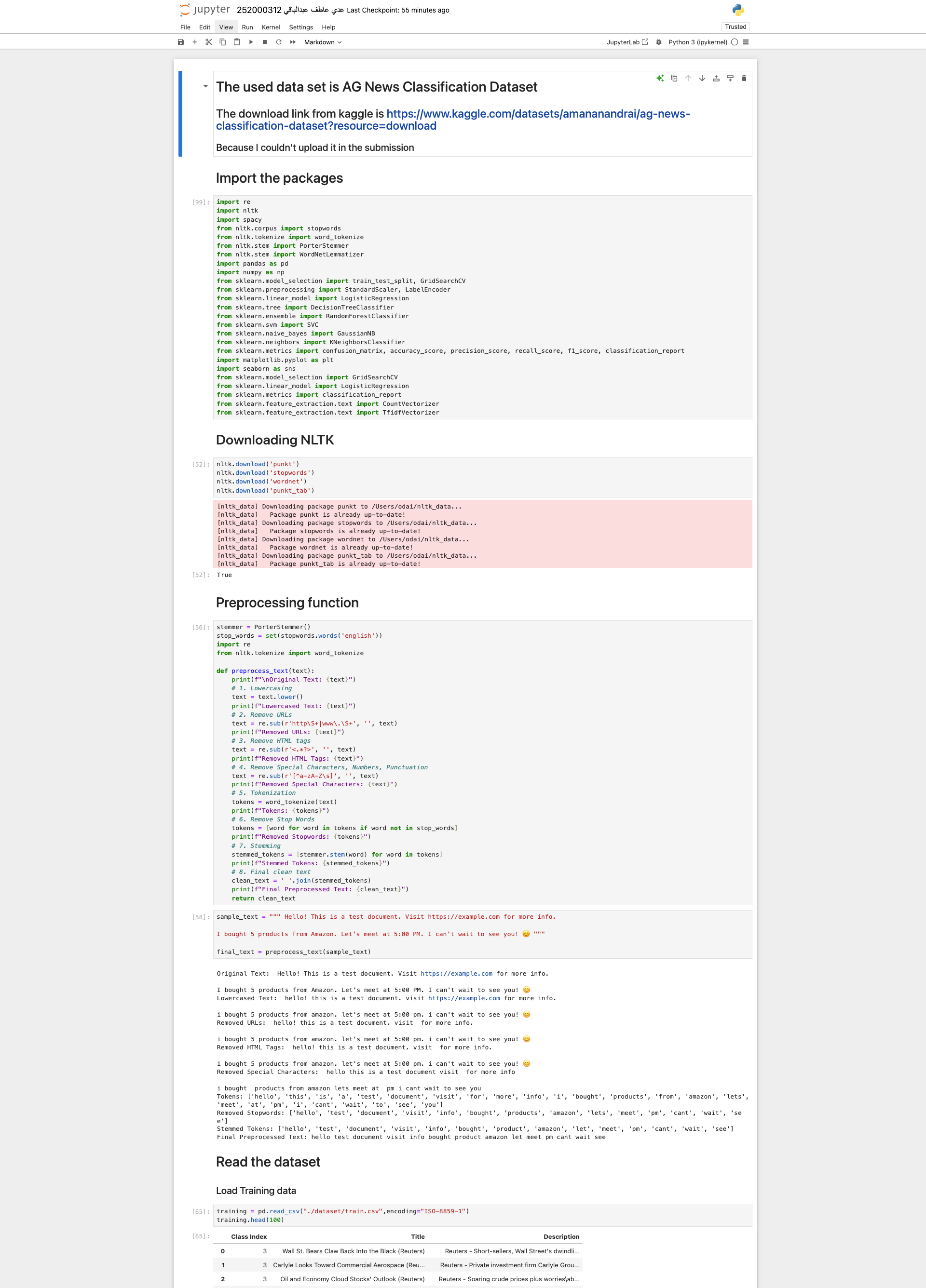


The used data set is AG News Classification Dataset

The download link from kaggle is https://www.kaggle.com/datasets/amananandrai/ag-news-classification-dataset?resource=download

Because I couldn't upload it in the submission

Import the packages

[99]:

```python
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
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, recall_score, f1_score, classification_report
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 CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
```

Downloading NLTK

[52]:

```python
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt to /Users/odai/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /Users/odai/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /Users/odai/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt_tab to /Users/odai/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

[52]:

```
True
```

Preprocessing function

[56]:

```python
stemmer = PorterStemmer()
stop_words = set(stopwords.words('english'))
import re
from nltk.tokenize import word_tokenize

def preprocess_text(text):
    print(f"\nOriginal Text: {text}")
    # 1. Lowercasing
    text = text.lower()
    print(f"Lowercased Text: {text}")
    # 2. Remove URLs
    text = re.sub(r'http\S+|www\.\S+', '', text)
    print(f"Removed URLs: {text}")
    # 3. Remove HTML tags
    text = re.sub(r'<.*?>', '', text)
    print(f"Removed HTML Tags: {text}")
    # 4. Remove Special Characters, Numbers, Punctuation
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    print(f"Removed Special Characters: {text}")
    # 5. Tokenization
    tokens = word_tokenize(text)
    print(f"Tokens: {tokens}")
    # 6. Remove Stop Words
    tokens = [word for word in tokens if word not in stop_words]
    print(f"Removed Stopwords: {tokens}")
    # 7. Stemming
    stemmed_tokens = [stemmer.stem(word) for word in tokens]
    print(f"Stemmed Tokens: {stemmed_tokens}")
    # 8. Final clean text
    clean_text = ' '.join(stemmed_tokens)
    print(f"Final Preprocessed Text: {clean_text}")
    return clean_text
```

[58]:

```python
sample_text = """ Hello! This is a test document. Visit https://example.com for more info.

I bought 5 products from Amazon. Let's meet at 5:00 PM. I can't wait to see you! 😊 """

final_text = preprocess_text(sample_text)
```

```

Original Text:  Hello! This is a test document. Visit https://example.com for more info.

I bought 5 products from Amazon. Let's meet at 5:00 PM. I can't wait to see you! 😊
Lowercased Text:  hello! this is a test document. visit https://example.com for more info.

i bought 5 products from amazon. let's meet at 5:00 pm. i can't wait to see you! 😊
Removed URLs:  hello! this is a test document. visit  for more info.

i bought 5 products from amazon. let's meet at 5:00 pm. i can't wait to see you! 😊
Removed HTML Tags:  hello! this is a test document. visit  for more info.

i bought 5 products from amazon. let's meet at 5:00 pm. i can't wait to see you! 😊
Removed Special Characters:  hello this is a test document visit  for more info

i bought  products from amazon lets meet at  pm i cant wait to see you
Tokens: ['hello', 'this', 'is', 'a', 'test', 'document', 'visit', 'for', 'more', 'info', 'i', 'bought', 'products', 'from', 'amazon', 'lets', 
'meet', 'at', 'pm', 'i', 'cant', 'wait', 'to', 'see', 'you']
Removed Stopwords: ['hello', 'test', 'document', 'visit', 'info', 'bought', 'products', 'amazon', 'lets', 'meet', 'pm', 'cant', 'wait', 'se
e']
Stemmed Tokens: ['hello', 'test', 'document', 'visit', 'info', 'bought', 'product', 'amazon', 'let', 'meet', 'pm', 'cant', 'wait', 'see']
Final Preprocessed Text: hello test document visit info bought product amazon let meet pm cant wait see
```

Read the dataset

Load Training data

[65]:

```python
training = pd.read_csv("./dataset/train.csv",encoding="ISO-8859-1")
training.head(100)
```

[65]:

	Class Index	Title	Description
0	3	Wall St. Bears Claw Back Into the Black (Reuters)	Reuters - Short-sellers, Wall Street's dwindli...
1	3	Carlyle Looks Toward Commercial Aerospace (Reu...	Reuters - Private investment firm Carlyle Grou...
2	3	Oil and Economy Cloud Stocks' Outlook (Reuters)	Reuters - Soaring crude prices plus worries\ab...

3	3	Iraq Halts Oil Exports from Main Southern Pipe...	Reuters - Authorities have halted oil export\f...
4	3	Oil prices soar to all-time record, posing new...	AFP - Tearaway world oil prices, toppling reco...
...	...	...	...
95	4	Gene Blocker Turns Monkeys Into Workaholics - ...	Reuters - Procrastinating monkeys were turned\...
96	4	Dolphins Too Have Born Socialites (Reuters)	Reuters - Some people are born to be the life ...
97	4	What's in a Name? Well, Matt Is Sexier Than Pa...	Reuters - As Shakespeare said, a rose by any o...
98	4	UK Scientists Allowed to Clone Human Embryos (...	Reuters - British scientists said on Wednesday...
99	4	Russian Alien Spaceship Claims Raise Eyebrows,...	SPACE.com - An expedition of Russian researche...

100 rows × 3 columns

[69]:

```
len(training)
```

[69]:

```
120000
```

[71]:

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

[71]:

```
Class Index
3    30000
4    30000
2    30000
1    30000
Name: count, dtype: int64
```

[73]:

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

Load Testing Data (Data was splited by Kaggle Trainig & Testing)

[81]:

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

[81]:

	Class Index	Title	Description
0	3	Fears for T N pension after talks	Unions representing workers at Turner Newall...
1	4	The Race is On: Second Private Team Sets Launc...	SPACE.com - TORONTO, Canada -- A second\team o...
2	4	Ky. Company Wins Grant to Study Peptides (AP)	AP - A company founded by a chemistry research...
3	4	Prediction Unit Helps Forecast Wildfires (AP)	AP - It's barely dawn when Mike Fitzpatrick st...
4	4	Calif. Aims to Limit Farm-Related Smog (AP)	AP - Southern California's smog-fighting agenc...

[83]:

```
len(testing)
```

[83]:

```
7600
```

[85]:

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

[87]:

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

Clean the Training & Testing data

[90]:

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

```
Original Text: Explosion Rocks Baghdad Neighborhood BAGHDAD, Iraq, August 24 -- A car bomb exploded near the gate of a US-funded Iraqi tele
vision network in Baghdad on Tuesday, killing at least two people and wounding two others, authorities and witnesses said.
Lowercased Text: explosion rocks baghdad neighborhood baghdad, iraq, august 24 -- a car bomb exploded near the gate of a us-funded iraqi te
levision network in baghdad on tuesday, killing at least two people and wounding two others, authorities and witnesses said.
Removed URLs: explosion rocks baghdad neighborhood baghdad, iraq, august 24 -- a car bomb exploded near the gate of a us-funded iraqi telev
ision network in baghdad on tuesday, killing at least two people and wounding two others, authorities and witnesses said.
Removed HTML Tags: explosion rocks baghdad neighborhood baghdad, iraq, august 24 -- a car bomb exploded near the gate of a us-funded iraqi
television network in baghdad on tuesday, killing at least two people and wounding two others, authorities and witnesses said.
Removed Special Characters: explosion rocks baghdad neighborhood baghdad iraq august   a car bomb exploded near the gate of a usfunded iraq
i television network in baghdad on tuesday killing at least two people and wounding two others authorities and witnesses said
Tokens: ['explosion', 'rocks', 'baghdad', 'neighborhood', 'baghdad', 'iraq', 'august', 'a', 'car', 'bomb', 'exploded', 'near', 'the', 'gat
e', 'of', 'a', 'usfunded', 'iraqi', 'television', 'network', 'in', 'baghdad', 'on', 'tuesday', 'killing', 'at', 'least', 'two', 'people',
'and', 'wounding', 'two', 'others', 'authorities', 'and', 'witnesses', 'said']
Removed Stopwords: ['explosion', 'rocks', 'baghdad', 'neighborhood', 'baghdad', 'iraq', 'august', 'car', 'bomb', 'exploded', 'near', 'gat
e', 'usfunded', 'iraqi', 'television', 'network', 'baghdad', 'tuesday', 'killing', 'least', 'two', 'people', 'wounding', 'two', 'others',
'authorities', 'witnesses', 'said']
Stemmed Tokens: ['explos', 'rock', 'baghdad', 'neighborhood', 'baghdad', 'iraq', 'august', 'car', 'bomb', 'explod', 'near', 'gate', 'usfun
```

[92]:

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

[94]:

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

Feature representaion

[126]:

```
vectorizer = TfidfVectorizer(ngram_range=(1, 1))
X_train_Unigram = vectorizer.fit_transform(X_train)
X_test_Unigram = vectorizer.transform(X_test)
vectorizer_b = TfidfVectorizer(ngram_range=(1, 2))
X_train_U_B_nigram = vectorizer_b.fit_transform(X_train)
X_test_U_B_nigram = vectorizer_b.transform(X_test)
vectorizer_t = TfidfVectorizer(ngram_range=(1, 3))
X_train_U_B_T_nigram = vectorizer_t.fit_transform(X_train)
X_test_U_B_T_nigram = vectorizer_t.transform(X_test)
```

Evauate the models

[104]:

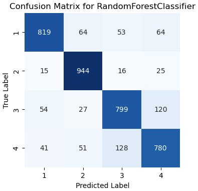
```
def evaluate_model(model_name, y_true, y_pred):
    # Calculate metrics
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    cm = confusion_matrix(y_true, y_pred)

    # Create classification 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
```

Train and use models

Random Forest Classifier

[109]:

```python
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
best_rf_classifier = grid_search.best_estimator_

# Make predictions
y_pred = best_rf_classifier.predict(X_test_Unigram)

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

# Print evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: {value}")

# Print best hyperparameters
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```

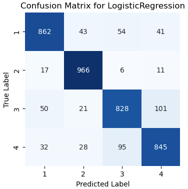


```
Model Name: RandomForestClassifier
Accuracy: 0.8355
Precision: 0.8354
Recall: 0.8355
F1 Score: 0.8348
              precision    recall  f1-score   support

           1       0.88      0.82      0.85      1000
           2       0.87      0.94      0.91      1000
           3       0.80      0.80      0.80      1000
           4       0.79      0.78      0.78      1000

    accuracy                           0.84      4000
   macro avg       0.84      0.84      0.83      4000
weighted avg       0.84      0.84      0.83      4000


Best hyperparameters found by GridSearchCV:
{'max_depth': None, 'n_estimators': 200}
```

LogisticRegression

[112]:

```python
# Define the Logistic Regression model
LR = LogisticRegression(random_state=42, max_iter=1000)

# Define the hyperparameters for grid search
param_grid = {
    'solver': ['liblinear', 'lbfgs'],
}

# 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
best_LR = grid_search.best_estimator_

# Make predictions
y_pred = best_LR.predict(X_test_Unigram)

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

# Print evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: {value}")

# Print best hyperparameters
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```

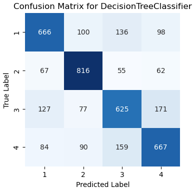


```
Model Name: LogisticRegression
Accuracy: 0.8752
Precision: 0.8748
Recall: 0.8752
F1 Score: 0.8747
              precision    recall  f1-score   support

           1       0.90      0.86      0.88      1000
           2       0.91      0.97      0.94      1000
           3       0.84      0.83      0.84      1000
           4       0.85      0.84      0.85      1000

    accuracy                           0.88      4000
   macro avg       0.87      0.88      0.87      4000
weighted avg       0.87      0.88      0.87      4000


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

DecisionTreeClassifier

[117]:

```python
DT = DecisionTreeClassifier(random_state=42)

# Define the hyperparameters for grid search
param_grid = {
    'criterion': ['gini', 'entropy'],    # measure the quality of a split
    'max_depth': [None, 10, 20, 30],    # maximum depth of the tree
}

# 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
best_DT = grid_search.best_estimator_

# Make predictions
y_pred = best_DT.predict(X_test_U_B_nigram)

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

# Print evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: {value}")

# Print best hyperparameters
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```



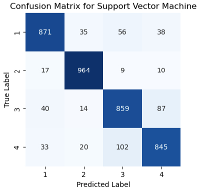
```
Model Name: DecisionTreeClassifier
Accuracy: 0.6935
Precision: 0.6921
Recall: 0.6935
F1 Score: 0.6923
              precision    recall  f1-score   support

           1       0.71      0.67      0.69      1000
           2       0.75      0.82      0.78      1000
           3       0.64      0.62      0.63      1000
           4       0.67      0.67      0.67      1000

    accuracy                           0.69      4000
   macro avg       0.69      0.69      0.69      4000
weighted avg       0.69      0.69      0.69      4000


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

Support Vector Machine

[122]:

```python
SVM = SVC(random_state=42)
# Define the SVM model
SVM = SVC()

# 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
best_SVM = grid_search.best_estimator_

# Make predictions
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 evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: {value}")

# Print best hyperparameters
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```

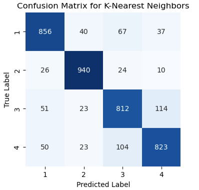

Confusion Matrix for Support Vector Machine

```
Model Name: Support Vector Machine
Accuracy: 0.8848
Precision: 0.8848
Recall: 0.8848
F1 Score: 0.8845
              precision    recall  f1-score   support

           1       0.91      0.87      0.89      1000
           2       0.93      0.96      0.95      1000
           3       0.84      0.86      0.85      1000
           4       0.86      0.84      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': 'linear'}
```

KNeighborsClassifier

[129]:

```
# Define KNN model
knn = KNeighborsClassifier()

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

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

# Fit the model
grid_search.fit(X_train_U_B_T_nigram, y_train)

# Best model
best_knn = grid_search.best_estimator_

# Predictions
y_pred = best_knn.predict(X_test_U_B_T_nigram)

# Evaluation
evaluation_results = evaluate_model('K-Nearest Neighbors', y_test, y_pred)

# Print results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: {value}")

# Best parameters
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'}
```

[ ]:

[ ]:

[ ]:

[ ]: