

```
In [32]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, r
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
import re
import nltk
import spacy
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
```

```
In [33]: nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]   C:\Users\ale1x\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]   C:\Users\ale1x\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]   C:\Users\ale1x\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt_tab to
[nltk_data]   C:\Users\ale1x\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

```
Out[33]: True
```

```
In [34]: from nltk.tokenize import word_tokenize

text = "This is a simple test."
tokens = word_tokenize(text)
print(tokens)
```

```
['This', 'is', 'a', 'simple', 'test', '.']
```

```
In [35]: df = pd.read_csv("fake_news_full_dataset.csv")
df.head()
```

Out[35]:

|   | id | title                                             | author   | text                                              | label |
|---|----|---------------------------------------------------|----------|---------------------------------------------------|-------|
| 0 | 0  | Breaking Event World True Event Analysis Fake ... | Author_0 | event event world government fake breaking inc... | 0     |
| 1 | 1  | Analysis Breaking True True Report Story Analy... | Author_1 | government incident world world report breakin... | 0     |
| 2 | 2  | News Incident Politics Story Event Incident Go... | Author_2 | analysis story event fake incident analysis in... | 1     |
| 3 | 3  | Incident Incident Report Incident Fake Breakin... | Author_3 | breaking report update event incident fake        | 0     |
| 4 | 4  | Breaking Government Incident Update Politics G... | Author_4 | event true update true true fake world news up... | 1     |

In [36]: `df['label'].value_counts()`

Out[36]:

```
label
0    10428
1    10372
Name: count, dtype: int64
```

In [38]: `df['clean_text'] = df['text'].apply(clean_text)`  
`Input_X = df['clean_text']`

In [39]:

```
stemmer = PorterStemmer()
# Stop words List
stop_words = set(stopwords.words('english'))
def clean_text(text):
    # 1. Lowercasing
    text = text.lower()

    # 2. Remove URLs
    text = re.sub(r'http\S+|www.\S+', '', text)

    # 3. Remove HTML tags
    text = re.sub(r'<.*?>', '', text)

    # 4. Remove Special Characters, Numbers, Punctuation
    text = re.sub(r'^a-zA-Z\s', '', text)

    # 5. Tokenization
    tokens = word_tokenize(text)

    # 6. Remove Stop Words
    tokens = [word for word in tokens if word not in stop_words]

    # 7. Stemming
    stemmed_tokens = [stemmer.stem(word) for word in tokens]

    # 8. Final clean text reconstruction (optional)
    clean_text = ' '.join(stemmed_tokens)

    return clean_text
```

In [40]: `df.head()`

Out[40]:

|          | id | title                                                   | author   | text                                                    | label | clean_text                                              |
|----------|----|---------------------------------------------------------|----------|---------------------------------------------------------|-------|---------------------------------------------------------|
| <b>0</b> | 0  | Breaking Event World<br>True Event Analysis<br>Fake ... | Author_0 | event event world<br>government fake<br>breaking inc... | 0     | event event world<br>govern fake break<br>incid even... |
| <b>1</b> | 1  | Analysis Breaking<br>True True Report<br>Story Analy... | Author_1 | government incident<br>world world report<br>breakin... | 0     | govern incid world<br>world report break<br>fake new... |
| <b>2</b> | 2  | News Incident<br>Politics Story Event<br>Incident Go... | Author_2 | analysis story event<br>fake incident analysis<br>in... | 1     | analysi stori event<br>fake incid analysi<br>incid w... |
| <b>3</b> | 3  | Incident Incident<br>Report Incident Fake<br>Breakin... | Author_3 | breaking report<br>update event<br>incident fake        | 0     | break report updat<br>event incid fake                  |
| <b>4</b> | 4  | Breaking<br>Government Incident<br>Update Politics G... | Author_4 | event true update<br>true true fake world<br>news up... | 1     | event true updat<br>true true fake world<br>news updat  |

```
In [41]: # 1. Define the mapping rules dict {original_value : new_value}
label_rules = {
    1: 'fake',
    0: 'real'
}
```

```
In [42]: # 2. Create the new 'label' column by mapping the 'level' column to the rules
df['label_text'] = df['label'].map(label_rules)
```

```
In [43]: print("\nResult with new 'label_text' column:")
df.head()
```

Result with new 'label\_text' column:

Out[43]:

|          | id | title                                                      | author   | text                                                       | label | clean_text                                                 | label_text |
|----------|----|------------------------------------------------------------|----------|------------------------------------------------------------|-------|------------------------------------------------------------|------------|
| <b>0</b> | 0  | Breaking Event<br>World True Event<br>Analysis Fake ...    | Author_0 | event event<br>world<br>government fake<br>breaking inc... | 0     | event event<br>world govern<br>fake break incid<br>even... | real       |
| <b>1</b> | 1  | Analysis Breaking<br>True True Report<br>Story Analy...    | Author_1 | government<br>incident world<br>world report<br>breakin... | 0     | govern incid<br>world world<br>report break<br>fake new... | real       |
| <b>2</b> | 2  | News Incident<br>Politics Story<br>Event Incident<br>Go... | Author_2 | analysis story<br>event fake<br>incident analysis<br>in... | 1     | analysi stori<br>event fake incid<br>analysi incid w...    | fake       |
| <b>3</b> | 3  | Incident Incident<br>Report Incident<br>Fake Breakin...    | Author_3 | breaking report<br>update event<br>incident fake           | 0     | break report<br>updat event<br>incid fake                  | real       |
| <b>4</b> | 4  | Breaking<br>Government<br>Incident Update<br>Politics G... | Author_4 | event true<br>update true true<br>fake world news<br>up... | 1     | event true<br>updat true true<br>fake world<br>news updat  | fake       |

In [44]:

```
# X = input (text), y = output (fake or real)
Input_X = df["text"]
Output_Y = df["label_text"]

# Split data: 80% training, 20% testing
X_train, X_test, y_train, y_test = train_test_split(
    Input_X, Output_Y, test_size=0.3, random_state=42
)
```

In [45]:

```
df.head()
```

Out[45]:

|   | id | title                                                      | author   | text                                                       | label | clean_text                                                 | label_text |
|---|----|------------------------------------------------------------|----------|------------------------------------------------------------|-------|------------------------------------------------------------|------------|
| 0 | 0  | Breaking Event<br>World True Event<br>Analysis Fake ...    | Author_0 | event event<br>world<br>government fake<br>breaking inc... | 0     | event event<br>world govern<br>fake break incid<br>even... | real       |
| 1 | 1  | Analysis Breaking<br>True True Report<br>Story Analy...    | Author_1 | government<br>incident world<br>world report<br>breakin... | 0     | govern incid<br>world world<br>report break<br>fake new... | real       |
| 2 | 2  | News Incident<br>Politics Story<br>Event Incident<br>Go... | Author_2 | analysis story<br>event fake<br>incident analysis<br>in... | 1     | analysi stori<br>event fake incid<br>analysis incid w...   | fake       |
| 3 | 3  | Incident Incident<br>Report Incident<br>Fake Breakin...    | Author_3 | breaking report<br>update event<br>incident fake           | 0     | break report<br>updat event<br>incid fake                  | real       |
| 4 | 4  | Breaking<br>Government<br>Incident Update<br>Politics G... | Author_4 | event true<br>update true true<br>fake world news<br>up... | 1     | event true<br>updat true true<br>fake world<br>news updat  | fake       |

In [59]: *# Remove or fill NaN in the Labels (y\_train)*  
y\_train = y\_train.fillna('')

*# Create the model*  
model = LogisticRegression(max\_iter=2000)  
  
*# Train the model using training data*  
model.fit(X\_train\_tfidf, y\_train)

Out[59]:

▼ LogisticRegression ⓘ ?  
 ► Parameters

In [60]: from sklearn.feature\_extraction.text import TfidfVectorizer

*# TF-IDF Vectorizer*  
tfidf\_vectorizer = TfidfVectorizer(  
 max\_features=5000,  
 ngram\_range=(1, 2)  
)  
  
*# Fit on training data, transform both train and test*  
X\_train\_tfidf = tfidf\_vectorizer.fit\_transform(X\_train)  
X\_test\_tfidf = tfidf\_vectorizer.transform(X\_test)

In [61]: def evaluate\_model(model\_name, y\_true, y\_pred):

*# Calculate metrics*  
accuracy = accuracy\_score(y\_true, y\_pred)  
precision = precision\_score(y\_true, y\_pred, average='weighted')  
recall = recall\_score(y\_true, y\_pred, average='weighted')  
f1 = f1\_score(y\_true, y\_pred, average='weighted')

```

cm = confusion_matrix(y_true, y_pred)

# Create a report
report = classification_report(y_true, y_pred)

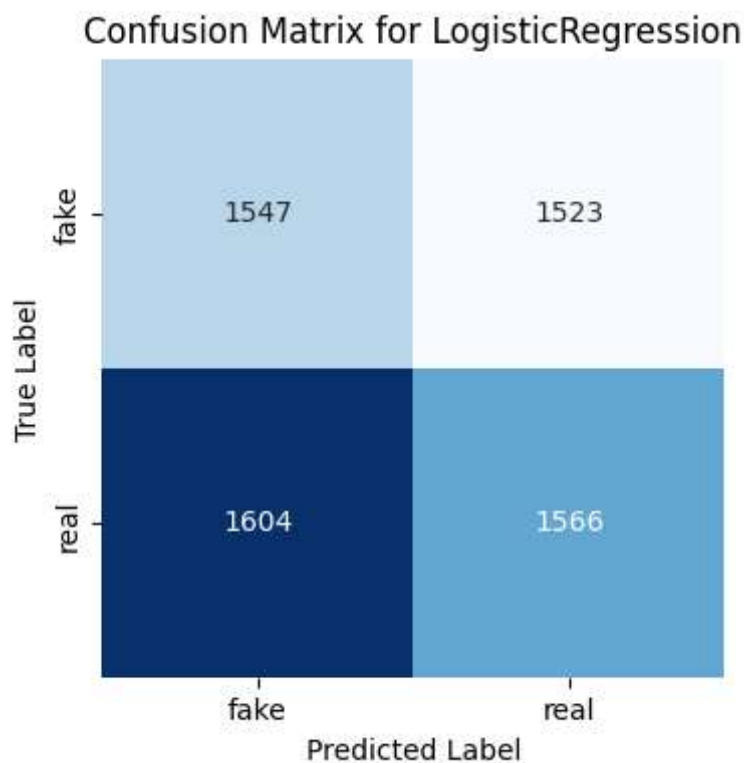
# Output results
metrics = {
    'Model Name': model_name,
    'Accuracy': accuracy,
    'Precision': precision,
    'Recall': recall,
    'F1 Score': f1,
    'Classification Report': report
}
# Plot Confusion Matrix
plt.figure(figsize=(4, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=r
plt.title(f'Confusion Matrix for {model_name}')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
return metrics

```

```

In [63]: # Define the Logistic Regression model
LR = LogisticRegression(random_state=42)
# Define the hyperparameters for grid search
param_grid = { 'solver': ['liblinear', 'lbfgs'],          # Solver type
}
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=LR, param_grid=param_grid, cv=5,
scoring='accuracy', n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(X_train_tfidf, y_train)
# Get the best estimator from grid search
best_LR = grid_search.best_estimator_
# Make predictions using the best model
y_pred = best_LR.predict(X_test_tfidf)
# Evaluate the model
evaluation_results = evaluate_model('LogisticRegression', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
# Print the best hyperparameters found by grid search
# print("\nBest hyperparameters found by GridSearchCV:")
# print(grid_search.best_params_)

```



Model Name:

LogisticRegression

Accuracy: 0.4989

Precision: 0.4991

Recall: 0.4989

F1 Score: 0.4989

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| fake         | 0.49      | 0.50   | 0.50     | 3070    |
| real         | 0.51      | 0.49   | 0.50     | 3170    |
| accuracy     |           |        | 0.50     | 6240    |
| macro avg    | 0.50      | 0.50   | 0.50     | 6240    |
| weighted avg | 0.50      | 0.50   | 0.50     | 6240    |

```
In [69]: from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression

# Example training data
texts = ["Aliens invaded NY!", "NASA launches new telescope", "Government lies a
labels = [0, 1, 0] # 0=fake, 1=real

# Train TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer()
X = tfidf_vectorizer.fit_transform(texts)

# Train model
model = LogisticRegression()
model.fit(X, labels)
```

```
Out[69]: LogisticRegression ⓘ ?
Parameters
```

```
In [70]: predict_news("Aliens invaded New York last night!")
predict_news("NASA launches a new telescope into space.")
```

```
Out[70]: 'REAL NEWS ✔'
```

```
In [74]: def predict_news(text):
          cleaned = clean_text(text)
          print("Cleaned text:", cleaned)

          vectorized = tfidf_vectorizer.transform([cleaned])
          print("Vectorized shape:", vectorized.shape)

          y_pred = model.predict(vectorized)[0]
          print("Raw model prediction:", y_pred)

          return "FAKE NEWS ✖" if y_pred == 0 else "REAL NEWS ✔"
```

```
In [76]: # Step 1: Imports
          from sklearn.feature_extraction.text import TfidfVectorizer
          from sklearn.linear_model import LogisticRegression
          import gradio as gr

          # Step 2: Simple clean_text function
          def clean_text(text):
              return text.lower()

          # Step 3: Tiny example dataset
          texts = [
              "Aliens invaded New York!",          # fake
              "NASA launches new telescope",        # real
              "Government lies about taxes",        # fake
              "Scientists discover water on Mars"    # real
          ]
          labels = [0, 1, 0, 1] # 0=fake, 1=real

          # Step 4: Train TF-IDF and model
          tfidf_vectorizer = TfidfVectorizer()
          X = tfidf_vectorizer.fit_transform([clean_text(t) for t in texts])
          model = LogisticRegression()
          model.fit(X, labels)

          # Step 5: Prediction function
          def predict_news(text):
              cleaned = clean_text(text)
              vectorized = tfidf_vectorizer.transform([cleaned])
              y_pred = model.predict(vectorized)[0]
              return "FAKE NEWS ✖" if y_pred == 0 else "REAL NEWS ✔"

          # Step 6: Gradio interface
          iface = gr.Interface(
              fn=predict_news,
              inputs=gr.Textbox(label="Enter news title", placeholder="Paste news article"),
              outputs=gr.Textbox(label="Result"),
              title="Fake News Detector",
              description="Enter a news article and find out if it's REAL or FAKE!"
          )
```

```
# Step 7: Launch
iface.launch()
```

- \* Running on local URL: http://127.0.0.1:7879
- \* To create a public link, set `share=True` in `launch()`.

# Fake News Detector

Enter a news article and find out if it's REAL or FAKE!

Enter news title

Paste news article text here...

Clear

Submit

Result

Flag

Built with Gradio  · Settings 

Out[76]:

In [ ]: