

```
In [1]: # Step 1: Connect Google Drive to Google Colab

from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

In [2]: # Step 2: Create a Project Folder in Google Drive

import os

project_path = "/content/drive/MyDrive/Arabic_English_Sentiment_Analysis"

os.makedirs(project_path, exist_ok=True)

print("Project folder created successfully:")
print(project_path)

Project folder created successfully:
/content/drive/MyDrive/Arabic_English_Sentiment_Analysis

In [3]: # Step 3: Create Folders for Datasets, Models, and Results

datasets_path = os.path.join(project_path, "Datasets")
models_path = os.path.join(project_path, "Models")
results_path = os.path.join(project_path, "Results")

os.makedirs(datasets_path, exist_ok=True)
os.makedirs(models_path, exist_ok=True)
os.makedirs(results_path, exist_ok=True)

print("Project folders created successfully:")
print("Datasets:", datasets_path)
print("Models:", models_path)
print("Results:", results_path)

Project folders created successfully:
Datasets: /content/drive/MyDrive/Arabic_English_Sentiment_Analysis/Datasets
Models: /content/drive/MyDrive/Arabic_English_Sentiment_Analysis/Models
Results: /content/drive/MyDrive/Arabic_English_Sentiment_Analysis/Results

In [5]: # Step 4: Check the Dataset Files in Google Drive

import os

files = os.listdir(datasets_path)

print("Dataset files:")
for file in files:
    print("-", file)

Dataset files:
- Twitter Data Set for Arabic Sentiment Analysis.zip
- Sentiment140 dataset with 1.6 million tweets.zip

In [6]: # Step 5: Extract the Arabic Sentiment Dataset

import zipfile
import os

arabic_zip = os.path.join(
    datasets_path,
    "Twitter Data Set for Arabic Sentiment Analysis.zip"
)

arabic_extract_path = os.path.join(
    datasets_path,
    "Arabic_Sentiment_Dataset"
)

os.makedirs(arabic_extract_path, exist_ok=True)

with zipfile.ZipFile(arabic_zip, 'r') as zip_ref:
    zip_ref.extractall(arabic_extract_path)

print("Arabic dataset extracted successfully.")
print("Extracted to:", arabic_extract_path)

print("\nExtracted files and folders:")
for item in os.listdir(arabic_extract_path):
    print("-", item)

Arabic dataset extracted successfully.
Extracted to: /content/drive/MyDrive/Arabic_English_Sentiment_Analysis/Datasets/Arabic_Sentiment_Dataset

Extracted files and folders:
- Twitter

In [7]: # Step 6: Check the Arabic Dataset Structure and Count Positive and Negative Samples

twitter_path = os.path.join(arabic_extract_path, "Twitter")

positive_path = os.path.join(twitter_path, "Positive")
negative_path = os.path.join(twitter_path, "Negative")

positive_files = os.listdir(positive_path)
negative_files = os.listdir(negative_path)

print("Arabic Dataset Summary:")
print("Positive samples:", len(positive_files))
print("Negative samples:", len(negative_files))
print("Total samples:", len(positive_files) + len(negative_files))

Arabic Dataset Summary:
Positive samples: 1000
Negative samples: 1000
Total samples: 2000
```

In [9]: # Step 7: Convert Arabic Text Files into a Pandas DataFrame with Encoding Handling

```
import pandas as pd
import os

arabic_data = []

# Function to read Arabic text files with different encodings
def read_arabic_file(file_path):
    encodings = ["utf-8", "windows-1256", "iso-8859-6"]

    for encoding in encodings:
        try:
            with open(file_path, "r", encoding=encoding) as file:
                return file.read().strip()
        except UnicodeDecodeError:
            continue

    return None

# Read Positive samples
for file_name in positive_files:
    file_path = os.path.join(positive_path, file_name)
    text = read_arabic_file(file_path)

    if text is not None:
        arabic_data.append([text, "positive"])

# Read Negative samples
for file_name in negative_files:
    file_path = os.path.join(negative_path, file_name)
    text = read_arabic_file(file_path)

    if text is not None:
        arabic_data.append([text, "negative"])

# Create DataFrame
arabic_df = pd.DataFrame(
    arabic_data,
    columns=["text", "sentiment"]
)

print("Arabic DataFrame created successfully.")
print("Shape:", arabic_df.shape)

print("\nSentiment distribution:")
print(arabic_df["sentiment"].value_counts())

display(arabic_df.head())
```

Arabic DataFrame created successfully.
Shape: (2000, 2)

Sentiment distribution:

sentiment	
positive	1000
negative	1000

Name: count, dtype: int64

	text	sentiment
0	حقاً	positive
1	صبح جداً	positive
2	أه و النعمة	positive
3	كلامك جميل ورائع	positive
4	و خير الكلام ما قل و دل	positive

In [11]: # Step 8: Check the Arabic Dataset for Missing Values, Duplicates, and Empty Text

```
print("Arabic Dataset Information:")
print("Shape:", arabic_df.shape)

print("\nMissing values:")
print(arabic_df.isnull().sum())

print("\nDuplicate rows:")
print(arabic_df.duplicated().sum())

print("\nEmpty text samples:")
print((arabic_df["text"].str.strip() == "").sum())

print("\nSentiment distribution:")
print(arabic_df["sentiment"].value_counts())
```

Arabic Dataset Information:
Shape: (2000, 2)

Missing values:

text	0
sentiment	0

dtype: int64

Duplicate rows:
24

Empty text samples:
4

Sentiment distribution:

sentiment	
positive	1000
negative	1000

Name: count, dtype: int64

In [12]: # Step 9: Remove Empty Texts and Duplicate Rows from the Arabic Dataset

```
# Remove empty text samples
arabic_df = arabic_df[arabic_df["text"].str.strip() != ""]

# Remove duplicate rows
arabic_df = arabic_df.drop_duplicates()

# Reset index
arabic_df = arabic_df.reset_index(drop=True)

print("Arabic dataset cleaned successfully.")

print("\nDataset shape after cleaning:")
print(arabic_df.shape)

print("\nRemaining duplicate rows:")
print(arabic_df.duplicated().sum())

print("\nRemaining empty text samples:")
print((arabic_df["text"].str.strip() == "").sum())

print("\nSentiment distribution after cleaning:")
print(arabic_df["sentiment"].value_counts())
```

Arabic dataset cleaned successfully.

Dataset shape after cleaning:
(1975, 2)

Remaining duplicate rows:
0

Remaining empty text samples:
0

Sentiment distribution after cleaning:

sentiment	
positive	997
negative	978

Name: count, dtype: int64

In [13]: # Step 10: Preprocess and Clean the Arabic Text

```
import re

def clean_arabic_text(text):

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

    # Remove mentions
    text = re.sub(r'@\w+', ' ', text)

    # Remove hashtags symbol but keep the hashtag word
    text = text.replace('#', ' ')

    # Remove Arabic diacritics
    arabic_diacritics = re.compile(r"""
-   | # Shadda
-   | # Fatha
-   | # Tanwin Fath
-   | # Damma
-   | # Tanwin Damm
-   | # Kasra
-   | # Tanwin Kasr
-   | # Sukun
-   # Tatweel
""", re.VERBOSE)

    text = re.sub(arabic_diacritics, '', text)

    # Normalize Arabic letters
    text = re.sub("[\u0600-\u06FF]", lambda x: x.lower(), text)
    text = re.sub("ى", "ي", text)
    text = re.sub("ة", "ت", text)
    text = re.sub("ي", "ى", text)

    # Keep Arabic letters and spaces only
    text = re.sub(r'[^ا-ز \u0600-\u06FF\s]', ' ', text)

    # Remove extra spaces
    text = re.sub(r'\s+', ' ', text).strip()

    return text

# Apply preprocessing
arabic_df["clean_text"] = arabic_df["text"].apply(clean_arabic_text)

print("Arabic text preprocessing completed successfully.")

# Display original text and cleaned text
display(arabic_df[["text", "clean_text", "sentiment"]].head(10))
```

Arabic text preprocessing completed successfully.

	text	clean_text	sentiment
0	حقا	حقا	positive
1	صبح جدا	صبح جدا	positive
2	اه و النعمة	اه و النعمة	positive
3	كلامك جميل ورائع	كلامك جميل ورائع	positive
4	و خير الكلام ما قل و دل	و خير الكلام ما قل و دل	positive
5	ولا تنسو تبسمك في وجه اخيك صدقه	ولا تنسو تبسمك في وجه اخيك صدقه	positive
6	لبيتنا نستطيع الابتسامه في الاوقات المؤلمه	لبيتنا نستطيع الابتسامه في الاوقات المؤلمه	positive
7	و يحب الناس الراقه الي بتضحك على طول	و يحب الناس الراقه الي بتضحك على طول	positive
8	...ابتسم امام كل الناس لتسرهم و تجبر خواطرحم فتنس	...ابتسم امام كل الناس لتسرهم و تجبر خواطرحم فتنس	positive
9	و الله انت حبيبي	و الله انت حبيبي	positive

In [14]: # Step 11: Remove Arabic Stop Words from the Cleaned Text

```
import nltk
from nltk.corpus import stopwords

# Download NLTK stopwords
nltk.download('stopwords')

# Load Arabic stop words
arabic_stopwords = set(stopwords.words('arabic'))

# Function to remove Arabic stop words
def remove_arabic_stopwords(text):
    words = text.split()
    filtered_words = [
        word for word in words
        if word not in arabic_stopwords
    ]
    return " ".join(filtered_words)

# Apply stop word removal
arabic_df["processed_text"] = arabic_df["clean_text"].apply(
    remove_arabic_stopwords
)

print("Arabic stop words removed successfully.")

# Display the results
display(
    arabic_df[
        ["text", "clean_text", "processed_text", "sentiment"]
    ].head(10)
)
```

Arabic stop words removed successfully.

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Unzipping corpora/stopwords.zip.

	text	clean_text	processed_text	sentiment
0	حقا	حقا		positive
1	صبح جدا	صبح جدا	صبح جدا	positive
2	اه و النعمة	اه و النعمة	اه و النعمة	positive
3	كلامك جميل ورائع	كلامك جميل ورائع	كلامك جميل ورائع	positive
4	و خير الكلام ما قل و دل	و خير الكلام ما قل و دل	خير الكلام قل دل	positive
5	ولا تنسو تبسمك في وجه اخيك صدقه	ولا تنسو تبسمك في وجه اخيك صدقه	تنسو تبسمك وجه اخيك صدقه	positive
6	لبيتنا نستطيع الابتسامه في الاوقات المؤلمه	لبيتنا نستطيع الابتسامه في الاوقات المؤلمه	لبيتنا نستطيع الابتسامه الاوقات المؤلمه	positive
7	و يحب الناس الراقه الي بتضحك على طول	و يحب الناس الراقه الي بتضحك على طول	يحب الناس الراقه الي بتضحك على طول	positive
8	...ابتسم امام كل الناس لتسرهم و تجبر خواطرحم فتنس	...ابتسم امام كل الناس لتسرهم و تجبر خواطرحم فتنس	...ابتسم امام الناس لتسرهم تجبر خواطرحم فتنسبك و ج	positive
9	و الله انت حبيبي	و الله انت حبيبي	الله انت حبيبي	positive

```
In [15]: # Step 12: Remove Empty Texts After Stop Word Removal

# Count empty processed texts
empty_processed = (arabic_df["processed_text"].str.strip() == "").sum()

print("Empty processed texts before removal:", empty_processed)

# Remove empty processed texts
arabic_df = arabic_df[
    arabic_df["processed_text"].str.strip() != ""
].reset_index(drop=True)

print("\nArabic dataset after preprocessing:")
print("Shape:", arabic_df.shape)

print("\nSentiment distribution:")
print(arabic_df["sentiment"].value_counts())

print("\nRemaining empty processed texts:")
print((arabic_df["processed_text"].str.strip() == "").sum())
```

Empty processed texts before removal: 7

Arabic dataset after preprocessing:
Shape: (1968, 4)

Sentiment distribution:
sentiment
positive 994
negative 974
Name: count, dtype: int64

Remaining empty processed texts:
0

```
In [16]: # Step 13: Convert Arabic Text into Numerical Features Using Bag of Words (BoW)

from sklearn.feature_extraction.text import CountVectorizer

# Create Bag of Words vectorizer
arabic_bow = CountVectorizer()

# Convert processed Arabic text into numerical features
X_arabic_bow = arabic_bow.fit_transform(
    arabic_df["processed_text"]
)

# Target labels
y_arabic = arabic_df["sentiment"]

print("Bag of Words created successfully.")

print("\nNumber of samples:")
print(X_arabic_bow.shape[0])

print("\nNumber of unique features (words):")
print(X_arabic_bow.shape[1])

print("\nBag of Words matrix shape:")
print(X_arabic_bow.shape)

print("\nFirst 20 features:")
print(arabic_bow.get_feature_names_out()[:20])
```

Bag of Words created successfully.

Number of samples:
1968

Number of unique features (words):
6543

Bag of Words matrix shape:
(1968, 6543)

First 20 features:
['ااااا' 'ااا' 'اااا' 'ااااا' 'اااااا' 'ااااااا' 'اااااااا' 'ااااااااا' 'اااااااااا' 'ااااااااااا' 'اااااااااااا' 'ااااااااااااا' 'اااااااااااااا' 'ااااااااااااااا' 'اااااااااااااااا' 'ااااااااااااااااا' 'اااااااااااااااااا' 'ااااااااااااااااااا' 'اااااااااااااااااااا' 'ااااااااااااااااااااا']

```
In [17]: # Step 14: Convert Arabic Text into Numerical Features Using TF-IDF

from sklearn.feature_extraction.text import TfidfVectorizer

# Create TF-IDF vectorizer
arabic_tfidf = TfidfVectorizer()

# Convert processed Arabic text into TF-IDF features
X_arabic_tfidf = arabic_tfidf.fit_transform(
    arabic_df["processed_text"]
)

print("TF-IDF features created successfully.")

print("\nNumber of samples:")
print(X_arabic_tfidf.shape[0])

print("\nNumber of unique features:")
print(X_arabic_tfidf.shape[1])

print("\nTF-IDF matrix shape:")
print(X_arabic_tfidf.shape)

print("\nFirst 20 features:")
print(arabic_tfidf.get_feature_names_out()[:20])
```

TF-IDF features created successfully.

Number of samples:
1968

Number of unique features:
6543

TF-IDF matrix shape:
(1968, 6543)

First 20 features:
['ااااا' 'ااا' 'اااا' 'ااااا' 'اااااا' 'ااااااا' 'اااااااا' 'ااااااااا' 'اااااااااا' 'ااااااااااا' 'اااااااااااا' 'ااااااااااااا' 'اااااااااااااا' 'ااااااااااااااا' 'اااااااااااااااا' 'ااااااااااااااااا' 'اااااااااااااااااا' 'ااااااااااااااااااا' 'اااااااااااااااااااا' 'ااااااااااااااااااااا']

In [18]: # Step 15: Create TF-IDF Features Using Unigrams, Bigrams, and Trigrams

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Unigram (1 word)
tfidf_unigram = TfidfVectorizer(ngram_range=(1, 1))
X_unigram = tfidf_unigram.fit_transform(arabic_df["processed_text"])

# Bigram (2 words)
tfidf_bigram = TfidfVectorizer(ngram_range=(2, 2))
X_bigram = tfidf_bigram.fit_transform(arabic_df["processed_text"])

# Trigram (3 words)
tfidf_trigram = TfidfVectorizer(ngram_range=(3, 3))
X_trigram = tfidf_trigram.fit_transform(arabic_df["processed_text"])

print("N-gram features created successfully.")

print("\nUnigram shape:")
print(X_unigram.shape)

print("\nBigram shape:")
print(X_bigram.shape)

print("\nTrigram shape:")
print(X_trigram.shape)

print("\nFirst 10 Unigrams:")
print(tfidf_unigram.get_feature_names_out()[:10])

print("\nFirst 10 Bigrams:")
print(tfidf_bigram.get_feature_names_out()[:10])

print("\nFirst 10 Trigrams:")
print(tfidf_trigram.get_feature_names_out()[:10])
```

N-gram features created successfully.

Unigram shape:
(1968, 6543)

Bigram shape:
(1968, 10480)

Trigram shape:
(1968, 9341)

First 10 Unigrams:
['أبتسم' 'أبتد' 'أبتسمه' 'أبتد' 'أب' 'أب' 'أب' 'أب' 'أب' 'أب']

First 10 Bigrams:
['أب الله القرآن' 'أب الله ياتوبه' 'أب الله ياتوبه' 'أب الله ياتوبه' 'أب الله ياتوبه' 'أب الله ياتوبه' 'أب الله ياتوبه' 'أب الله ياتوبه' 'أب الله ياتوبه' 'أب الله ياتوبه']

First 10 Trigrams:
['أب الله القرآن فقلت' 'أب الله ياتوبه عافاالله' 'أب الله ياتوبه عافاالله' 'أب الله ياتوبه عافاالله' 'أب الله ياتوبه عافاالله' 'أب الله ياتوبه عافاالله' 'أب الله ياتوبه عافاالله' 'أب الله ياتوبه عافاالله' 'أب الله ياتوبه عافاالله' 'أب الله ياتوبه عافاالله']

In [19]: # Step 16: Split the Arabic Dataset into Training and Testing Sets

```
from sklearn.model_selection import train_test_split

# Input text and target labels
X = arabic_df["processed_text"]
y = arabic_df["sentiment"]

# Split the dataset: 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.20,
    random_state=42,
    stratify=y
)

print("Arabic dataset split successfully.")

print("\nTraining samples:")
print(len(X_train))

print("\nTesting samples:")
print(len(X_test))

print("\nTraining sentiment distribution:")
print(y_train.value_counts())

print("\nTesting sentiment distribution:")
print(y_test.value_counts())
```

Arabic dataset split successfully.

Training samples:
1574

Testing samples:
394

Training sentiment distribution:
sentiment
positive 795
negative 779
Name: count, dtype: int64

Testing sentiment distribution:
sentiment
positive 199
negative 195
Name: count, dtype: int64

In [20]: # Step 17: Create TF-IDF Features for Training and Testing Data

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Create TF-IDF vectorizer using Unigrams and Bigrams
tfidf_model = TfidfVectorizer(
    ngram_range=(1, 2)
)

# Fit TF-IDF only on the training data
X_train_tfidf = tfidf_model.fit_transform(X_train)

# Transform the testing data using the same vectorizer
X_test_tfidf = tfidf_model.transform(X_test)

print("TF-IDF features created successfully.")

print("\nTraining TF-IDF shape:")
print(X_train_tfidf.shape)

print("\nTesting TF-IDF shape:")
print(X_test_tfidf.shape)

print("\nNumber of features:")
print(len(tfidf_model.get_feature_names_out()))
```

TF-IDF features created successfully.

Training TF-IDF shape:
(1574, 13997)

Testing TF-IDF shape:
(394, 13997)

Number of features:
13997

In [21]: # Step 18: Train and Evaluate Logistic Regression for Arabic Sentiment Analysis

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# Create and train the Logistic Regression model
logistic_model = LogisticRegression(
    max_iter=1000,
    random_state=42
)

logistic_model.fit(X_train_tfidf, y_train)

# Make predictions
y_pred_logistic = logistic_model.predict(X_test_tfidf)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred_logistic)
precision = precision_score(
    y_test, y_pred_logistic,
    pos_label="positive"
)
recall = recall_score(
    y_test, y_pred_logistic,
    pos_label="positive"
)
f1 = f1_score(
    y_test, y_pred_logistic,
    pos_label="positive"
)

print("Logistic Regression trained successfully.")

print("\nEvaluation Results:")
print(f"Accuracy : {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall : {recall:.4f}")
print(f"F1-Score : {f1:.4f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred_logistic))

print("\nConfusion Matrix:")
print(
    confusion_matrix(
        y_test,
        y_pred_logistic,
        labels=["negative", "positive"]
    )
)
```

```
Logistic Regression trained successfully.

Evaluation Results:
Accuracy : 0.8249
Precision: 0.8316
Recall   : 0.8191
F1-Score : 0.8253

Classification Report:
              precision    recall  f1-score   support

   negative       0.82       0.83       0.82       195
   positive       0.83       0.82       0.83       199

   accuracy                0.82       0.82       0.82       394
  macro avg       0.82       0.82       0.82       394
weighted avg       0.82       0.82       0.82       394

Confusion Matrix:
[[162  33]
 [ 36 163]]
```

In [22]: # Step 19: Visualize the Logistic Regression Confusion Matrix

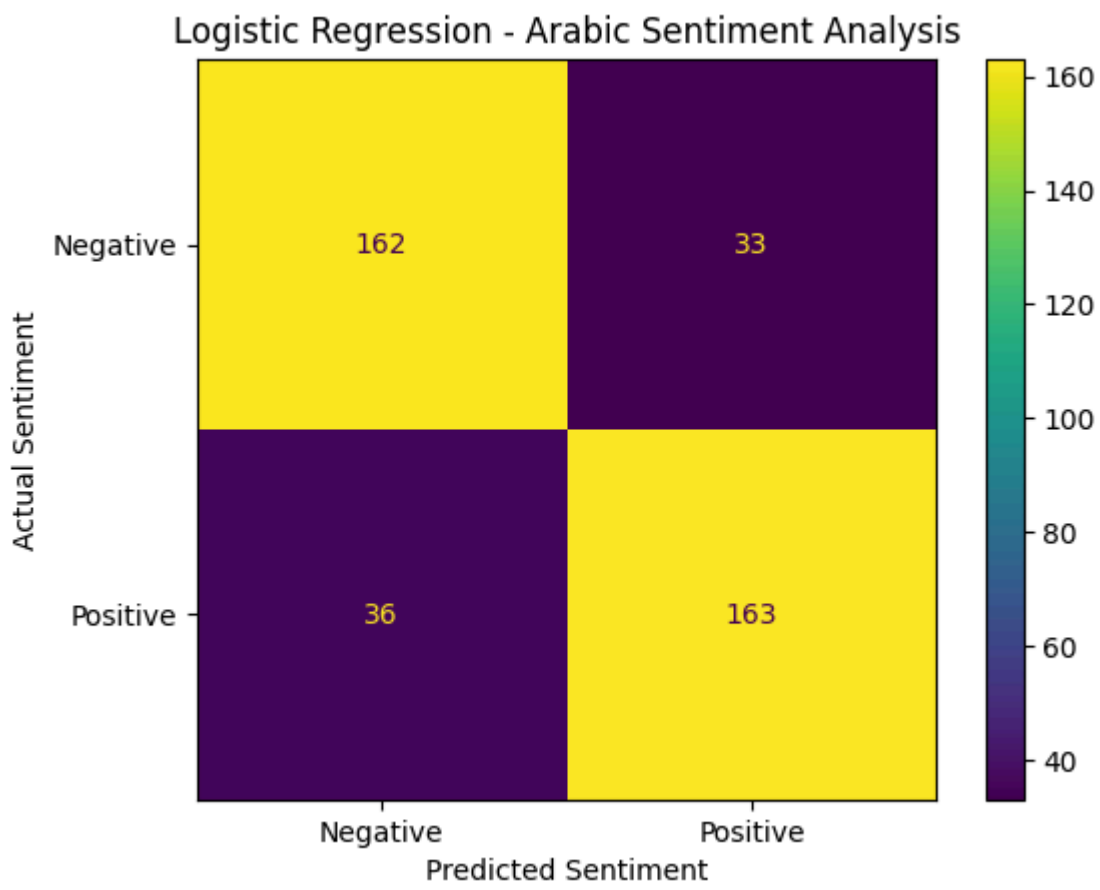
```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

cm = confusion_matrix(
    y_test,
    y_pred_logistic,
    labels=["negative", "positive"]
)

disp = ConfusionMatrixDisplay(
    confusion_matrix=cm,
    display_labels=["Negative", "Positive"]
)

disp.plot(values_format="d")

plt.title("Logistic Regression - Arabic Sentiment Analysis")
plt.xlabel("Predicted Sentiment")
plt.ylabel("Actual Sentiment")
plt.show()
```



In [23]: # Step 20: Train and Evaluate SVM for Arabic Sentiment Analysis

```
from sklearn.svm import LinearSVC
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# Create and train the SVM model
svm_model = LinearSVC(
    random_state=42
)

svm_model.fit(X_train_tfidf, y_train)

# Make predictions
y_pred_svm = svm_model.predict(X_test_tfidf)

# Calculate evaluation metrics
svm_accuracy = accuracy_score(y_test, y_pred_svm)

svm_precision = precision_score(
    y_test,
    y_pred_svm,
    pos_label="positive"
)

svm_recall = recall_score(
    y_test,
    y_pred_svm,
    pos_label="positive"
)

svm_f1 = f1_score(
    y_test,
    y_pred_svm,
    pos_label="positive"
)

print("SVM trained successfully.")

print("\nEvaluation Results:")
print(f"Accuracy : {svm_accuracy:.4f}")
print(f"Precision: {svm_precision:.4f}")
print(f"Recall   : {svm_recall:.4f}")
print(f"F1-Score  : {svm_f1:.4f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred_svm))

print("\nConfusion Matrix:")
print(
    confusion_matrix(
        y_test,
        y_pred_svm,
        labels=["negative", "positive"]
    )
)
```

SVM trained successfully.

Evaluation Results:
Accuracy : 0.8477
Precision: 0.8458
Recall : 0.8543
F1-Score : 0.8500

Classification Report:				
	precision	recall	f1-score	support
negative	0.85	0.84	0.85	195
positive	0.85	0.85	0.85	199
accuracy			0.85	394
macro avg	0.85	0.85	0.85	394
weighted avg	0.85	0.85	0.85	394

Confusion Matrix:
[[164 31]
 [29 170]]

In [24]: # Step 21: Train and Evaluate Naive Bayes for Arabic Sentiment Analysis

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# Create and train the Naive Bayes model
nb_model = MultinomialNB()

nb_model.fit(X_train_tfidf, y_train)

# Make predictions
y_pred_nb = nb_model.predict(X_test_tfidf)

# Calculate evaluation metrics
nb_accuracy = accuracy_score(y_test, y_pred_nb)

nb_precision = precision_score(
    y_test,
    y_pred_nb,
    pos_label="positive"
)

nb_recall = recall_score(
    y_test,
    y_pred_nb,
    pos_label="positive"
)

nb_f1 = f1_score(
    y_test,
    y_pred_nb,
    pos_label="positive"
)

print("Naive Bayes trained successfully.")

print("\nEvaluation Results:")
print(f"Accuracy : {nb_accuracy:.4f}")
print(f"Precision: {nb_precision:.4f}")
print(f"Recall   : {nb_recall:.4f}")
print(f"F1-Score  : {nb_f1:.4f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred_nb))

print("\nConfusion Matrix:")
print(
    confusion_matrix(
        y_test,
        y_pred_nb,
        labels=["negative", "positive"]
    )
)
```

Naive Bayes trained successfully.

Evaluation Results:
Accuracy : 0.8528
Precision: 0.8079
Recall : 0.9296
F1-Score : 0.8645

Classification Report:				
	precision	recall	f1-score	support
negative	0.92	0.77	0.84	195
positive	0.81	0.93	0.86	199
accuracy			0.85	394
macro avg	0.86	0.85	0.85	394
weighted avg	0.86	0.85	0.85	394

Confusion Matrix:
[[151 44]
 [14 185]]

In [25]: # Step 22: Train and Evaluate Random Forest for Arabic Sentiment Analysis

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# Create and train the Random Forest model
rf_model = RandomForestClassifier(
    n_estimators=200,
    random_state=42,
    n_jobs=-1
)

rf_model.fit(X_train_tfidf, y_train)

# Make predictions
y_pred_rf = rf_model.predict(X_test_tfidf)

# Calculate evaluation metrics
rf_accuracy = accuracy_score(y_test, y_pred_rf)

rf_precision = precision_score(
    y_test,
    y_pred_rf,
    pos_label="positive"
)

rf_recall = recall_score(
    y_test,
    y_pred_rf,
    pos_label="positive"
)

rf_f1 = f1_score(
    y_test,
    y_pred_rf,
    pos_label="positive"
)

print("Random Forest trained successfully.")

print("\nEvaluation Results:")
print(f"Accuracy : {rf_accuracy:.4f}")
print(f"Precision: {rf_precision:.4f}")
print(f"Recall   : {rf_recall:.4f}")
print(f"F1-Score  : {rf_f1:.4f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred_rf))

print("\nConfusion Matrix:")
print(
    confusion_matrix(
        y_test,
        y_pred_rf,
        labels=["negative", "positive"]
    )
)
```

Random Forest trained successfully.

Evaluation Results:
Accuracy : 0.7640
Precision: 0.8193
Recall : 0.6834
F1-Score : 0.7452

Classification Report:				
	precision	recall	f1-score	support
negative	0.72	0.85	0.78	195
positive	0.82	0.68	0.75	199
accuracy			0.76	394
macro avg	0.77	0.76	0.76	394
weighted avg	0.77	0.76	0.76	394

Confusion Matrix:
[[165 30]
 [63 136]]

In [26]: # Step 23: Train and Evaluate Decision Tree for Arabic Sentiment Analysis

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# Create and train the Decision Tree model
dt_model = DecisionTreeClassifier(
    random_state=42
)

dt_model.fit(X_train_tfidf, y_train)

# Make predictions
y_pred_dt = dt_model.predict(X_test_tfidf)

# Calculate evaluation metrics
dt_accuracy = accuracy_score(y_test, y_pred_dt)

dt_precision = precision_score(
    y_test,
    y_pred_dt,
    pos_label="positive"
)

dt_recall = recall_score(
    y_test,
    y_pred_dt,
    pos_label="positive"
)

dt_f1 = f1_score(
    y_test,
    y_pred_dt,
    pos_label="positive"
)

print("Decision Tree trained successfully.")

print("\nEvaluation Results:")
print(f"Accuracy : {dt_accuracy:.4f}")
print(f"Precision: {dt_precision:.4f}")
print(f"Recall   : {dt_recall:.4f}")
print(f"F1-Score  : {dt_f1:.4f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred_dt))

print("\nConfusion Matrix:")
print(
    confusion_matrix(
        y_test,
        y_pred_dt,
        labels=["negative", "positive"]
    )
)
```

Decision Tree trained successfully.

Evaluation Results:
Accuracy : 0.7284
Precision: 0.7447
Recall : 0.7035
F1-Score : 0.7235

Classification Report:				
	precision	recall	f1-score	support
negative	0.71	0.75	0.73	195
positive	0.74	0.70	0.72	199
accuracy			0.73	394
macro avg	0.73	0.73	0.73	394
weighted avg	0.73	0.73	0.73	394

Confusion Matrix:
[[147 48]
 [59 140]]

In [27]: # Step 24: Train and Evaluate KNN for Arabic Sentiment Analysis

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# Create the KNN model
knn_model = KNeighborsClassifier(
    n_neighbors=5
)

# Train the model
knn_model.fit(X_train_tfidf, y_train)

# Make predictions
y_pred_knn = knn_model.predict(X_test_tfidf)

# Calculate evaluation metrics
knn_accuracy = accuracy_score(y_test, y_pred_knn)

knn_precision = precision_score(
    y_test,
    y_pred_knn,
    pos_label="positive"
)

knn_recall = recall_score(
    y_test,
    y_pred_knn,
    pos_label="positive"
)

knn_f1 = f1_score(
    y_test,
    y_pred_knn,
    pos_label="positive"
)

print("KNN trained successfully.")

print("\nEvaluation Results:")
print(f"Accuracy : {knn_accuracy:.4f}")
print(f"Precision: {knn_precision:.4f}")
print(f"Recall   : {knn_recall:.4f}")
print(f"F1-Score  : {knn_f1:.4f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred_knn))

print("\nConfusion Matrix:")
print(
    confusion_matrix(
        y_test,
        y_pred_knn,
        labels=["negative", "positive"]
    )
)
```

```
KNN trained successfully.

Evaluation Results:
Accuracy : 0.8299
Precision: 0.8402
Recall   : 0.8191
F1-Score : 0.8295

Classification Report:
              precision    recall  f1-score   support

   negative         0.82         0.84         0.83         195
   positive         0.84         0.82         0.83         199

   accuracy                0.83         0.83         0.83         394
  macro avg              0.83         0.83         0.83         394
weighted avg              0.83         0.83         0.83         394


Confusion Matrix:
[[164  31]
 [ 36 163]]
```

In [28]: # Step 25: Compare All Arabic Sentiment Analysis Models

```
import pandas as pd

arabic_results = pd.DataFrame({
    "Model": [
        "Logistic Regression",
        "SVM",
        "Naive Bayes",
        "Random Forest",
        "Decision Tree",
        "KNN"
    ],
    "Accuracy": [
        accuracy,
        svm_accuracy,
        nb_accuracy,
        rf_accuracy,
        dt_accuracy,
        knn_accuracy
    ],
    "Precision": [
        precision,
        svm_precision,
        nb_precision,
        rf_precision,
        dt_precision,
        knn_precision
    ],
    "Recall": [
        recall,
        svm_recall,
        nb_recall,
        rf_recall,
        dt_recall,
        knn_recall
    ],
    "F1-Score": [
        f1,
        svm_f1,
        nb_f1,
        rf_f1,
        dt_f1,
        knn_f1
    ]
})

# Sort models by Accuracy
arabic_results = arabic_results.sort_values(
    by="Accuracy",
    ascending=False
).reset_index(drop=True)

# Convert results to percentages
arabic_results_display = arabic_results.copy()

for column in ["Accuracy", "Precision", "Recall", "F1-Score"]:
    arabic_results_display[column] = (
        arabic_results_display[column] * 100
    ).round(2)

print("Arabic Sentiment Analysis - Model Comparison")

display(arabic_results_display)
```

Arabic Sentiment Analysis - Model Comparison

	Model	Accuracy	Precision	Recall	F1-Score
0	Naive Bayes	85.28	80.79	92.96	86.45
1	SVM	84.77	84.58	85.43	85.00
2	KNN	82.99	84.02	81.91	82.95
3	Logistic Regression	82.49	83.16	81.91	82.53
4	Random Forest	76.40	81.93	68.34	74.52
5	Decision Tree	72.84	74.47	70.35	72.35

```
In [29]: # Step 26: Visualize the Performance of All Arabic Sentiment Models

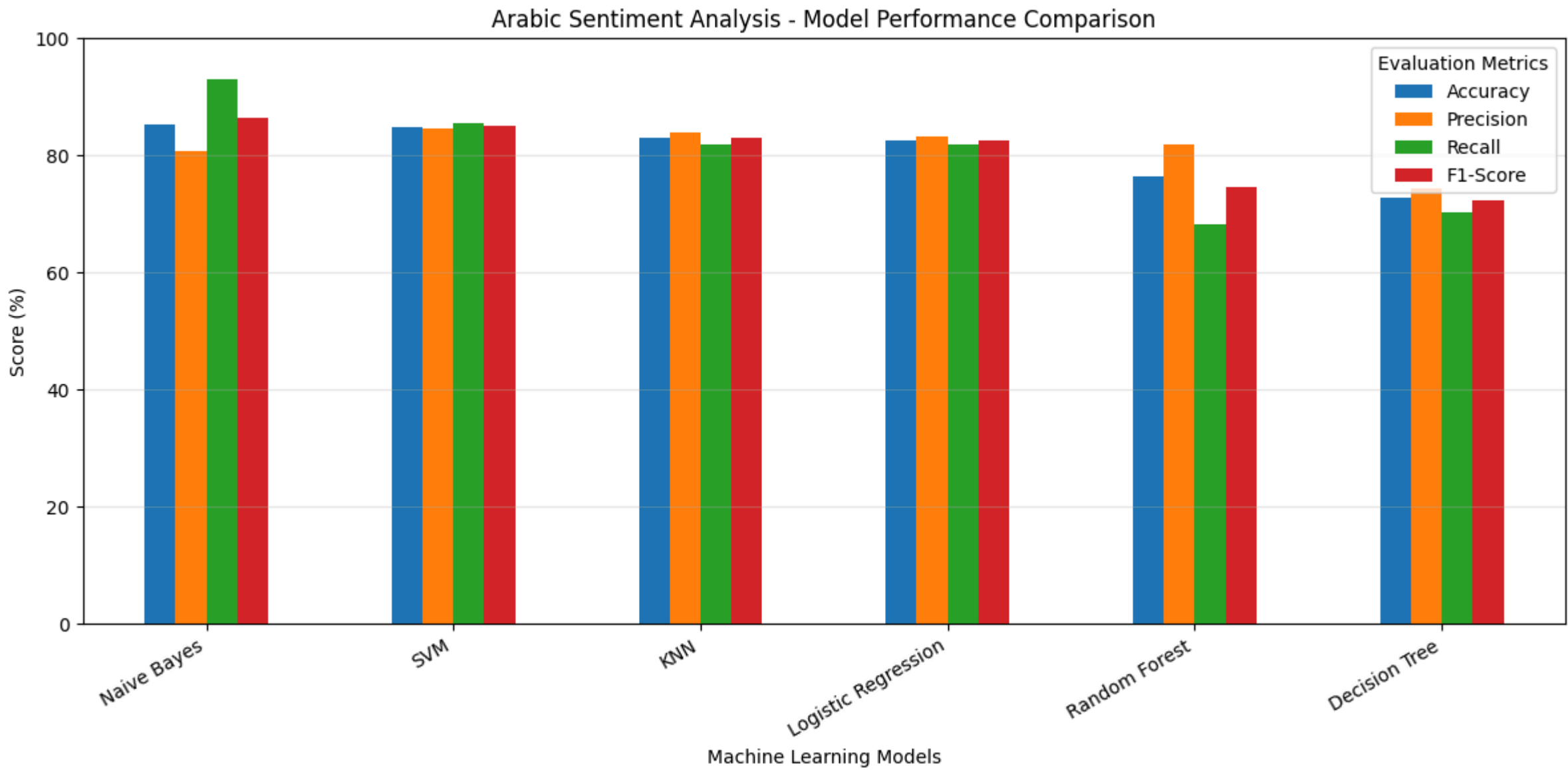
import matplotlib.pyplot as plt

# Prepare data for visualization
plot_data = arabic_results_display.set_index("Model")[
    ["Accuracy", "Precision", "Recall", "F1-Score"]
]

# Create comparison chart
ax = plot_data.plot(
    kind="bar",
    figsize=(12, 6)
)

plt.title("Arabic Sentiment Analysis - Model Performance Comparison")
plt.xlabel("Machine Learning Models")
plt.ylabel("Score (%)")
plt.ylim(0, 100)
plt.xticks(rotation=30, ha="right")
plt.legend(title="Evaluation Metrics")
plt.grid(axis="y", alpha=0.3)

plt.tight_layout()
plt.show()
```



```
In [30]: # Step 27: Save the Best Arabic Model and TF-IDF Vectorizer

import os
import joblib

# File paths
arabic_model_path = os.path.join(
    models_path,
    "arabic_naive_bayes_model.pkl"
)

arabic_tfidf_path = os.path.join(
    models_path,
    "arabic_tfidf_vectorizer.pkl"
)

# Save the trained Naive Bayes model
joblib.dump(nb_model, arabic_model_path)

# Save the TF-IDF vectorizer
joblib.dump(tfidf_model, arabic_tfidf_path)

print("Best Arabic model saved successfully.")
print("Model:", arabic_model_path)

print("\nArabic TF-IDF vectorizer saved successfully.")
print("Vectorizer:", arabic_tfidf_path)
```

```
Best Arabic model saved successfully.
Model: /content/drive/MyDrive/Arabic_English_Sentiment_Analysis/Models/arabic_naive_bayes_model.pkl

Arabic TF-IDF vectorizer saved successfully.
Vectorizer: /content/drive/MyDrive/Arabic_English_Sentiment_Analysis/Models/arabic_tfidf_vectorizer.pkl
```

```
In [31]: # Step 28: Test the Saved Arabic Sentiment Model with New Sentences

import joblib

# Load the saved Arabic model and TF-IDF vectorizer
loaded_arabic_model = joblib.load(arabic_model_path)
loaded_arabic_tfidf = joblib.load(arabic_tfidf_path)

# New Arabic sentences for testing
test_sentences = [
    "الخدمة ممتازة وأنا سعيد جداً",
    "التجربة رائعة وأنصح الجميع بها",
    "الخدمة سيئة جداً ولن أستخدمها مرة أخرى",
    "أنا غير راضي عن هذه التجربة"
]

# Apply the same Arabic preprocessing used during training
processed_sentences = [
    remove_arabic_stopwords(clean_arabic_text(text))
    for text in test_sentences
]

# Convert the processed sentences to TF-IDF features
test_features = loaded_arabic_tfidf.transform(processed_sentences)

# Predict sentiment
predictions = loaded_arabic_model.predict(test_features)

# Display results
print("Arabic Sentiment Prediction Results:\n")

for original_text, processed_text, prediction in zip(
    test_sentences,
    processed_sentences,
    predictions
):
    print("Text:", original_text)
    print("Processed:", processed_text)
    print("Prediction:", prediction)
    print("-" * 50)
```

```
Arabic Sentiment Prediction Results:

Text: الخدمة ممتازة وأنا سعيد جداً
Processed: الخدمة ممتازة وأنا سعيد جداً
Prediction: positive
-----
Text: التجربة رائعة وأنصح الجميع بها
Processed: التجربة رائعة وأنصح الجميع
Prediction: positive
-----
Text: الخدمة سيئة جداً ولن أستخدمها مرة أخرى
Processed: الخدمة سيئة جداً ولن أستخدمها مرة أخرى
Prediction: positive
-----
Text: أنا غير راضي عن هذه التجربة
Processed: أنا راضي التجربة
Prediction: positive
-----
```

```
In [32]: # Step 29: Analyze Wrong Arabic Predictions and Check Negation Words

# Sentences that were predicted incorrectly
wrong_sentences = [
    "الخدمة سيئة جدا وإن استخدمها مرة أخرى",
    "أنا غير راضي عن هذه التجربة"
]

# Important Arabic negation words
negation_words = [
    "ي",
    "إن",
    "لم",
    "ليس",
    "ليست",
    "غير",
    "ما",
    "مثل"
]

print("Arabic Negation Analysis:\n")

for sentence in wrong_sentences:
    cleaned = clean_arabic_text(sentence)
    processed = remove_arabic_stopwords(cleaned)

    print("Original Text :", sentence)
    print("Cleaned Text :", cleaned)
    print("Processed Text:", processed)

    found_negations = [
        word for word in negation_words
        if word in cleaned.split()
    ]

    print("Negation Words Found:", found_negations)

    removed_negations = [
        word for word in found_negations
        if word not in processed.split()
    ]

    print("Negation Words Removed:", removed_negations)
    print("-" * 60)
```

```
Arabic Negation Analysis:

Original Text : الخدمة سيئة جدا وإن استخدمها مرة أخرى
Cleaned Text  : الخدمة سيئة جدا وإن استخدمها مرة اخرى
Processed Text: الخدمة سيئة جدا وإن استخدمها مرة اخرى
Negation Words Found: []
Negation Words Removed: []
-----
Original Text : أنا غير راضي عن هذه التجربة
Cleaned Text  : أنا غير راضي عن هذه التجربة
Processed Text: أنا راضي التجربة
Negation Words Found: ['غير']
Negation Words Removed: ['غير']
-----
```

```
In [33]: # Step 30: Improve Arabic Preprocessing and Preserve Negation Words

import re
import nltk
from nltk.corpus import stopwords

nltk.download("stopwords")

# Load Arabic stop words
arabic_stopwords = set(stopwords.words("arabic"))

# Important negation words that must NOT be removed
arabic_negations = {
    "ي",
    "إن",
    "لم",
    "ليس",
    "ليست",
    "غير",
    "ما",
    "مثل"
}

# Remove negation words from the stop-word list
arabic_stopwords = arabic_stopwords - arabic_negations

def improved_arabic_preprocessing(text):

    # Apply the original cleaning function
    text = clean_arabic_text(text)

    # Separate common negation words when attached to "و"
    text = re.sub(r"(\b|و)", "و", text)
    text = re.sub(r"(\b|و)", "و", text)
    text = re.sub(r"(\b|و)", "و", text)
    text = re.sub(r"(\b|و)", "و", text)
    text = re.sub(r"(\b|و)", "و", text)
    text = re.sub(r"(\b|و)", "و", text)

    # Tokenize
    words = text.split()

    # Remove stop words while preserving negations
    words = [
        word for word in words
        if word not in arabic_stopwords
    ]

    return " ".join(words)

# Test the improved preprocessing
test_sentences = [
    "الخدمة سيئة جدا وإن استخدمها مرة أخرى",
    "أنا غير راضي عن هذه التجربة",
    "الخدمة ممتازة وأنا سعيد جدا",
    "التجربة رائعة والنصح للجميع بها"
]

print("Improved Arabic Preprocessing:\n")

for sentence in test_sentences:

    processed = improved_arabic_preprocessing(sentence)

    print("Original :", sentence)
    print("Processed:", processed)
    print("-" * 60)
```

```
Improved Arabic Preprocessing:

Original : الخدمة سيئة جدا وإن استخدمها مرة أخرى
Processed: الخدمة سيئة جدا إن استخدمها مرة اخرى
-----
Original : أنا غير راضي عن هذه التجربة
Processed: أنا غير راضي التجربة
-----
Original : الخدمة ممتازة وأنا سعيد جدا
Processed: الخدمة ممتازة وأنا سعيد جدا
-----
Original : التجربة رائعة والنصح للجميع بها
Processed: التجربة رائعة والنصح للجميع
-----

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
```

In [34]: # Step 31: Reprocess the Arabic Dataset with Improved Preprocessing

```
# Create a copy of the Arabic dataset
arabic_df_improved = arabic_df.copy()

# Apply the improved preprocessing
arabic_df_improved["processed_text"] = (
    arabic_df_improved["text"]
    .apply(improved_arabic_preprocessing)
)

# Check for empty processed texts
empty_count = (
    arabic_df_improved["processed_text"]
    .str.strip()
    .eq("")
    .sum()
)

print("Arabic dataset reprocessed successfully.")

print("\nDataset shape:")
print(arabic_df_improved.shape)

print("\nEmpty processed texts:")
print(empty_count)

print("\nSentiment distribution:")
print(arabic_df_improved["sentiment"].value_counts())

print("\nSample after improved preprocessing:")

display(
    arabic_df_improved[
        ["text", "processed_text", "sentiment"]
    ].head(10)
)
```

Arabic dataset reprocessed successfully.

Dataset shape:
(1968, 4)

Empty processed texts:
0

Sentiment distribution:
sentiment
positive 994
negative 974
Name: count, dtype: int64

Sample after improved preprocessing:

	text	processed_text	sentiment
0	صح جدا	صح جدا	positive
1	اه و النعمة	اه النعمة	positive
2	كلامك جميل ورائع	كلامك جميل ورائع	positive
3	و خير الكلام ما قل و دل	خير الكلام ما قل دل	positive
4	ولا تنسو تيسمك في وجه اخيك صدقه	لا تنسو تيسمك وجه اخيك صدقه	positive
5	ليتنا نستطيع الالتسامه في الاوقات المؤلمه	ليتنا نستطيع الالتسامه المولمه	positive
6	و يحب الناس الياقه الي بتضحك على طول	بحب الناس الياقه الي بتضحك علي طول	positive
7	...ابتسم امام كل الناس لئسهم و تجبر خواطريهم فتب	...ابتسم امام الناس لئسهم تجبر خواطريهم فتبسمك وج	positive
8	و الله انت حبيبي	الله انت حبيبي	positive
9	صح فالالتسامه تصنع السعاده	صح فالالتسامه تصنع السعاده	positive

In [35]: # Step 32: Split the Improved Arabic Dataset and Create TF-IDF Features

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

# Define input text and target labels
X = arabic_df_improved["processed_text"]
y = arabic_df_improved["sentiment"]

# Split dataset: 80% Training / 20% Testing
X_train_imp, X_test_imp, y_train_imp, y_test_imp = train_test_split(
    X,
    y,
    test_size=0.20,
    random_state=42,
    stratify=y
)

# Create improved TF-IDF vectorizer
tfidf_improved = TfidfVectorizer(
    ngram_range=(1, 2),
    min_df=1
)

# Learn vocabulary ONLY from training data
X_train_tfidf_imp = tfidf_improved.fit_transform(X_train_imp)

# Transform testing data using the same vocabulary
X_test_tfidf_imp = tfidf_improved.transform(X_test_imp)

print("Improved Arabic dataset prepared successfully.")

print("\nTraining samples:")
print(len(X_train_imp))

print("\nTesting samples:")
print(len(X_test_imp))

print("\nTraining TF-IDF shape:")
print(X_train_tfidf_imp.shape)

print("\nTesting TF-IDF shape:")
print(X_test_tfidf_imp.shape)

print("\nNumber of TF-IDF features:")
print(len(tfidf_improved.get_feature_names_out()))

print("\nTraining sentiment distribution:")
print(y_train_imp.value_counts())

print("\nTesting sentiment distribution:")
print(y_test_imp.value_counts())
```

Improved Arabic dataset prepared successfully.

Training samples:
1574

Testing samples:
394

Training TF-IDF shape:
(1574, 14266)

Testing TF-IDF shape:
(394, 14266)

Number of TF-IDF features:
14266

Training sentiment distribution:
sentiment
positive 795
negative 779
Name: count, dtype: int64

Testing sentiment distribution:
sentiment
positive 199
negative 195
Name: count, dtype: int64

In [36]: # Step 33: Train and Evaluate Improved Naive Bayes Model

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# Create the improved Naive Bayes model
nb_model_improved = MultinomialNB()

# Train the model
nb_model_improved.fit(
    X_train_tfidf_imp,
    y_train_imp
)

# Make predictions
y_pred_nb_improved = nb_model_improved.predict(
    X_test_tfidf_imp
)

# Calculate evaluation metrics
nb_imp_accuracy = accuracy_score(
    y_test_imp,
    y_pred_nb_improved
)

nb_imp_precision = precision_score(
    y_test_imp,
    y_pred_nb_improved,
    pos_label="positive"
)

nb_imp_recall = recall_score(
    y_test_imp,
    y_pred_nb_improved,
    pos_label="positive"
)

nb_imp_f1 = f1_score(
    y_test_imp,
    y_pred_nb_improved,
    pos_label="positive"
)

print("Improved Naive Bayes trained successfully.")

print("\nEvaluation Results:")
print(f"Accuracy : {nb_imp_accuracy:.4f}")
print(f"Precision: {nb_imp_precision:.4f}")
print(f"Recall   : {nb_imp_recall:.4f}")
print(f"F1-Score : {nb_imp_f1:.4f}")

print("\nClassification Report:")
print(
    classification_report(
        y_test_imp,
        y_pred_nb_improved
    )
)

print("\nConfusion Matrix:")
print(
    confusion_matrix(
        y_test_imp,
        y_pred_nb_improved,
        labels=["negative", "positive"]
    )
)

print("\nOld Naive Bayes Accuracy: 85.28%")
print(
    f"Improved Naive Bayes Accuracy: "
    f"{nb_imp_accuracy * 100:.2f}%"
)
```

Improved Naive Bayes trained successfully.

Evaluation Results:
Accuracy : 0.8503
Precision: 0.8125
Recall : 0.9146
F1-Score : 0.8605

Classification Report:				
	precision	recall	f1-score	support
negative	0.90	0.78	0.84	195
positive	0.81	0.91	0.86	199
accuracy			0.85	394
macro avg	0.86	0.85	0.85	394
weighted avg	0.86	0.85	0.85	394

Confusion Matrix:
[[153 42]
 [17 182]]

Old Naive Bayes Accuracy: 85.28%
Improved Naive Bayes Accuracy: 85.03%

In [37]: # Step 34: Test the Improved Arabic Model with New Sentences

```
# Test sentences
test_sentences_improved = [
    "الخدمة ممتازة وأنا سعيد جداً",
    "التجربة رائعة والصح الجميع بها",
    "الخدمة سيئة جداً وإن استخدمها مرة أخرى",
    "أنا غير راضي عن هذه التجربة"
]

# Apply the improved preprocessing
processed_test_sentences = [
    improved_arabic_preprocessing(text)
    for text in test_sentences_improved
]

# Transform using the NEW TF-IDF vectorizer
test_tfidf_improved = tfidf_improved.transform(
    processed_test_sentences
)

# Predict using the NEW Naive Bayes model
predictions_improved = nb_model_improved.predict(
    test_tfidf_improved
)

print("Improved Arabic Sentiment Prediction Results:\n")

for original, processed, prediction in zip(
    test_sentences_improved,
    processed_test_sentences,
    predictions_improved
):
    print("Original Text :", original)
    print("Processed Text:", processed)
    print("Prediction   :", prediction)
    print("-" * 60)
```

Improved Arabic Sentiment Prediction Results:

Original Text :	الخدمة ممتازة وأنا سعيد جداً
Processed Text:	الخدمة ممتازة وأنا سعيد جداً
Prediction	positive

Original Text :	التجربة رائعة والصح الجميع بها
Processed Text:	التجربة رائعة والصح الجميع
Prediction	positive

Original Text :	الخدمة سيئة جداً وإن استخدمها مرة أخرى
Processed Text:	الخدمة سيئة جداً لن استخدمها مرة أخرى
Prediction	positive

Original Text :	أنا غير راضي عن هذه التجربة
Processed Text:	أنا غير راضي التجربة
Prediction	negative

In [38]: # Step 35: Train and Test Improved SVM Model

```
from sklearn.svm import LinearSVC
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score
)

# Create SVM model
svm_model_improved = LinearSVC(
    random_state=42
)

# Train SVM using improved Arabic data
svm_model_improved.fit(
    X_train_tfidf_imp,
    y_train_imp
)

# Predict test dataset
y_pred_svm_improved = svm_model_improved.predict(
    X_test_tfidf_imp
)

# Calculate evaluation metrics
svm_imp_accuracy = accuracy_score(
    y_test_imp,
    y_pred_svm_improved
)

svm_imp_precision = precision_score(
    y_test_imp,
    y_pred_svm_improved,
    pos_label="positive"
)

svm_imp_recall = recall_score(
    y_test_imp,
    y_pred_svm_improved,
    pos_label="positive"
)

svm_imp_f1 = f1_score(
    y_test_imp,
    y_pred_svm_improved,
    pos_label="positive"
)

print("Improved SVM trained successfully.")

print("\nEvaluation Results:")
print(f"Accuracy : {svm_imp_accuracy:.4f}")
print(f"Precision: {svm_imp_precision:.4f}")
print(f"Recall   : {svm_imp_recall:.4f}")
print(f"F1-Score  : {svm_imp_f1:.4f}")

# Test SVM on the same four sentences
svm_manual_predictions = svm_model_improved.predict(
    test_tfidf_improved
)

print("\nManual Sentence Test:\n")

for original, prediction in zip(
    test_sentences_improved,
    svm_manual_predictions
):
    print("Text      :", original)
    print("Prediction:", prediction)
    print("-" * 60)
```

```
Improved SVM trained successfully.

Evaluation Results:
Accuracy : 0.8553
Precision: 0.8586
Recall   : 0.8543
F1-Score : 0.8564

Manual Sentence Test:

Text      : الخدمة ممتازة وأنا سعيد جدا
Prediction: positive
-----
Text      : التجربة رائعة والنصح الجميع بها
Prediction: positive
-----
Text      : الخدمة سيئة جدا وإن استخدما مرة أخرى
Prediction: positive
-----
Text      : أنا غير راضي عن هذه التجربة
Prediction: negative
-----
```

In [39]: # Step 36: Save the Final Improved Arabic SVM Model and TF-IDF Vectorizer

```
import os
import joblib

# Define final model paths
final_arabic_model_path = os.path.join(
    models_path,
    "arabic_svm_final_model.pkl"
)

final_arabic_tfidf_path = os.path.join(
    models_path,
    "arabic_tfidf_final_vectorizer.pkl"
)

# Save the improved SVM model
joblib.dump(
    svm_model_improved,
    final_arabic_model_path
)

# Save the improved TF-IDF vectorizer
joblib.dump(
    tfidf_improved,
    final_arabic_tfidf_path
)

print("Final Arabic SVM model saved successfully.")
print("Model:", final_arabic_model_path)

print("\nFinal Arabic TF-IDF vectorizer saved successfully.")
print("Vectorizer:", final_arabic_tfidf_path)

print("\nFinal Arabic Model Information:")
print("Model: Improved SVM")
print(f"Accuracy : {svm_imp_accuracy * 100:.2f}%")
print(f"Precision: {svm_imp_precision * 100:.2f}%")
print(f"Recall   : {svm_imp_recall * 100:.2f}%")
print(f"F1-Score : {svm_imp_f1 * 100:.2f}%")
```

```
Final Arabic SVM model saved successfully.
Model: /content/drive/MyDrive/Arabic_English_Sentiment_Analysis/Models/arabic_svm_final_model.pkl

Final Arabic TF-IDF vectorizer saved successfully.
Vectorizer: /content/drive/MyDrive/Arabic_English_Sentiment_Analysis/Models/arabic_tfidf_final_vectorizer.pkl

Final Arabic Model Information:
Model: Improved SVM
Accuracy : 85.53%
Precision: 85.86%
Recall   : 85.43%
F1-Score : 85.64%
```

In [40]: # Step 37: Extract the English Sentiment140 Dataset

```
import os
import zipfile

english_zip = os.path.join(
    datasets_path,
    "Sentiment140 dataset with 1.6 million tweets.zip"
)

english_extract_path = os.path.join(
    datasets_path,
    "Sentiment140_Dataset"
)

os.makedirs(english_extract_path, exist_ok=True)

with zipfile.ZipFile(english_zip, "r") as zip_ref:
    zip_ref.extractall(english_extract_path)

print("English Sentiment140 dataset extracted successfully.")
print("Extracted to:", english_extract_path)

print("\nExtracted files:")
for item in os.listdir(english_extract_path):
    print("-", item)
```

English Sentiment140 dataset extracted successfully.
Extracted to: /content/drive/MyDrive/Arabic_English_Sentiment_Analysis/Datasets/Sentiment140_Dataset

Extracted files:
- training.1600000.processed.noemoticon.csv

In [41]: # Step 38: Load and Inspect the English Sentiment140 Dataset

```
import pandas as pd
import os

english_csv_path = os.path.join(
    english_extract_path,
    "training.1600000.processed.noemoticon.csv"
)

# Sentiment140 columns
column_names = [
    "target",
    "id",
    "date",
    "flag",
    "user",
    "text"
]

# Load dataset
english_df = pd.read_csv(
    english_csv_path,
    encoding="latin-1",
    names=column_names,
    header=None
)

print("English Sentiment140 dataset loaded successfully.")

print("\nDataset shape:")
print(english_df.shape)

print("\nColumns:")
print(english_df.columns.tolist())

print("\nSentiment distribution:")
print(english_df["target"].value_counts())

print("\nFirst 10 samples:")
display(english_df.head(10))
```

English Sentiment140 dataset loaded successfully.

Dataset shape:
(1600000, 6)

Columns:
['target', 'id', 'date', 'flag', 'user', 'text']

Sentiment distribution:
target
0 800000
4 800000
Name: count, dtype: int64

First 10 samples:

	target	id	date	flag	user	text
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all...
5	0	1467811372	Mon Apr 06 22:20:00 PDT 2009	NO_QUERY	joy_wolf	@Kwesidei not the whole crew
6	0	1467811592	Mon Apr 06 22:20:03 PDT 2009	NO_QUERY	mybirch	Need a hug
7	0	1467811594	Mon Apr 06 22:20:03 PDT 2009	NO_QUERY	coZZ	@LOLTrish hey long time no see! Yes.. Rains a...
8	0	1467811795	Mon Apr 06 22:20:05 PDT 2009	NO_QUERY	2Hood4Hollywood	@Tatiana_K nope they didn't have it
9	0	1467812025	Mon Apr 06 22:20:09 PDT 2009	NO_QUERY	mimismo	@twittera que me muera ?

```
In [42]: # Step 39: Create a Balanced English Dataset Sample

import pandas as pd

# Select 50,000 negative tweets
negative_df = english_df[
    english_df["target"] == 0
].sample(
    n=50000,
    random_state=42
)

# Select 50,000 positive tweets
positive_df = english_df[
    english_df["target"] == 4
].sample(
    n=50000,
    random_state=42
)

# Combine both classes
english_sample_df = pd.concat(
    [negative_df, positive_df],
    ignore_index=True
)

# Shuffle the dataset
english_sample_df = english_sample_df.sample(
    frac=1,
    random_state=42
).reset_index(drop=True)

# Convert labels
english_sample_df["sentiment"] = english_sample_df["target"].map({
    0: "negative",
    4: "positive"
})

# Keep only required columns
english_sample_df = english_sample_df[
    ["text", "sentiment"]
]

print("Balanced English dataset created successfully.")

print("\nDataset shape:")
print(english_sample_df.shape)

print("\nSentiment distribution:")
print(english_sample_df["sentiment"].value_counts())

print("\nFirst 10 samples:")
display(english_sample_df.head(10))
```

Balanced English dataset created successfully.

Dataset shape:
(100000, 2)

Sentiment distribution:
sentiment
positive 50000
negative 50000
Name: count, dtype: int64

First 10 samples:

	text	sentiment
0	@stargazer60 that's awesome	positive
1	@cunningstunts22 yes... i am!	positive
2	N my bed aaalllll alone	negative
3	@Thorney88 re guinea fowl ive heard that lots ...	positive
4	def going to the movies tonite	positive
5	@MikePaetzold from http://bit.ly/15UN6p may i...	positive
6	Ok my belly is so full now,that apple pie was ...	positive
7	@Beans300 what's to be blue about? you're out ...	positive
8	@alvinlzg - how true. Eh? How come change twit...	positive
9	is happy that today is a good friday, don pani...	positive

```
In [43]: # Step 40: Inspect English Dataset Before Cleaning

print("English Dataset Information:")
print("Shape:", english_sample_df.shape)

print("\nMissing values:")
print(english_sample_df.isnull().sum())

print("\nDuplicate rows:")
print(english_sample_df.duplicated().sum())

print("\nEmpty text samples:")
print(
    (english_sample_df["text"]
     .astype(str)
     .str.strip() == "").sum()
)

print("\nSentiment distribution:")
print(english_sample_df["sentiment"].value_counts())

print("\nDataset sample:")
display(english_sample_df.head(10))
```

English Dataset Information:
Shape: (100000, 2)

Missing values:
text 0
sentiment 0
dtype: int64

Duplicate rows:
284

Empty text samples:
0

Sentiment distribution:
sentiment
positive 50000
negative 50000
Name: count, dtype: int64

Dataset sample:

	text	sentiment
0	@stargazer60 that's awesome	positive
1	@cunningstunts22 yes... i am!	positive
2	N my bed aaalllll alone	negative
3	@Thorney88 re guinea fowl ive heard that lots ...	positive
4	def going to the movies tonite	positive
5	@MikePaetzold from http://bit.ly/15UN6p may i...	positive
6	Ok my belly is so full now,that apple pie was ...	positive
7	@Beans300 what's to be blue about? you're out ...	positive
8	@alvinlzg - how true. Eh? How come change twit...	positive
9	is happy that today is a good friday, don pani...	positive

In [44]:

```
# Step 41: Clean the English Dataset

# Remove missing values
english_sample_df = english_sample_df.dropna(
    subset=["text", "sentiment"]
)

# Remove empty text
english_sample_df = english_sample_df[
    english_sample_df["text"].astype(str).str.strip() != ""
]

# Remove duplicate rows
english_sample_df = english_sample_df.drop_duplicates()

# Reset index
english_sample_df = english_sample_df.reset_index(drop=True)

print("English dataset cleaned successfully.")

print("\nDataset shape after cleaning:")
print(english_sample_df.shape)

print("\nRemaining duplicate rows:")
print(english_sample_df.duplicated().sum())

print("\nRemaining empty text samples:")
print(
    (english_sample_df["text"].astype(str).str.strip() == "").sum()
)

print("\nSentiment distribution after cleaning:")
print(english_sample_df["sentiment"].value_counts())
```

```
English dataset cleaned successfully.

Dataset shape after cleaning:
(99716, 2)

Remaining duplicate rows:
0

Remaining empty text samples:
0

Sentiment distribution after cleaning:
sentiment
positive    49896
negative    49820
Name: count, dtype: int64
```

In [45]:

```
# Step 42: English Text Preprocessing

import re

def clean_english_text(text):
    text = str(text).lower()

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

    # Remove Twitter usernames
    text = re.sub(r"@w+", " ", text)

    # Remove hashtag symbol but keep the word
    text = re.sub(r"#", "", text)

    # Replace HTML entity
    text = re.sub(r"&";, " and ", text)

    # Remove numbers
    text = re.sub(r"d+", " ", text)

    # Keep English letters and apostrophes
    text = re.sub(r"^[a-z']+", " ", text)

    # Remove extra spaces
    text = re.sub(r"s+", " ", text).strip()

    return text

# Apply preprocessing
english_sample_df["clean_text"] = (
    english_sample_df["text"]
    .apply(clean_english_text)
)

print("English text preprocessing completed successfully.")

print("\nOriginal text vs Cleaned text:")
display(
    english_sample_df[
        ["text", "clean_text", "sentiment"]
    ].head(10)
)

print("\nEmpty cleaned texts:")
print(
    (english_sample_df["clean_text"].str.strip() == "").sum()
)
```

```
English text preprocessing completed successfully.

Original text vs Cleaned text:
```

	text	clean_text	sentiment
0	@stargazer60 that's awesome	that's awesome	positive
1	@cunningstunts22 yes... i am!	yes i am	positive
2	N my bed aaaIIlll alone	n my bed aaaIIlll alone	negative
3	@Thorney88 re guinea fowl ive heard that lots ...	re guinea fowl ive heard that lots of people d...	positive
4	def going to the movies tonite	def going to the movies tonite	positive
5	@MikePaetzold from http://bit.ly/1SLN6p may i...	from may i request the free report plz thank you	positive
6	Ok my belly is so full now.that apple pie was ...	ok my belly is so full now that apple pie was ...	positive
7	@Beans300 what's to be blue about? you're out ...	what's to be blue about you're out on fi and i...	positive
8	@alvinlzg - how true. Eh? How come change twit...	how true eh how come change twitter display name	positive
9	is happy that today is a good friday, don pani...	is happy that today is a good friday don panic...	positive

```
Empty cleaned texts:
240
```

In [46]: # Step 43: Remove English Stop Words While Keeping Negation Words

```
import nltk
from nltk.corpus import stopwords

nltk.download("stopwords")

# Load English stop words
english_stop_words = set(stopwords.words("english"))

# Important negation words for sentiment analysis
negation_words = {
    "no", "not", "nor", "never",
    "don't", "doesn't", "didn't",
    "isn't", "aren't", "wasn't", "weren't",
    "can't", "couldn't", "won't", "wouldn't",
    "shouldn't", "haven't", "hasn't", "hadn't"
}

# Keep negation words
english_stop_words = english_stop_words - negation_words

def remove_english_stopwords(text):
    words = text.split()

    words = [
        word for word in words
        if word not in english_stop_words
    ]

    return " ".join(words)

# Apply stop-word removal
english_sample_df["processed_text"] = (
    english_sample_df["clean_text"]
    .apply(remove_english_stopwords)
)

print("English stop words removed successfully.")

print("\nOriginal vs Cleaned vs Processed Text:")

display(
    english_sample_df[
        ["text", "clean_text", "processed_text", "sentiment"]
    ].head(10)
)

print("\nEmpty processed texts:")
print(
    (english_sample_df["processed_text"].str.strip() == "").sum()
)
```

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!

English stop words removed successfully.

Original vs Cleaned vs Processed Text:

	text	clean_text	processed_text	sentiment
0	@stargazer60 that's awesome	that's awesome	that's awesome	positive
1	@cunningstunts22 yes... i am!	yes i am	yes	positive
2	N my bed aaalllll alone	n my bed aaalllll alone	n bed aaalllll alone	negative
3	@Thorney88 re guinea fowl ive heard that lots ...	re guinea fowl ive heard that lots of people d...	guinea fowl ive heard lots people dislike say ...	positive
4	def going to the movies tonite	def going to the movies tonite	def going movies tonite	positive
5	@MikePaetzold from http://bit.ly/15LN6p may i...	from may i request the free report plz thank you	may request free report plz thank	positive
6	Ok my belly is so full now,that apple pie was ...	ok my belly is so full now that apple pie was ...	ok belly full apple pie great addition milksha...	positive
7	@Beans300 what's to be blue about? you're out ...	what's to be blue about you're out on fi and i...	what's blue fi sunny day	positive
8	@alvinlzg - how true. Eh? How come change twit...	how true eh how come change twitter display name	true eh come change twitter display name	positive
9	is happy that today is a good friday, don pani...	is happy that today is a good friday don panic...	happy today good friday panic happy	positive

Empty processed texts:
465

In [47]: # Step 44: Finalize English Preprocessing

```
# Count empty processed texts before removal
empty_count = (
    english_sample_df["processed_text"]
    .str.strip()
    .eq("")
    .sum()
)

print("Empty processed texts before removal:", empty_count)

# Remove empty processed texts
english_sample_df = english_sample_df[
    english_sample_df["processed_text"].str.strip() != ""
].copy()

# Reset index
english_sample_df.reset_index(drop=True, inplace=True)

print("\nEnglish dataset after preprocessing:")

print("Shape:")
print(english_sample_df.shape)

print("\nSentiment distribution:")
print(english_sample_df["sentiment"].value_counts())

print("\nRemaining empty processed texts:")
print(
    english_sample_df["processed_text"]
    .str.strip()
    .eq("")
    .sum()
)

print("\nFinal sample:")
display(
    english_sample_df[
        ["text", "processed_text", "sentiment"]
    ].head(10)
)
```

Empty processed texts before removal: 465

English dataset after preprocessing:
Shape:
(99251, 4)

Sentiment distribution:
sentiment
positive 49632
negative 49619
Name: count, dtype: int64

Remaining empty processed texts:
0

Final sample:

	text	processed_text	sentiment
0	@stargazer60 that's awesome	that's awesome	positive
1	@cunningstunts22 yes... i am!	yes	positive
2	N my bed aaalllll alone	n bed aaalllll alone	negative
3	@Thorney88 re guinea fowl ive heard that lots ...	guinea fowl ive heard lots people dislike say ...	positive
4	def going to the movies tonite	def going movies tonite	positive
5	@MikePaetzold from http://bit.ly/15LN6p may i...	may request free report plz thank	positive
6	Ok my belly is so full now,that apple pie was ...	ok belly full apple pie great addition milksha...	positive
7	@Beans300 what's to be blue about? you're out ...	what's blue fi sunny day	positive
8	@alvinlzg - how true. Eh? How come change twit...	true eh come change twitter display name	positive
9	is happy that today is a good friday, don pani...	happy today good friday panic happy	positive

In [48]: # Step 45: Split English Dataset into Training and Testing Sets

```
from sklearn.model_selection import train_test_split

X = english_sample_df["processed_text"]
y = english_sample_df["sentiment"]

X_train_eng, X_test_eng, y_train_eng, y_test_eng = train_test_split(
    X,
    y,
    test_size=0.20,
    random_state=42,
    stratify=y
)

print("English dataset split successfully.")

print("\nTraining samples:")
print(len(X_train_eng))

print("\nTesting samples:")
print(len(X_test_eng))

print("\nTraining sentiment distribution:")
print(y_train_eng.value_counts())

print("\nTesting sentiment distribution:")
print(y_test_eng.value_counts())
```

```
English dataset split successfully.

Training samples:
79400

Testing samples:
19851

Training sentiment distribution:
sentiment
positive    39705
negative    39695
Name: count, dtype: int64

Testing sentiment distribution:
sentiment
positive     9927
negative     9924
Name: count, dtype: int64
```

In [49]: # Step 46: TF-IDF Feature Extraction for English Dataset

```
from sklearn.feature_extraction.text import TfidfVectorizer

english_tfidf = TfidfVectorizer(
    max_features=20000,
    ngram_range=(1, 2),
    min_df=2,
    max_df=0.95
)

# Fit only on training data
X_train_eng_tfidf = english_tfidf.fit_transform(X_train_eng)

# Transform testing data
X_test_eng_tfidf = english_tfidf.transform(X_test_eng)

print("English TF-IDF features created successfully.")

print("\nTraining TF-IDF shape:")
print(X_train_eng_tfidf.shape)

print("\nTesting TF-IDF shape:")
print(X_test_eng_tfidf.shape)

print("\nNumber of features:")
print(len(english_tfidf.get_feature_names_out()))

print("\nFirst 20 features:")
print(english_tfidf.get_feature_names_out()[:20])
```

```
English TF-IDF features created successfully.

Training TF-IDF shape:
(79400, 20000)

Testing TF-IDF shape:
(19851, 20000)

Number of features:
20000

First 20 features:
['aa' 'aaa' 'aaaaa' 'aaaaaah' 'aaaah' 'aaah' 'aaargh' 'aaw' 'aah' 'aahh'
 'aaron' 'aaw' 'aaww' 'ab' 'abandoned' 'abba' 'abc' 'abdc' 'ability'
 'abit']
```

In [50]: # Step 47: Train and Evaluate Logistic Regression - English

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# Create model
lr_eng = LogisticRegression(
    max_iter=1000,
    random_state=42
)

# Train model
lr_eng.fit(X_train_eng_tfidf, y_train_eng)

# Predictions
y_pred_lr_eng = lr_eng.predict(X_test_eng_tfidf)

# Evaluation
lr_eng_accuracy = accuracy_score(y_test_eng, y_pred_lr_eng)
lr_eng_precision = precision_score(
    y_test_eng, y_pred_lr_eng, pos_label="positive"
)
lr_eng_recall = recall_score(
    y_test_eng, y_pred_lr_eng, pos_label="positive"
)
lr_eng_f1 = f1_score(
    y_test_eng, y_pred_lr_eng, pos_label="positive"
)

print("English Logistic Regression trained successfully.")

print("\nEvaluation Results:")
print(f"Accuracy : {lr_eng_accuracy:.4f}")
print(f"Precision: {lr_eng_precision:.4f}")
print(f"Recall   : {lr_eng_recall:.4f}")
print(f"F1-Score  : {lr_eng_f1:.4f}")

print("\nClassification Report:")
print(classification_report(y_test_eng, y_pred_lr_eng))

print("\nConfusion Matrix:")
print(
    confusion_matrix(
        y_test_eng,
        y_pred_lr_eng,
        labels=["negative", "positive"]
    )
)
```

English Logistic Regression trained successfully.

Evaluation Results:
Accuracy : 0.7799
Precision: 0.7747
Recall : 0.7895
F1-Score : 0.7820

Classification Report:				
	precision	recall	f1-score	support
negative	0.79	0.77	0.78	9924
positive	0.77	0.79	0.78	9927
accuracy			0.78	19851
macro avg	0.78	0.78	0.78	19851
weighted avg	0.78	0.78	0.78	19851

Confusion Matrix:
[[7645 2279]
 [2090 7837]]

In [51]: # Step 48: Train and Evaluate SVM - English

```
from sklearn.svm import LinearSVC
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# Create SVM model
svm_eng = LinearSVC(
    random_state=42,
    max_iter=5000
)

# Train model
svm_eng.fit(X_train_eng_tfidf, y_train_eng)

# Prediction
y_pred_svm_eng = svm_eng.predict(X_test_eng_tfidf)

# Evaluation
svm_eng_accuracy = accuracy_score(y_test_eng, y_pred_svm_eng)

svm_eng_precision = precision_score(
    y_test_eng,
    y_pred_svm_eng,
    pos_label="positive"
)

svm_eng_recall = recall_score(
    y_test_eng,
    y_pred_svm_eng,
    pos_label="positive"
)

svm_eng_f1 = f1_score(
    y_test_eng,
    y_pred_svm_eng,
    pos_label="positive"
)

print("English SVM trained successfully.")

print("\nEvaluation Results:")
print(f"Accuracy : {svm_eng_accuracy:.4f}")
print(f"Precision: {svm_eng_precision:.4f}")
print(f"Recall   : {svm_eng_recall:.4f}")
print(f"F1-Score  : {svm_eng_f1:.4f}")

print("\nClassification Report:")
print(
    classification_report(
        y_test_eng,
        y_pred_svm_eng
    )
)

print("\nConfusion Matrix:")
print(
    confusion_matrix(
        y_test_eng,
        y_pred_svm_eng,
        labels=["negative", "positive"]
    )
)
```

English SVM trained successfully.

Evaluation Results:
Accuracy : 0.7662
Precision: 0.7623
Recall : 0.7737
F1-Score : 0.7680

Classification Report:				
	precision	recall	f1-score	support
negative	0.77	0.76	0.76	9924
positive	0.76	0.77	0.77	9927
accuracy			0.77	19851
macro avg	0.77	0.77	0.77	19851
weighted avg	0.77	0.77	0.77	19851

Confusion Matrix:
[[7529 2395]
 [2246 7681]]

In [52]: # Step 49: Train and Evaluate Naive Bayes - English

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# Create Naive Bayes model
nb_eng = MultinomialNB()

# Train model
nb_eng.fit(X_train_eng_tfidf, y_train_eng)

# Prediction
y_pred_nb_eng = nb_eng.predict(X_test_eng_tfidf)

# Evaluation
nb_eng_accuracy = accuracy_score(y_test_eng, y_pred_nb_eng)

nb_eng_precision = precision_score(
    y_test_eng,
    y_pred_nb_eng,
    pos_label="positive"
)

nb_eng_recall = recall_score(
    y_test_eng,
    y_pred_nb_eng,
    pos_label="positive"
)

nb_eng_f1 = f1_score(
    y_test_eng,
    y_pred_nb_eng,
    pos_label="positive"
)

print("English Naive Bayes trained successfully.")

print("\nEvaluation Results:")
print(f"Accuracy : {nb_eng_accuracy:.4f}")
print(f"Precision: {nb_eng_precision:.4f}")
print(f"Recall   : {nb_eng_recall:.4f}")
print(f"F1-Score  : {nb_eng_f1:.4f}")

print("\nClassification Report:")
print(
    classification_report(
        y_test_eng,
        y_pred_nb_eng
    )
)

print("\nConfusion Matrix:")
print(
    confusion_matrix(
        y_test_eng,
        y_pred_nb_eng,
        labels=["negative", "positive"]
    )
)
```

English Naive Bayes trained successfully.

Evaluation Results:
Accuracy : 0.7660
Precision: 0.7698
Recall : 0.7589
F1-Score : 0.7643

Classification Report:				
	precision	recall	f1-score	support
negative	0.76	0.77	0.77	9924
positive	0.77	0.76	0.76	9927
accuracy			0.77	19851
macro avg	0.77	0.77	0.77	19851
weighted avg	0.77	0.77	0.77	19851

Confusion Matrix:
[[7671 2253]
 [2393 7534]]

In [53]: # Step 50: Train and Evaluate Random Forest - English

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# Create Random Forest model
rf_eng = RandomForestClassifier(
    n_estimators=100,
    random_state=42,
    n_jobs=-1
)

# Train model
rf_eng.fit(X_train_eng_tfidf, y_train_eng)

# Prediction
y_pred_rf_eng = rf_eng.predict(X_test_eng_tfidf)

# Evaluation
rf_eng_accuracy = accuracy_score(y_test_eng, y_pred_rf_eng)

rf_eng_precision = precision_score(
    y_test_eng,
    y_pred_rf_eng,
    pos_label="positive"
)

rf_eng_recall = recall_score(
    y_test_eng,
    y_pred_rf_eng,
    pos_label="positive"
)

rf_eng_f1 = f1_score(
    y_test_eng,
    y_pred_rf_eng,
    pos_label="positive"
)

print("English Random Forest trained successfully.")

print("\nEvaluation Results:")
print(f"Accuracy : {rf_eng_accuracy:.4f}")
print(f"Precision: {rf_eng_precision:.4f}")
print(f"Recall   : {rf_eng_recall:.4f}")
print(f"F1-Score  : {rf_eng_f1:.4f}")

print("\nClassification Report:")
print(classification_report(y_test_eng, y_pred_rf_eng))

print("\nConfusion Matrix:")
print(
    confusion_matrix(
        y_test_eng,
        y_pred_rf_eng,
        labels=["negative", "positive"]
    )
)
```

English Random Forest trained successfully.

Evaluation Results:
Accuracy : 0.7638
Precision: 0.7717
Recall : 0.7495
F1-Score : 0.7604

Classification Report:				
	precision	recall	f1-score	support
negative	0.76	0.78	0.77	9924
positive	0.77	0.75	0.76	9927
accuracy			0.76	19851
macro avg	0.76	0.76	0.76	19851
weighted avg	0.76	0.76	0.76	19851

Confusion Matrix:
[[7723 2281]
 [2487 7440]]

In [54]: # Step 51: Train and Evaluate Decision Tree - English

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# Create Decision Tree model
dt_eng = DecisionTreeClassifier(
    random_state=42
)

# Train model
dt_eng.fit(X_train_eng_tfidf, y_train_eng)

# Prediction
y_pred_dt_eng = dt_eng.predict(X_test_eng_tfidf)

# Evaluation
dt_eng_accuracy = accuracy_score(y_test_eng, y_pred_dt_eng)

dt_eng_precision = precision_score(
    y_test_eng,
    y_pred_dt_eng,
    pos_label="positive"
)

dt_eng_recall = recall_score(
    y_test_eng,
    y_pred_dt_eng,
    pos_label="positive"
)

dt_eng_f1 = f1_score(
    y_test_eng,
    y_pred_dt_eng,
    pos_label="positive"
)

print("English Decision Tree trained successfully.")

print("\nEvaluation Results:")
print(f"Accuracy : {dt_eng_accuracy:.4f}")
print(f"Precision: {dt_eng_precision:.4f}")
print(f"Recall   : {dt_eng_recall:.4f}")
print(f"F1-Score  : {dt_eng_f1:.4f}")

print("\nClassification Report:")
print(classification_report(y_test_eng, y_pred_dt_eng))

print("\nConfusion Matrix:")
print(
    confusion_matrix(
        y_test_eng,
        y_pred_dt_eng,
        labels=["negative", "positive"]
    )
)
```

English Decision Tree trained successfully.

Evaluation Results:
Accuracy : 0.7131
Precision: 0.7126
Recall : 0.7145
F1-Score : 0.7135

Classification Report:				
	precision	recall	f1-score	support
negative	0.71	0.71	0.71	9924
positive	0.71	0.71	0.71	9927
accuracy			0.71	19851
macro avg	0.71	0.71	0.71	19851
weighted avg	0.71	0.71	0.71	19851

Confusion Matrix:
[[7063 2861]
 [2834 7093]]

```
In [55]: # Step 52: Train and Evaluate KNN - English

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# Create KNN model
knn_eng = KNeighborsClassifier(
    n_neighbors=5,
    n_jobs=-1
)

# Train model
knn_eng.fit(X_train_eng_tfidf, y_train_eng)

# Prediction
y_pred_knn_eng = knn_eng.predict(X_test_eng_tfidf)

# Evaluation
knn_eng_accuracy = accuracy_score(y_test_eng, y_pred_knn_eng)

knn_eng_precision = precision_score(
    y_test_eng,
    y_pred_knn_eng,
    pos_label="positive"
)

knn_eng_recall = recall_score(
    y_test_eng,
    y_pred_knn_eng,
    pos_label="positive"
)

knn_eng_f1 = f1_score(
    y_test_eng,
    y_pred_knn_eng,
    pos_label="positive"
)

print("English KNN trained successfully.")

print("\nEvaluation Results:")
print(f"Accuracy : {knn_eng_accuracy:.4f}")
print(f"Precision: {knn_eng_precision:.4f}")
print(f"Recall   : {knn_eng_recall:.4f}")
print(f"F1-Score  : {knn_eng_f1:.4f}")

print("\nClassification Report:")
print(classification_report(y_test_eng, y_pred_knn_eng))

print("\nConfusion Matrix:")
print(
    confusion_matrix(
        y_test_eng,
        y_pred_knn_eng,
        labels=["negative", "positive"]
    )
)
```

English KNN trained successfully.

Evaluation Results:
Accuracy : 0.6011
Precision: 0.5775
Recall : 0.7538
F1-Score : 0.6540

Classification Report:				
	precision	recall	f1-score	support
negative	0.65	0.45	0.53	9924
positive	0.58	0.75	0.65	9927
accuracy			0.60	19851
macro avg	0.61	0.60	0.59	19851
weighted avg	0.61	0.60	0.59	19851

Confusion Matrix:
[[4449 5475]
 [2444 7483]]

In [56]: # Step 53: English Sentiment Analysis - Model Comparison

```
import pandas as pd

english_results = pd.DataFrame({
    "Model": [
        "Logistic Regression",
        "SVM",
        "Naive Bayes",
        "Random Forest",
        "Decision Tree",
        "KNN"
    ],
    "Accuracy": [
        lr_eng_accuracy,
        svm_eng_accuracy,
        nb_eng_accuracy,
        rf_eng_accuracy,
        dt_eng_accuracy,
        knn_eng_accuracy
    ],
    "Precision": [
        lr_eng_precision,
        svm_eng_precision,
        nb_eng_precision,
        rf_eng_precision,
        dt_eng_precision,
        knn_eng_precision
    ],
    "Recall": [
        lr_eng_recall,
        svm_eng_recall,
        nb_eng_recall,
        rf_eng_recall,
        dt_eng_recall,
        knn_eng_recall
    ],
    "F1-Score": [
        lr_eng_f1,
        svm_eng_f1,
        nb_eng_f1,
        rf_eng_f1,
        dt_eng_f1,
        knn_eng_f1
    ]
})

# Sort models by Accuracy
english_results = english_results.sort_values(
    by="Accuracy",
    ascending=False
).reset_index(drop=True)

# Convert results to percentages
english_results_display = english_results.copy()

for column in ["Accuracy", "Precision", "Recall", "F1-Score"]:
    english_results_display[column] = (
        english_results_display[column] * 100
    ).round(2)

print("English Sentiment Analysis - Model Comparison")

display(english_results_display)
```

English Sentiment Analysis - Model Comparison

	Model	Accuracy	Precision	Recall	F1-Score
0	Logistic Regression	77.99	77.47	78.95	78.20
1	SVM	76.62	76.23	77.37	76.80
2	Naive Bayes	76.60	76.98	75.89	76.43
3	Random Forest	76.38	77.17	74.95	76.04
4	Decision Tree	71.31	71.26	71.45	71.35
5	KNN	60.11	57.75	75.38	65.40

In [57]: # Step 54: English Sentiment Analysis - Model Performance Comparison

```
import matplotlib.pyplot as plt
import numpy as np

models = english_results_display["Model"]

accuracy = english_results_display["Accuracy"]
precision = english_results_display["Precision"]
recall = english_results_display["Recall"]
f1 = english_results_display["F1-Score"]

x = np.arange(len(models))
width = 0.20

plt.figure(figsize=(14, 7))

plt.bar(x - 1.5*width, accuracy, width, label="Accuracy")
plt.bar(x - 0.5*width, precision, width, label="Precision")
plt.bar(x + 0.5*width, recall, width, label="Recall")
plt.bar(x + 1.5*width, f1, width, label="F1-Score")

plt.xlabel("Machine Learning Models")
plt.ylabel("Score (%)")

plt.title(
    "English Sentiment Analysis - Model Performance Comparison"
)

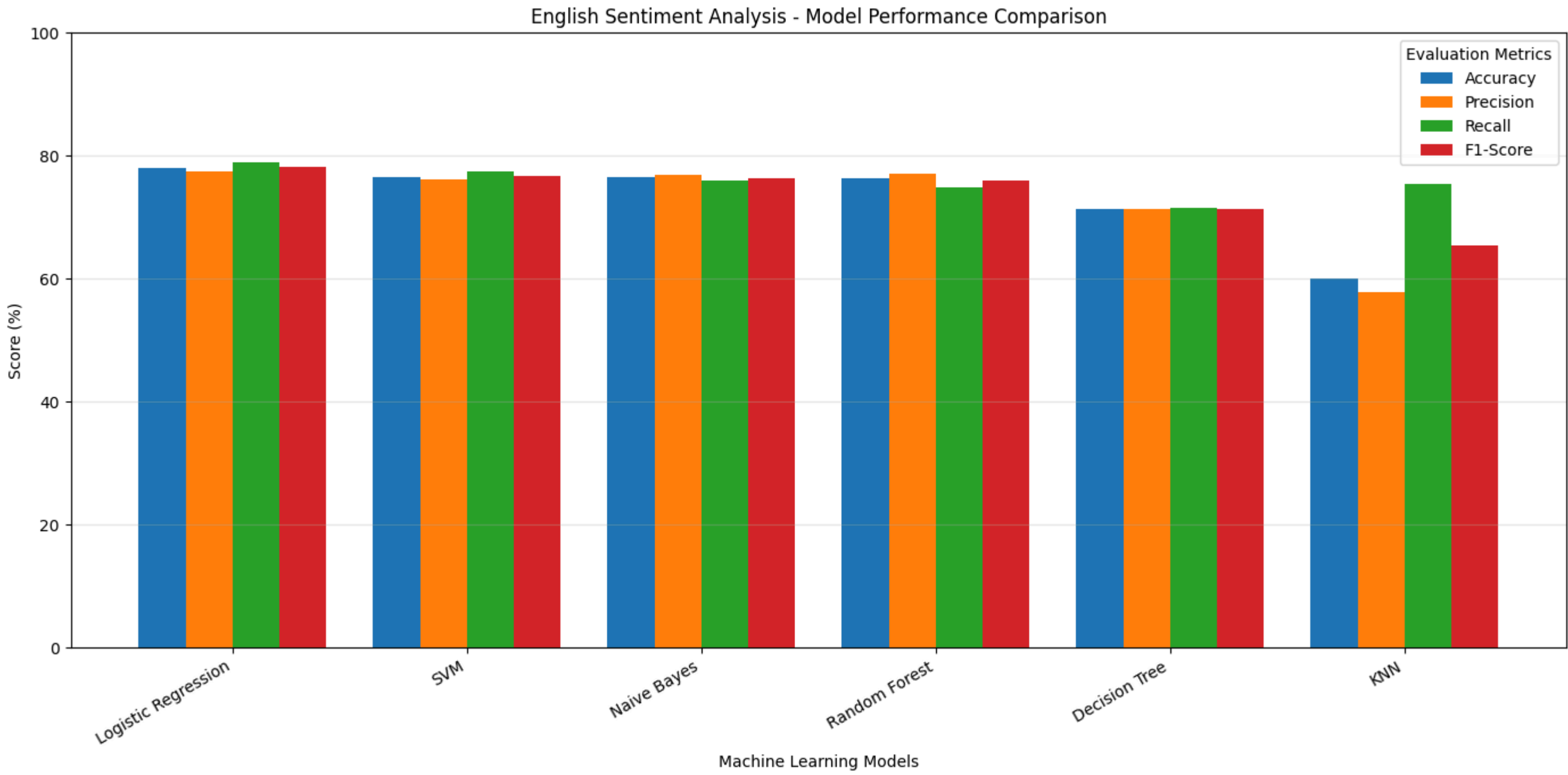
plt.xticks(
    x,
    models,
    rotation=30,
    ha="right"
)

plt.ylim(0, 100)

plt.legend(
    title="Evaluation Metrics"
)

plt.grid(
    axis="y",
    alpha=0.3
)

plt.tight_layout()
plt.show()
```



In [59]: # Step 55: Test Best English Model on New Sentences

```
import re
from nltk.corpus import stopwords

english_stop_words = set(stopwords.words("english"))

# Keep negation words because they are important for sentiment
for word in ["not", "no", "nor"]:
    english_stop_words.discard(word)

def process_english_text(text):

    # Convert to lowercase
    text = str(text).lower()

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

    # Remove @mentions
    text = re.sub(r"@w+", " ", text)

    # Remove special characters and numbers
    text = re.sub(r"[^a-z\s']", " ", text)

    # Remove extra spaces
    text = re.sub(r"\s+", " ", text).strip()

    # Remove stopwords but keep negation
    words = text.split()
    words = [word for word in words if word not in english_stop_words]

    return " ".join(words)

test_sentences_eng = [
    "The service was excellent and I am very happy",
    "I really loved this experience",
    "The service was terrible and I will never use it again",
    "I am not satisfied with this experience"
]

print("English Sentiment Prediction Results:\n")

for original_text in test_sentences_eng:

    processed_text = process_english_text(original_text)

    # TF-IDF
    text_tfidf = english_tfidf.transform([processed_text])

    # Logistic Regression prediction
    prediction = lr_eng.predict(text_tfidf)[0]

    print("Original Text :", original_text)
    print("Processed Text:", processed_text)
    print("Prediction      :", prediction)
    print("-" * 60)
```

```
English Sentiment Prediction Results:

Original Text : The service was excellent and I am very happy
Processed Text: service excellent happy
Prediction      : positive
-----
Original Text : I really loved this experience
Processed Text: really loved experience
Prediction      : positive
-----
Original Text : The service was terrible and I will never use it again
Processed Text: service terrible never use
Prediction      : negative
-----
Original Text : I am not satisfied with this experience
Processed Text: not satisfied experience
Prediction      : positive
-----
```

In [60]: # Step 56: Improve English Negation Handling

```
import re

def handle_english_negation(text):
    """
    Combine negation words with the following word.
    Example:
    not satisfied -> not_satisfied
    not good      -> not_good
    never use      -> never_use
    """

    words = text.split()
    result = []

    negation_words = ["not", "no", "never"]

    i = 0

    while i < len(words):

        if words[i] in negation_words and i + 1 < len(words):

            combined_word = words[i] + "_" + words[i + 1]

            result.append(combined_word)

            i += 2

        else:
            result.append(words[i])
            i += 1

    return " ".join(result)

# Test negation handling
negation_test_sentences = [
    "not satisfied experience",
    "not good service",
    "never use again",
    "excellent service",
    "very happy experience"
]

print("English Negation Handling Test:\n")

for sentence in negation_test_sentences:

    processed = handle_english_negation(sentence)

    print("Before :", sentence)
    print("After  :", processed)
    print("-" * 50)
```

```
English Negation Handling Test:

Before : not satisfied experience
After  : not_satisfied experience
-----
Before : not good service
After  : not_good service
-----
Before : never use again
After  : never_use again
-----
Before : excellent service
After  : excellent service
-----
Before : very happy experience
After  : very happy experience
-----
```

In [61]: # Step 57: Apply Improved Negation Handling to the English Dataset

```
# Create a copy of the English dataset
english_df_improved = english_sample_df.copy()

# Apply negation handling to processed text
english_df_improved["processed_text"] = (
    english_df_improved["processed_text"]
    .apply(handle_english_negation)
)

print("Improved English preprocessing applied successfully.")

print("\nDataset shape:")
print(english_df_improved.shape)

print("\nSentiment distribution:")
print(english_df_improved["sentiment"].value_counts())

print("\nSample after improved negation handling:")

display(
    english_df_improved[
        ["text", "processed_text", "sentiment"]
    ].head(10)
)
```

Improved English preprocessing applied successfully.

Dataset shape:
(99251, 4)

Sentiment distribution:
sentiment
positive 49632
negative 49619
Name: count, dtype: int64

Sample after improved negation handling:

	text	processed_text	sentiment
0	@stargazer60 that's awesome	that's awesome	positive
1	@cunningstunts22 yes... i am!	yes	positive
2	N my bed aaalllll alone	n bed aaalllll alone	negative
3	@Thorney88 re guinea fowl ive heard that lots ...	guinea fowl ive heard lots people dislike say ...	positive
4	def going to the movies tonite	def going movies tonite	positive
5	@MikePaetzold from http://bit.ly/15UN6p may i...	may request free report plz thank	positive
6	Ok my belly is so full now,that apple pie was ...	ok belly full apple pie great addition milksha...	positive
7	@Beans300 what's to be blue about? you're out ...	what's blue fi sunny day	positive
8	@alvinlzg - how true. Eh? How come change twit...	true eh come change twitter display name	positive
9	is happy that today is a good friday, don pani...	happy today good friday panic happy	positive

In [62]: # Step 58: Split Improved English Dataset and Create TF-IDF Features

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

# Features and labels
X_eng_imp = english_df_improved["processed_text"]
y_eng_imp = english_df_improved["sentiment"]

# Train/Test split - 80/20
X_train_eng_imp, X_test_eng_imp, y_train_eng_imp, y_test_eng_imp = train_test_split(
    X_eng_imp,
    y_eng_imp,
    test_size=0.20,
    random_state=42,
    stratify=y_eng_imp
)

# Create new TF-IDF vectorizer
english_tfidf_imp = TfidfVectorizer(
    max_features=20000,
    ngram_range=(1, 2)
)

# Fit only on training data
X_train_eng_imp_tfidf = english_tfidf_imp.fit_transform(X_train_eng_imp)

# Transform testing data
X_test_eng_imp_tfidf = english_tfidf_imp.transform(X_test_eng_imp)

print("Improved English dataset prepared successfully.")

print("\nTraining samples:")
print(X_train_eng_imp_tfidf.shape[0])

print("\nTesting samples:")
print(X_test_eng_imp_tfidf.shape[0])

print("\nTraining TF-IDF shape:")
print(X_train_eng_imp_tfidf.shape)

print("\nTesting TF-IDF shape:")
print(X_test_eng_imp_tfidf.shape)

print("\nNumber of TF-IDF features:")
print(len(english_tfidf_imp.get_feature_names_out()))

print("\nTraining sentiment distribution:")
print(y_train_eng_imp.value_counts())

print("\nTesting sentiment distribution:")
print(y_test_eng_imp.value_counts())
```

```
Improved English dataset prepared successfully.

Training samples:
79400

Testing samples:
19851

Training TF-IDF shape:
(79400, 20000)

Testing TF-IDF shape:
(19851, 20000)

Number of TF-IDF features:
20000

Training sentiment distribution:
sentiment
positive    39705
negative    39695
Name: count, dtype: int64

Testing sentiment distribution:
sentiment
positive    9927
negative    9924
Name: count, dtype: int64
```

In [63]: # Step 59: Train Improved English Logistic Regression Model

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

# Train Logistic Regression
lr_eng_imp = LogisticRegression(
    max_iter=1000,
    random_state=42
)

lr_eng_imp.fit(
    X_train_eng_imp_tfidf,
    y_train_eng_imp
)

# Predictions
y_pred_lr_eng_imp = lr_eng_imp.predict(
    X_test_eng_imp_tfidf
)

# Evaluation
lr_imp_accuracy = accuracy_score(
    y_test_eng_imp,
    y_pred_lr_eng_imp
)

lr_imp_precision = precision_score(
    y_test_eng_imp,
    y_pred_lr_eng_imp,
    pos_label="positive"
)

lr_imp_recall = recall_score(
    y_test_eng_imp,
    y_pred_lr_eng_imp,
    pos_label="positive"
)

lr_imp_f1 = f1_score(
    y_test_eng_imp,
    y_pred_lr_eng_imp,
    pos_label="positive"
)

print("Improved English Logistic Regression trained successfully.")

print("\nEvaluation Results:")
print(f"Accuracy : {lr_imp_accuracy:.4f}")
print(f"Precision: {lr_imp_precision:.4f}")
print(f"Recall   : {lr_imp_recall:.4f}")
print(f"F1-Score  : {lr_imp_f1:.4f}")

print("\nClassification Report:")
print(
    classification_report(
        y_test_eng_imp,
        y_pred_lr_eng_imp,
        labels=["negative", "positive"]
    )
)

print("Confusion Matrix:")
print(
    confusion_matrix(
        y_test_eng_imp,
        y_pred_lr_eng_imp,
        labels=["negative", "positive"]
    )
)

print("\nOld Logistic Regression Accuracy: 77.99%")
print(
    f"Improved Logistic Regression Accuracy: "
    f"{lr_imp_accuracy * 100:.2f}%"
)
```

Improved English Logistic Regression trained successfully.

Evaluation Results:
Accuracy : 0.7770
Precision: 0.7699
Recall : 0.7902
F1-Score : 0.7799

	precision	recall	f1-score	support
negative	0.78	0.76	0.77	9924
positive	0.77	0.79	0.78	9927
accuracy			0.78	19851
macro avg	0.78	0.78	0.78	19851
weighted avg	0.78	0.78	0.78	19851

Confusion Matrix:
[[7580 2344]
 [2083 7844]]

Old Logistic Regression Accuracy: 77.99%
Improved Logistic Regression Accuracy: 77.70%

In [64]: # Step 60: Test Improved English Model

```
test_sentences_imp = [
    "The service was excellent and I am very happy",
    "I really loved this experience",
    "The service was terrible and I will never use it again",
    "I am not satisfied with this experience",
    "This product is not good",
    "I will never buy this product again"
]

print("Improved English Sentiment Prediction Results:\n")

for original_text in test_sentences_imp:

    # Apply original preprocessing
    processed_text = process_english_text(original_text)

    # Apply improved negation handling
    processed_text = handle_english_negation(processed_text)

    # Transform using improved TF-IDF
    text_tfidf = english_tfidf_imp.transform([processed_text])

    # Predict using improved Logistic Regression
    prediction = lr_eng_imp.predict(text_tfidf)[0]

    print("Original Text :", original_text)
    print("Processed Text:", processed_text)
    print("Prediction   :", prediction)
    print("-" * 60)
```

Improved English Sentiment Prediction Results:

Original Text : The service was excellent and I am very happy
Processed Text: service excellent happy
Prediction : positive

Original Text : I really loved this experience
Processed Text: really loved experience
Prediction : positive

Original Text : The service was terrible and I will never use it again
Processed Text: service terrible never_use
Prediction : negative

Original Text : I am not satisfied with this experience
Processed Text: not_satisfied experience
Prediction : positive

Original Text : This product is not good
Processed Text: product not_good
Prediction : negative

Original Text : I will never buy this product again
Processed Text: never_buy product
Prediction : positive

```
In [65]: # Step 61: Save Final Best English Model and TF-IDF Vectorizer

import os
import joblib

# Models folder
models_path = "/content/drive/MyDrive/Arabic_English_Sentiment_Analysis/Models"

os.makedirs(models_path, exist_ok=True)

# File paths
english_model_path = os.path.join(
    models_path,
    "english_logistic_regression_final_model.pkl"
)

english_tfidf_path = os.path.join(
    models_path,
    "english_tfidf_final_vectorizer.pkl"
)

# Save ORIGINAL Logistic Regression model
# Best English model: Accuracy = 77.99%
joblib.dump(lr_eng, english_model_path)

# Save ORIGINAL English TF-IDF vectorizer
joblib.dump(english_tfidf, english_tfidf_path)

print("Final English model saved successfully.")
print("Model:", english_model_path)

print("\nFinal English TF-IDF vectorizer saved successfully.")
print("Vectorizer:", english_tfidf_path)

print("\nFinal English Model Information:")
print("Model    : Logistic Regression")
print("Accuracy  : 77.99%")
print("Precision: 77.47%")
print("Recall    : 78.95%")
print("F1-Score  : 78.20%")
```

```
Final English model saved successfully.
Model: /content/drive/MyDrive/Arabic_English_Sentiment_Analysis/Models/english_logistic_regression_final_model.pkl

Final English TF-IDF vectorizer saved successfully.
Vectorizer: /content/drive/MyDrive/Arabic_English_Sentiment_Analysis/Models/english_tfidf_final_vectorizer.pkl

Final English Model Information:
Model    : Logistic Regression
Accuracy : 77.99%
Precision: 77.47%
Recall   : 78.95%
F1-Score : 78.20%
```

```
In [66]: # Step 62: Build a Unified Arabic-English Sentiment Prediction Function
```

```
import re
import joblib

# Load final saved models
arabic_model = joblib.load(
    "/content/drive/MyDrive/Arabic_English_Sentiment_Analysis/Models/arabic_svm_final_model.pkl"
)

arabic_tfidf = joblib.load(
    "/content/drive/MyDrive/Arabic_English_Sentiment_Analysis/Models/arabic_tfidf_final_vectorizer.pkl"
)

english_model = joblib.load(
    "/content/drive/MyDrive/Arabic_English_Sentiment_Analysis/Models/english_logistic_regression_final_model.pkl"
)

english_tfidf = joblib.load(
    "/content/drive/MyDrive/Arabic_English_Sentiment_Analysis/Models/english_tfidf_final_vectorizer.pkl"
)

# Detect language based on Arabic characters
def detect_language(text):

    arabic_chars = re.findall(r'[\u0600-\u06FF]', text)
    english_chars = re.findall(r'[A-Za-z]', text)

    if len(arabic_chars) > len(english_chars):
        return "Arabic"
    else:
        return "English"

# Unified prediction function
def predict_sentiment(text):

    language = detect_language(text)

    # Arabic
    if language == "Arabic":

        processed_text = improved_arabic_preprocessing(text)

        features = arabic_tfidf.transform(
            [processed_text]
        )

        prediction = arabic_model.predict(features)[0]

    # English
    else:

        processed_text = process_english_text(text)

        features = english_tfidf.transform(
            [processed_text]
        )

        prediction = english_model.predict(features)[0]

    return {
        "Language": language,
        "Original Text": text,
        "Processed Text": processed_text,
        "Sentiment": prediction
    }
```

```
print("Unified Arabic-English sentiment prediction system created successfully.")
```

```
Unified Arabic-English sentiment prediction system created successfully.
```

In [67]: # Step 63: Test Unified Arabic-English Sentiment System

```
test_sentences = [
    # Arabic Positive
    "الخدمة ممتازة وأنا سعيد جداً",

    # Arabic Negative
    "الخدمة سيئة جداً وإن استخدمتها مرة أخرى",

    # Arabic Negative with negation
    "أنا غير راضي عن هذه التجربة",

    # English Positive
    "The service was excellent and I am very happy",

    # English Negative
    "The service was terrible and I will never use it again",

    # English Positive
    "I really loved this experience"
]
```

print("Unified Arabic-English Sentiment Analysis Results:\n")

for text in test_sentences:

result = predict_sentiment(text)

```
print("Language      :", result["Language"])
print("Original Text  :", result["Original Text"])
print("Processed Text  :", result["Processed Text"])
print("Sentiment       :", result["Sentiment"])
print("-" * 70)
```

Unified Arabic-English Sentiment Analysis Results:

Language	: Arabic
Original Text	: الخدمة ممتازة وأنا سعيد جداً
Processed Text	: الخدمة ممتازة وأنا سعيد جداً
Sentiment	: positive

Language	: Arabic
Original Text	: الخدمة سيئة جداً وإن استخدمتها مرة أخرى
Processed Text	: الخدمة سيئة جداً إن استخدمتها مرة أخرى
Sentiment	: positive

Language	: Arabic
Original Text	: أنا غير راضي عن هذه التجربة
Processed Text	: أنا غير راضي التجربة
Sentiment	: negative

Language	: English
Original Text	: The service was excellent and I am very happy
Processed Text	: service excellent happy
Sentiment	: positive

Language	: English
Original Text	: The service was terrible and I will never use it again
Processed Text	: service terrible never use
Sentiment	: negative

Language	: English
Original Text	: I really loved this experience
Processed Text	: really loved experience
Sentiment	: positive

In [68]: # Step 64: Improve Arabic Negation Handling

```
def handle_arabic_negation(text):
    """
    Combine Arabic negation words with the following word.

    Examples:
    غير راضي -> غير_راضي
    لن استخدم -> لن_استخدم
    لا احب -> لا_احب
    ليس جيد -> ليس_جيد
    """

    words = text.split()
    result = []

    negation_words = {
        "و",
        "ان",
        "له",
        "ليس",
        "ليست",
        "غير",
        "وما",
        "مش"
    }

    i = 0

    while i < len(words):

        if words[i] in negation_words and i + 1 < len(words):

            combined_word = words[i] + "_" + words[i + 1]

            result.append(combined_word)

            i += 2

        else:
            result.append(words[i])

            i += 1

    return " ".join(result)

# Test Arabic negation handling
arabic_negation_test = [
    "أنا غير راضي عن التجربة",
    "الخدمة سيئة جداً إن استخدمتها مرة أخرى",
    "أنا لا احب هذا المنتج",
    "هذا المنتج ليس جيد",
    "التجربة ممتازة جداً"
]

print("Arabic Negation Handling Test:\n")

for sentence in arabic_negation_test:

    # Apply existing Arabic preprocessing first
    processed = improved_arabic_preprocessing(sentence)

    # Apply improved negation handling
    processed_with_negation = handle_arabic_negation(processed)

    print("Original :", sentence)
    print("Before  :", processed)
    print("After   :", processed_with_negation)
    print("-" * 60)
```

Arabic Negation Handling Test:

Original	: أنا غير راضي عن التجربة
Before	: أنا غير راضي التجربة
After	: أنا غير_راضي التجربة

Original	: الخدمة سيئة جداً لن استخدمها مرة أخرى
Before	: الخدمة سيئة جداً لن استخدمها مرة أخرى
After	: الخدمة سيئة جداً لن_استخدمها مرة أخرى

Original	: أنا لا احب هذا المنتج
Before	: أنا لا احب المنتج
After	: أنا لا_احب المنتج

Original	: هذا المنتج ليس جيد
Before	: المنتج ليس جيد
After	: المنتج_ليس_جيد

Original	: التجربة ممتازة جداً
Before	: التجربة ممتازة جداً
After	: التجربة ممتازة جداً

```
In [69]: # =====
# Step 65: Apply Improved Arabic Negation to Full Dataset
#         + Train/Test Split + TF-IDF
# =====

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

print("Step 65: Preparing Arabic dataset with improved negation...\n")

# -----
# 1. Create a copy of the Arabic dataset
# -----

arabic_df_final = arabic_df.copy()

# -----
# 2. Apply Arabic preprocessing + negation handling
# -----

def final_arabic_preprocessing(text):

    # Apply our previous Arabic preprocessing
    processed = improved_arabic_preprocessing(text)

    # Preserve negation meaning
    processed = handle_arabic_negation(processed)

    return processed

arabic_df_final["processed_text"] = (
    arabic_df_final["text"]
    .astype(str)
    .apply(final_arabic_preprocessing)
)

# -----
# 3. Remove empty processed texts
# -----

arabic_df_final = arabic_df_final[
    arabic_df_final["processed_text"].str.strip() != ""
].copy()

arabic_df_final.reset_index(drop=True, inplace=True)

print("Arabic dataset prepared successfully.")

print("\nDataset shape:")
print(arabic_df_final.shape)

print("\nSentiment distribution:")
print(arabic_df_final["sentiment"].value_counts())

# -----
# 4. Train / Test Split
# -----

X_train_ar_final, X_test_ar_final, y_train_ar_final, y_test_ar_final = (
    train_test_split(
        arabic_df_final["processed_text"],
        arabic_df_final["sentiment"],
        test_size=0.20,
        random_state=42,
        stratify=arabic_df_final["sentiment"]
    )
)

print("\nTraining samples:")
print(len(X_train_ar_final))

print("\nTesting samples:")
print(len(X_test_ar_final))

# -----
# 5. Create New Arabic TF-IDF
# -----

arabic_tfidf_final = TfidfVectorizer(
    max_features=20000,
    ngram_range=(1, 2),
    sublinear_tf=True
)

X_train_ar_tfidf_final = arabic_tfidf_final.fit_transform(
    X_train_ar_final
)

X_test_ar_tfidf_final = arabic_tfidf_final.transform(
    X_test_ar_final
)

print("\nTraining TF-IDF shape:")
print(X_train_ar_tfidf_final.shape)

print("\nTesting TF-IDF shape:")
print(X_test_ar_tfidf_final.shape)

print("\nNumber of TF-IDF features:")
print(len(arabic_tfidf_final.get_feature_names_out()))

# -----
# 6. Show Negation Examples
# -----

print("\nNegation examples:")

examples = [
    "انا غير راضى عن التجربة",
    "الخدمة سيئة جدا ولن استخدمها مرة اخرى",
    "انا لا احب هذا المنتج",
    "هذا المنتج ليس جيد",
    "الخدمة ممتازة جدا"
]

for text in examples:

    processed = final_arabic_preprocessing(text)

    print("\nOriginal :", text)
    print("Processed:", processed)

print("\nStep 65 completed successfully.")
```

Step 65: Preparing Arabic dataset with improved negation...

Arabic dataset prepared successfully.

Dataset shape:
(1968, 4)

Sentiment distribution:
sentiment
positive 994
negative 974
Name: count, dtype: int64

Training samples:
1574

Testing samples:
394

Training TF-IDF shape:
(1574, 14098)

Testing TF-IDF shape:
(394, 14098)

Number of TF-IDF features:
14098

Negation examples:

Original : انا غير راضى عن التجربة
Processed: انا غير راضى التجربة

Original : الخدمة سيئة جدا ولن استخدمها مرة اخرى

Processed: الخدمة سيئة جدا لنـ استخدمها مرة اخرى

Original : انا لا احب هذا المنتج
Processed: انا لا احب المنتج

Original : هذا المنتج ليس جيد
Processed: المنتج ليس جيد

Original : الخدمة ممتازة جدا
Processed: الخدمة ممتازة جدا

Step 65 completed successfully.

```
In [70]: # =====
# Step 66: Train Final Improved Arabic SVM
# =====

from sklearn.svm import LinearSVC
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix
)

print("Step 66: Training Improved Arabic SVM...\n")

# -----
# 1. Train SVM
# -----

arabic_svm_final = LinearSVC(
    C=1.0,
    random_state=42
)

arabic_svm_final.fit(
    X_train_ar_tfidf_final,
    y_train_ar_final
)

# -----
# 2. Predictions
# -----

y_pred_ar_final = arabic_svm_final.predict(
    X_test_ar_tfidf_final
)

# -----
# 3. Evaluation
# -----

ar_final_accuracy = accuracy_score(
    y_test_ar_final,
    y_pred_ar_final
)

ar_final_precision = precision_score(
    y_test_ar_final,
    y_pred_ar_final,
    pos_label="positive"
)

ar_final_recall = recall_score(
    y_test_ar_final,
    y_pred_ar_final,
    pos_label="positive"
)

ar_final_f1 = f1_score(
    y_test_ar_final,
    y_pred_ar_final,
    pos_label="positive"
)

print("Improved Arabic SVM trained successfully.\n")

print("Evaluation Results:")
print(f"Accuracy : {ar_final_accuracy:.4f}")
print(f"Precision : {ar_final_precision:.4f}")
print(f"Recall : {ar_final_recall:.4f}")
print(f"F1-Score : {ar_final_f1:.4f}")

# -----
# 4. Classification Report
# -----

print("\nClassification Report:")

print(
    classification_report(
        y_test_ar_final,
        y_pred_ar_final,
        labels=["negative", "positive"]
    )
)

# -----
# 5. Confusion Matrix
# -----

print("Confusion Matrix:")

print(
    confusion_matrix(
        y_test_ar_final,
        y_pred_ar_final,
        labels=["negative", "positive"]
    )
)

# -----
# 6. Compare with Previous Best Arabic Model
# -----

old_accuracy = 0.8553

print("\nPrevious Arabic SVM Accuracy : 85.53%")

print(
    f"New Arabic SVM Accuracy : "
    f"{ar_final_accuracy * 100:.2f}%"
)

difference = (
    ar_final_accuracy - old_accuracy
) * 100

print(
    f"Accuracy Difference : "
    f"{difference:+.2f}%"
)

print("\nStep 66 completed successfully.")
```

Step 66: Training Improved Arabic SVM...
Improved Arabic SVM trained successfully.

Evaluation Results:
Accuracy : 0.8579
Precision : 0.8557
Recall : 0.8643
F1-Score : 0.8600

Classification Report:

	precision	recall	f1-score	support
negative	0.86	0.85	0.86	195
positive	0.86	0.86	0.86	199
accuracy			0.86	394
macro avg	0.86	0.86	0.86	394
weighted avg	0.86	0.86	0.86	394

Confusion Matrix:
[[166 29]
[27 172]]

Previous Arabic SVM Accuracy : 85.53%
New Arabic SVM Accuracy : 85.79%
Accuracy Difference : +0.26%

Step 66 completed successfully.

```
In [71]: # =====
# Step 67: Save Final Improved Arabic Model
# =====

import os
import joblib

print("Step 67: Saving Final Arabic Model...\n")

# Models folder
models_path = (
    "/content/drive/MyDrive/"
    "Arabic_English_Sentiment_Analysis/Models"
)

os.makedirs(models_path, exist_ok=True)

# -----
# File paths
# -----

arabic_model_final_path = os.path.join(
    models_path,
    "arabic_svm_final_model.pkl"
)

arabic_tfidf_final_path = os.path.join(
    models_path,
    "arabic_tfidf_final.pkl"
)

# -----
# Save model
# -----

joblib.dump(
    arabic_svm_final,
    arabic_model_final_path
)

# -----
# Save TF-IDF
# -----

joblib.dump(
    arabic_tfidf_final,
    arabic_tfidf_final_path
)

# -----
# Verify files
# -----

print("Final Arabic SVM model saved successfully.")
print("Model:")
print(arabic_model_final_path)

print("\nFinal Arabic TF-IDF vectorizer saved successfully.")
print("Vectorizer:")
print(arabic_tfidf_final_path)

print("\nFinal Arabic Model Information:")
print("Model      : Improved SVM")
print(f"Accuracy   : {ar_final_accuracy * 100:.2f}%")
print(f"Precision  : {ar_final_precision * 100:.2f}%")
print(f"Recall     : {ar_final_recall * 100:.2f}%")
print(f"F1-Score   : {ar_final_f1 * 100:.2f}%")

print("\nStep 67 completed successfully.")
```

```
Step 67: Saving Final Arabic Model...

Final Arabic SVM model saved successfully.
Model:
/content/drive/MyDrive/Arabic_English_Sentiment_Analysis/Models/arabic_svm_final_model.pkl

Final Arabic TF-IDF vectorizer saved successfully.
Vectorizer:
/content/drive/MyDrive/Arabic_English_Sentiment_Analysis/Models/arabic_tfidf_final.pkl

Final Arabic Model Information:
Model      : Improved SVM
Accuracy   : 85.79%
Precision  : 85.57%
Recall     : 86.43%
F1-Score   : 86.00%

Step 67 completed successfully.
```

In [72]:

```
# =====
# Step 68: Update and Test the Final Unified Arabic-English System
# =====

import re
import joblib

print("Step 68: Loading final Arabic and English models...\n")

# -----
# 1. Load Final Arabic Model and TF-IDF
# -----

final_arabic_model = joblib.load(
    "/content/drive/MyDrive/Arabic_English_Sentiment_Analysis/"
    "Models/arabic_svm_final_model.pkl"
)

final_arabic_tfidf = joblib.load(
    "/content/drive/MyDrive/Arabic_English_Sentiment_Analysis/"
    "Models/arabic_tfidf_final.pkl"
)

# -----
# 2. Load Final English Model and TF-IDF
# -----

final_english_model = joblib.load(
    "/content/drive/MyDrive/Arabic_English_Sentiment_Analysis/"
    "Models/english_logistic_regression_final_model.pkl"
)

final_english_tfidf = joblib.load(
    "/content/drive/MyDrive/Arabic_English_Sentiment_Analysis/"
    "Models/english_tfidf_final_vectorizer.pkl"
)

# -----
# 3. Automatic Language Detection
# -----

def detect_final_language(text):

    arabic_chars = len(
        re.findall(r"[\u0600-\u06FF]", text)
    )

    english_chars = len(
        re.findall(r"[A-Za-z]", text)
    )

    if arabic_chars > english_chars:
        return "Arabic"

    return "English"

# -----
# 4. Final Unified Prediction Function
# -----

def final_predict_sentiment(text):

    language = detect_final_language(text)

    # Arabic prediction
    if language == "Arabic":

        processed_text = final_arabic_preprocessing(text)

        features = final_arabic_tfidf.transform(
            [processed_text]
        )

        prediction = final_arabic_model.predict(
            features
        )[0]

    # English prediction
    else:

        processed_text = process_english_text(text)

        features = final_english_tfidf.transform(
            [processed_text]
        )

        prediction = final_english_model.predict(
            features
        )[0]

    return {
        "Language": language,
        "Original Text": text,
        "Processed Text": processed_text,
        "Sentiment": prediction
    }

# -----
# 5. Final Test Sentences
# -----

final_test_sentences = [

    # Arabic Positive
    "الخدمة ممتازة وأنا سعيد جدا",

    # Arabic Negative
    "الخدمة سيئة جدا وإن استخدمها مرة أخرى",

    # Arabic Negative with Negation
    "أنا غير راضي عن هذه التجربة",

    # Arabic Negative
    "أنا لا أحب هذا المنتج",

    # English Positive
    "The service was excellent and I am very happy",

    # English Negative
    "The service was terrible and I will never use it again",

    # English Positive
    "I really loved this experience",

    # English Negative
    "This product is not good"
]

print("Final Unified Arabic-English Sentiment Results:\n")

for text in final_test_sentences:

    result = final_predict_sentiment(text)

    print("Language      :", result["Language"])
    print("Original Text  :", result["Original Text"])
    print("Processed Text :", result["Processed Text"])
    print("Sentiment      :", result["Sentiment"])

    print("-" * 70)

print("\nStep 68 completed successfully.")
```

```
Step 68: Loading final Arabic and English models...

Final Unified Arabic-English Sentiment Results:

Language      : Arabic
Original Text  : الخدمة ممتازة وأنا سعيد جدا
Processed Text : الخدمة ممتازة وأنا سعيد جدا
Sentiment      : positive
-----
Language      : Arabic
Original Text  : الخدمة سيئة جدا وإن استخدمها مرة أخرى
Processed Text : الخدمة سيئة جدا وإن استخدمها مرة أخرى
Sentiment      : positive
-----
Language      : Arabic
Original Text  : أنا غير راضي عن هذه التجربة
Processed Text : أنا غير راضي التجربة
Sentiment      : negative
-----
Language      : Arabic
```

```
Original Text : أنا لا أحب هذا المنتج
Processed Text : أنا لا أحب المنتج
Sentiment : negative
-----
Language : English
Original Text : The service was excellent and I am very happy
Processed Text : service excellent happy
Sentiment : positive
-----
Language : English
Original Text : The service was terrible and I will never use it again
Processed Text : service terrible never use
Sentiment : negative
-----
Language : English
Original Text : I really loved this experience
Processed Text : really loved experience
Sentiment : positive
-----
Language : English
Original Text : This product is not good
Processed Text : product not good
Sentiment : negative
-----

Step 68 completed successfully.
```

```
print("\nStep 69 completed successfully.")
```

about:blank

```
Language : English
Processed : service terrible never use
Sentiment : negative
-----

Step 69 completed successfully.
```

```
In [74]: # =====
# Step 70: Build Gradio Interface for Arabic-English Sentiment Analysis
# =====

!pip install -q gradio

import gradio as gr

def gradio_sentiment_analysis(text):

    if not text or text.strip() == "":
        return "Please enter a text.", "", ""

    result = final_predict_sentiment_v2(text)

    language = result["Language"]
    sentiment = result["Sentiment"]
    processed_text = result["Processed Text"]

    if sentiment == "positive":
        sentiment_display = "Positive 😊"
    else:
        sentiment_display = "Negative 😞"

    return language, sentiment_display, processed_text

with gr.Blocks() as demo:

    gr.Markdown(
        """
        # Arabic & English Sentiment Analysis
        Enter Arabic or English text and the system will automatically
        detect the language and classify the sentiment as Positive or Negative.
        """
    )

    text_input = gr.Textbox(
        label="Enter Text / أدخل النص",
        placeholder="اكتب جملة بالعربي أو الإنجليزي",
        lines=4
    )

    analyze_button = gr.Button(
        "Analyze Sentiment"
    )

    language_output = gr.Textbox(
        label="Detected Language"
    )

    sentiment_output = gr.Textbox(
        label="Sentiment"
    )

    processed_output = gr.Textbox(
        label="Processed Text"
    )

    analyze_button.click(
        fn=gradio_sentiment_analysis,
        inputs=text_input,
        outputs=[
            language_output,
            sentiment_output,
            processed_output
        ]
    )

demo.launch(share=True)
```

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: <https://c333f10156f8419715.gradio.live>

This share link is temporary and will last for up to 1 week (best effort). For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (<https://huggingface.co/spaces>)



Loading...

In [75]: # Step 71: Final Testing for the Unified Arabic-English Sentiment System

```
final_validation_sentences = [
    # Arabic Positive
    "الخدمة ممتازة جدا",
    "التجربة رائعة وأنا سعيد",
    "المنتج جميل والنصح به",

    # Arabic Negative
    "الخدمة سيئة جدا",
    "أنا غير راضي عن هذه التجربة",
    "لن أشتري هذا المنتج مرة أخرى",

    # English Positive
    "This service is excellent",
    "I really love this product",
    "The experience was amazing",

    # English Negative
    "This service is terrible",
    "I am not happy with this product",
    "I will never buy this again"
]

print("Final System Validation Results:\n")

for text in final_validation_sentences:
    result = final_predict_sentiment_v2(text)

    print("Language :", result["Language"])
    print("Text      :", result["Original Text"])
    print("Processed :", result["Processed Text"])
    print("Sentiment :", result["Sentiment"])
    print("-" * 70)

print("\nStep 71 completed successfully.")
```

```
Final System Validation Results:

Language : Arabic
Text      : الخدمة ممتازة جدا
Processed : الخدمة ممتازة جدا
Sentiment : positive
-----
Language : Arabic
Text      : التجربة رائعة وأنا سعيد
Processed : التجربة رائعة وأنا سعيد
Sentiment : positive
-----
Language : Arabic
Text      : المنتج جميل والنصح به
Processed : المنتج جميل والنصح
Sentiment : positive
-----
Language : Arabic
Text      : الخدمة سيئة جدا
Processed : الخدمة سيئة جدا
Sentiment : negative
-----
Language : Arabic
Text      : أنا غير راضي عن هذه التجربة
Processed : أنا غير راضي التجربة
Sentiment : negative
-----
Language : Arabic
Text      : لن أشتري هذا المنتج مرة أخرى
Processed : لن أشتري المنتج مرة اخري
Sentiment : negative
-----
Language : English
Text      : This service is excellent
Processed : service excellent
Sentiment : positive
-----
Language : English
Text      : I really love this product
Processed : really love product
Sentiment : positive
-----
Language : English
Text      : The experience was amazing
Processed : experience amazing
Sentiment : positive
-----
Language : English
Text      : This service is terrible
Processed : service terrible
Sentiment : negative
-----
Language : English
Text      : I am not happy with this product
Processed : not happy product
Sentiment : negative
-----
Language : English
Text      : I will never buy this again
Processed : never buy
Sentiment : negative
-----

Step 71 completed successfully.
```

In [76]: # Step 72: Create and Save Final Project Results Summary

```
import pandas as pd
import os

print("Step 72: Creating Final Project Summary...\n")

# Final results
final_results = pd.DataFrame({
    "Language": ["Arabic", "English"],
    "Final Model": ["Improved SVM", "Logistic Regression"],
    "Accuracy (%)": [85.79, 77.99],
    "Precision (%)": [85.57, 77.47],
    "Recall (%)": [86.43, 78.95],
    "F1-Score (%)": [86.00, 78.20]
})

print("Final Arabic-English Sentiment Analysis Results:")
display(final_results)

# Save results
results_path = (
    "/content/drive/MyDrive/"
    "Arabic_English_Sentiment_Analysis/"
    "final_project_results.csv"
)

final_results.to_csv(
    results_path,
    index=False,
    encoding="utf-8-sig"
)

print("\nFinal results saved successfully.")
print("File:", results_path)

print("\nProject Summary:")
print("-" * 60)
print("Arabic Model : Improved SVM")
print("Arabic Accuracy: 85.79%")
print("English Model : Logistic Regression")
print("English Accuracy: 77.99%")
print("-" * 60)

print("\nArabic & English Sentiment Analysis Project")
print("COMPLETED SUCCESSFULLY ✅")
print("\nStep 72 completed successfully.")
```

Step 72: Creating Final Project Summary...

Final Arabic-English Sentiment Analysis Results:

	Language	Final Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
0	Arabic	Improved SVM	85.79	85.57	86.43	86.0
1	English	Logistic Regression	77.99	77.47	78.95	78.2

Final results saved successfully.
File: /content/drive/MyDrive/Arabic_English_Sentiment_Analysis/final_project_results.csv

Project Summary:

Arabic Model : Improved SVM
Arabic Accuracy: 85.79%
English Model : Logistic Regression
English Accuracy: 77.99%

Arabic & English Sentiment Analysis Project
COMPLETED SUCCESSFULLY ✅

Step 72 completed successfully.