

```
from IPython.display import Image, display
display(Image(filename='Lecture 7.png'))
```



Download dataset:

AG News Classification Dataset

The link dataset: <https://www.kaggle.com/datasets/amananandrai/ag-news-classification-dataset> The AG's news topic classification dataset is constructed by choosing 4 largest classes from the original corpus. Each class contains 30,000 training samples and 1,900 testing samples. The total number of training samples is 120,000 and testing 7,600.

Import necessary libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import GridSearchCV
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, recall_score, f1_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.metrics import classification_report
from sklearn.feature_extraction.text import TfidfVectorizer
```

Reading training set

```
training = pd.read_csv("train.csv", encoding="ISO-8859-1")
training.head()
```

	Class	Index	Title \
0	3	Wall St. Bears Claw Back Into the Black (Reuters)	
1	3	Carlyle Looks Toward Commercial Aerospace (Reu...	
2	3	Oil and Economy Cloud Stocks' Outlook (Reuters)	
3	3	Iraq Halts Oil Exports from Main Southern Pipe...	
4	3	Oil prices soar to all-time record, posing new...	

	Description
0	Reuters - Short-sellers, Wall Street's dwindli...
1	Reuters - Private investment firm Carlyle Grou...
2	Reuters - Soaring crude prices plus worries\ab...
3	Reuters - Authorities have halted oil export\f...
4	AFP - Tearaway world oil prices, toppling reco...

```
len(training)
```

```
120000
```

```
training['Class Index'].value_counts()
```

```
Class Index
```

```
3    30000
```

```
4    30000
```

```
2    30000
```

```
1    30000
```

```
Name: count, dtype: int64
```

```
training = training.groupby('Class Index',
group_keys=False).apply(lambda x: x.sample(n=1000, random_state=42))
training = training.reset_index(drop=True)
```

```
C:\Users\hager\AppData\Local\Temp\ipykernel_39148\1402402670.py:1:
DeprecationWarning: DataFrameGroupBy.apply operated on the grouping
columns. This behavior is deprecated, and in a future version of
pandas the grouping columns will be excluded from the operation.
Either pass `include_groups=False` to exclude the groupings or
explicitly select the grouping columns after groupby to silence this
warning.
```

```
training = training.groupby('Class Index',
group_keys=False).apply(lambda x: x.sample(n=1000, random_state=42))
```

Reading testing set

```
testing = pd.read_csv("test.csv", encoding="ISO-8859-1")
testing.head()
```

	Class	Index	Title \
0		3	Fears for T N pension after talks
1		4	The Race is On: Second Private Team Sets Launc...
2		4	Ky. Company Wins Grant to Study Peptides (AP)
3		4	Prediction Unit Helps Forecast Wildfires (AP)
4		4	Calif. Aims to Limit Farm-Related Smog (AP)

	Description
0	Unions representing workers at Turner Newall...
1	SPACE.com - TORONTO, Canada -- A second\team o...
2	AP - A company founded by a chemistry research...
3	AP - It's barely dawn when Mike Fitzpatrick st...
4	AP - Southern California's smog-fighting agenc...

```
len(testing)
```

```
7600
```

```
testing = testing.groupby('Class Index',
group_keys=False).apply(lambda x: x.sample(n=1000, random_state=42))
testing = testing.reset_index(drop=True)
```

```
C:\Users\hager\AppData\Local\Temp\ipykernel_39148\3048157673.py:1:
DeprecationWarning: DataFrameGroupBy.apply operated on the grouping
columns. This behavior is deprecated, and in a future version of
pandas the grouping columns will be excluded from the operation.
Either pass `include_groups=False` to exclude the groupings or
explicitly select the grouping columns after groupby to silence this
warning.
```

```
testing = testing.groupby('Class Index',
group_keys=False).apply(lambda x: x.sample(n=1000, random_state=42))
```

```
training['text'] = training['Title'] + ' ' + training['Description']
testing['text'] = testing['Title'] + ' ' + testing['Description']
```

Cleaning dataset

```
import re
import spacy
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer

import nltk
nltk.download('punkt_tab')
```

```

nltk.download('stopwords')
nltk.download('wordnet')

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

True

stemmer = PorterStemmer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    text = text.lower()

    text = re.sub(r'http\S+|www.\S+', '', text)
    text = re.sub(r'<.*?>', '', text)
    text = re.sub(r'^a-zA-Z\s', '', text)

    tokens = word_tokenize(text)

    tokens = [word for word in tokens if word not in stop_words]

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

    clean_text = ' '.join(stemmed_tokens)

    return clean_text

training["clean_text"] = training["text"].apply(preprocess_text)
training = training.dropna()

testing["clean_text"] = testing["text"].apply(preprocess_text)
testing = testing.dropna()

```

Determining X_train, y_train, X_test, y_test

```

X_train= training['clean_text'].values
y_train = training['Class Index'].values

```

```
X_test= testing['clean_text'].values
y_test = testing['Class Index'].values
```

Feature representation

TF-IDF with Unigrams, Bigrams, and Trigrams

Three types of n-grams

- Unigrams: "machine", "learning", "fun"
- Bigrams: "machine learning", "deep learning"
- Trigrams: "machine learning is", "deep learning is"
- ngram_range=(min_n, max_n)
 - Unigrams(1,1)
 - Bigrams(2,2)
 - Unigrams + Bigrams (1,2)
 - Unigrams + Bigrams + Trigrams(1,3)

Unigram

```
vectorizer = TfidfVectorizer(ngram_range=(1, 1))

X_train_Unigram = vectorizer.fit_transform(X_train)
X_test_Unigram = vectorizer.transform(X_test)
```

Evaluation models

```
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=np.unique(y_true),
                yticklabels=np.unique(y_true))
    plt.title(f'Confusion Matrix for {model_name}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()

    return metrics

```

Random Forest Classifier

```

# Define the Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Define the hyperparameters for grid search
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier,
                           param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train_Unigram, y_train)

# Get the best estimator from grid search
best_rf_classifier = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_rf_classifier.predict(X_test_Unigram)

# Evaluate the model
evaluation_results = evaluate_model('RandomForestClassifier', 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

```

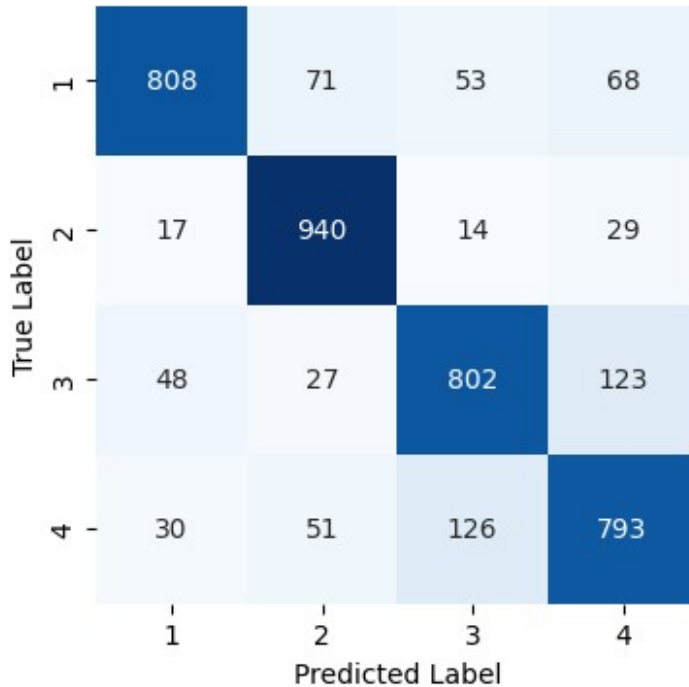
```

else:
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

# Print the best parameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

Confusion Matrix for RandomForestClassifier



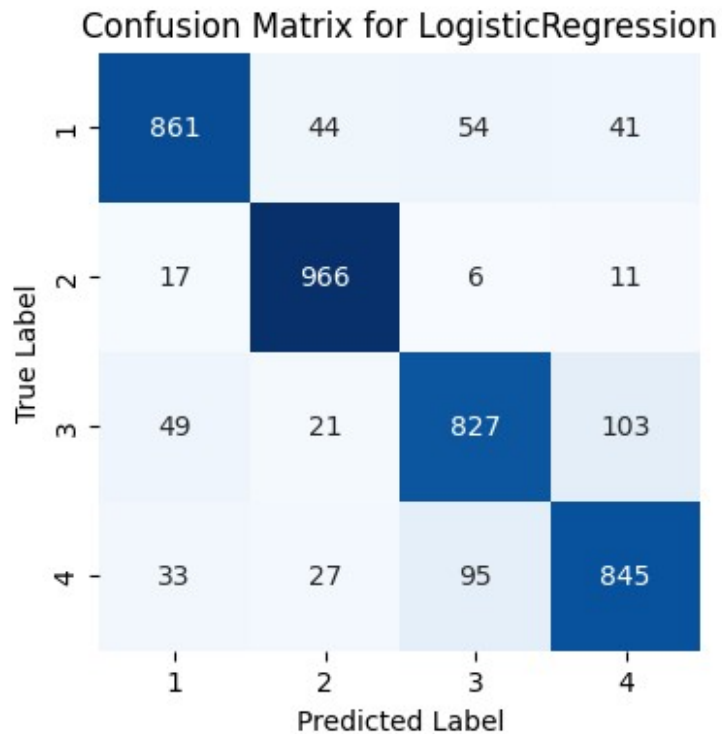
Model Name:
RandomForestClassifier
Accuracy: 0.8357
Precision: 0.8367
Recall: 0.8357
F1 Score: 0.8353

	precision	recall	f1-score	support
1	0.89	0.81	0.85	1000
2	0.86	0.94	0.90	1000
3	0.81	0.80	0.80	1000
4	0.78	0.79	0.79	1000
accuracy			0.84	4000
macro avg	0.84	0.84	0.84	4000
weighted avg	0.84	0.84	0.84	4000

```
Best hyperparameters found by GridSearchCV:  
{'max_depth': None, 'n_estimators': 300}
```

LogisticRegression

```
# 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_Unigram, 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_Unigram)  
  
# 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: 0.8748
Precision: 0.8743
Recall: 0.8748
F1 Score: 0.8742

	precision	recall	f1-score	support
1	0.90	0.86	0.88	1000
2	0.91	0.97	0.94	1000
3	0.84	0.83	0.83	1000
4	0.84	0.84	0.84	1000
accuracy			0.87	4000
macro avg	0.87	0.87	0.87	4000
weighted avg	0.87	0.87	0.87	4000

Best hyperparameters found by GridSearchCV:
{'solver': 'lbfgs'}

Unigrams + Bigrams

```
vectorizer = TfidfVectorizer(ngram_range=(1, 2))

X_train_U_B_nigram = vectorizer.fit_transform(X_train)
X_test_U_B_nigram = vectorizer.transform(X_test)
```

DecisionTreeClassifier

```
# Define the Decision Tree model
DT = DecisionTreeClassifier(random_state=42)

# Define the hyperparameters for grid search
param_grid = {
    'criterion': ['gini', 'entropy'],           # The function to
    'max_depth': [None, 10, 20, 30],           # The maximum depth
    'min_samples_split': [2, 5, 10],           # The minimum number of samples
    'min_samples_leaf': [1, 2, 5]              # The minimum number of samples
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=DT, param_grid=param_grid, cv=5,
scoring='accuracy', n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train_U_B_nigram , y_train)

# Get the best estimator from grid search
best_DT = grid_search.best_estimator_

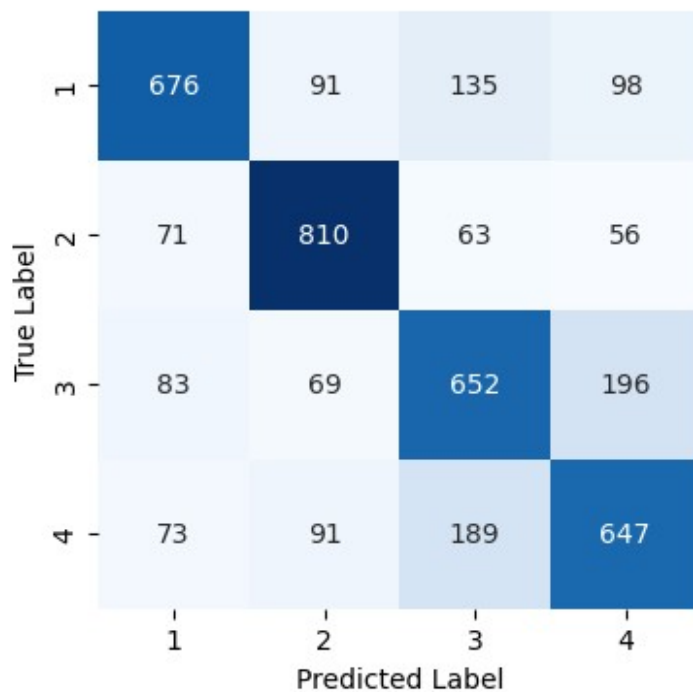
# Make predictions using the best model
y_pred = best_DT.predict(X_test_U_B_nigram )

# Evaluate the model
evaluation_results = evaluate_model('DecisionTreeClassifier', 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_)
```

Confusion Matrix for DecisionTreeClassifier



Model Name:
DecisionTreeClassifier
Accuracy: 0.6963
Precision: 0.6971
Recall: 0.6963
F1 Score: 0.6960

	precision	recall	f1-score	support
1	0.75	0.68	0.71	1000
2	0.76	0.81	0.79	1000
3	0.63	0.65	0.64	1000
4	0.65	0.65	0.65	1000
accuracy			0.70	4000
macro avg	0.70	0.70	0.70	4000
weighted avg	0.70	0.70	0.70	4000

Best hyperparameters found by GridSearchCV:
{'criterion': 'gini', 'max_depth': None}

Support Vector Machine

```
SVM = SVC(random_state=42)

# Define the hyperparameters for grid search
```

```

param_grid = {
    'kernel': ['linear', 'rbf', 'poly'],    # Kernel type
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=SVM, param_grid=param_grid, cv=5,
    scoring='accuracy', n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train_U_B_nigram, y_train)

# Get the best estimator from grid search
best_SVM = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_SVM.predict(X_test_U_B_nigram)

# Evaluate the model
evaluation_results = evaluate_model('Support Vector Machine', 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_)

```

Confusion Matrix for Support Vector Machine

True Label	1	856	46	54	44
	2	14	969	5	12
	3	40	18	833	109
	4	33	21	81	865
		1	2	3	4
		Predicted Label			

Model Name:

Support Vector Machine

Accuracy: 0.8808

Precision: 0.8808

Recall: 0.8808

F1 Score: 0.8803

	precision	recall	f1-score	support
1	0.91	0.86	0.88	1000
2	0.92	0.97	0.94	1000
3	0.86	0.83	0.84	1000
4	0.84	0.86	0.85	1000
accuracy			0.88	4000
macro avg	0.88	0.88	0.88	4000
weighted avg	0.88	0.88	0.88	4000

Best hyperparameters found by GridSearchCV:
{'kernel': 'rbf'}

To apply TF-IDF using Unigrams + Bigrams + Trigrams,

```
vectorizer = TfidfVectorizer(ngram_range=(1, 3))
```

```
X_train_U_B_T_nigram = vectorizer.fit_transform(X_train)
```

```
X_test_U_B_T_nigram = vectorizer.transform(X_test)
```

KNeighborsClassifier

```
# Define the K-Nearest Neighbors model
knn = KNeighborsClassifier()

# Define the hyperparameters for grid search
param_grid = {
    'n_neighbors': [1, 3, 5, 7, 9],          # Number of neighbors
    'weights': ['uniform', 'distance'],      # Weight function used
    'metric': ['euclidean', 'manhattan'],    # Distance metric
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'] #
    Algorithm to compute the nearest neighbors
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5,
scoring='accuracy', n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train_U_B_T_nigram, y_train)

# Get the best estimator from grid search
best_knn = grid_search.best_estimator_

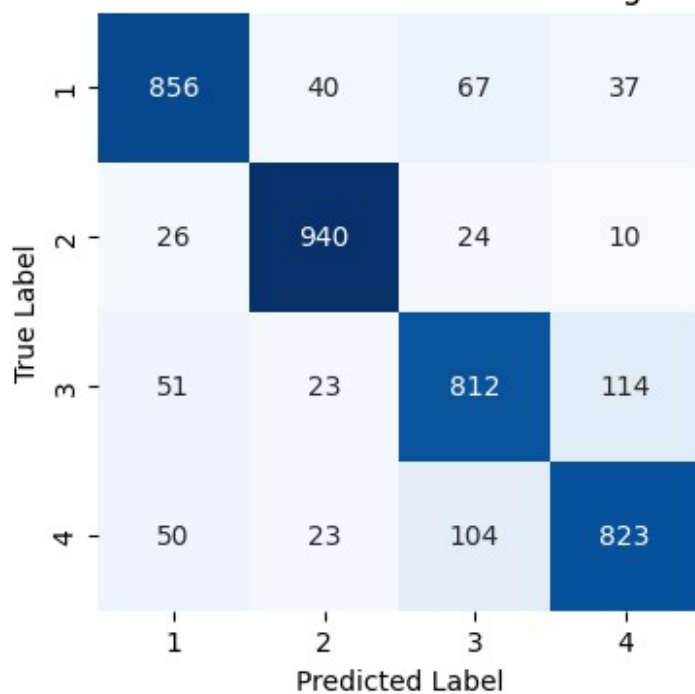
# Make predictions using the best model
y_pred = best_knn.predict(X_test_U_B_T_nigram)

# Evaluate the model
evaluation_results = evaluate_model('K-Nearest Neighbors', 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_)
```

Confusion Matrix for K-Nearest Neighbors



Model Name:

K-Nearest Neighbors

Accuracy: 0.8578

Precision: 0.8574

Recall: 0.8578

F1 Score: 0.8575

	precision	recall	f1-score	support
1	0.87	0.86	0.86	1000
2	0.92	0.94	0.93	1000
3	0.81	0.81	0.81	1000
4	0.84	0.82	0.83	1000
accuracy			0.86	4000
macro avg	0.86	0.86	0.86	4000
weighted avg	0.86	0.86	0.86	4000

Best hyperparameters found by GridSearchCV:

```
{'algorithm': 'auto', 'metric': 'euclidean', 'n_neighbors': 7, 'weights': 'distance'}
```

Other application using json format

Download dataset:

News Category Dataset

<https://www.kaggle.com/code/stellamer/starter-news-category-dataset-58e2a5f9-2> This dataset contains around 210k news headlines from 2012 to 2022 from HuffPost. This is one of the biggest news datasets and can serve as a benchmark for a variety of computational linguistic tasks

Reading dataset

```
df=pd.read_json('News_Category_Dataset_v3.json',lines=True)
df.head()
```

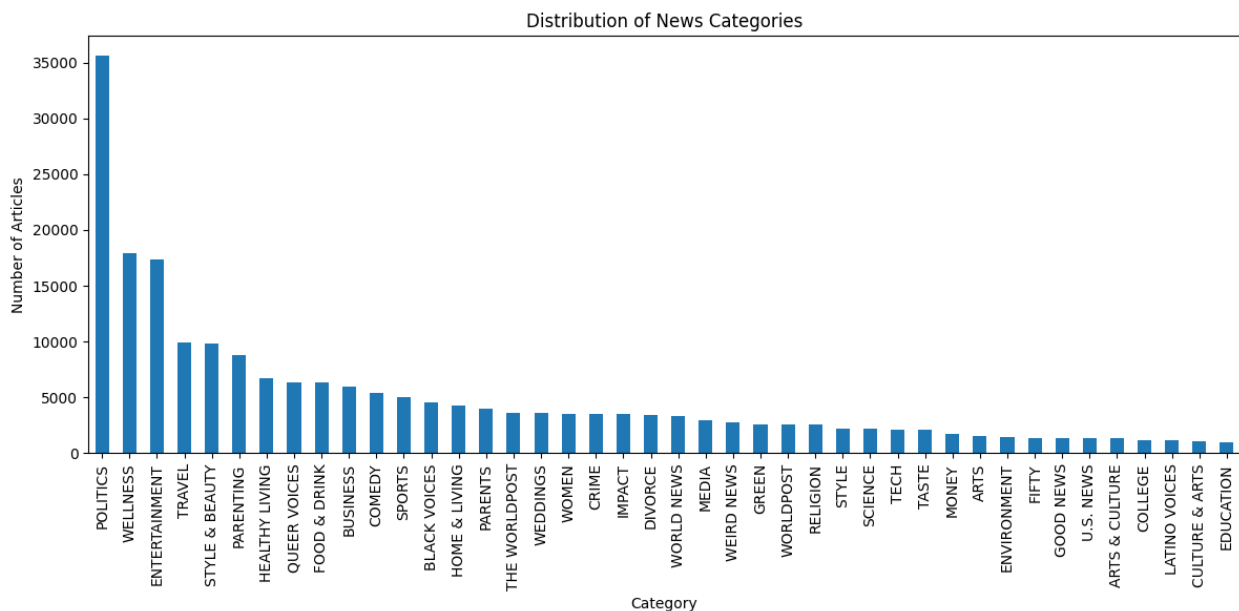
```
link \
0 https://www.huffpost.com/entry/covid-boosters-...
1 https://www.huffpost.com/entry/american-airlin...
2 https://www.huffpost.com/entry/funniest-tweets...
3 https://www.huffpost.com/entry/funniest-parent...
4 https://www.huffpost.com/entry/amy-cooper-lose...
```

```
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 Gologowski
```

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

```
import matplotlib.pyplot as plt
# Count unique categories
category_counts = df['category'].value_counts()
# Plot the category distribution
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()
```



Filtered dataset

```
# Define the categories to keep
categories_to_keep = ['POLITICS', 'WELLNESS', 'ENTERTAINMENT',
                      'TRAVEL', 'STYLE & BEAUTY', 'PARENTING']
```

```
# Filter the DataFrame
```

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

	link \
3	https://www.huffpost.com/entry/funniest-parent...
20	https://www.huffpost.com/entry/golden-globes-r...
21	https://www.huffpost.com/entry/biden-us-forces...
24	https://www.huffpost.com/entry/ukraine-festiva...
28	https://www.huffpost.com/entry/james-cameron-f...

	headline
category \	

3	The Funniest Tweets From Parents This Week (Se...	PARENTING
20	Golden Globes Returning To NBC In January Afte...	ENTERTAINMENT
21	Biden Says U.S. Forces Would Defend Taiwan If ...	POLITICS
24	'Beautiful And Sad At The Same Time': Ukrainia...	POLITICS
28	James Cameron Says He 'Clashed' With Studio Be...	ENTERTAINMENT

	short_description	authors \
3	"Accidentally put grown-up toothpaste on my to...	Caroline Bologna
20	For the past 18 months, Hollywood has effectiv...	
21	President issues vow as tensions with China rise.	
24	An annual celebration took on a different feel...	Jonathan Nicholson
28	The "Avatar" director said aspects of his 2009...	Ben Blanchet

	date
3	2022-09-23
20	2022-09-20
21	2022-09-19
24	2022-09-19
28	2022-09-18

```
len(filtered_df)
```

```
99414
```

```
filtered_df = filtered_df.groupby('category',
group_keys=False).apply(lambda x: x.sample(n=1000, random_state=42))
filtered_df = filtered_df.reset_index(drop=True)
```

```
C:\Users\hager\AppData\Local\Temp\ipykernel_29940\1799294979.py:1:
DeprecationWarning: DataFrameGroupBy.apply operated on the grouping
columns. This behavior is deprecated, and in a future version of
pandas the grouping columns will be excluded from the operation.
Either pass `include_groups=False` to exclude the groupings or
explicitly select the grouping columns after groupby to silence this
warning.
```

```
filtered_df = filtered_df.groupby('category',
group_keys=False).apply(lambda x: x.sample(n=1000, random_state=42))
```

```
filtered_df['text'] = filtered_df['headline'] + ' ' +
filtered_df['short_description']
```

Cleaning dataset

```
filtered_df["clean_text"] = filtered_df["text"].apply(preprocess_text)
filtered_df = filtered_df.dropna()
```

Define features and target variable

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

Bag of Words

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(X)
```

Splitting data

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.30, random_state=42, stratify=y)
```

Random Forest Classifier

```
# Define the Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Define the hyperparameters for grid search
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier,
                           param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best estimator from grid search
best_rf_classifier = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_rf_classifier.predict(X_test)

# Evaluate the model
evaluation_results = evaluate_model('RandomForestClassifier', 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 parameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

Confusion Matrix for RandomForestClassifier

True Label	ENTERTAINMENT	235	16	24	12	7	6
	PARENTING	27	225	9	5	6	28
	POLITICS	33	7	239	2	8	11
	STYLE & BEAUTY	40	8	6	228	2	16
	TRAVEL	38	16	4	14	215	13
	WELLNESS	37	13	9	9	23	209
		ENTERTAINMENT	PARENTING	POLITICS	STYLE & BEAUTY	TRAVEL	WELLNESS
		Predicted Label					

Model Name:
 RandomForestClassifier
 Accuracy: 0.7506
 Precision: 0.7651
 Recall: 0.7506
 F1 Score: 0.7539

	precision	recall	f1-score	support
ENTERTAINMENT	0.57	0.78	0.66	300
PARENTING	0.79	0.75	0.77	300
POLITICS	0.82	0.80	0.81	300
STYLE & BEAUTY	0.84	0.76	0.80	300
TRAVEL	0.82	0.72	0.77	300
WELLNESS	0.74	0.70	0.72	300
accuracy			0.75	1800
macro avg	0.77	0.75	0.75	1800
weighted avg	0.77	0.75	0.75	1800

Best hyperparameters found by GridSearchCV:
{'max_depth': None, 'n_estimators': 100}

LogisticRegression

```
# 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, 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)

# Evaluate the model
evaluation_results = evaluate_model('LogisticRegression', y_test,
y_pred)

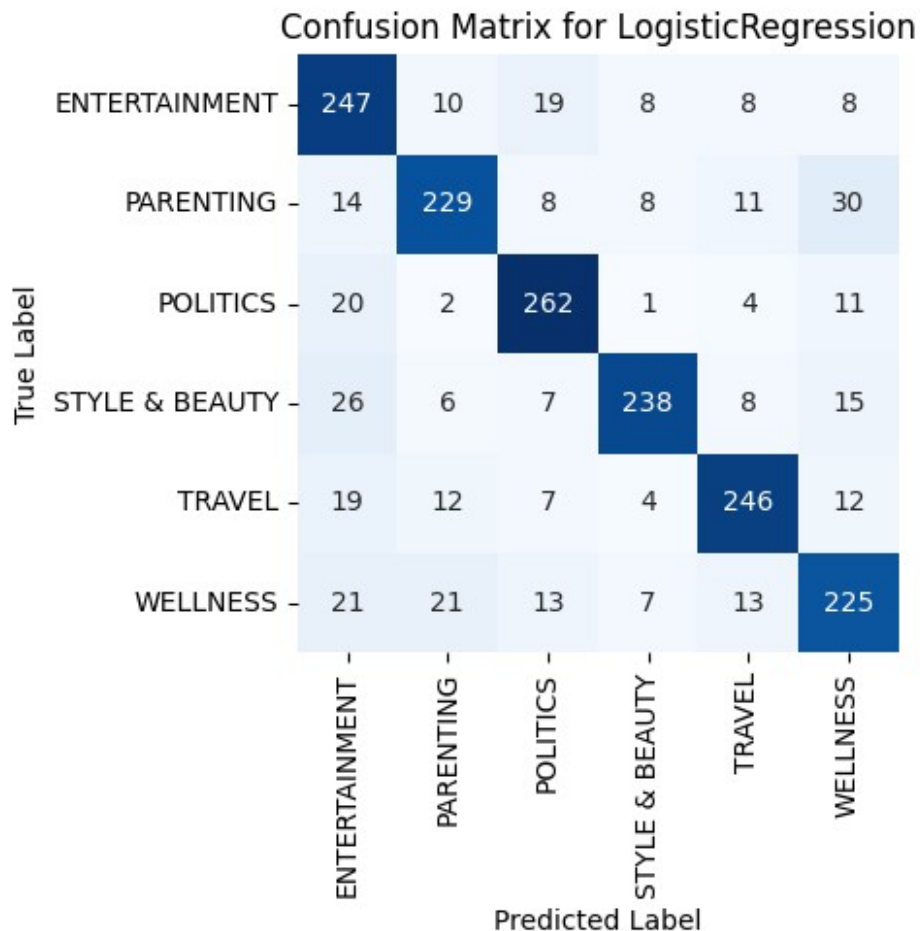
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
```

```

        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: 0.8039
 Precision: 0.8082
 Recall: 0.8039
 F1 Score: 0.8046

	precision	recall	f1-score	support
ENTERTAINMENT	0.71	0.82	0.76	300
PARENTING	0.82	0.76	0.79	300
POLITICS	0.83	0.87	0.85	300

STYLE & BEAUTY	0.89	0.79	0.84	300
TRAVEL	0.85	0.82	0.83	300
WELLNESS	0.75	0.75	0.75	300
accuracy			0.80	1800
macro avg	0.81	0.80	0.80	1800
weighted avg	0.81	0.80	0.80	1800

Best hyperparameters found by GridSearchCV:
{'solver': 'liblinear'}