

Student Name: Ammar Mohammad Wassas

Course Name: Practical_Image_Processing_and_Natural_Language_Processing_Second
Trimester 2026

Project 2

Step1: Download the NLP dataset for the classification problem.

Dataset Name: News Category Dataset (HuffPost)

Dataset link: <https://www.kaggle.com/datasets/rmisra/news-category-dataset>

```
[1]: from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
[2]: # Locate the JSON file inside "Project 2 - NLP" under your Datasets folder
import glob

matches = glob.glob('/content/drive/MyDrive/**/Project 2 - NLP/*.json',
                    ↪recursive=True)
print(matches)
```

```
['/content/drive/MyDrive/Colab Notebooks/Datasets/Project 2 -
NLP/News_Category_Dataset_v3.json']
```

```
[3]: # Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
[4]: # Reading dataset
DATA_PATH = '/content/drive/MyDrive/Colab Notebooks/Datasets/Project 2 - NLP/
        ↪News_Category_Dataset_v3.json'

df = pd.read_json(DATA_PATH, lines=True)
df.head()
```

```
[4]:                                     link \
0  https://www.huffpost.com/entry/covid-boosters-...
1  https://www.huffpost.com/entry/american-airlin...
2  https://www.huffpost.com/entry/funniest-tweets...
3  https://www.huffpost.com/entry/funniest-parent...
4  https://www.huffpost.com/entry/amy-cooper-lose...

                                     headline  category \
0  Over 4 Million Americans Roll Up Sleeves For O...  U.S. NEWS
1  American Airlines Flyer Charged, Banned For Li...  U.S. NEWS
2  23 Of The Funniest Tweets About Cats And Dogs ...  COMEDY
```

```

3 The Funniest Tweets From Parents This Week (Se... PARENTING
4 Woman Who Called Cops On Black Bird-Watcher Lo... U.S. NEWS

```

```

                                short_description          authors \
0 Health experts said it is too early to predict... Carla K. Johnson, AP
1 He was subdued by passengers and crew when he ... Mary Papenfuss
2 "Until you have a dog you don't understand wha... Elyse Wanshel
3 "Accidentally put grown-up toothpaste on my to... Caroline Bologna
4 Amy Cooper accused investment firm Franklin Te... Nina Golgowski

```

```

                                date
0 2022-09-23
1 2022-09-23
2 2022-09-23
3 2022-09-23
4 2022-09-22

```

```

[5]: # Shape and category info
      print(df.shape)
      df['category'].value_counts()

```

```

(209527, 6)

```

```

[5]: category
POLITICS          35602
WELLNESS          17945
ENTERTAINMENT     17362
TRAVEL            9900
STYLE & BEAUTY    9814
PARENTING         8791
HEALTHY LIVING    6694
QUEER VOICES      6347
FOOD & DRINK      6340
BUSINESS          5992
COMEDY            5400
SPORTS            5077
BLACK VOICES      4583
HOME & LIVING     4320
PARENTS           3955
THE WORLDPOST     3664
WEDDINGS          3653
WOMEN             3572
CRIME             3562
IMPACT            3484
DIVORCE           3426
WORLD NEWS        3299
MEDIA             2944
WEIRD NEWS        2777

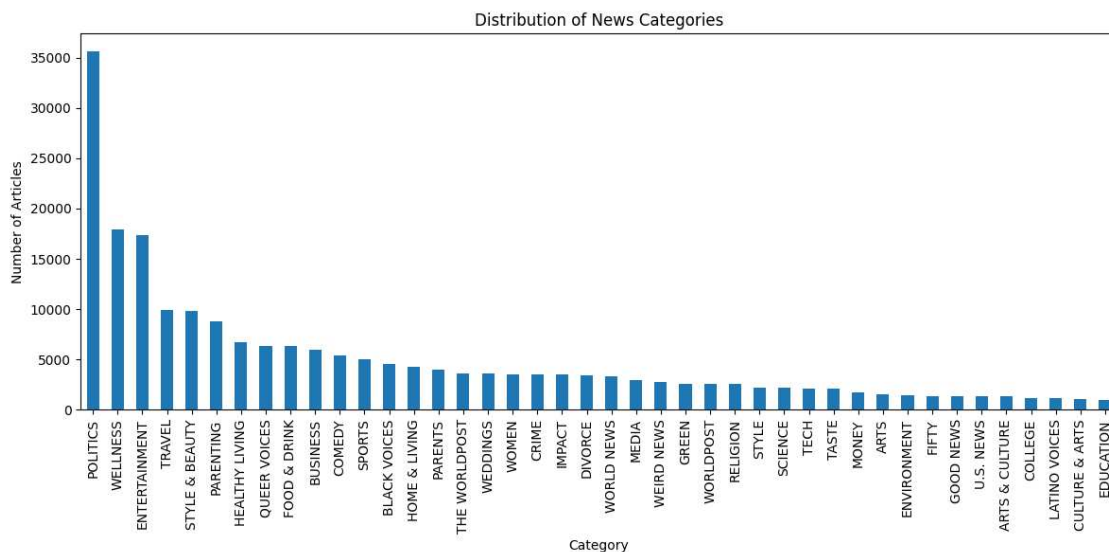
```

GREEN	2622
WORLDPOST	2579
RELIGION	2577
STYLE	2254
SCIENCE	2206
TECH	2104
TASTE	2096
MONEY	1756
ARTS	1509
ENVIRONMENT	1444
FIFTY	1401
GOOD NEWS	1398
U.S. NEWS	1377
ARTS & CULTURE	1339
COLLEGE	1144
LATINO VOICES	1130
CULTURE & ARTS	1074
EDUCATION	1014

Name: count, dtype: int64

```
[6]: # Count unique categories and plot the distribution
category_counts = df['category'].value_counts()

plt.figure(figsize=(12, 6))
category_counts.plot(kind='bar')
plt.title('Distribution of News Categories')
plt.xlabel('Category')
plt.ylabel('Number of Articles')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



```
[10]: # Filtered dataset - keep the same 6 categories
categories_to_keep = ['POLITICS', 'WELLNESS', 'ENTERTAINMENT',
                      'TRAVEL', 'STYLE & BEAUTY', 'PARENTING']

filtered_df = df[df['category'].isin(categories_to_keep)]
print(len(filtered_df))
filtered_df.head()
```

99414

```
[10]:
```

	link \	headline	category \	short_description	authors \	date
3	https://www.huffpost.com/entry/funniest-parent...	The Funniest Tweets From Parents This Week (Se...	PARENTING	"Accidentally put grown-up toothpaste on my to...	Caroline Bologna	2022-09-23
20	https://www.huffpost.com/entry/golden-globes-r...	Golden Globes Returning To NBC In January Afte...	ENTERTAINMENT	For the past 18 months, Hollywood has effectiv...		2022-09-20
21	https://www.huffpost.com/entry/biden-us-forces...	Biden Says U.S. Forces Would Defend Taiwan If ...	POLITICS	President issues vow as tensions with China rise.		2022-09-19
24	https://www.huffpost.com/entry/ukraine-festiva...	'Beautiful And Sad At The Same Time': Ukrainia...	POLITICS	An annual celebration took on a different feel...	Jonathan Nicholson	2022-09-19
28	https://www.huffpost.com/entry/james-cameron-f...	James Cameron Says He 'Clashed' With Studio Be...	ENTERTAINMENT	The "Avatar" director said aspects of his 2009...	Ben Blanchet	2022-09-18

```
[11]: # Sample 1000 per category (clean, non-deprecated groupby().sample())
filtered_df = filtered_df.groupby('category', group_keys=False).sample(n=1000,
↪random_state=42)
filtered_df['category'].value_counts()
```

```
[11]: category
ENTERTAINMENT    1000
```

```
PARENTING          1000
POLITICS           1000
STYLE & BEAUTY     1000
TRAVEL             1000
WELLNESS           1000
Name: count, dtype: int64
```

```
[12]: # Combine headline + short_description into one text field
filtered_df['text'] = filtered_df['headline'] + ' ' +
    ↪filtered_df['short_description']
filtered_df[['category', 'text']].head()
```

```
[12]:
```

	category	text
49579	ENTERTAINMENT	Kanye West's 'Erratic' Medication Dosage Repor...
2347	ENTERTAINMENT	Regina King Reveals Her Plan To Push Diversity...
98112	ENTERTAINMENT	A 13-Year-Old Girl Was Largely Responsible For...
4697	ENTERTAINMENT	YouTuber Jake Paul Charged With Trespassing In...
104903	ENTERTAINMENT	A Woman Proposed To Her Girlfriend During Ryan...

Step 2: Data preprocessing

```
[13]: # Import necessary libraries for preprocessing
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer

# Download NLTK resources (only first time)
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
```

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

```
[13]: True
```

```
[14]: # Preprocessing function
stemmer = PorterStemmer()
stop_words = set(stopwords.words('english'))

def preprocess_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
clean_text = ' '.join(stemmed_tokens)

return clean_text

```

```

[15]: # Cleaning dataset
filtered_df["clean_text"] = filtered_df["text"].apply(preprocess_text)
filtered_df = filtered_df.dropna()
filtered_df[["category", "text", "clean_text"]].head()

```

```

[15]:
      category      text \
49579  ENTERTAINMENT  Kanye West's 'Erratic' Medication Dosage Repor...
2347   ENTERTAINMENT  Regina King Reveals Her Plan To Push Diversity...
98112  ENTERTAINMENT  A 13-Year-Old Girl Was Largely Responsible For...
4697   ENTERTAINMENT  YouTuber Jake Paul Charged With Trespassing In...
104903 ENTERTAINMENT  A Woman Proposed To Her Girlfriend During Ryan...

      clean_text
49579  kanye west errat medic dosag reportedli led hos...
2347   regina king reveal plan push divers cann film ...
98112  yearold girl larg respons start johnni depp ca...
4697   youtub jake paul charg trespass loot arizona m...
104903 woman propos girlfriend ryan gosl directori debut

```

```

[16]: # Sanity check on the cleaning
print("Original :", filtered_df['text'].iloc[0])
print("Cleaned  :", filtered_df['clean_text'].iloc[0])

```

Original : Kanye West's 'Erratic' Medication Dosage Reportedly Led To Hospitalization The rapper has been open about his experiences with depression and anxiety.

Cleaned : kany west errat medic dosag reportedli led hospit rapper open experi depress anxieti

Step 3: Split the dataset

```
[17]: # Define features and target variable
```

```
X = filtered_df['clean_text']
```

```
y = filtered_df['category']
```

```
[18]: # Split data into training and testing sets
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(
```

```
    X, y, test_size=0.30, random_state=42, stratify=y
```

```
)
```

```
print("Train shape:", X_train.shape)
```

```
print("Test shape:", X_test.shape)
```

```
print("\nTrain class distribution:\n", y_train.value_counts())
```

```
print("\nTest class distribution:\n", y_test.value_counts())
```

Train shape: (4200,)

Test shape: (1800,)

Train class distribution:

category	
TRAVEL	700
POLITICS	700
STYLE & BEAUTY	700
ENTERTAINMENT	700
WELLNESS	700
PARENTING	700

Name: count, dtype: int64

Test class distribution:

category	
ENTERTAINMENT	300
POLITICS	300
STYLE & BEAUTY	300
PARENTING	300
TRAVEL	300
WELLNESS	300

Name: count, dtype: int64

Step 4: Feature representation methods

```
[19]: # Import feature representation libraries
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
```

```
[20]: # Bag of Words
vectorizer = CountVectorizer()
X_train_bow = vectorizer.fit_transform(X_train)
X_test_bow = vectorizer.transform(X_test)
print("BoW shape:", X_train_bow.shape)
```

BoW shape: (4200, 11758)

```
[21]: # TF-IDF Unigram
vectorizer_uni = TfidfVectorizer(ngram_range=(1, 1))
X_train_Unigram = vectorizer_uni.fit_transform(X_train)
X_test_Unigram = vectorizer_uni.transform(X_test)
print("Unigram TF-IDF shape:", X_train_Unigram.shape)
```

Unigram TF-IDF shape: (4200, 11758)

```
[22]: # TF-IDF Unigrams + Bigrams
vectorizer_ub = TfidfVectorizer(ngram_range=(1, 2))
X_train_U_B_nigram = vectorizer_ub.fit_transform(X_train)
X_test_U_B_nigram = vectorizer_ub.transform(X_test)
print("Unigram+Bigram TF-IDF shape:", X_train_U_B_nigram.shape)
```

Unigram+Bigram TF-IDF shape: (4200, 77406)

```
[23]: # TF-IDF Unigrams + Bigrams + Trigrams
vectorizer_ubt = TfidfVectorizer(ngram_range=(1, 3))
X_train_U_B_T_nigram = vectorizer_ubt.fit_transform(X_train)
X_test_U_B_T_nigram = vectorizer_ubt.transform(X_test)
print("Unigram+Bigram+Trigram TF-IDF shape:", X_train_U_B_T_nigram.shape)
```

Unigram+Bigram+Trigram TF-IDF shape: (4200, 147054)

```
[24]: # Tokenize clean_text for Word2Vec
from nltk.tokenize import word_tokenize

def preprocess(text):
    tokens = word_tokenize(text.lower())
    return tokens

filtered_df['tokens'] = filtered_df['clean_text'].apply(preprocess)
filtered_df['tokens'].head()
```

```
[24]: 49579    [kany, west, errat, medic, dosag, reportedli, ...
      2347    [regina, king, reveal, plan, push, divers, can...
      98112   [yearold, girl, larg, respons, start, johnni, ...
      4697    [youtub, jake, paul, charg, trespass, loot, ar...
```

104903 [woman, propos, girlfriend, ryan, gosl, direct...
Name: tokens, dtype: object

```
[26]: # Install gensim (not pre-installed on Colab)
!pip install -q gensim
```

27.9/27.9 MB

37.6 MB/s eta 0:00:00

```
[27]: # Train CBOW and Skip-gram Word2Vec models
from gensim.models import Word2Vec

cbow_model = Word2Vec(sentences=filtered_df['tokens'], vector_size=100,
                      window=5, min_count=1, sg=0) # sg=0 for CBOW
skipgram_model = Word2Vec(sentences=filtered_df['tokens'], vector_size=100,
                          window=5, min_count=1, sg=1) # sg=1 for Skip-gram
print("CBOW vocab size:", len(cbow_model.wv))
print("Skip-gram vocab size:", len(skipgram_model.wv))
```

CBOW vocab size: 14097

Skip-gram vocab size: 14097

```
[28]: # Average Word Vectors per document
import numpy as np

def get_sentence_vector(tokens, model):
    vectors = [model.wv[word] for word in tokens if word in model.wv]
    if vectors:
        return np.mean(vectors, axis=0)
    else:
        return np.zeros(model.vector_size)

filtered_df['Vector_Cbow'] = filtered_df['tokens'].apply(lambda x:
    ↪get_sentence_vector(x, cbow_model))
filtered_df['Vector_skipgram'] = filtered_df['tokens'].apply(lambda x:
    ↪get_sentence_vector(x, skipgram_model))
```

```
[29]: # Build train/test matrices for Word2Vec features
X_train_cbow = np.vstack(filtered_df.loc[X_train.index, 'Vector_Cbow'].values)
X_test_cbow = np.vstack(filtered_df.loc[X_test.index, 'Vector_Cbow'].values)

X_train_sg = np.vstack(filtered_df.loc[X_train.index, 'Vector_skipgram'].values)
X_test_sg = np.vstack(filtered_df.loc[X_test.index, 'Vector_skipgram'].values)

print("Word2Vec CBOW train shape:", X_train_cbow.shape)
print("Word2Vec Skip-gram train shape:", X_train_sg.shape)
```

Word2Vec CBOW train shape: (4200, 100)

Word2Vec Skip-gram train shape: (4200, 100)

Step 5: train and evaluate models.

```
[30]: # Import evaluation libraries
from sklearn.metrics import (
    confusion_matrix, accuracy_score, precision_score,
    recall_score, f1_score, classification_report
)
import seaborn as sns

[31]: # Evaluation function
def evaluate_model(model_name, y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted',
    ↪zero_division=0)
    recall = recall_score(y_true, y_pred, average='weighted', zero_division=0)
    f1 = f1_score(y_true, y_pred, average='weighted', zero_division=0)
    cm = confusion_matrix(y_true, y_pred)
    report = classification_report(y_true, y_pred, zero_division=0)

    metrics = {
        'Model Name': model_name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1,
        'Classification Report': report,
    }

    plt.figure(figsize=(4, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
    ↪xticklabels=sorted(y_true.unique()), yticklabels=sorted(y_true.
    ↪unique()))
    plt.title(f'Confusion Matrix for {model_name}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

    return metrics

[32]: # Collect all feature sets built in Step 4 for looping over models
feature_sets = {
    'BoW': (X_train_bow, X_test_bow),
    'TFIDF-Unigram': (X_train_Unigram, X_test_Unigram),
    'TFIDF-Uni+Bi': (X_train_U_B_nigram, X_test_U_B_nigram),
```

```

    'TFIDF-Uni+Bi+Tri': (X_train_U_B_T_nigram, X_test_U_B_T_nigram),
    'W2V-CBOW':         (X_train_cbow, X_test_cbow),
    'W2V-SkipGram':      (X_train_sg, X_test_sg),
}
results = [] # we'll append each model/feature result here as we go

```

```

[33]: # Model 1: Random Forest Classifier - across all feature representations
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
}

for feat_name, (Xtr, Xte) in feature_sets.items():
    rf_classifier = RandomForestClassifier(random_state=42)
    grid_search = GridSearchCV(estimator=rf_classifier,
    ↪ param_grid=param_grid_rf,
                                cv=5, scoring='accuracy', n_jobs=-1)
    grid_search.fit(Xtr, y_train)
    best_rf = grid_search.best_estimator_
    y_pred = best_rf.predict(Xte)

    res = evaluate_model(f'RandomForest ({feat_name})', y_test, y_pred)
    res['Feature'] = feat_name
    results.append(res)

    print(f"[{feat_name}] Accuracy={res['Accuracy']:.4f} F1={res['F1 Score']:.
    ↪ 4f} Best={grid_search.best_params_}")

```

Confusion Matrix for RandomForest (BoW)

True Label	ENTERTAINMENT	225	14	27	11	12	11
	PARENTING	23	217	7	9	9	35
	POLITICS	36	12	221	0	10	21
	STYLE & BEAUTY	31	7	3	238	7	14
	TRAVEL	44	10	5	26	198	17
	WELLNESS	26	13	5	7	11	238
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

[BoW] Accuracy=0.7428 F1=0.7451 Best={'max_depth': None, 'n_estimators': 300}

Confusion Matrix for RandomForest (TFIDF-Unigram)

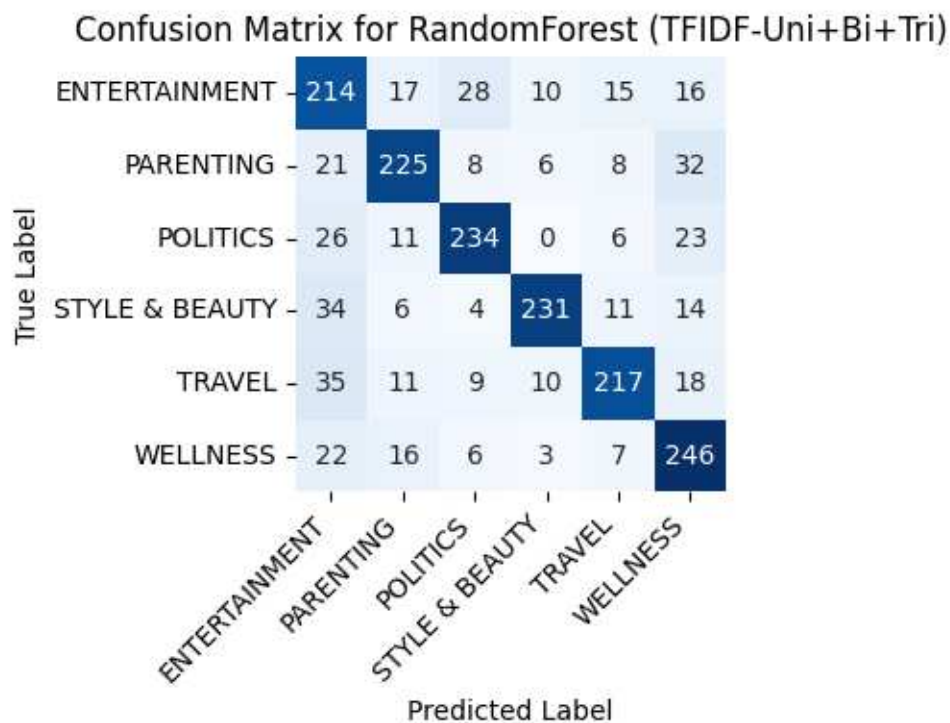
True Label	ENTERTAINMENT	218	14	30	12	13	13
	PARENTING	19	223	5	10	8	35
	POLITICS	33	10	221	1	11	24
	STYLE & BEAUTY	27	10	4	237	10	12
	TRAVEL	46	7	4	22	200	21
	WELLNESS	20	13	6	5	11	245
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

[TFIDF-Unigram] Accuracy=0.7467 F1=0.7484 Best={'max_depth': None, 'n_estimators': 200}

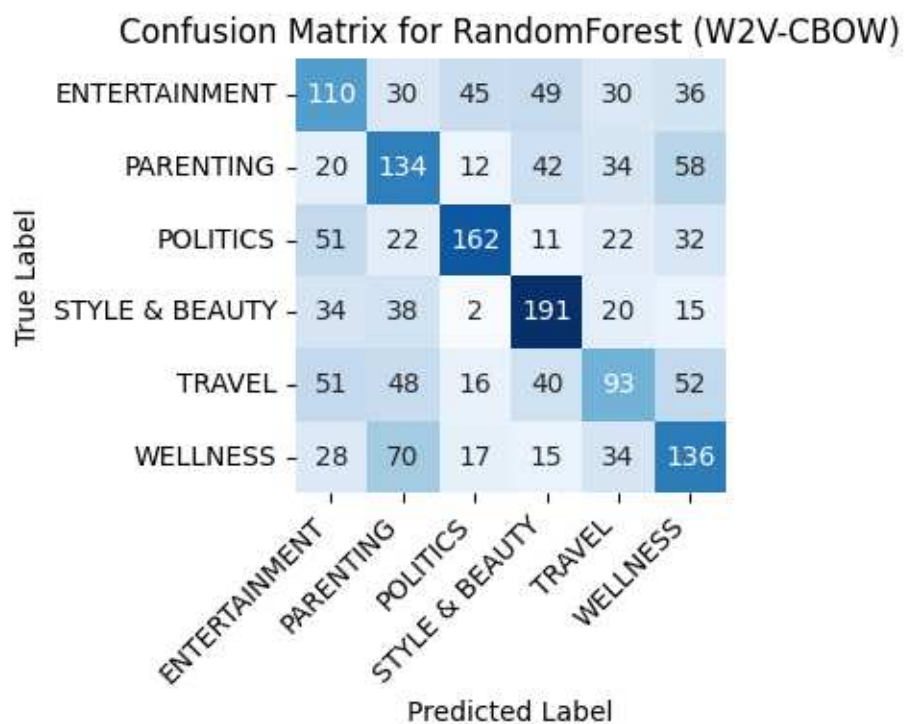
Confusion Matrix for RandomForest (TFIDF-Uni+Bi)

True Label	ENTERTAINMENT	215	16	32	12	12	13
	PARENTING	20	228	6	6	10	30
	POLITICS	30	12	228	1	8	21
	STYLE & BEAUTY	28	5	3	240	11	13
	TRAVEL	36	9	9	17	214	15
	WELLNESS	19	19	8	4	11	239
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

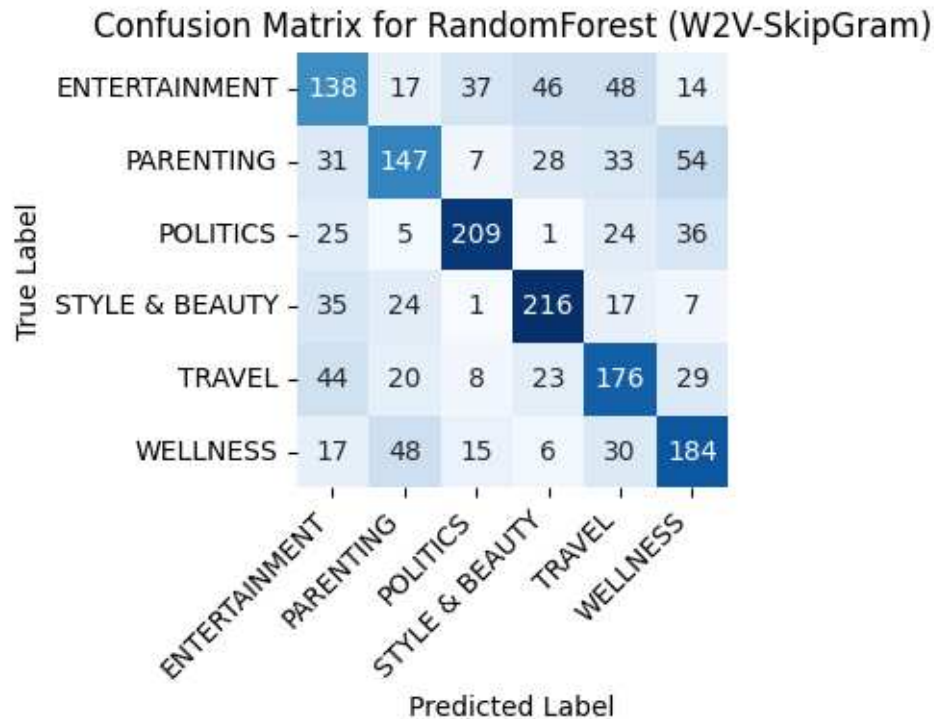
[TFIDF-Uni+Bi] Accuracy=0.7578 F1=0.7595 Best={'max_depth': 20, 'n_estimators': 300}



[TFIDF-Uni+Bi+Tri] Accuracy=0.7594 F1=0.7619 Best={'max_depth': 30, 'n_estimators': 300}



[W2V-CBOW] Accuracy=0.4589 F1=0.4573 Best={'max_depth': None, 'n_estimators': 300}



[W2V-SkipGram] Accuracy=0.5944 F1=0.5939 Best={'max_depth': None, 'n_estimators': 300}

```
[34]: # Model 2: Logistic Regression
from sklearn.linear_model import LogisticRegression

param_grid_lr = {
    'solver': ['liblinear', 'lbfgs'],
}

for feat_name, (Xtr, Xte) in feature_sets.items():
    LR = LogisticRegression(random_state=42, max_iter=1000)
    grid_search = GridSearchCV(estimator=LR, param_grid=param_grid_lr,
                               cv=5, scoring='accuracy', n_jobs=-1)
    grid_search.fit(Xtr, y_train)
    best_LR = grid_search.best_estimator_
    y_pred = best_LR.predict(Xte)
```

```

res = evaluate_model(f'LogisticRegression ({feat_name})', y_test, y_pred)
res['Feature'] = feat_name
results.append(res)

print(f"[{feat_name}] Accuracy={res['Accuracy']:.4f} F1={res['F1 Score']:.4f} Best={grid_search.best_params_}")

```

Confusion Matrix for LogisticRegression (BoW)

True Label	ENTERTAINMENT	238	13	25	6	12	6
	PARENTING	18	220	5	12	7	38
	POLITICS	27	8	239	1	8	17
	STYLE & BEAUTY	20	8	2	251	8	11
	TRAVEL	23	6	10	13	231	17
	WELLNESS	9	15	12	3	9	252
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

[BoW] Accuracy=0.7950 F1=0.7957 Best={'solver': 'liblinear'}

Confusion Matrix for LogisticRegression (TFIDF-Unigram)

True Label	ENTERTAINMENT	227	11	26	10	13	13
	PARENTING	11	224	7	13	7	38
	POLITICS	20	8	241	0	11	20
	STYLE & BEAUTY	18	6	3	249	10	14
	TRAVEL	20	5	5	13	243	14
	WELLNESS	10	10	5	0	10	265
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

[TFIDF-Unigram] Accuracy=0.8050 F1=0.8054 Best={'solver': 'lbfgs'}

Confusion Matrix for LogisticRegression (TFIDF-Uni+Bi)

True Label	ENTERTAINMENT	210	16	28	14	13	19
	PARENTING	8	234	6	9	8	35
	POLITICS	13	12	240	0	11	24
	STYLE & BEAUTY	12	7	3	253	8	17
	TRAVEL	11	6	4	15	248	16
	WELLNESS	1	13	5	1	9	271
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

[TFIDF-Uni+Bi] Accuracy=0.8089 F1=0.8087 Best={'solver': 'liblinear'}

Confusion Matrix for LogisticRegression (TFIDF-Uni+Bi+Tri)

True Label	ENTERTAINMENT	199	19	29	18	13	22
	PARENTING	7	237	5	9	7	35
	POLITICS	10	14	240	1	11	24
	STYLE & BEAUTY	9	8	4	254	8	17
	TRAVEL	11	8	4	16	244	17
	WELLNESS	1	13	5	1	8	272
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

[TFIDF-Uni+Bi+Tri] Accuracy=0.8033 F1=0.8026 Best={'solver': 'liblinear'}

Confusion Matrix for LogisticRegression (W2V-CBOW)

True Label	ENTERTAINMENT	162	45	17	18	35	23
	PARENTING	50	135	5	24	41	45
	POLITICS	111	66	76	11	12	24
	STYLE & BEAUTY	78	81	1	99	24	17
	TRAVEL	119	94	1	12	35	39
	WELLNESS	80	101	8	6	45	60
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

[W2V-CBOW] Accuracy=0.3150 F1=0.3101 Best={'solver': 'lbfgs'}

Confusion Matrix for LogisticRegression (W2V-SkipGram)

True Label	ENTERTAINMENT	116	22	49	56	43	14
	PARENTING	28	127	14	41	42	48
	POLITICS	30	6	212	4	21	27
	STYLE & BEAUTY	36	22	1	221	10	10
	TRAVEL	58	37	9	34	130	32
	WELLNESS	16	74	42	12	30	126
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

[W2V-SkipGram] Accuracy=0.5178 F1=0.5118 Best={'solver': 'liblinear'}

```
[35]: # Model 3: Decision Tree
from sklearn.tree import DecisionTreeClassifier

param_grid_dt = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
}

for feat_name, (Xtr, Xte) in feature_sets.items():
    DT = DecisionTreeClassifier(random_state=42)
    grid_search = GridSearchCV(estimator=DT, param_grid=param_grid_dt,
                               cv=5, scoring='accuracy', n_jobs=-1)
    grid_search.fit(Xtr, y_train)
    best_DT = grid_search.best_estimator_
    y_pred = best_DT.predict(Xte)

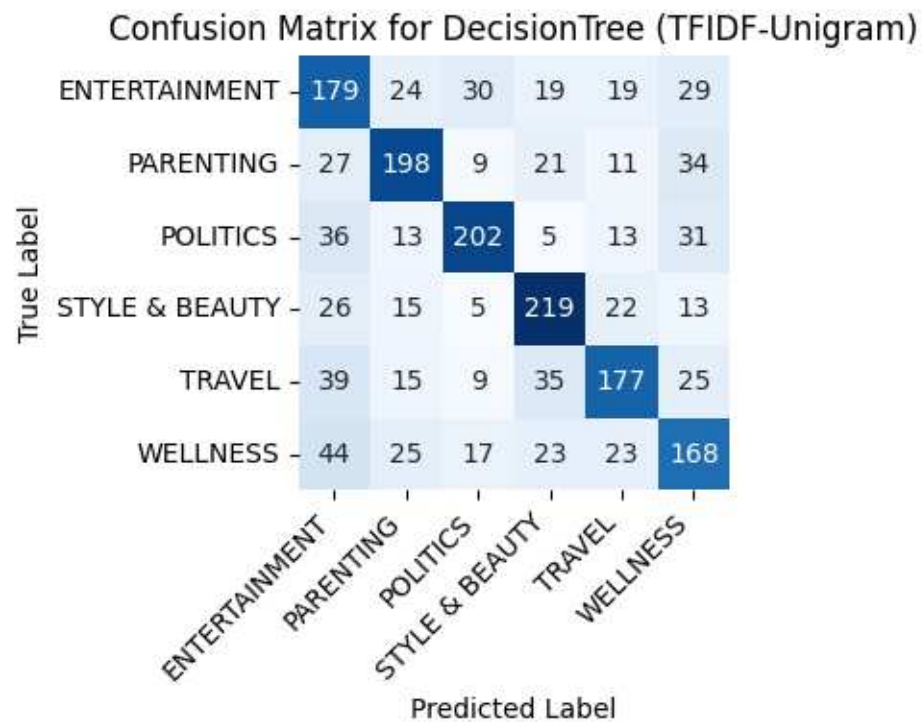
    res = evaluate_model(f'DecisionTree ({feat_name})', y_test, y_pred)
    res['Feature'] = feat_name
    results.append(res)

    print(f"[{feat_name}] Accuracy={res['Accuracy']:.4f} F1={res['F1 Score']:.4f} Best={grid_search.best_params_}")
```

Confusion Matrix for DecisionTree (BoW)

True Label	ENTERTAINMENT	182	19	30	18	20	31
	PARENTING	16	198	16	14	17	39
	POLITICS	33	12	208	2	14	31
	STYLE & BEAUTY	36	14	4	212	20	14
	TRAVEL	35	16	17	31	174	27
	WELLNESS	31	18	17	12	24	198
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

[BoW] Accuracy=0.6511 F1=0.6524 Best={'criterion': 'gini', 'max_depth': None}



[TFIDF-Unigram] Accuracy=0.6350 F1=0.6364 Best={'criterion': 'gini', 'max_depth': None}

Confusion Matrix for DecisionTree (TFIDF-Uni+Bi)

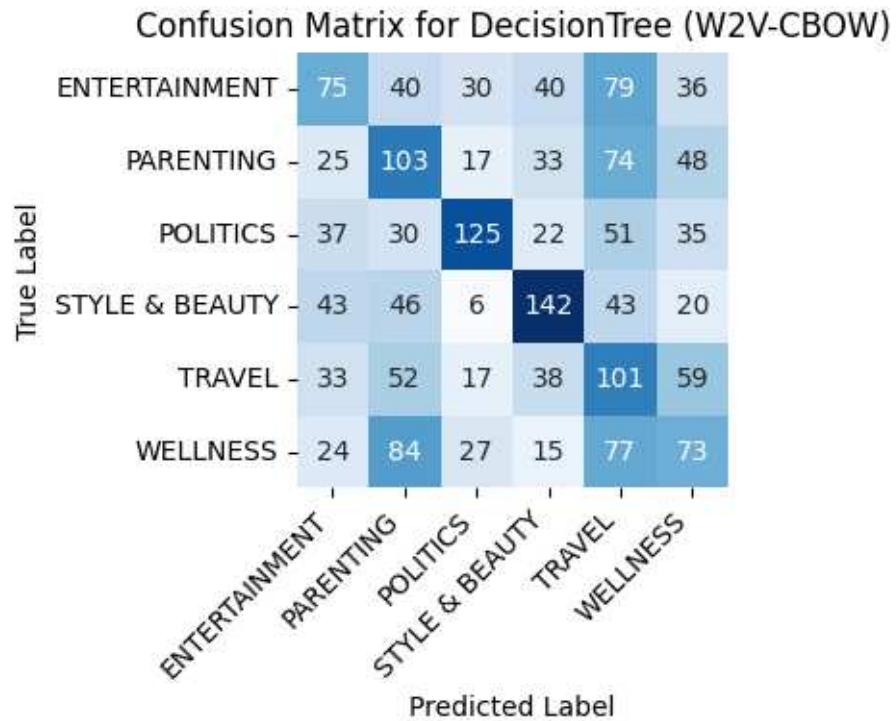
True Label	ENTERTAINMENT	184	27	35	17	14	23
	PARENTING	33	192	6	22	16	31
	POLITICS	46	11	193	4	14	32
	STYLE & BEAUTY	40	11	6	216	15	12
	TRAVEL	45	15	16	33	165	26
	WELLNESS	57	27	20	14	19	163
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

[TFIDF-Uni+Bi] Accuracy=0.6183 F1=0.6211 Best={'criterion': 'gini', 'max_depth': None}

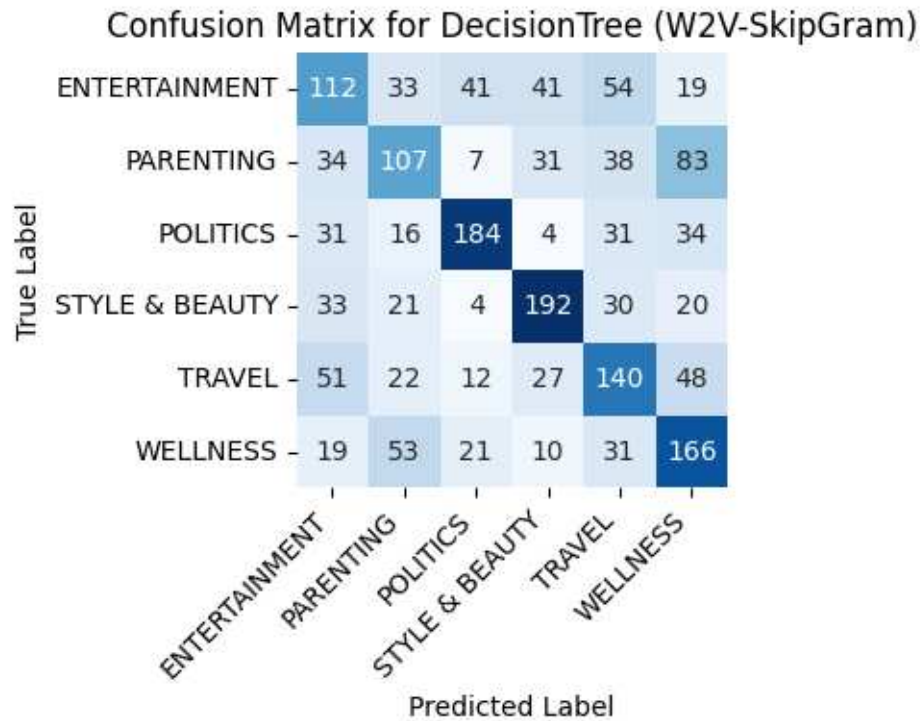
Confusion Matrix for DecisionTree (TFIDF-Uni+Bi+Tri)

True Label	ENTERTAINMENT	170	24	35	11	19	41
	PARENTING	26	191	9	16	12	46
	POLITICS	46	14	193	1	13	33
	STYLE & BEAUTY	31	12	11	206	15	25
	TRAVEL	47	13	14	31	163	32
	WELLNESS	41	29	18	17	18	177
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

[TFIDF-Uni+Bi+Tri] Accuracy=0.6111 F1=0.6147 Best={'criterion': 'gini', 'max_depth': None}



[W2V-CBOW] Accuracy=0.3439 F1=0.3481 Best={'criterion': 'gini', 'max_depth': 10}



[W2V-SkipGram] Accuracy=0.5006 F1=0.4999 Best={'criterion': 'gini', 'max_depth': 10}

```
[36]: # Model 4: SVM
from sklearn.svm import SVC

param_grid_svm = {
    'kernel': ['linear', 'rbf', 'poly'],
}

for feat_name, (Xtr, Xte) in feature_sets.items():
    SVM = SVC(random_state=42)
    grid_search = GridSearchCV(estimator=SVM, param_grid=param_grid_svm,
                               cv=5, scoring='accuracy', n_jobs=-1)
    grid_search.fit(Xtr, y_train)
    best_SVM = grid_search.best_estimator_
    y_pred = best_SVM.predict(Xte)

    res = evaluate_model(f'SVM ({feat_name})', y_test, y_pred)
    res['Feature'] = feat_name
    results.append(res)
```

```
print(f"[{feat_name}] Accuracy={res['Accuracy']:.4f} F1={res['F1 Score']:.4f} Best={grid_search.best_params_}")
```

Confusion Matrix for SVM (BoW)

True Label	ENTERTAINMENT	238	9	26	8	13	6
	PARENTING	24	214	7	14	12	29
	POLITICS	35	11	233	0	6	15
	STYLE & BEAUTY	29	8	3	238	12	10
	TRAVEL	34	7	14	12	216	17
	WELLNESS	18	23	11	7	13	228
		Predicted Label					
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS

[BoW] Accuracy=0.7594 F1=0.7610 Best={'kernel': 'linear'}

Confusion Matrix for SVM (TFIDF-Unigram)

True Label	ENTERTAINMENT	226	13	25	12	11	13
	PARENTING	16	226	8	9	6	35
	POLITICS	25	9	239	0	7	20
	STYLE & BEAUTY	21	9	2	247	9	12
	TRAVEL	20	3	6	11	238	22
	WELLNESS	9	15	6	1	10	259
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

[TFIDF-Unigram] Accuracy=0.7972 F1=0.7982 Best={'kernel': 'linear'}

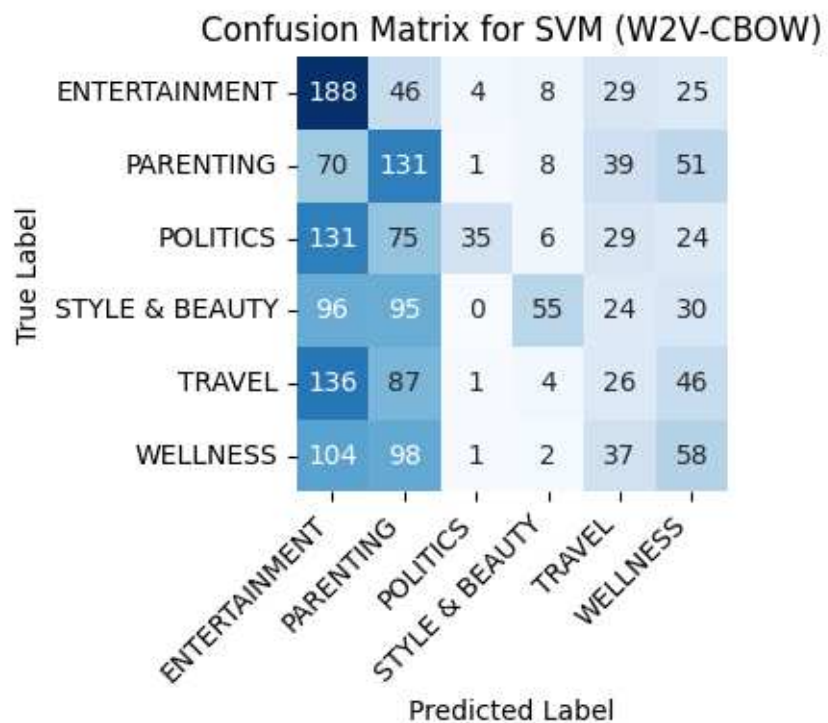
Confusion Matrix for SVM (TFIDF-Uni+Bi)

True Label	ENTERTAINMENT	214	14	30	13	12	17
	PARENTING	9	231	6	10	7	37
	POLITICS	16	11	238	0	11	24
	STYLE & BEAUTY	9	11	3	255	6	16
	TRAVEL	13	7	4	13	244	19
	WELLNESS	3	15	5	0	8	269
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

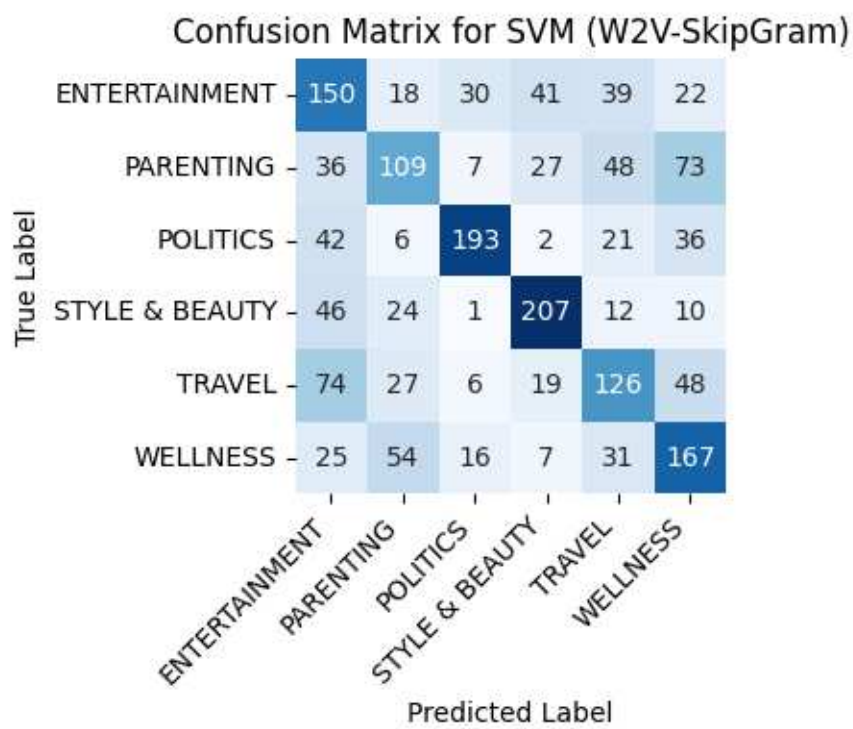
[TFIDF-Uni+Bi] Accuracy=0.8061 F1=0.8062 Best={'kernel': 'linear'}

True Label	ENTERTAINMENT	206	14	31	16	11	22
	PARENTING	6	231	6	13	5	39
	POLITICS	11	11	241	0	10	27
	STYLE & BEAUTY	7	10	4	256	7	16
	TRAVEL	9	11	4	14	241	21
	WELLNESS	1	15	5	0	8	271
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

[TFIDF-Uni+Bi+Tri] Accuracy=0.8033 F1=0.8034 Best={'kernel': 'linear'}



[W2V-CBOW] Accuracy=0.2739 F1=0.2497 Best={'kernel': 'linear'}



[W2V-SkipGram] Accuracy=0.5289 F1=0.5302 Best={'kernel': 'linear'}

```
[37]: # K-Nearest Neighbors
from sklearn.neighbors import KNeighborsClassifier

param_grid_knn = {
    'n_neighbors': [1, 3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
}

for feat_name, (Xtr, Xte) in feature_sets.items():
    knn = KNeighborsClassifier()
    grid_search = GridSearchCV(estimator=knn, param_grid=param_grid_knn,
                               cv=5, scoring='accuracy', n_jobs=-1)
    grid_search.fit(Xtr, y_train)
    best_knn = grid_search.best_estimator_
    y_pred = best_knn.predict(Xte)

    res = evaluate_model(f'KNN ({feat_name})', y_test, y_pred)
    res['Feature'] = feat_name
    results.append(res)

    print(f"[{feat_name}] Accuracy={res['Accuracy']:.4f} F1={res['F1 Score']:.4f} Best={grid_search.best_params_}")
```

True Label \ Predicted Label	ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
ENTERTAINMENT	59	0	15	4	222	0
PARENTING	56	9	13	5	217	0
POLITICS	16	1	111	2	170	0
STYLE & BEAUTY	53	0	3	53	191	0
TRAVEL	29	0	2	1	268	0
WELLNESS	37	0	9	1	239	14

```

/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_search.py:1108:
UserWarning: One or more of the test scores are non-finite: [0.58857143
0.58857143 0.62642857 0.63238095 0.66785714 0.67857143
0.68904762 0.69428571 0.70452381 0.7097619          nan 0.22261905
          nan 0.26309524          nan 0.30119048          nan 0.28928571
          nan 0.26047619 0.58857143 0.58857143 0.62642857 0.63238095
0.66785714 0.67857143 0.68904762 0.69428571 0.70452381 0.7097619
          nan 0.22261905          nan 0.26309524          nan 0.30119048
          nan 0.28928571          nan 0.26047619 0.58857143 0.58857143
0.62642857 0.63238095 0.66785714 0.67857143 0.68904762 0.69428571
0.70452381 0.7097619          nan 0.22261905          nan 0.26309524
          nan 0.30119048          nan 0.28928571          nan 0.26047619
0.58857143 0.58857143 0.62642857 0.63238095 0.66785714 0.67857143
0.68904762 0.69428571 0.70452381 0.7097619          nan 0.22261905
          nan 0.26309524          nan 0.30119048          nan 0.28928571
          nan 0.26047619]
warnings.warn(

```

Confusion Matrix for KNN (TFIDF-Unigram)

True Label	ENTERTAINMENT	186	27	32	23	17	15
	PARENTING	18	196	10	23	9	44
	POLITICS	11	13	243	5	11	17
	STYLE & BEAUTY	24	16	9	225	12	14
	TRAVEL	12	22	11	12	217	26
	WELLNESS	5	38	11	3	14	229
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

[TFIDF-Unigram] Accuracy=0.7200 F1=0.7197 Best={'algorithm': 'auto', 'metric': 'euclidean', 'n_neighbors': 9, 'weights': 'distance'}

/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_search.py:1108:

UserWarning: One or more of the test scores are non-finite: [0.59333333

0.59333333 0.63285714 0.64095238 0.66880952 0.67190476

0.68833333 0.6952381 0.70142857 0.71285714 nan 0.19571429

nan 0.20690476 nan 0.19714286 nan 0.18619048

nan 0.18333333 0.59333333 0.59333333 0.63285714 0.64095238

0.66880952 0.67190476 0.68833333 0.6952381 0.70142857 0.71285714

nan 0.19571429 nan 0.20690476 nan 0.19714286

nan 0.18619048 nan 0.18333333 0.59333333 0.59333333

0.63285714 0.64095238 0.66880952 0.67190476 0.68833333 0.6952381

0.70142857 0.71285714 nan 0.19571429 nan 0.20690476

nan 0.19714286 nan 0.18619048 nan 0.18333333

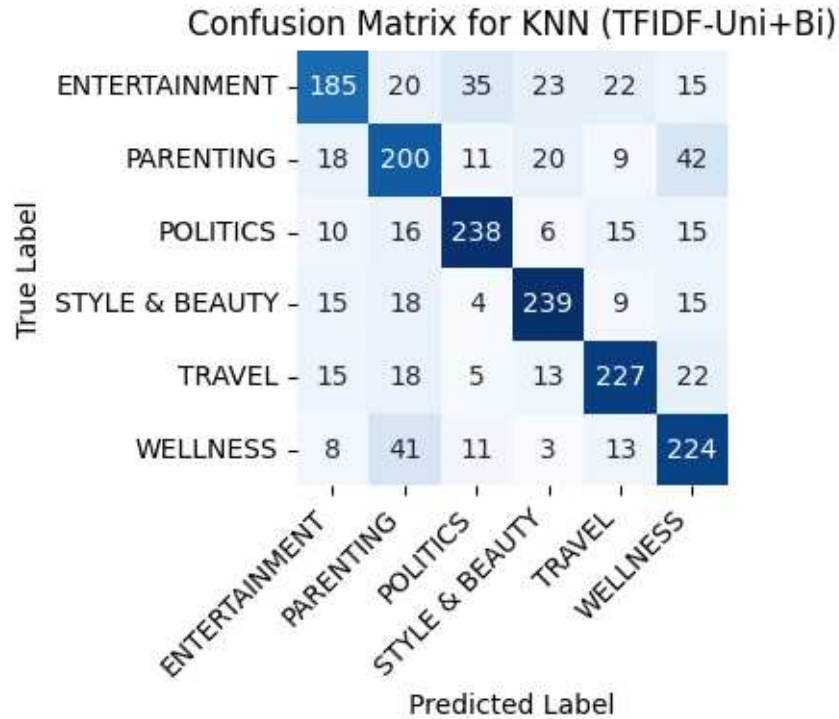
0.59333333 0.59333333 0.63285714 0.64095238 0.66880952 0.67190476

0.68833333 0.6952381 0.70142857 0.71285714 nan 0.19571429

nan 0.20690476 nan 0.19714286 nan 0.18619048

nan 0.18333333]

warnings.warn(



[TFIDF-Uni+Bi] Accuracy=0.7294 F1=0.7290 Best={'algorithm': 'auto', 'metric': 'euclidean', 'n_neighbors': 9, 'weights': 'distance'}

/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_search.py:1108:

UserWarning: One or more of the test scores are non-finite: [0.59309524

0.59309524 0.63166667 0.63738095 0.66428571 0.67119048

0.68928571 0.69666667 0.70357143 0.71071429 nan 0.1852381

nan 0.1852381 nan 0.18214286 nan 0.18071429

nan 0.1797619 0.59309524 0.59309524 0.63166667 0.63738095

0.66428571 0.67119048 0.68928571 0.69666667 0.70357143 0.71071429

nan 0.1852381 nan 0.1852381 nan 0.18214286

nan 0.18071429 nan 0.1797619 0.59309524 0.59309524

0.63166667 0.63738095 0.66428571 0.67119048 0.68928571 0.69666667

0.70357143 0.71071429 nan 0.1852381 nan 0.1852381

nan 0.18214286 nan 0.18071429 nan 0.1797619

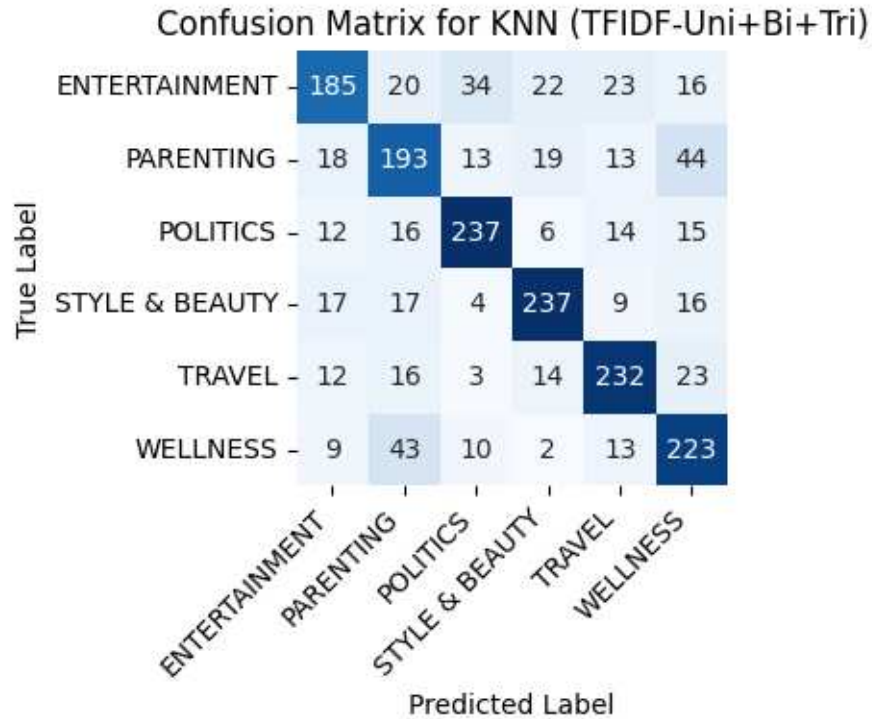
0.59309524 0.59309524 0.63166667 0.63738095 0.66428571 0.67119048

0.68928571 0.69666667 0.70357143 0.71071429 nan 0.1852381

nan 0.1852381 nan 0.18214286 nan 0.18071429

nan 0.1797619]

warnings.warn(



```
[TFIDF-Uni+Bi+Tri] Accuracy=0.7261 F1=0.7257 Best={'algorithm': 'auto',
'metric': 'euclidean', 'n_neighbors': 9, 'weights': 'distance'}
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_search.py:1108:
```

```
UserWarning: One or more of the test scores are non-finite: [0.34642857
```

```
0.34642857 0.34166667 0.3547619 0.35119048 0.36595238
```

```
0.34857143 0.36095238 0.34309524 0.35928571 nan 0.3502381
```

```
nan 0.36047619 nan 0.36428571 nan 0.35738095
```

```
nan 0.35833333 0.34642857 0.34642857 0.34166667 0.3547619
```

```
0.35119048 0.36595238 0.34857143 0.36095238 0.34309524 0.35928571
```

```
0.3502381 0.3502381 0.3452381 0.36047619 0.35142857 0.36428571
```

```
0.34142857 0.35738095 0.34428571 0.35833333 0.34642857 0.34642857
```

```
0.34166667 0.3547619 0.35119048 0.36595238 0.34857143 0.36095238
```

```
0.34309524 0.35928571 0.3502381 0.3502381 0.3452381 0.36047619
```

```
0.35142857 0.36428571 0.34142857 0.35738095 0.34428571 0.35833333
```

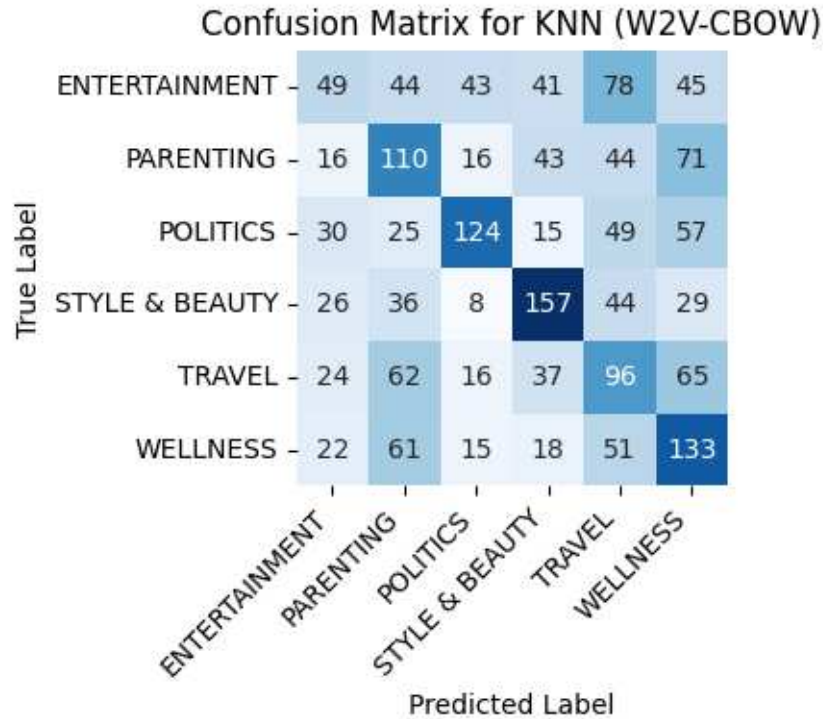
```
0.34642857 0.34642857 0.34166667 0.3547619 0.35119048 0.36595238
```

```
0.34857143 0.36095238 0.34309524 0.35928571 nan 0.3502381
```

```
nan 0.36047619 nan 0.36428571 nan 0.35738095
```

```
nan 0.35833333]
```

```
warnings.warn(
```



[W2V-CBOW] Accuracy=0.3717 F1=0.3690 Best={'algorithm': 'auto', 'metric': 'euclidean', 'n_neighbors': 5, 'weights': 'distance'}

/usr/local/lib/python3.12/dist-packages/sklearn/model_selection/_search.py:1108:

UserWarning: One or more of the test scores are non-finite: [0.45333333

0.45333333 0.49 0.48261905 0.50095238 0.50404762

0.52333333 0.52071429 0.52333333 0.5252381 nan 0.45619048

nan 0.48190476 nan 0.50880952 nan 0.51690476

nan 0.52785714 0.45333333 0.45333333 0.49 0.48261905

0.50095238 0.50404762 0.52333333 0.52071429 0.52333333 0.5252381

0.45619048 0.45619048 0.48404762 0.48190476 0.5047619 0.50880952

0.51666667 0.51690476 0.52880952 0.52785714 0.45333333 0.45333333

0.49 0.48261905 0.50095238 0.50404762 0.52333333 0.52071429

0.52333333 0.5252381 0.45619048 0.45619048 0.48404762 0.48190476

0.5047619 0.50880952 0.51666667 0.51690476 0.52880952 0.52785714

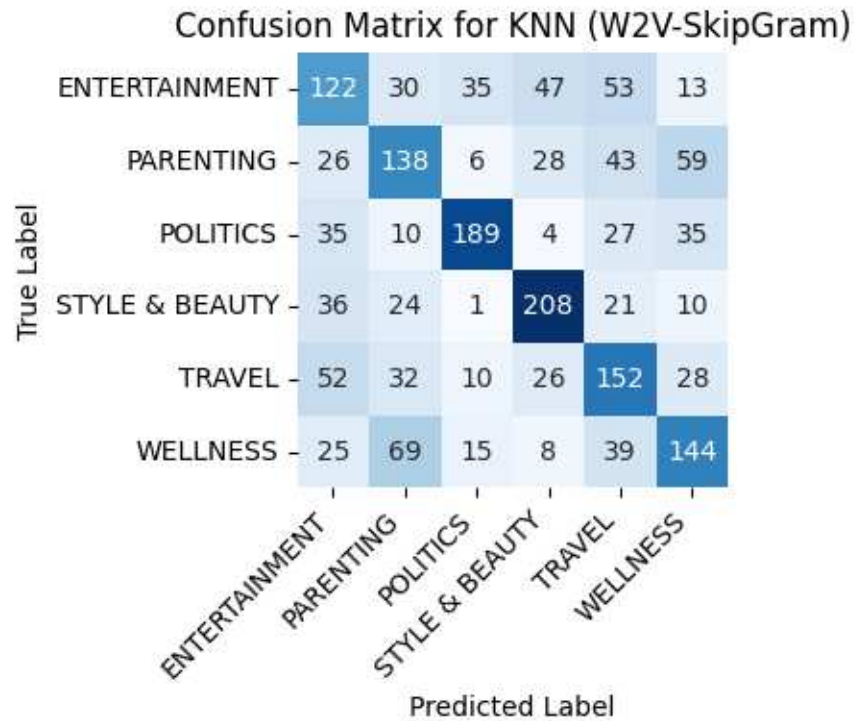
0.45333333 0.45333333 0.49 0.48261905 0.50095238 0.50404762

0.52333333 0.52071429 0.52333333 0.5252381 nan 0.45619048

nan 0.48190476 nan 0.50880952 nan 0.51690476

nan 0.52785714]

warnings.warn(



[W2V-SkipGram] Accuracy=0.5294 F1=0.5308 Best={'algorithm': 'ball_tree', 'metric': 'manhattan', 'n_neighbors': 9, 'weights': 'uniform'}

```
[38]: # Naive Bayes
# MultinomialNB for the sparse BoW/TF-IDF features, GaussianNB for the dense
# Word2Vec features
from sklearn.naive_bayes import MultinomialNB, GaussianNB

dense_features = {'W2V-CBOW', 'W2V-SkipGram'}

for feat_name, (Xtr, Xte) in feature_sets.items():
    if feat_name in dense_features:
        nb = GaussianNB()
        param_grid_nb = {'var_smoothing': [1e-9, 1e-8]}
    else:
        nb = MultinomialNB()
        param_grid_nb = {'alpha': [0.5, 1.0]}

    grid_search = GridSearchCV(estimator=nb, param_grid=param_grid_nb,
                               cv=5, scoring='accuracy', n_jobs=-1)
    grid_search.fit(Xtr, y_train)
    best_nb = grid_search.best_estimator_
    y_pred = best_nb.predict(Xte)
```

```

res = evaluate_model(f'NaiveBayes ({feat_name})', y_test, y_pred)
res['Feature'] = feat_name
results.append(res)

print(f"[{feat_name}] Accuracy={res['Accuracy']:.4f} F1={res['F1 Score']:.4f} Best={grid_search.best_params_}")

```

Confusion Matrix for NaiveBayes (BoW)

ENTERTAINMENT	194	26	24	28	14	14
PARENTING	7	229	9	13	9	33
POLITICS	9	20	248	3	7	13
STYLE & BEAUTY	9	13	4	252	12	10
TRAVEL	7	16	7	8	247	15
WELLNESS	2	38	6	0	9	245

True Label

Predicted Label

[BoW] Accuracy=0.7861 F1=0.7860 Best={'alpha': 0.5}

Confusion Matrix for NaiveBayes (TFIDF-Unigram)

True Label	ENTERTAINMENT	193	24	28	27	15	13
	PARENTING	6	232	9	12	5	36
	POLITICS	7	18	250	3	8	14
	STYLE & BEAUTY	10	16	4	252	7	11
	TRAVEL	5	23	6	8	240	18
	WELLNESS	0	36	5	0	7	252
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

[TFIDF-Unigram] Accuracy=0.7883 F1=0.7885 Best={'alpha': 0.5}

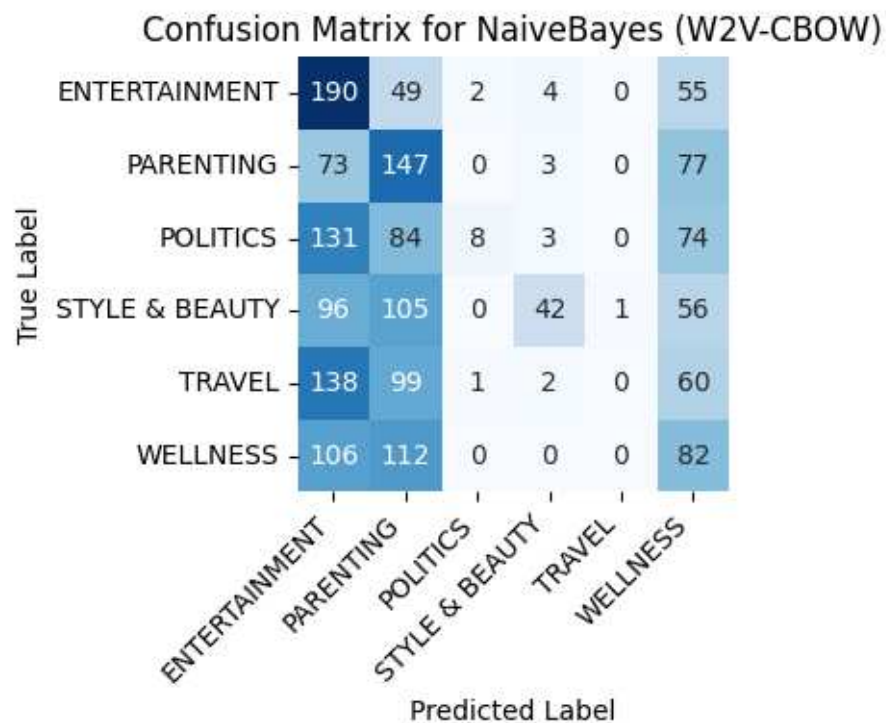
Confusion Matrix for NaiveBayes (TFIDF-Uni+Bi)

True Label	ENTERTAINMENT	189	25	28	25	16	17
	PARENTING	6	234	6	15	7	32
	POLITICS	6	14	255	2	7	16
	STYLE & BEAUTY	6	16	4	255	8	11
	TRAVEL	5	19	5	9	245	17
	WELLNESS	0	28	5	1	8	258
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

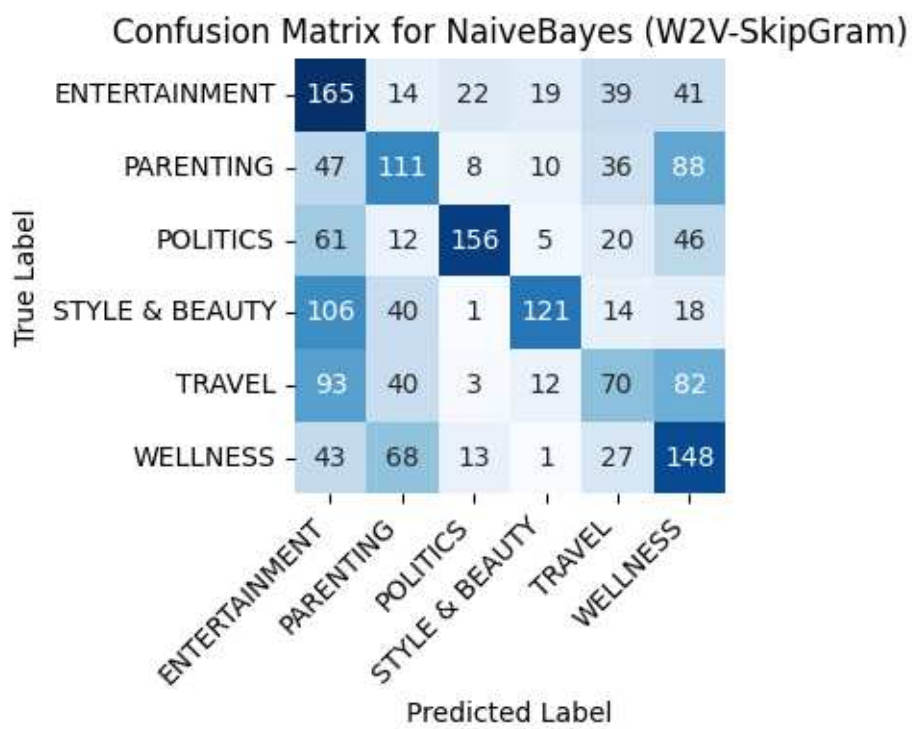
[TFIDF-Uni+Bi] Accuracy=0.7978 F1=0.7970 Best={'alpha': 0.5}

True Label	ENTERTAINMENT	189	24	28	25	16	18
	PARENTING	6	234	5	16	7	32
	POLITICS	6	14	255	2	7	16
	STYLE & BEAUTY	5	15	4	256	8	12
	TRAVEL	4	17	4	12	245	18
	WELLNESS	0	26	5	1	8	260
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

[TFIDF-Uni+Bi+Tri] Accuracy=0.7994 F1=0.7985 Best={'alpha': 0.5}



[W2V-CBOW] Accuracy=0.2606 F1=0.2029 Best={'var_smoothing': 1e-09}



[W2V-SkipGram] Accuracy=0.4283 F1=0.4346 Best={'var_smoothing': 1e-09}

```
[39]: # Build results DataFrame and show top combinations
results_df = pd.DataFrame(results)
summary = results_df[['Feature', 'Model Name', 'Accuracy', 'Precision',
↳ 'Recall', 'F1 Score']] \
    .sort_values('F1 Score', ascending=False).reset_index(drop=True)
summary.head(15)
```

```
[39]:
```

	Feature	Model Name	Accuracy \
0	TFIDF-Uni+Bi	LogisticRegression (TFIDF-Uni+Bi)	0.808889
1	TFIDF-Uni+Bi	SVM (TFIDF-Uni+Bi)	0.806111
2	TFIDF-Unigram	LogisticRegression (TFIDF-Unigram)	0.805000
3	TFIDF-Uni+Bi+Tri	SVM (TFIDF-Uni+Bi+Tri)	0.803333
4	TFIDF-Uni+Bi+Tri	LogisticRegression (TFIDF-Uni+Bi+Tri)	0.803333
5	TFIDF-Uni+Bi+Tri	NaiveBayes (TFIDF-Uni+Bi+Tri)	0.799444
6	TFIDF-Unigram	SVM (TFIDF-Unigram)	0.797222
7	TFIDF-Uni+Bi	NaiveBayes (TFIDF-Uni+Bi)	0.797778
8	BoW	LogisticRegression (BoW)	0.795000
9	TFIDF-Unigram	NaiveBayes (TFIDF-Unigram)	0.788333
10	BoW	NaiveBayes (BoW)	0.786111
11	TFIDF-Uni+Bi+Tri	RandomForest (TFIDF-Uni+Bi+Tri)	0.759444
12	BoW	SVM (BoW)	0.759444
13	TFIDF-Uni+Bi	RandomForest (TFIDF-Uni+Bi)	0.757778
14	TFIDF-Unigram	RandomForest (TFIDF-Unigram)	0.746667

	Precision	Recall	F1 Score
0	0.814345	0.808889	0.808677
1	0.811630	0.806111	0.806211
2	0.809712	0.805000	0.805380
3	0.812125	0.803333	0.803435
4	0.809895	0.803333	0.802640
5	0.808174	0.799444	0.798501
6	0.802831	0.797222	0.798171
7	0.806185	0.797778	0.796972
8	0.799598	0.795000	0.795669
9	0.797322	0.788333	0.788492
10	0.792152	0.786111	0.785996
11	0.769943	0.759444	0.761920
12	0.767269	0.759444	0.760983
13	0.764609	0.757778	0.759538
14	0.756737	0.746667	0.748448

```
[40]: # Extract clean model names (strip the "(Feature)" suffix) for pivoting
results_df['Model'] = results_df['Model Name'].str.extract(r'^([^\(]+)')[0].str.
↳ strip()
```

```

pivot = results_df.pivot(index='Model', columns='Feature', values='F1 Score')
pivot = pivot[['BoW', 'TFIDF-Unigram', 'TFIDF-Uni+Bi', 'TFIDF-Uni+Bi+Tri', '
↳ 'W2V-CBOW', 'W2V-SkipGram']]
pivot

```

```

[40]: Feature          BoW  TFIDF-Unigram  TFIDF-Uni+Bi  TFIDF-Uni+Bi+Tri  \
Model
DecisionTree      0.652377      0.636355      0.621127      0.614669
KNN                0.245834      0.719721      0.729045      0.725691
LogisticRegression 0.795669      0.805380      0.808677      0.802640
NaiveBayes         0.785996      0.788492      0.796972      0.798501
RandomForest      0.745052      0.748448      0.759538      0.761920
SVM                0.760983      0.798171      0.806211      0.803435

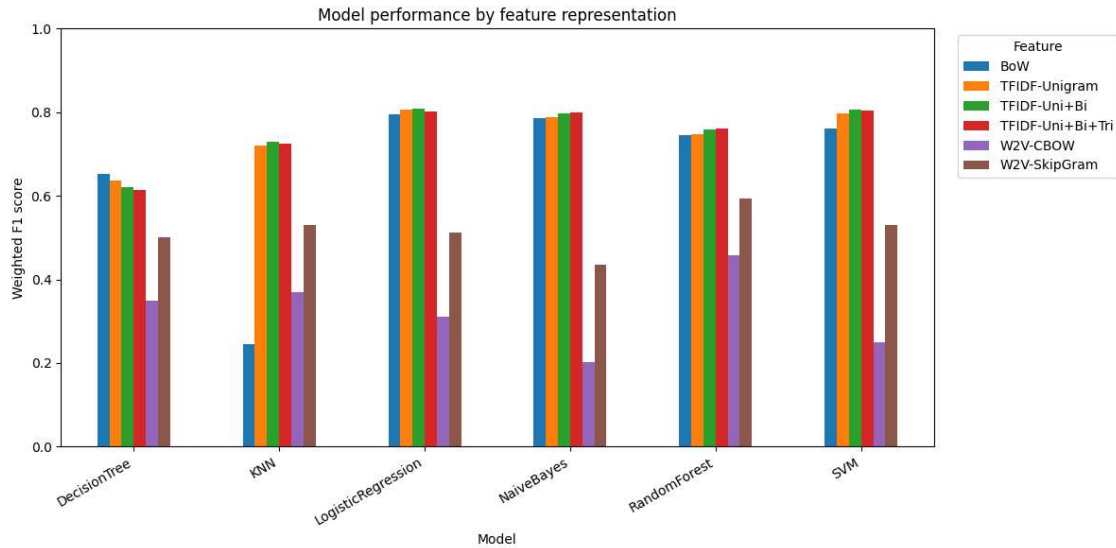
Feature          W2V-CBOW  W2V-SkipGram
Model
DecisionTree      0.348072      0.499931
KNN                0.368953      0.530759
LogisticRegression 0.310132      0.511826
NaiveBayes         0.202887      0.434642
RandomForest      0.457260      0.593887
SVM                0.249714      0.530239

```

```

[41]: # Bar chart comparing all models across all feature representations
ax = pivot.plot(kind='bar', figsize=(12, 6))
ax.set_ylabel('Weighted F1 score')
ax.set_title('Model performance by feature representation')
ax.set_ylim(0, 1)
plt.xticks(rotation=30, ha='right')
plt.legend(title='Feature', bbox_to_anchor=(1.02, 1), loc='upper left')
plt.tight_layout()
plt.show()

```



```
[42]: # Best overall combination and its full classification report
best_row = summary.iloc[0]
best_match = results_df[
    (results_df['Model'] == best_row['Model Name'].split(' ')[0]) &
    (results_df['Feature'] == best_row['Feature'])
].iloc[0]

print(f"Best combination: {best_row['Model Name']}")
print(f"Accuracy={best_row['Accuracy']:.4f} Precision={best_row['Precision']:.4f} "
      f"Recall={best_row['Recall']:.4f} F1={best_row['F1 Score']:.4f}\n")
print(best_match['Classification Report'])
```

Best combination: LogisticRegression (TFIDF-Uni+Bi)
 Accuracy=0.8089 Precision=0.8143 Recall=0.8089 F1=0.8087

	precision	recall	f1-score	support
ENTERTAINMENT	0.82	0.70	0.76	300
PARENTING	0.81	0.78	0.80	300
POLITICS	0.84	0.80	0.82	300
STYLE & BEAUTY	0.87	0.84	0.85	300
TRAVEL	0.84	0.83	0.83	300
WELLNESS	0.71	0.90	0.79	300
accuracy			0.81	1800
macro avg	0.81	0.81	0.81	1800
weighted avg	0.81	0.81	0.81	1800