matrix, where true positive, false positive, true negative and false negative are very clearly visible.

Project Code

Task 1 Code:

```
def min_edit_distance(s1, s2):
    m = len(s1)
    n = len(s2)
    dp = [[0 for _ in range(n+1)] for _ in range(m+1)]
    for i in range(m+1):
        dp[i][0] = i
    for j in range(n+1):
        dp[0][j] = j
    for i in range(1, m+1):
        for j in range(1, n+1):
            if s1[i-1] == s2[j-1]:
                cost = 0
            else:
                cost = 1
            dp[i][j] = min(
                dp[i-1][j] + 1,
                dp[i][j-1] + 1,
                dp[i-1][j-1] + cost
            )
    return dp[m][n]

text1 = input("Enter first text: ")
text2 = input("Enter second text: ")
distance = min_edit_distance(text1, text2)
similarity = 1 - (distance / max(len(text1), len(text2)))
print("Minimum Edit Distance:", distance)
print("Similarity:", round(similarity, 4))
```

Task 2 Code :

```
from google.colab import drive
drive.mount('/content/drive')


import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.tokenize import word_tokenize
import re
import string
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt


fake = pd.read_csv('/content/drive/MyDrive/Fake_news_detection/fake.csv')
true = pd.read_csv('/content/drive/MyDrive/Fake_news_detection/true.csv')
print("Fake shape:", fake.shape)
print("True shape:", true.shape)
```

```
fake['label'] = 'fake'
true['label'] = 'true'
data = pd.concat([fake, true], axis=0).reset_index(drop=True)
data = data.sample(frac=1).reset_index(drop=True)
data.head()


df =
pd.read_csv("/content/drive/MyDrive/Fake_news_detection/merged_fake_true_news.csv")
df.drop_duplicates(subset=['text'], inplace=True)
df.dropna(subset=['text'], inplace=True)
df = df[df['text'].str.len() > 20]
df = df[~df['text'].str.match(r'^[0-9\W]+$', na=False)]
df = df[~df['text'].str.contains(r'^http', na=False)]
df = df[~df['text'].str.contains("subscribe|click here|read more|follow us", case=False)]
def clean_basic(text):
    text = re.sub(r"http\S+|www.\S+", " ", text)
    text = re.sub(r"<.*?>", " ", text)
    text = re.sub(r"\s+", " ", text)
    return text.strip()
df["clean_text"] = df["text"].apply(clean_basic)
df.reset_index(drop=True, inplace=True)
df.head()


df["clean_text_lower"] = df["clean_text"].str.lower()
df[["clean_text", "clean_text_lower"]].head()


punct_mapping = str.maketrans("", "", string.punctuation)
df["clean_text_no_punct"] = df["clean_text_lower"].apply(lambda x:
x.translate(punct_mapping))
df["clean_text_no_punct"] = df["clean_text_no_punct"].str.strip()
df[["clean_text_lower", "clean_text_no_punct"]].head()
```

```
df["tokens"] = df["clean_text_no_punct"].apply(lambda x: x.split())
df[["clean_text_no_punct", "tokens"]].head()


stop_words = set(stopwords.words("english"))
df["tokens_no_stopwords"] = df["tokens"].apply(
    lambda words: [word for word in words if word not in stop_words]
)
df[["tokens", "tokens_no_stopwords"]].head()


ps = PorterStemmer()
df["stemmed_tokens"] = df["tokens_no_stopwords"].apply(lambda tokens: [ps.stem(word)
for word in tokens])
df[["tokens_no_stopwords", "stemmed_tokens"]].head()


lemmatizer = WordNetLemmatizer()
df["lemmatized_tokens"] = df["stemmed_tokens"].apply(lambda tokens:
[lemmatizer.lemmatize(word) for word in tokens])
df[["stemmed_tokens", "lemmatized_tokens"]].head()


df["final_text"] = df["lemmatized_tokens"].apply(lambda x: " ".join(x))
df[["lemmatized_tokens", "final_text"]].head()


tfidf = TfidfVectorizer(
    max_features=10000,
    ngram_range=(1,2)
)
X_tfidf = tfidf.fit_transform(df["final_text"])
print(X_tfidf.shape)
```

```
X = X_tfidf
y = df["label"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
print(X_train.shape, X_test.shape)

model = MultinomialNB()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test,y_pred)
print("Accuracy:", accuracy)
print(classification_report(y_test, y_pred))


cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
```