# PROJECT: 8

**PROJECT ID:** Proj\_225023\_Team\_3

**NAME:** S Paul Benjamin Felix

# FAKE NEWS DETECTION USING NLP

**PROBLEM STATEMENT**

The task at hand involves the creation of a dependable and efficient model to identify counterfeit news within text-based materials, including articles, social media posts, and news reports. The aim is to devise a model or software solution capable of autonomously discerning between trustworthy, evidence-based information and misleading or deceitful content.

# DESIGN THINKING

The iterative problem-solving approach known as design thinking can be utilized for the purpose of detecting fake news through the implementation of Natural Language Processing (NLP). This approach encompasses various crucial stages, including Empathize, Define, Ideate, Prototype, and Test.

# Empathize:

**Data Collection:** Collect pertinent information, comprising of categorized sets of fabricated and authentic news articles from a reputable online platform such as Kaggle.

**Dataset Link:** https://[www.kaggle.com/datasets/clmentbisaillon/fake-](http://www.kaggle.com/datasets/clmentbisaillon/fake-) and-real-news-dataset

# Define:

**Problem Statement:** The task at hand involves the creation of a dependable and efficient machine learning framework to identify fabricated information within text-based materials, including articles, social media posts, and news reports.

# Ideate:

**Feature Selection:** Please finalize the NLP techniques and features to be utilized in the detection of fake news. This may encompass sentiment analysis, topic modeling, word embeddings, and additional methods.

# Prototype:

**Data Preprocessing:** Prepare and clean the data, including text normalization, tokenization, and feature extraction.

**Model Development:** Create a prototype of the NLP-based fake news detection system using various machine learning algorithms.

# Test:

**Evaluation:** Evaluate the model's performance by utilizing appropriate evaluation metrics, including accuracy, confusion matrix, and false positive calculation. Validate the model's accuracy by testing it with labeled datasets.

DATASETS

The datasets used for Fake News Detection are True.csv with 21,417 records of data and Fake.csv with 23,481 records of data. Both the files have the following columns,

* + Title
  + Text
  + Subject
  + Date

# DATA PREPROCESSING

**Step 1:** Import the required libraries for the project. Download the libraries if necessary.

**Step 2:** Use basic data analysis functions such as head( ) to see the datasets inside the Jupyter notebook or Google Colab.

**Step 3:** Add labels to both datasets in order to distinguish fake and true news articles. Fake = 0 and True = 1

**Step 4:** Separate last 10 records of data from both files for future use as manual testing data.

**Step 5:** Concatenate both the datasets into a variable **df.**

**Step 6:** Only the necessary columns such as **text** and **class** are kept, other columns that do not contribute to Fake News Detection project are dropped.

**Step 7:** Shuffle the dataset and reset the index values to prevent overfitting in the machine learning model. Removing the extra index.

**Step 8:** All the text in the dataset should be changed to lowercase, and any empty spaces, punctuations, HTML tags, and escape characters should be removed using regular expressions. This function should be applied uniformly to all the text in the dataset.

**Step 9:** Remove stop words from the text column using the nltk library to eliminate articles and other unnecessary words for training a model. This step is essential in NLP models, but not in deep learning models.

**Step 10:** Optionally use wordcloud to visualize the frequency of words in the text column.

# FEATURE EXTRACTION

**Step 1:** Utilize TfidfVectorizer to examine the textual components of fabricated and authentic content. Partition the information into training and testing sets. TfidfVectorizer is a numerical representation employed in natural language processing and information retrieval to evaluate and contrast the significance of words in a document in relation to a corpus.

**Term Frequency (TF):** This component quantifies the frequency of a word's occurrence within a document by calculating the ratio of the number of times the word appears to the total number of words in the document.

**Inverse Document Frequency (IDF):** IDF measures the importance of a word in the entire corpus. Words that appear frequently in many documents have a lower IDF value, while words that are more unique have a higher IDF value. IDF is calculated as the logarithm of the total number of documents in the corpus divided by the number of documents containing the word.

# MODEL TRAINING AND EVALUATION

**Step 1:** Machine learning models are imported from sklearn library. Metrics such as accuracy\_score, classification\_report and confusion\_meatrix are imported.

**Step 2:** Models are built and trained. After the model has been trained print the values of evaluation metrics. The following table showcases the evaluation metrics.

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **MODEL** | **ACCURACY** | **FALSE POSITIVES** |
| 1. | Logistic Regression | 98.69 % | 65 |
| 2. | Decision Tree Classifier | 99.49 % | 25 |
| 3. | Random Forest Classifier | 99.05 % | 46 |
| 4. | Gradient Boosting  Classifier | 99.38 % | 41 |

# CODING

**!**pip install pandas

**!**pip install matplotlib seaborn scikit-learn

**!**pip install wordcloud

**!**pip install nltk

**import** pandas **as** pd **import** numpy **as** np **import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.metrics **import** accuracy\_score

**from** sklearn.metrics **import** classification\_report

**import** re

**import** string

**from** wordcloud **import** WordCloud **import** nltk nltk**.**download('stopwords')

**from** nltk.corpus **import** stopwords

**from** sklearn.feature\_extraction.text **import** TfidfVectorizer

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.metrics **import** accuracy\_score, classification\_report from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier from sklearn.ensemble import GradientBoostingClassifier

df\_fake['class'] **=** 0

df\_true['class'] **=** 1

df\_fake\_testing\_data **=** df\_fake**.**tail(10)

**for** i **in** range(23480, 23470, **-**1): df\_fake**.**drop([i], axis **=** 0, inplace **= True**)

df\_true\_testing\_data **=** df\_true**.**tail(10)

**for** i **in** range(21416, 21406, **-**1): df\_true**.**drop([i], axis **=** 0, inplace **= True**)

df\_testing **=** pd**.**concat([df\_fake\_testing\_data, df\_true\_testing\_data], axis **=** 0) df **=** pd**.**concat([df\_fake, df\_true], axis **=** 0)

df **=** df**.**drop(["title", "subject", "date"], axis **=** 1) print(df**.**isnull()**.**sum())

A black and white text  Description automatically generated

df **=** df**.**sample(frac **=** 1) df**.**reset\_index(inplace **= True**) df**.**drop(['index'], axis **=** 1, inplace **= True**)

**def** wordopt(text): text **=** text**.**lower()

text **=** re**.**sub('\[.\*?\]', '', text) text **=** re**.**sub("\\W"," ",text)

text **=** re**.**sub('https?://\S+|www\.\S+', '', text) text **=** re**.**sub('<.\*?>+', '', text)

text **=** re**.**sub('[%s]' **%** re**.**escape(string**.**punctuation), '', text) text **=** re**.**sub('\n', '', text)

text **=** re**.**sub('\w\*\d\w\*', '', text)

**return** text

df["text"] **=** df["text"]**.**apply(wordopt)

**def** remove\_stopwords(text):

stop\_words **=** set(stopwords**.**words('english')) words **=** text**.**split()

filtered\_words **=** [word **for** word **in** words **if** word**.**lower() **not in** stop\_words]

**return** ' '**.**join(filtered\_words)

df['text'] **=** df['text']**.**apply(remove\_stopwords)

*# Splitting the dataset into training and testing sets*

x **=** df["text"]

y **=** df["class"]

x\_train, x\_test, y\_train, y\_test **=** train\_test\_split(x, y, test\_size**=**0.2, random\_state**=**42)

*# Vectorize the text using TF-IDF*

vectorization **=** TfidfVectorizer()

xv\_train **=** vectorization**.**fit\_transform(x\_train) xv\_test **=** vectorization**.**transform(x\_test)

# Initialize and train a Logistic Regression model LR = LogisticRegression()

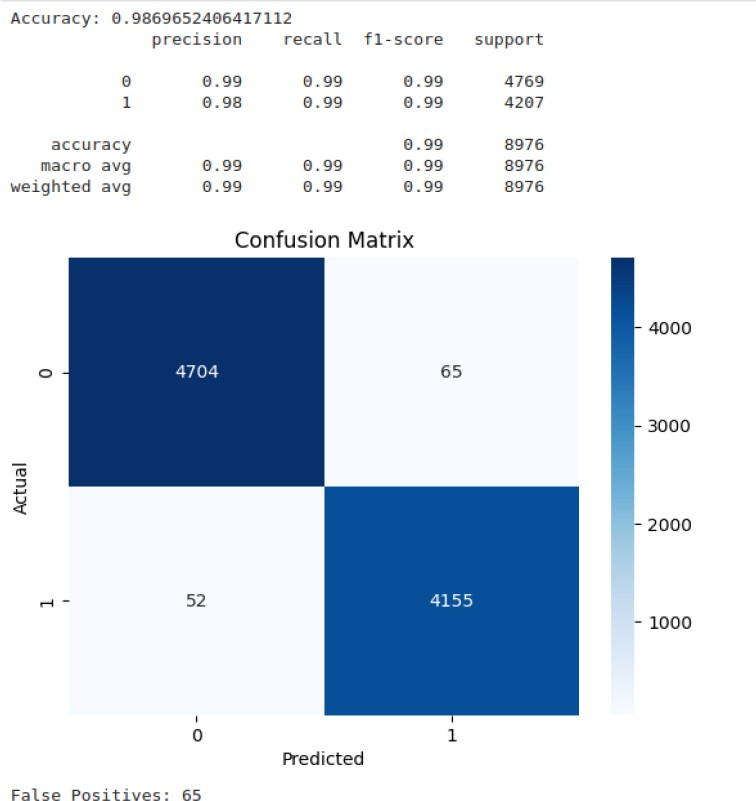
LR.fit(xv\_train, y\_train) pred\_lr = LR.predict(xv\_test)

accuracy = accuracy\_score(y\_test, pred\_lr) print("Accuracy:", accuracy) print(classification\_report(y\_test, pred\_lr)) conf\_matrix = confusion\_matrix(y\_test, pred\_lr) TN, FP, FN, TP = conf\_matrix.ravel() false\_positives = FP

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues') plt.xlabel('Predicted')

plt.ylabel('Actual') plt.title('Confusion Matrix') plt.show()

print("False Positives:", false\_positives)



# Initialize and train a Decision Tree Classifier model DT = DecisionTreeClassifier()

DT.fit(xv\_train, y\_train)

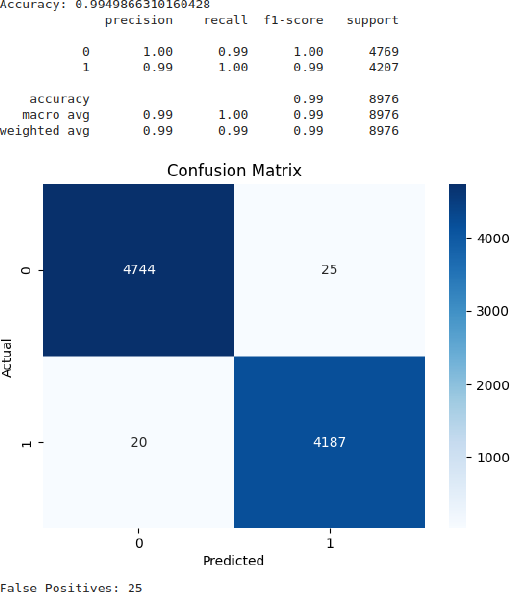
pred\_dt = DT.predict(xv\_test)

accuracy = accuracy\_score(y\_test, pred\_dt) print("Accuracy:", accuracy) print(classification\_report(y\_test, pred\_dt)) conf\_matrix = confusion\_matrix(y\_test, pred\_dt) TN, FP, FN, TP = conf\_matrix.ravel() false\_positives = FP

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues') plt.xlabel('Predicted')

plt.ylabel('Actual') plt.title('Confusion Matrix') plt.show()

print("False Positives:", false\_positives)



# Initialize and train a Random Forest Classifier model RFC = RandomForestClassifier(random\_state=0)

RFC.fit(xv\_train, y\_train)

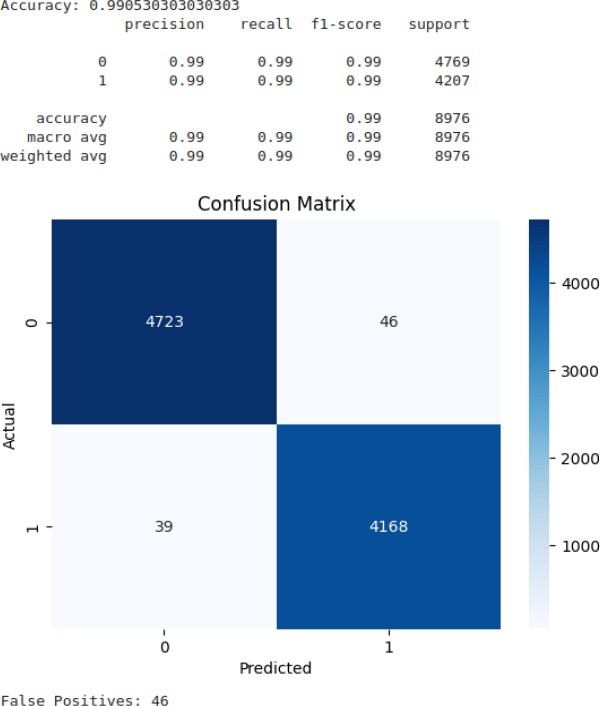
pred\_rfc = RFC.predict(xv\_test)

accuracy = accuracy\_score(y\_test, pred\_rfc) print("Accuracy:", accuracy) print(classification\_report(y\_test, pred\_rfc)) conf\_matrix = confusion\_matrix(y\_test, pred\_rfc) TN, FP, FN, TP = conf\_matrix.ravel() false\_positives = FP

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues') plt.xlabel('Predicted')

plt.ylabel('Actual') plt.title('Confusion Matrix') plt.show()

print("False Positives:", false\_positives)



# Initialize and train a Gradient Boosting Classifier model GBC = GradientBoostingClassifier(random\_state=0) GBC.fit(xv\_train, y\_train)

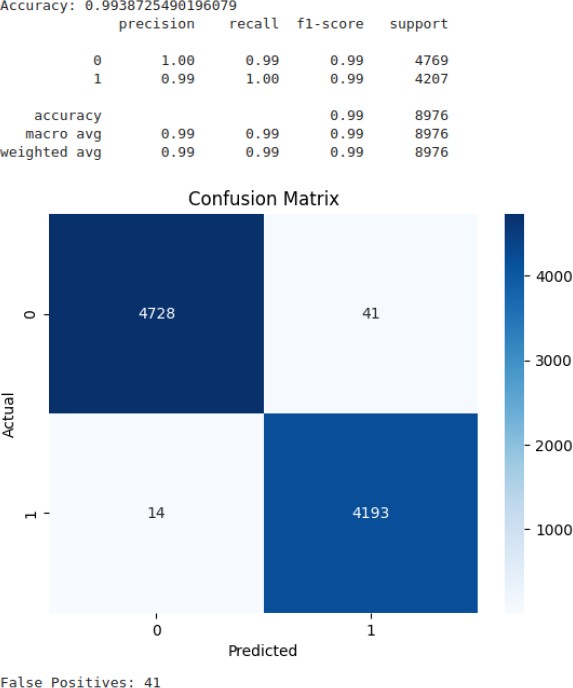
pred\_gbc = GBC.predict(xv\_test)

accuracy = accuracy\_score(y\_test, pred\_gbc) print("Accuracy:", accuracy) print(classification\_report(y\_test, pred\_gbc)) conf\_matrix = confusion\_matrix(y\_test, pred\_gbc) TN, FP, FN, TP = conf\_matrix.ravel() false\_positives = FP

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues') plt.xlabel('Predicted')

plt.ylabel('Actual') plt.title('Confusion Matrix') plt.show()

print("False Positives:", false\_positives)



# # Functions to test the text content with all the models.

def output\_class(n): if n == 0:

return "Fake News" elif n == 1:

return "True News"

def manual\_testing(news): testing\_news = {"text":[news]}

input\_news = pd.DataFrame(testing\_news)

input\_news ["text"] = input\_news ["text"].apply(wordopt) input\_news ["text"]= input\_news ["text"].apply(remove\_stopwords) new\_x\_test = input\_news["text"]

new\_xv\_test = vectorization.transform(new\_x\_test) pred\_LR = LR.predict(new\_xv\_test)

pred\_DT = DT.predict(new\_xv\_test) pred\_GBC = GBC.predict(new\_xv\_test) pred\_RFC = RFC.predict(new\_xv\_test)

return print("\n\nLR Prediction: {} \nDT Prediction: {} \nGBC Prediction: {}

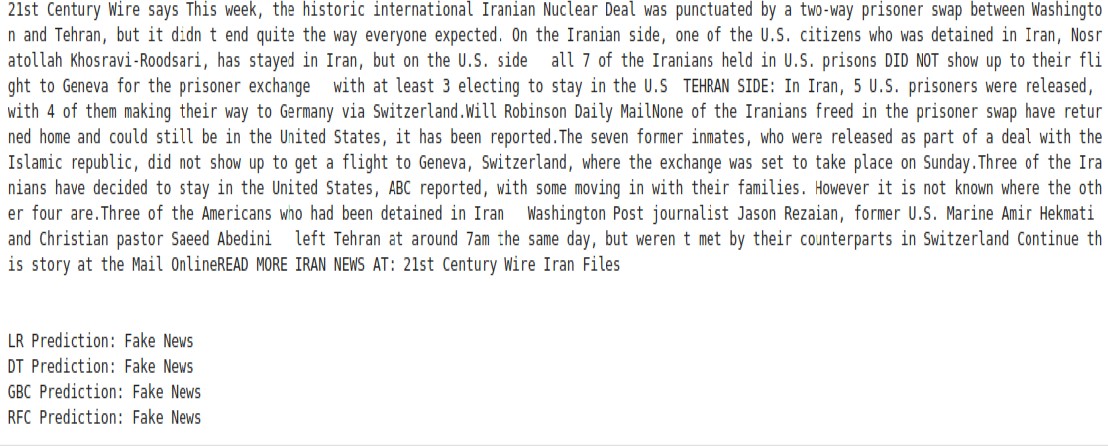
\nRFC Prediction: {}".format(output\_class(pred\_LR[0]), output\_class(pred\_DT[0]), output\_class(pred\_GBC[0]), output\_class(pred\_RFC[0])))

# #To make the user type in the news content then str( ) is used

news = str(input()) manual\_testing(news)

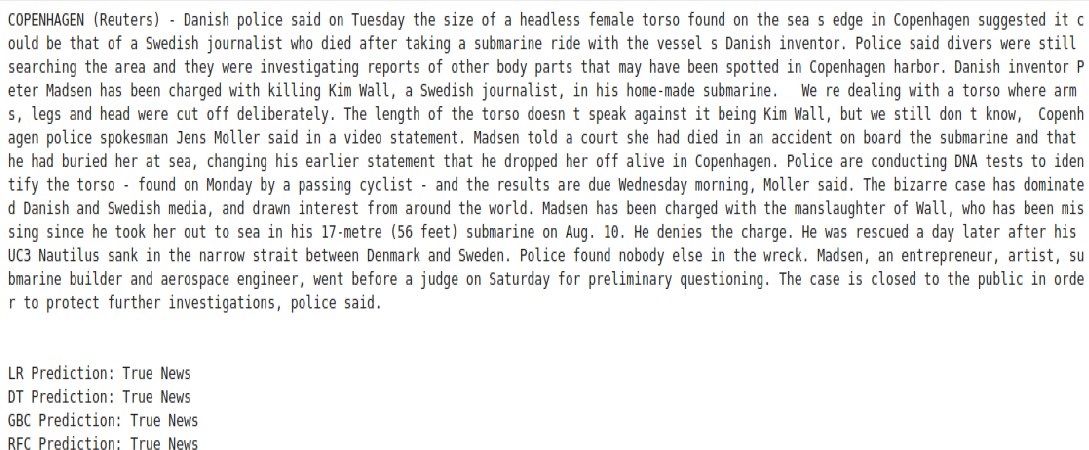
# # Using random fake news from manual test data defined at the beginning # to test the models that were built.

test\_fake\_news = df\_fake\_testing\_data[‘text’][23471] news = test\_fake\_news

print(news) manual\_testing(news)

test\_true\_news = df\_true\_testing\_data[‘text’][21410] news = test\_true\_news

print(news) manual\_testing(news)



# INFERENCE

All 4 machine learning models provide above 98 % accuracy. Decision Tree Classifier provide highest accuracy 99.49% with 25 false positive values which is the lowest among the four.