Problem Statement

The primary objective of this project is to build a classification model that can automatically

categorize news articles into different predefined categories. The model will be trained using

a labeled dataset of news articles and will output the most likely category (e.g., sports,

politics, or technology) for any given article.

In [383…

```
# IMPORTING NECESSARY LIBRARIES ===
import pandas as pd
import numpy as np
import re
import string
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

from sklearn.model_selection import train_test_split, cross_val_score,StratifiedKFold, GridSearchCV
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import loguniform
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
```

```python
from sklearn.metrics import classification_report, ConfusionMatrixDisplay, accuracy_score

import gensim
from gensim.models import Word2Vec

nltk.download('stopwords')
nltk.download('punkt')
```

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

Out[383… True

In [385…

```python
## Loading the data
df=pd.read_csv("news.csv")
## top 5 rows details
print(df.head())
```

```
  category                                         headline  \
0  WELLNESS              143 Miles in 35 Days: Lessons Learned
1  WELLNESS       Talking to Yourself: Crazy or Crazy Helpful?
2  WELLNESS  Crenezumab: Trial Will Gauge Whether Alzheimer...
3  WELLNESS                     Oh, What a Difference She Made
4  WELLNESS                                   Green Superfoods

                                               links  \
0  https://www.huffingtonpost.com/entry/running-l...
1  https://www.huffingtonpost.com/entry/talking-t...
2  https://www.huffingtonpost.com/entry/crenezuma...
3  https://www.huffingtonpost.com/entry/meaningfu...
4  https://www.huffingtonpost.com/entry/green-sup...

                                   short_description  \
0  Resting is part of training. I've confirmed wh...
1  Think of talking to yourself as a tool to coac...
2  The clock is ticking for the United States to ...
3  If you want to be busy, keep trying to be perf...
4  First, the bad news: Soda bread, corned beef a...

                             keywords
0                     running-lessons
1           talking-to-yourself-crazy
2  crenezumab-alzheimers-disease-drug
3                     meaningful-life
4                    green-superfoods
```

In [324…

```
print("Dataset loaded. Shape:", df.shape)
```

```
Dataset loaded. Shape: (50000, 5)
```

In [326…

```
# Check for missing values
print(df.isnull().sum())
```

```
category                0
headline                0
links                   0
short_description       0
keywords             2668
dtype: int64
```

Taking only the relevants columns for more accurate results

In [328...

```
# Combining headline + short_description
df['text'] = df['headline'] + " " + df['short_description']
df.dropna(subset=['text', 'category'], inplace=True)
```

In [330...

```
# Focusing on top 5-7 categories to reduce noise
top_categories = df['category'].value_counts().nlargest(7).index.tolist()
df = df[df['category'].isin(top_categories)].reset_index(drop=True)
```

Preprocessing

In [335...

```
#  Clean text
stop_words = set(stopwords.words('english'))

def clean_text(text):
    text = text.lower()         ## Lowercase
    text = re.sub(r'\[.*?\]', '', text)
    text = re.sub(r'http\S+|www\S+', '', text)
    text = re.sub(r'<.*?>+', '', text)
    text = re.sub(r'[^a-zA-Z\s]', '', text)        ## removing stop-words
    text = re.sub(r'\s+', ' ', text).strip()
    text = ' '.join(word for word in text.split() if word not in stop_words)   ## text splitting
    return text
```

In [337...

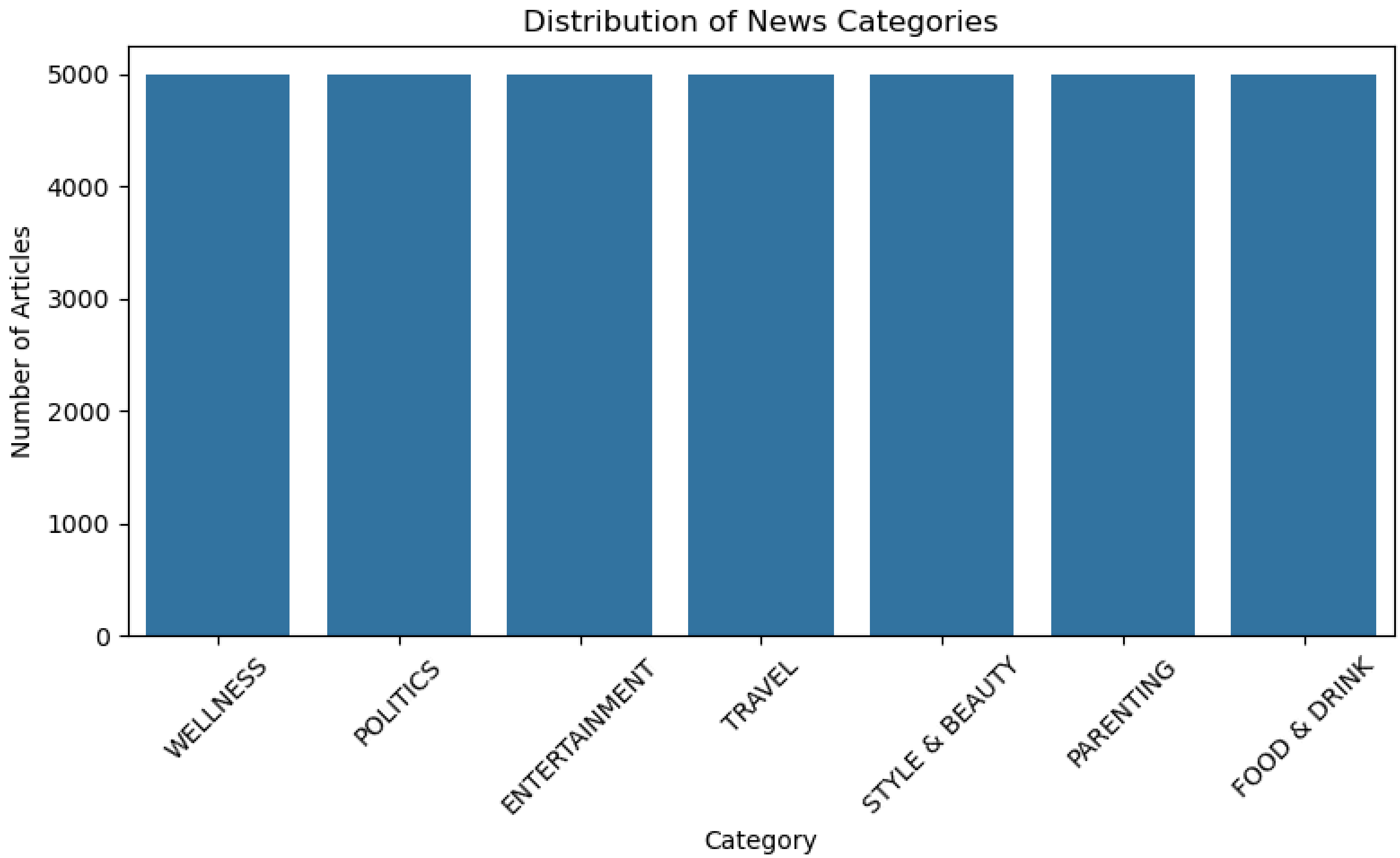
```
df['clean_text'] = df['text'].apply(clean_text)
```

Category Distribution(EDA process)

In [340...

```
plt.figure(figsize=(8, 5))
sns.countplot(data=df, x='category', order=df['category'].value_counts().index)
plt.title("Distribution of News Categories")
plt.xlabel("Category")
plt.ylabel("Number of Articles")
```

```
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Feature Extraction

In [342…

```
#  FEATURE EXTRACTION
# BoW
bow_vectorizer = CountVectorizer(max_features=5000, stop_words='english')
X_bow = bow_vectorizer.fit_transform(df['clean_text'])
```

In [394…

```
## BoW results
print(X_bow)
```

```
(0, 2851)	1
(0, 1123)	2
(0, 2573)	1
(0, 2548)	1
(0, 4597)	2
(0, 2327)	2
(0, 916)	2
(0, 4130)	1
(0, 2468)	1
(0, 2192)	3
(0, 577)	2
(0, 3822)	1
(0, 2006)	1
(0, 4947)	1
(0, 4865)	1
(0, 2664)	1
(0, 1052)	1
(0, 3279)	1
(0, 4497)	1
(0, 3785)	1
(1, 4409)	2
(1, 1025)	2
(1, 2060)	1
(1, 4504)	1
(1, 4558)	1
:	:
(34997, 753)	1
(34997, 665)	1
(34997, 1787)	1
(34997, 2837)	1
(34997, 1784)	1
(34997, 4647)	1
(34997, 1316)	2
(34998, 1393)	1
(34998, 4210)	1
(34998, 2037)	2
(34998, 4848)	1
(34998, 1746)	2
(34998, 1632)	2
(34998, 872)	2
(34998, 144)	1
```

```
  (34998, 974)  1
  (34998, 3611) 1
  (34998, 1281) 1
  (34999, 1025) 1
  (34999, 2668) 4
  (34999, 2455) 3
  (34999, 3234) 1
  (34999, 766)  2
  (34999, 1257) 1
  (34999, 1353) 1
```

In [344…

```
## vectorizing model

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(max_features=8000, ngram_range=(1,2))
X = vectorizer.fit_transform(df['clean_text'])
y = df['category']
```

In [346…

```
# Word2Vec
tokenized_text = df['clean_text'].apply(word_tokenize)
w2v_model = Word2Vec(sentences=tokenized_text, vector_size=100, window=5, min_count=2, workers=4)
```

In [347…

```
def document_vector(doc):
    doc = [word for word in doc if word in w2v_model.wv]
    return np.mean(w2v_model.wv[doc], axis=0) if doc else np.zeros(100)

X_w2v = np.vstack([document_vector(doc) for doc in tokenized_text])
```

In [398…

```
## Word2Vec results
print(X_w2v)
```

```
[[-0.34040597  0.32966942  0.15053    ... -0.60315204  0.07026497
  -0.09102805]
 [-0.27951324  0.25751325  0.03571267 ... -0.7341765   0.1607964
  -0.2143683 ]
 [-0.30829832  0.29086486  0.02457564 ... -0.5803688   0.08964723
  -0.12368248]
 ...
 [-0.25475678  0.2448282  -0.00404433 ... -0.6254412   0.27053663
  -0.1776912 ]
 [-0.23737246  0.38088384  0.0339792  ... -0.7405287   0.24074261
  -0.20853588]
 [-0.14719456  0.1497331   0.07144123 ... -0.7442189   0.0117287
  -0.19597055]]
```

In [350…

```
print("TF-IDF shape:", X.shape)
```

```
TF-IDF shape: (35000, 8000)
```

LabelEncoder to convert categorical to numerical

In [21]:

```
label_encoder = LabelEncoder()
df['label'] = label_encoder.fit_transform(df['category'])

category_word_freq = defaultdict(lambda: defaultdict(int))

for i, row in df.iterrows():
    for word in row['clean_text'].split():
        category_word_freq[row['category']][word] += 1
```

In [23]:

```
# Display top 10 words per category
for category in df['category'].unique():
    print(f"\nTop words for category: {category}")
    sorted_words = sorted(category_word_freq[category].items(), key=lambda x: x[1], reverse=True)
    for word, count in sorted_words[:10]:
        print(f"{word}: {count}")
```

```
Top words for category: WELLNESS
us: 534
life: 534
time: 492
people: 485
one: 482
health: 421
like: 347
new: 332
get: 305
make: 295

Top words for category: POLITICS
us: 311
new: 265
one: 249
state: 247
people: 242
would: 239
said: 223
president: 222
states: 174
political: 151

Top words for category: ENTERTAINMENT
new: 378
one: 272
film: 271
like: 193
first: 186
show: 180
time: 168
movie: 168
star: 163
years: 159

Top words for category: TRAVEL
one: 490
travel: 425
new: 393
like: 338
```

```
time: 328
us: 303
city: 293
world: 284
get: 271
best: 259

Top words for category: STYLE & BEAUTY
check: 547
style: 524
fashion: 483
want: 421
new: 401
sure: 400
look: 368
twitter: 329
one: 327
like: 309

Top words for category: PARENTING
kids: 673
children: 602
one: 531
parents: 473
time: 464
like: 395
child: 379
know: 328
dont: 300
day: 293

Top words for category: FOOD & DRINK
food: 397
one: 395
make: 315
like: 313
time: 274
us: 216
new: 211
dont: 210
best: 209
```

```
get: 198

Top words for category: WORLD NEWS
people: 392
us: 328
one: 285
world: 251
said: 247
years: 218
new: 215
president: 208
country: 201
government: 189

Top words for category: BUSINESS
new: 438
one: 356
business: 344
people: 338
us: 294
time: 293
many: 265
company: 235
get: 234
years: 232

Top words for category: SPORTS
game: 292
first: 256
one: 234
team: 212
sports: 196
nfl: 189
football: 188
new: 166
two: 163
players: 158
```

In [352…

```
#  Train/Test Split ===
from sklearn.model_selection import train_test_split
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Logistic Regression

In [355…

```python
## Hyperparamter tuning
params = {
    'C': loguniform(0.1, 100),
    'solver': ['liblinear', 'lbfgs']
}
```

In [357…

```python
model_lr=LogisticRegression()
```

In [359…

```python
rand_lr = RandomizedSearchCV(LogisticRegression(max_iter=2000), param_distributions=params,
                             n_iter=20, cv=5, scoring='accuracy', random_state=42, n_jobs=-1)
rand_lr.fit(X_train, y_train)
```

Out[359…

```
▸ RandomizedSearchCV
▸ best_estimator_: LogisticRegression
    ▸ LogisticRegression
```

In [361…

```python
print("Training Score",rand_lr.score(X_train,y_train))
print("Testing Score",rand_lr.score(X_test,y_test))
```

```
Training Score 0.9296785714285715
Testing Score 0.8362857142857143
```

The Testing Score is 83.6% which a good model predictions

In [363…

```python
print("Best Parameters for Logistic Regression:", rand_lr.best_params_)
print("Best Score (Train CV):", rand_lr.best_score_)
```

```
Best Parameters for Logistic Regression: {'C': 2.5135566617708283, 'solver': 'liblinear'}
Best Score (Train CV): 0.8357857142857142
```

In [365…

```
y_pred_lr = rand_lr.predict(X_test)
```

In [367…

```
## predicted
print(y_pred_lr)
```

```
['TRAVEL' 'POLITICS' 'POLITICS' ... 'ENTERTAINMENT' 'STYLE & BEAUTY'
 'TRAVEL']
```

In [369…

```
## Actual and predicted values in DataFrame form
df=pd.DataFrame({'Actual':y_test,'Predicted':y_pred_lr})
df
```

Out[369…

	Actual	Predicted
17813	TRAVEL	TRAVEL
6857	POLITICS	POLITICS
7672	POLITICS	POLITICS
9704	POLITICS	POLITICS
14303	ENTERTAINMENT	ENTERTAINMENT
...	...	...
8045	POLITICS	POLITICS
33786	FOOD & DRINK	STYLE & BEAUTY
13208	ENTERTAINMENT	ENTERTAINMENT
24073	STYLE & BEAUTY	STYLE & BEAUTY
20318	STYLE & BEAUTY	TRAVEL

7000 rows × 2 columns

In [371…

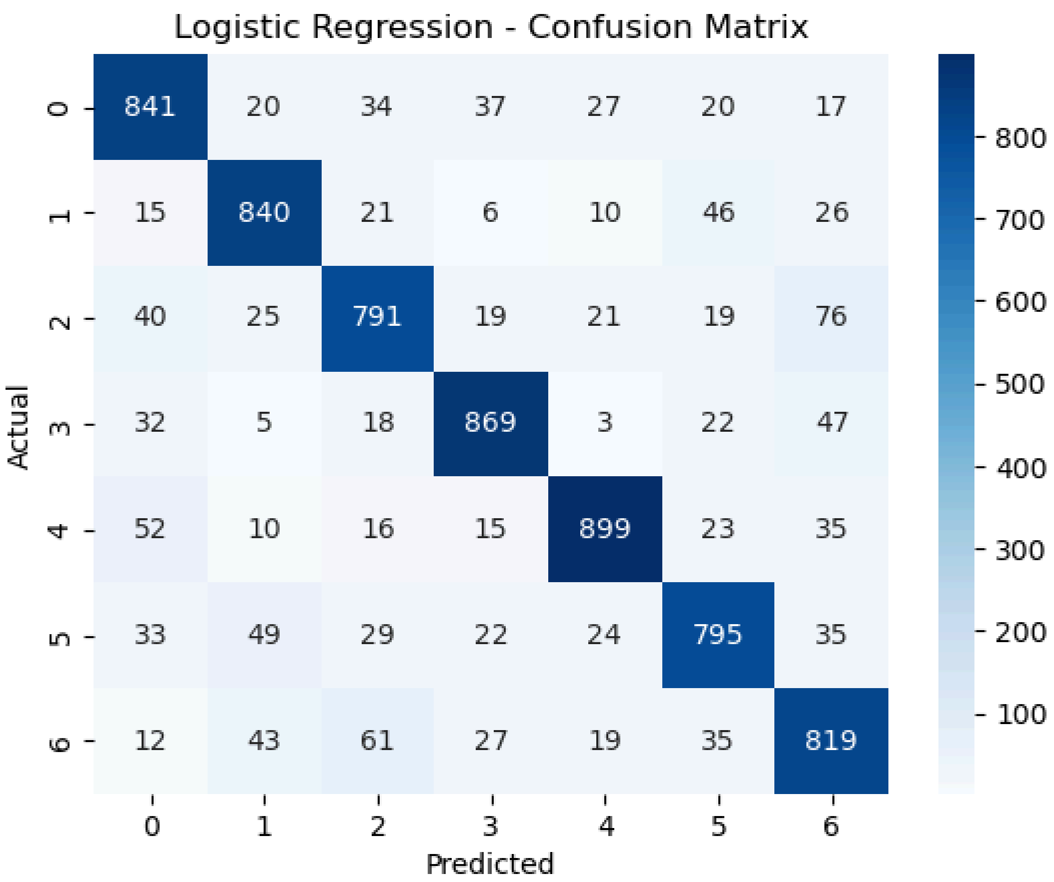
```
cm_lr=confusion_matrix(y_test,y_pred_lr)
print(cm_lr)
print("Classification Report:\n", classification_report(y_test, y_pred_lr))
```

```
[[841  20  34  37  27  20  17]
 [ 15 840  21   6  10  46  26]
 [ 40  25 791  19  21  19  76]
 [ 32   5  18 869   3  22  47]
 [ 52  10  16  15 899  23  35]
 [ 33  49  29  22  24 795  35]
 [ 12  43  61  27  19  35 819]]
Classification Report:
                 precision    recall  f1-score   support

 ENTERTAINMENT       0.82      0.84      0.83       996
  FOOD & DRINK       0.85      0.87      0.86       964
     PARENTING       0.82      0.80      0.81       991
      POLITICS       0.87      0.87      0.87       996
STYLE & BEAUTY       0.90      0.86      0.88      1050
        TRAVEL       0.83      0.81      0.82       987
      WELLNESS       0.78      0.81      0.79      1016

      accuracy                           0.84      7000
     macro avg       0.84      0.84      0.84      7000
  weighted avg       0.84      0.84      0.84      7000
```

The Logistic Regression accuracy is 84%, which is predicting a really good model

In [373…

```
sns.heatmap(cm_lr, annot=True, fmt='d',cmap='Blues')
plt.title(f"Logistic Regression - Confusion Matrix")
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Support Vector Machine

In [279…

```
## Hyperparameter tuning
svm_params = {'C': [0.01, 0.1, 1, 10]}
svm_model = GridSearchCV(LinearSVC(), svm_params, cv=5)
```

In [281…

```
svm_model.fit(X_train,y_train)
```

Out[281…

GridSearchCV

best_estimator_: LinearSVC

LinearSVC

In [283…

```
y_pred_svm=svm_model.predict(X_test)
```

In [285…

```
print(y_pred_svm)
```

```
['TRAVEL' 'POLITICS' 'POLITICS' ... 'ENTERTAINMENT' 'STYLE & BEAUTY'
 'TRAVEL']
```

In [289…

```
print("Training Score",svm_model.score(X_train,y_train))
print("Testing Score",svm_model.score(X_test,y_test))
```

```
Training Score 0.90575
Testing Score 0.8377142857142857
```

The model testing score is 83.7% is good model predictions

In [291…

```
df=pd.DataFrame({'Actual':y_test,'Predicted':y_pred_svm})
df
```

Out[291…

	Actual	Predicted
17813	TRAVEL	TRAVEL
6857	POLITICS	POLITICS
7672	POLITICS	POLITICS
9704	POLITICS	POLITICS
14303	ENTERTAINMENT	ENTERTAINMENT
...	...	...
8045	POLITICS	POLITICS
33786	FOOD & DRINK	STYLE & BEAUTY
13208	ENTERTAINMENT	ENTERTAINMENT
24073	STYLE & BEAUTY	STYLE & BEAUTY
20318	STYLE & BEAUTY	TRAVEL

7000 rows × 2 columns

In [293…

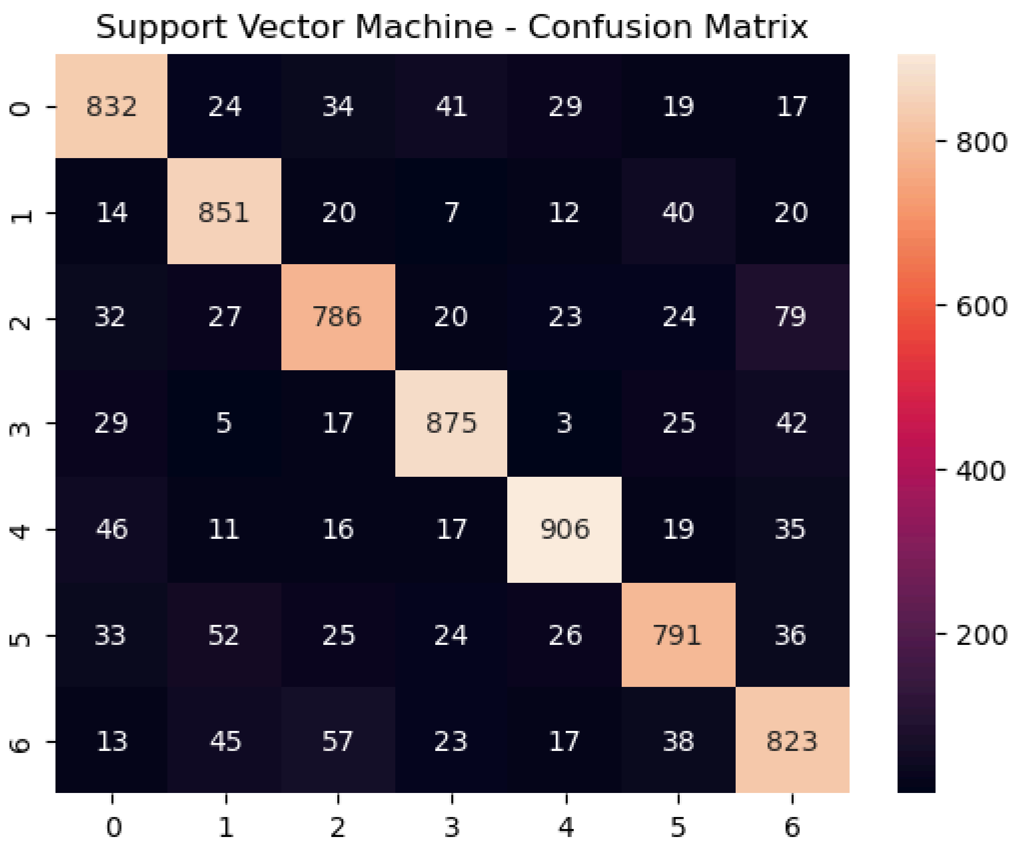
```
cm_rf=confusion_matrix(y_test,y_pred_svm)
print(cm_rf)
print(classification_report(y_test,y_pred_svm))
```

```
[[832  24  34  41  29  19  17]
 [ 14 851  20   7  12  40  20]
 [ 32  27 786  20  23  24  79]
 [ 29   5  17 875   3  25  42]
 [ 46  11  16  17 906  19  35]
 [ 33  52  25  24  26 791  36]
 [ 13  45  57  23  17  38 823]]
                precision    recall  f1-score   support

 ENTERTAINMENT       0.83      0.84      0.83       996
  FOOD & DRINK       0.84      0.88      0.86       964
     PARENTING       0.82      0.79      0.81       991
      POLITICS       0.87      0.88      0.87       996
STYLE & BEAUTY       0.89      0.86      0.88      1050
        TRAVEL       0.83      0.80      0.81       987
      WELLNESS       0.78      0.81      0.80      1016

      accuracy                           0.84      7000
     macro avg       0.84      0.84      0.84      7000
  weighted avg       0.84      0.84      0.84      7000
```

The SVM accuracy is 84% predicting good results

In [295…

```
sns.heatmap(cm_rf, annot=True, fmt='d')
plt.title(f"Support Vector Machine - Confusion Matrix")
plt.show()
```



Naive Bayes

In [298…

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

Out[298…

```
▾ MultinomialNB
MultinomialNB()
```

In [300…

```
nb_preds = nb_model.predict(X_test)
```

In [302...

```
print(nb_preds)
```

```
['TRAVEL' 'POLITICS' 'POLITICS' ... 'ENTERTAINMENT' 'STYLE & BEAUTY'
 'TRAVEL']
```

In [304...

```
print("Training Score",nb_model.score(X_train,y_train))
print("Testing Score",nb_model.score(X_test,y_test))
```

```
Training Score 0.8734285714285714
Testing Score 0.824
```

The Naive bayes testing score is 82.4% which is good but not that good compared to the other models

In [306...

```
df=pd.DataFrame({'Actual':y_test,'Predicted':nb_preds})
df
```

Out[306…

	Actual	Predicted
17813	TRAVEL	TRAVEL
6857	POLITICS	POLITICS
7672	POLITICS	POLITICS
9704	POLITICS	POLITICS
14303	ENTERTAINMENT	STYLE & BEAUTY
...	...	...
8045	POLITICS	POLITICS
33786	FOOD & DRINK	FOOD & DRINK
13208	ENTERTAINMENT	ENTERTAINMENT
24073	STYLE & BEAUTY	STYLE & BEAUTY
20318	STYLE & BEAUTY	TRAVEL

7000 rows × 2 columns

In [308…

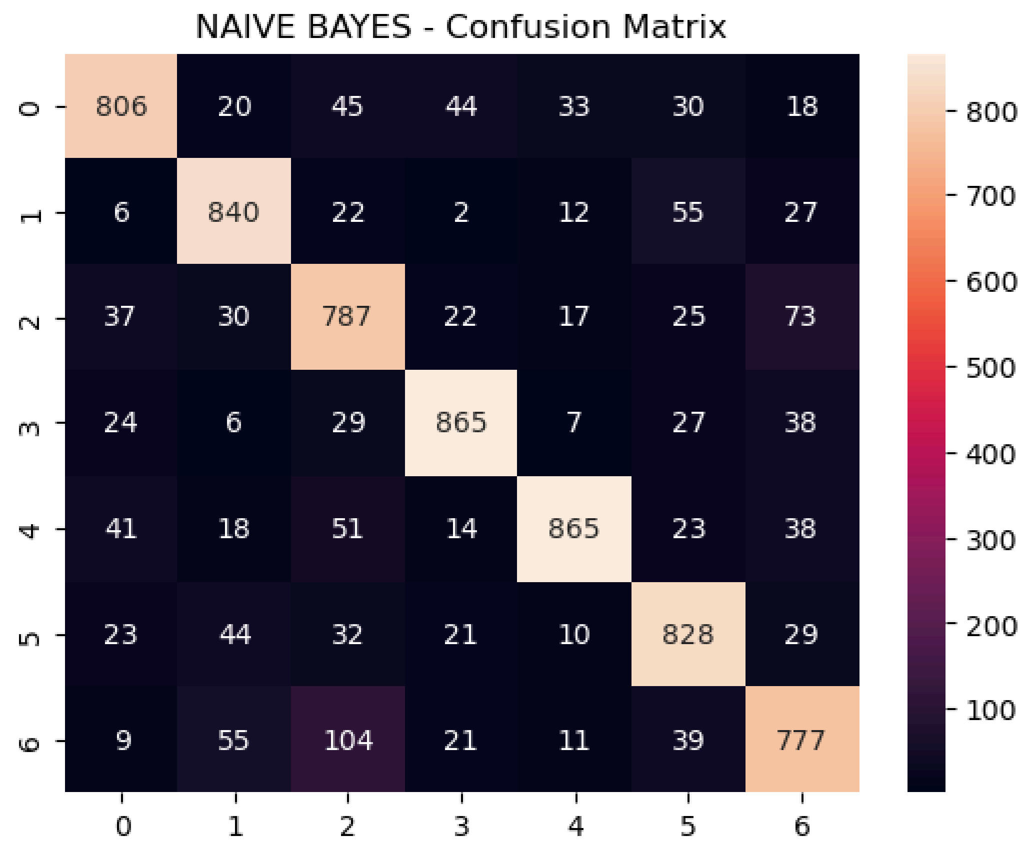
```
cm_rf=confusion_matrix(y_test,nb_preds)
print(cm_rf)
print(classification_report(y_test,nb_preds))
```

```
[[806  20  45  44  33  30  18]
 [  6 840  22   2  12  55  27]
 [ 37  30 787  22  17  25  73]
 [ 24   6  29 865   7  27  38]
 [ 41  18  51  14 865  23  38]
 [ 23  44  32  21  10 828  29]
 [  9  55 104  21  11  39 777]]
                precision    recall  f1-score   support

 ENTERTAINMENT       0.85      0.81      0.83       996
  FOOD & DRINK       0.83      0.87      0.85       964
     PARENTING       0.74      0.79      0.76       991
      POLITICS       0.87      0.87      0.87       996
STYLE & BEAUTY       0.91      0.82      0.86      1050
        TRAVEL       0.81      0.84      0.82       987
      WELLNESS       0.78      0.76      0.77      1016

      accuracy                           0.82      7000
     macro avg       0.83      0.82      0.82      7000
  weighted avg       0.83      0.82      0.82      7000
```

The Naive Bayes model prediction is least compared to all the other models i.e 82%

In [310…

```
sns.heatmap(cm_rf, annot=True, fmt='d')
plt.title(f"NAIVE BAYES - Confusion Matrix")
plt.show()
```



Cross-Validation process

In [377…

```
# === Stratified K-Fold ===
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

In [379…

```
# === Models ===
models = {
    "Naive Bayes": MultinomialNB(),
    "Logistic Regression": LogisticRegression(C=5, max_iter=2000, solver='liblinear'),
    "Linear SVM": LinearSVC(C=1),
}
```

In [387…

```
# === Voting Classifier (Ensemble) ===
voting_clf = VotingClassifier(estimators=[
    ('lr', models["Logistic Regression"]),
    ('nb', models["Naive Bayes"]),
    ('svm', models["Linear SVM"])
], voting='hard')

models["Voting Ensemble"] = voting_clf
```

In [391…

```
# === Cross-validation scores ===
print("\n🔍 Cross-Validation Accuracy Scores:")
for name, model in models.items():
    scores = cross_val_score(model, X, y, cv=cv, scoring='accuracy', n_jobs=-1)
    print(f"{name:20}: Mean: {scores.mean():.4f} ± Std: {scores.std():.4f}")
```

```
🔍 Cross-Validation Accuracy Scores:
Naive Bayes         : Mean: 0.8247 ± Std: 0.0041
Logistic Regression : Mean: 0.8373 ± Std: 0.0037
Linear SVM          : Mean: 0.8293 ± Std: 0.0045
Voting Ensemble     : Mean: 0.8381 ± Std: 0.0038
```

All models performed well, achieving over 82% accuracy, which is strong for a multi-class text classification task with diverse topics.

Logistic Regression and SVM delivered the best individual performance, with Logistic Regression slightly outperforming others in consistency and interpretability.

Naive Bayes, despite being simpler, performed very competitively, indicating that the preprocessing and TF-IDF vectorization were highly effective.

The Voting Ensemble (combining all three models) achieved the highest overall cross-validation score, reinforcing the advantage of model diversity in ensemble learning.

VIDEO EXPLANATION

In [ ]:

```
https://drive.google.com/file/d/10GjSjZYA67LIcRuqYD9slTI1K-Z9Rc2l/view?usp=sharing
```