

مشروع تصنيف النصوص باستخدام تقنيات (NLP) معالجة اللغة الطبيعية

1. استيراد المكتبات (Import Libraries)

في هذه المرحلة يتم تجهيز بيئة العمل الخاصة بالمشروع من خلال استيراد المكتبات اللازمة لمعالجة البيانات النصية وبناء نماذج التعلم الآلي. تشمل هذه المكتبات أدوات لمعالجة اللغة الطبيعية وتقنيات تحويل النصوص إلى بيانات رقمية، بالإضافة إلى خوارزميات متعددة لتصنيف (NLP)، النصوص وتقييم أدائها.

```
In [2]: import pandas as pd
import numpy as np
import re
import nltk

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, classification_report, confusion_mat

import matplotlib.pyplot as plt
import seaborn as sns

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

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

Out[2]: True

تحميل البيانات وتحليل توزيع الفئات

ثم إجراء تحليل أولي لتوزيع Pandas في هذه المرحلة تم تحميل مجموعة البيانات باستخدام مكتبة يساعد هذا التحليل في اكتشاف ما إذا كانت البيانات متوازنة أو غير (Class Distribution). الفئات متوازنة، وهو أمر مهم لأنه يؤثر بشكل مباشر على أداء نماذج التصنيف ويحدد الحاجة لاستخدام تقنيات مثل معالجة عدم التوازن.

```
In [4]: df = pd.read_csv("unbalanceddataset.csv")

print("Class Distribution:")
print(df['sentiment'].value_counts())
```

```
Class Distribution:
sentiment
positive    2000
negative     600
Name: count, dtype: int64
```

معالجة النصوص (Text Preprocessing and Cleaning)

لتنظيف النصوص (NLP) في هذه المرحلة تم تطبيق مجموعة من تقنيات معالجة اللغة الطبيعية وتحضيرها للتحليل. شملت هذه الخطوات:

والرموز الخاصة HTML إزالة الروابط وعلامات (Lowercasing) تحويل النص إلى أحرف صغيرة
Stemming تطبيق (Stopwords) حذف الكلمات الشائعة (Tokenization) تقسيم النص إلى كلمات
لتقليل الكلمات إلى جذورها

تهدف هذه العمليات إلى تقليل الضوضاء في البيانات وتحسين جودة التمثيل النصي، مما ينعكس إيجابيًا على أداء نماذج التصنيف.

```
In [5]: stemmer = PorterStemmer()
stop_words = set(stopwords.words('english'))

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

    tokens = word_tokenize(text)
    tokens = [word for word in tokens if word not in stop_words]
    tokens = [stemmer.stem(word) for word in tokens]

    return ' '.join(tokens)

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

مجموعة اختبار (Data Splitting Strategy) تقسيم البيانات إلى تدريب واختبار

ومجموعة اختبار (Training Set) في هذه المرحلة تم تقسيم البيانات إلى مجموعتين: مجموعة تدريب حيث تم تخصيص 70% من البيانات للتدريب و30% للاختبار. كما تم استخدام خاصية (Testing Set) لضمان الحفاظ على نفس توزيع الفئات في كلا المجموعتين، مما يساعد في تقييم النموذج بدقة وبشكل عادل ودقيق.

```
In [6]: X = df['clean_text']
y = df['sentiment']

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3,
    random_state=42,
```

```
stratify=y  
)
```

n-grams و TF-IDF تمثيل النصوص باستخدام

TF-IDF (Term Frequency–Inverse Document Frequency) في هذه المرحلة تم تحويل النصوص إلى تمثيل رقمي باستخدام تقنية n-grams والتي تقيس أهمية الكلمات داخل النصوص. كما تم استخدام n-grams (Unigrams + Bigrams) مما يساعد النموذج على فهم السياق بشكل أفضل وتحسين دقة التصنيف.

```
In [7]: vectorizer = TfidfVectorizer(ngram_range=(1,2))  
  
X_train_vec = vectorizer.fit_transform(X_train)  
X_test_vec = vectorizer.transform(X_test)
```

(Model Evaluation Framework) تقييم أداء النماذج

في هذه المرحلة تم إنشاء دالة مخصصة لتقييم أداء نماذج التصنيف بشكل موحد. تعتمد عملية التقييم على عدة مقاييس مهمة مثل:

لتحليل الأداء بشكل تفصيلي لكل فئة F1-score و Recall و Precision لقياس الدقة العامة Accuracy لعرض الأخطاء والتنبؤات بشكل بصري Confusion Matrix

يساعد هذا الإطار في مقارنة أداء النماذج المختلفة واختيار النموذج الأفضل بناءً على نتائج دقيقة وشاملة.

```
In [8]: def evaluate(model, name):  
    y_pred = model.predict(X_test_vec)  
  
    print(f"\n==== {name} =====")  
    print("Accuracy:", accuracy_score(y_test, y_pred))  
    print(classification_report(y_test, y_pred))  
  
    cm = confusion_matrix(y_test, y_pred)  
    sns.heatmap(cm, annot=True, fmt='d')  
    plt.title(name)  
    plt.show()
```

(Logistic Regression Model Training) تدريب نموذج الانحدار اللوجستي

باستخدام البيانات المعالجة والممثلة بتقنية Logistic Regression في هذه المرحلة تم تدريب نموذج يُعتبر هذا النموذج من النماذج الخطية القوية في تصنيف النصوص، وغالبًا يُستخدم TF-IDF. للمقارنة مع باقي الخوارزميات (Baseline) كنموذج أساسي.

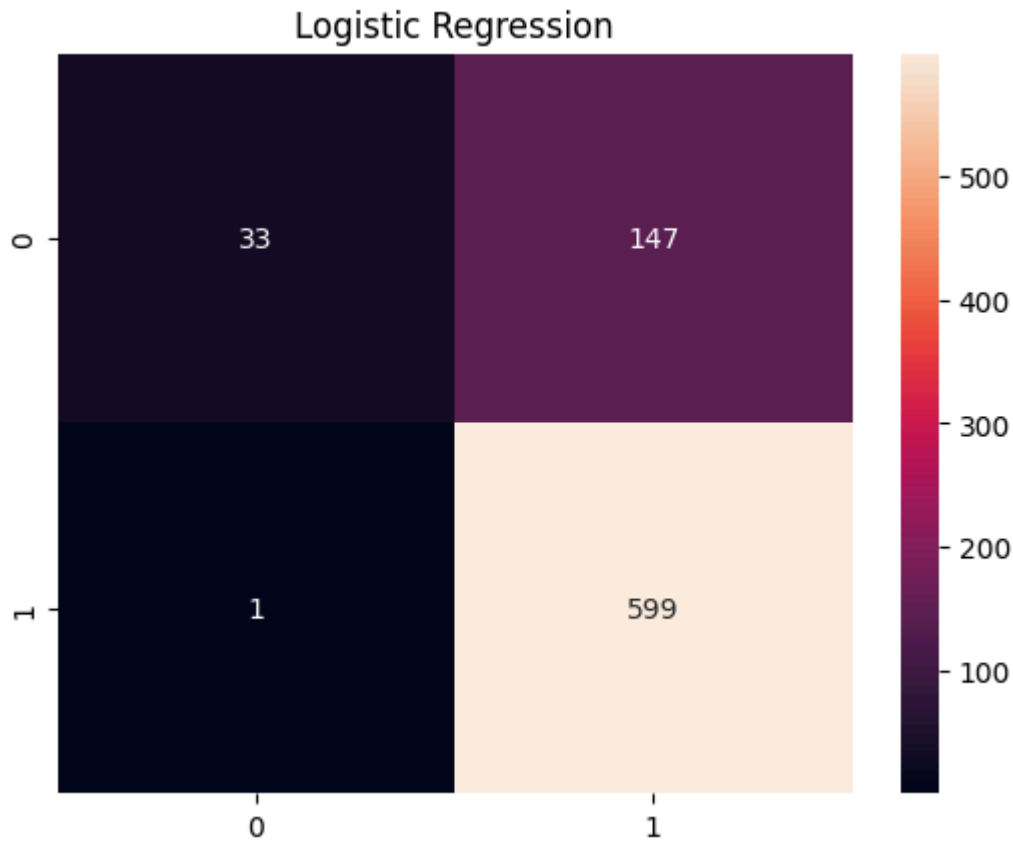
```
In [9]: lr = LogisticRegression(max_iter=1000)  
lr.fit(X_train_vec, y_train)  
evaluate(lr, "Logistic Regression")
```

```

===== Logistic Regression =====
Accuracy: 0.8102564102564103

```

	precision	recall	f1-score	support
negative	0.97	0.18	0.31	180
positive	0.80	1.00	0.89	600
accuracy			0.81	780
macro avg	0.89	0.59	0.60	780
weighted avg	0.84	0.81	0.76	780



في تصنيف النصوص Naive Bayes تطبيق نموذج

لتصنيف النصوص بناءً على الاحتمالات وتكرار الكلمات. أظهرت Naive Bayes تم تدريب نموذج النتائج أن النموذج يحقق دقة جيدة بشكل عام، لكنه يميل بشكل واضح إلى التنبؤ بالفئة الإيجابية بسبب عدم توازن البيانات، حيث كانت قدرته على اكتشاف الفئة السلبية ضعيفة جدًا. يشير ذلك إلى أن النموذج متحيز للفئة الأكبر، مما يستدعي استخدام تقنيات إضافية لمعالجة عدم التوازن وتحسين الأداء.

```

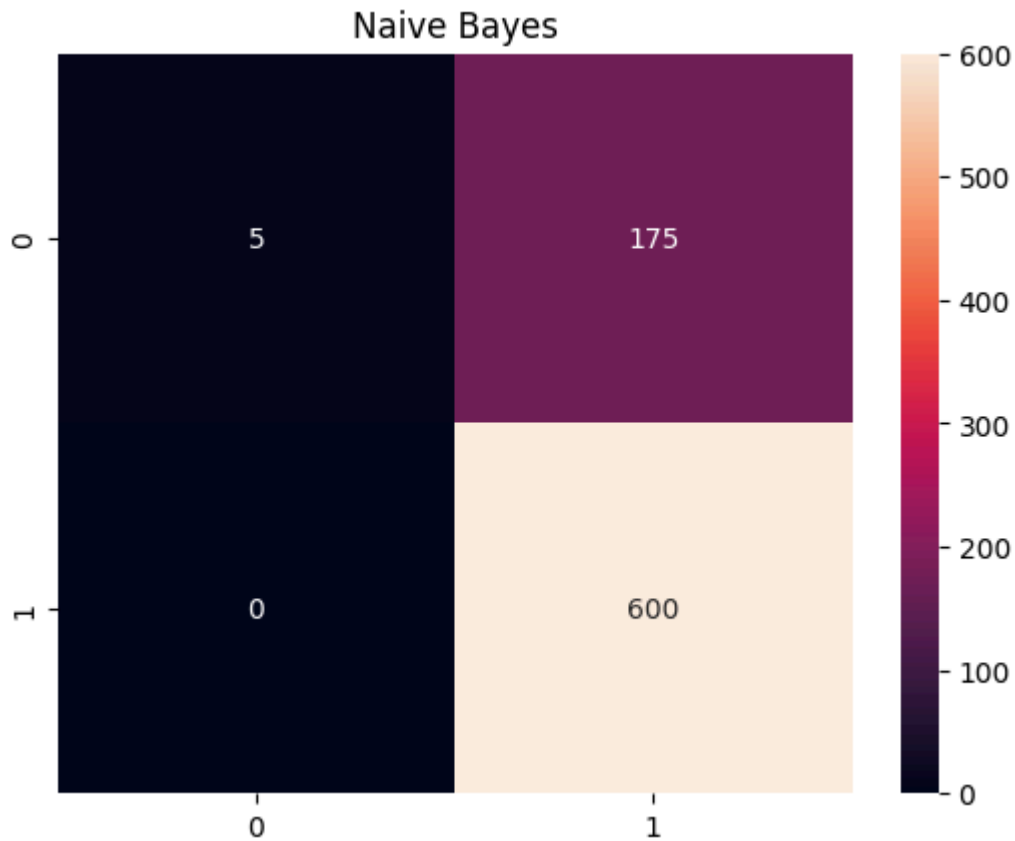
In [10]: nb = MultinomialNB()
nb.fit(X_train_vec, y_train)
evaluate(nb, "Naive Bayes")

```

===== Naive Bayes =====

Accuracy: 0.7756410256410257

	precision	recall	f1-score	support
negative	1.00	0.03	0.05	180
positive	0.77	1.00	0.87	600
accuracy			0.78	780
macro avg	0.89	0.51	0.46	780
weighted avg	0.83	0.78	0.68	780



وتحليل أدائه Support Vector Machine (SVM) تدريب نموذج

لتصنيف النصوص، وهو من أقوى النماذج في SVM (Support Vector Machine) تم تدريب نموذج مهام التصنيف خاصة مع البيانات النصية. أظهرت النتائج دقة مرتفعة (~86%)، مع توازن أفضل في الأداء بين الفئتين مقارنة بالنماذج السابقة.

من خلال النتائج:

أداء قوي جدًا للفئة الإيجابية النموذج Naive Bayes تحسن ملحوظ في التنبؤ بالفئة السلبية مقارنة بـ أقل تأثرًا بمشكلة عدم توازن البيانات

Confusion Matrix: من خلال

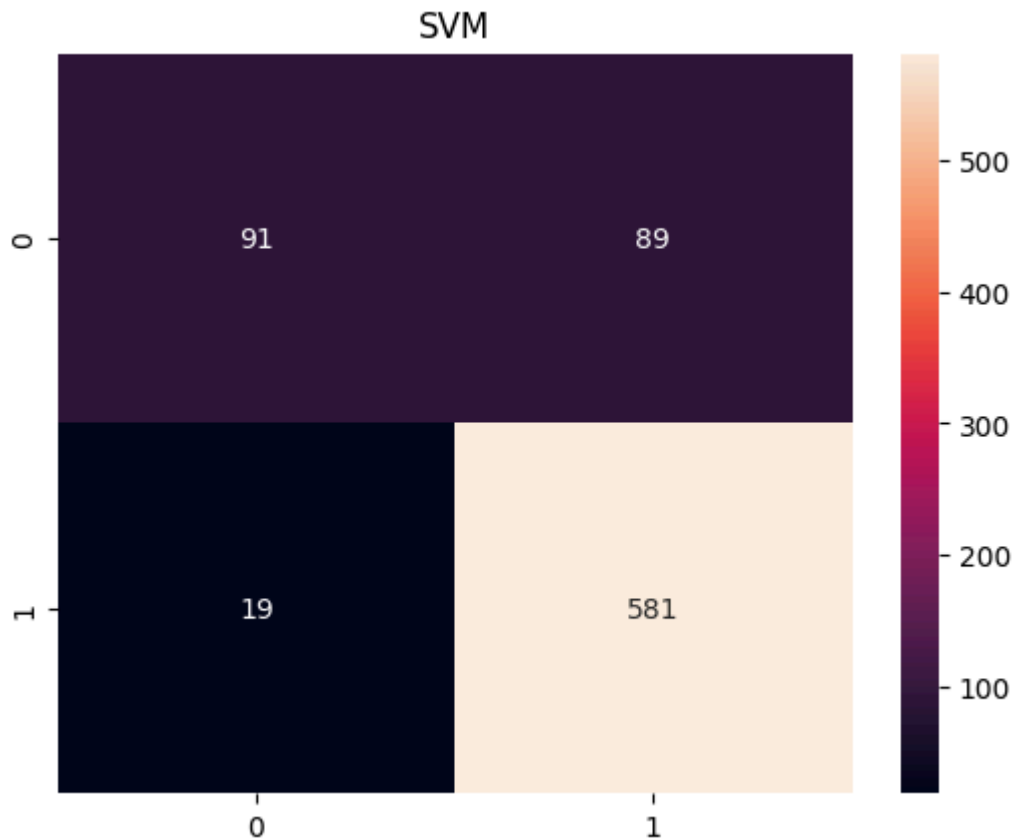
النموذج استطاع التمييز بين الفئتين بشكل أفضل عدد الأخطاء أقل مقارنة بالنماذج السابقة

```
In [11]: svm = LinearSVC()
svm.fit(X_train_vec, y_train)
evaluate(svm, "SVM")
```

===== SVM =====

Accuracy: 0.8615384615384616

	precision	recall	f1-score	support
negative	0.83	0.51	0.63	180
positive	0.87	0.97	0.91	600
accuracy			0.86	780
macro avg	0.85	0.74	0.77	780
weighted avg	0.86	0.86	0.85	780



Random Forest تدريب وتحسين نموذج

لاختيار أفضل المعاملات وتحسين الأداء. GridSearchCV مع Random Forest تم استخدام نموذج في SVM حقق النموذج دقة جيدة (~84%) مع أداء قوي للفئة الإيجابية، لكنه لا يزال أقل كفاءة من التعامل مع الفئة السلبية.

```
In [13]: rf = RandomForestClassifier()

param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20]
}

grid = GridSearchCV(rf, param_grid, cv=3)
grid.fit(X_train_vec, y_train)

best_rf = grid.best_estimator_

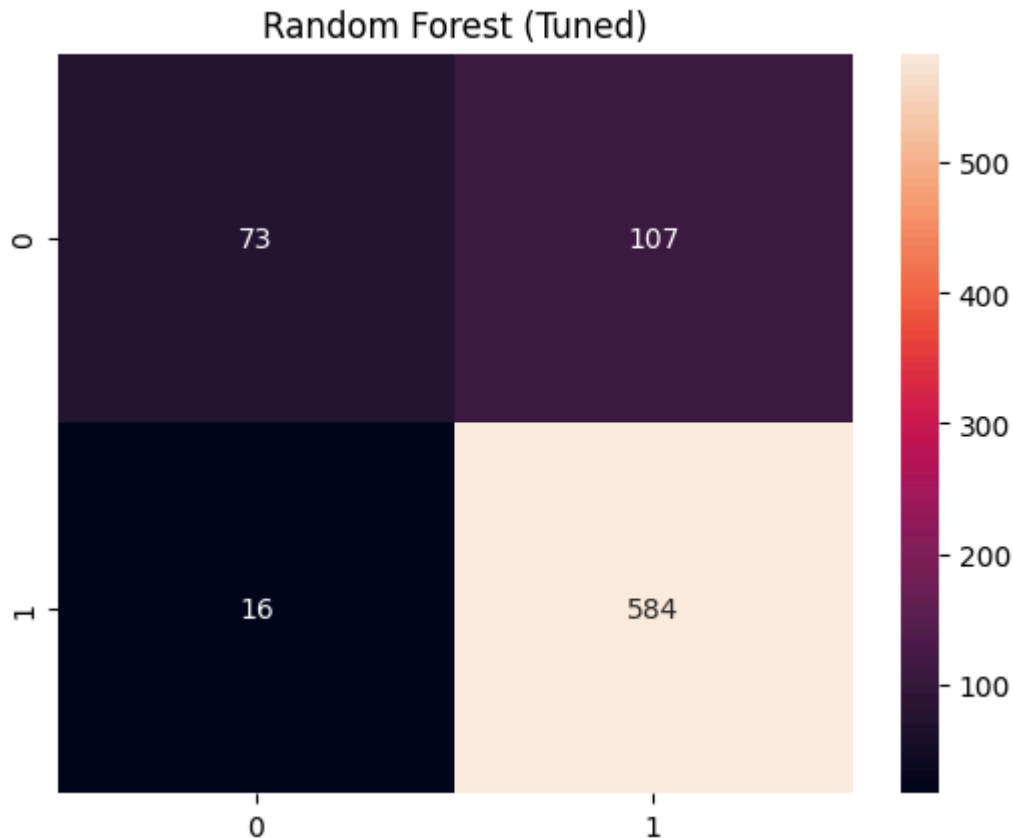
evaluate(best_rf, "Random Forest (Tuned)")

print("Best Parameters:", grid.best_params_)
```

==== Random Forest (Tuned) =====

Accuracy: 0.8423076923076923

	precision	recall	f1-score	support
negative	0.82	0.41	0.54	180
positive	0.85	0.97	0.90	600
accuracy			0.84	780
macro avg	0.83	0.69	0.72	780
weighted avg	0.84	0.84	0.82	780



Best Parameters: {'max_depth': None, 'n_estimators': 200}

Decision Tree تدريب نموذج

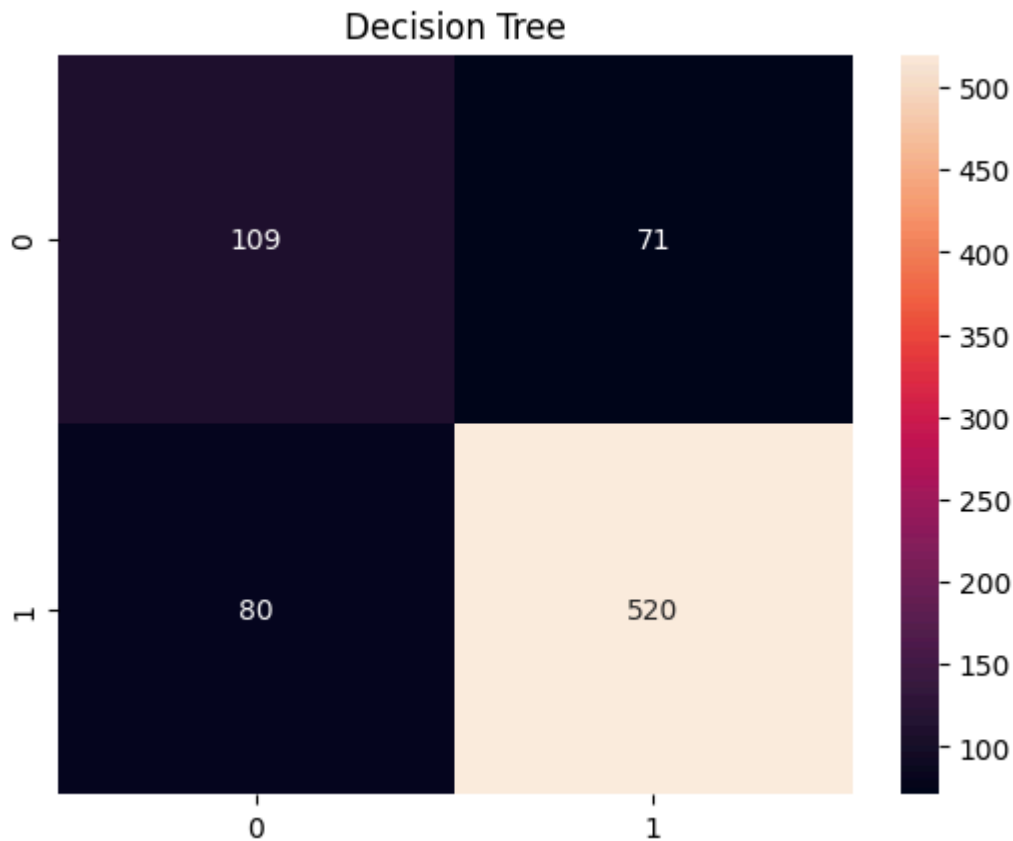
لتصنيف النصوص، وحقق دقة جيدة (~81%) مع أداء مقبول في Decision Tree تم تدريب نموذج Logistic Regression و SVM كلا الفئتين، لكنه أقل كفاءة مقارنة بنماذج مثل

```
In [14]: dt = DecisionTreeClassifier()
dt.fit(X_train_vec, y_train)
evaluate(dt, "Decision Tree")
```

==== Decision Tree =====

Accuracy: 0.8064102564102564

	precision	recall	f1-score	support
negative	0.58	0.61	0.59	180
positive	0.88	0.87	0.87	600
accuracy			0.81	780
macro avg	0.73	0.74	0.73	780
weighted avg	0.81	0.81	0.81	780



تدريب نموذج K-Nearest Neighbors (KNN)

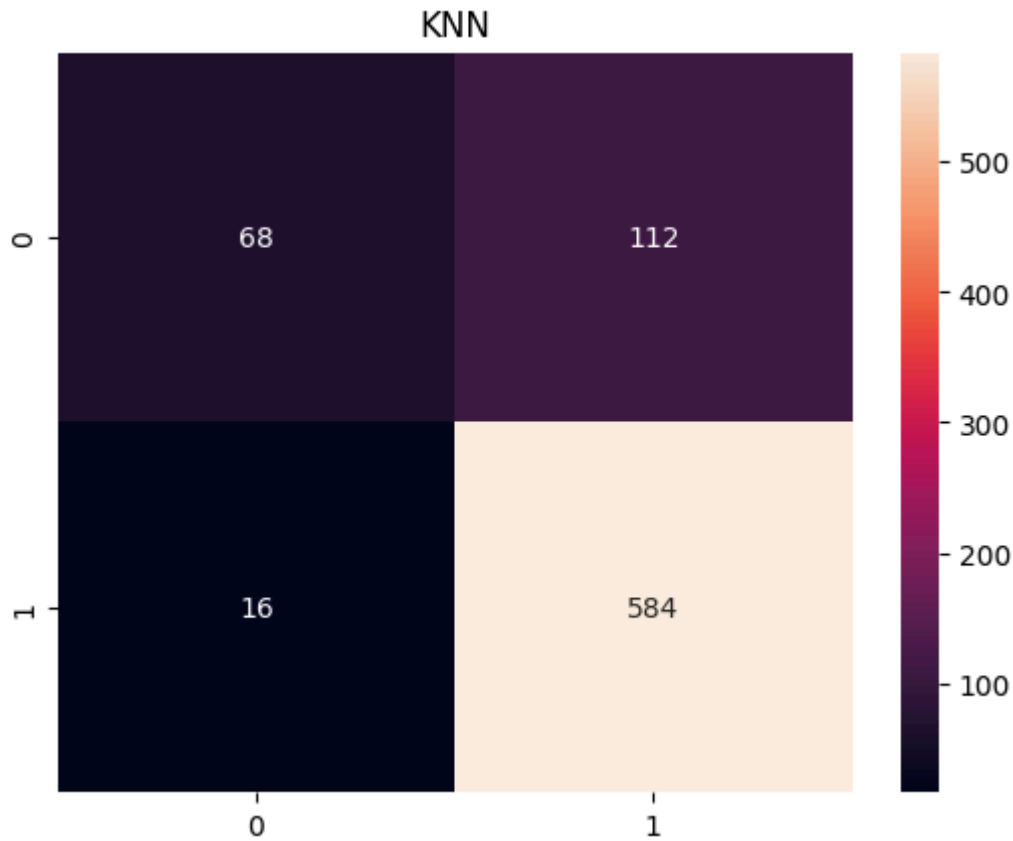
لتصنيف النصوص بناءً على أقرب الجيران، وحقق دقة جيدة (~84%) مع أداء KNN تم تدريب نموذج SVM. قوي للفئة الإيجابية، لكنه أقل كفاءة في التنبؤ بالفئة السلبية مقارنة بالنماذج الأقوى مثل

```
In [15]: knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_vec, y_train)
evaluate(knn, "KNN")
```

===== KNN =====

Accuracy: 0.8358974358974359

	precision	recall	f1-score	support
negative	0.81	0.38	0.52	180
positive	0.84	0.97	0.90	600
accuracy			0.84	780
macro avg	0.82	0.68	0.71	780
weighted avg	0.83	0.84	0.81	780



SMOTE معالجة عدم توازن البيانات باستخدام

لإنشاء عينات جديدة للفئة الأقل تمثيلًا، بهدف تحقيق توازن بين الفئات SMOTE تم استخدام تقنية في بيانات التدريب، مما يساعد على تحسين أداء النماذج خاصة في التنبؤ بالفئة السلبية.

```
In [18]: from imblearn.over_sampling import SMOTE

smote = SMOTE()

X_train_res, y_train_res = smote.fit_resample(X_train_vec, y_train)
```

(Imbalanced-Learn) تثبيت مكتبة معالجة البيانات غير المتوازنة

التي توفر أدوات لمعالجة مشكلة عدم توازن البيانات، مثل تقنية imbalanced-learn تم تثبيت مكتبة والتي تُستخدم لاحقًا لتحسين أداء النماذج على الفئات الأقل تمثيلًا، SMOTE.

```
In [17]: !pip install imbalanced-learn
```

Collecting imbalanced-learn

Downloading imbalanced_learn-0.14.1-py3-none-any.whl.metadata (8.9 kB)

Requirement already satisfied: numpy<3,>=1.25.2 in c:\users\bashr\miniconda3\lib\site-packages (from imbalanced-learn) (2.4.1)

Requirement already satisfied: scipy<2,>=1.11.4 in c:\users\bashr\miniconda3\lib\site-packages (from imbalanced-learn) (1.17.0)

Requirement already satisfied: scikit-learn<2,>=1.4.2 in c:\users\bashr\miniconda3\lib\site-packages (from imbalanced-learn) (1.8.0)

Collecting sklearn-compat<0.2,>=0.1.5 (from imbalanced-learn)

Downloading sklearn_compat-0.1.5-py3-none-any.whl.metadata (20 kB)

Requirement already satisfied: joblib<2,>=1.2.0 in c:\users\bashr\miniconda3\lib\site-packages (from imbalanced-learn) (1.5.3)

Requirement already satisfied: threadpoolctl<4,>=2.0.0 in c:\users\bashr\miniconda3\lib\site-packages (from imbalanced-learn) (3.6.0)

Downloading imbalanced_learn-0.14.1-py3-none-any.whl (235 kB)

Downloading sklearn_compat-0.1.5-py3-none-any.whl (20 kB)

Installing collected packages: sklearn-compat, imbalanced-learn

```
----- 1/2 [imbalanced-learn]
----- 1/2 [imbalanced-learn]
----- 1/2 [imbalanced-learn]
----- 1/2 [imbalanced-learn]
----- 1/2 [imbalanced-learn]
----- 1/2 [imbalanced-learn]
----- 2/2 [imbalanced-learn]
```

Successfully installed imbalanced-learn-0.14.1 sklearn-compat-0.1.5

عرض بيانات التدريب بعد إعادة التوازن

حيث أصبحت الفئات متوازنة، مما يساهم في SMOTE، تم عرض بيانات التدريب بعد تطبيق تقنية تحسين قدرة النماذج على التعلم وتقليل التحيز للفئة الأكبر.

```
In [19]: X_train_res, y_train_res
```

```
Out[19]: (<Compressed Sparse Row sparse matrix of dtype 'float64'
          with 47114 stored elements and shape (2800, 13909)>,
0         positive
1         positive
2         positive
3         positive
4         positive
...
2795      negative
2796      negative
2797      negative
2798      negative
2799      negative
Name: sentiment, Length: 2800, dtype: object)
```

تدريب نموذج بعد إعادة التوازن

```
In [20]: lr_smote = LogisticRegression(max_iter=1000)
lr_smote.fit(X_train_res, y_train_res)

y_pred_smote = lr_smote.predict(X_test_vec)
```

رسم Confusion Matrix بعد SMOTE

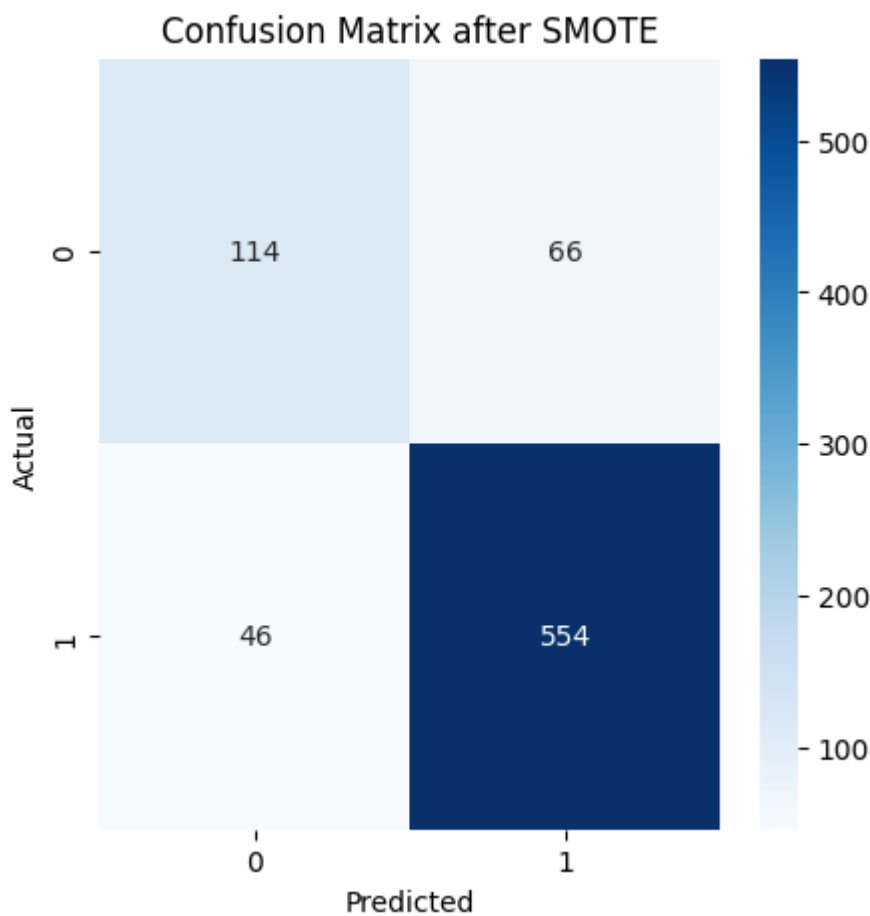
```
In [21]: from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred_smote)

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.title("Confusion Matrix after SMOTE")
plt.xlabel("Predicted")
plt.ylabel("Actual")

plt.show()
```



حيث أظهرت النتائج تحسناً ملحوظاً في SMOTE، تمت مقارنة أداء النموذج قبل وبعد تطبيق تقنية قدرة النموذج على التنبؤ بالفئة السلبية، مع الحفاظ على دقة جيدة للفئة الإيجابية، مما يؤكد فعالية معالجة عدم توازن البيانات في تحسين الأداء العام.

```
In [22]: print("Before SMOTE:")
print(classification_report(y_test, lr.predict(X_test_vec)))

print("\nAfter SMOTE:")
print(classification_report(y_test, y_pred_smote))
```

Before SMOTE:

	precision	recall	f1-score	support
negative	0.97	0.18	0.31	180
positive	0.80	1.00	0.89	600
accuracy			0.81	780
macro avg	0.89	0.59	0.60	780
weighted avg	0.84	0.81	0.76	780

After SMOTE:

	precision	recall	f1-score	support
negative	0.71	0.63	0.67	180
positive	0.89	0.92	0.91	600
accuracy			0.86	780
macro avg	0.80	0.78	0.79	780
weighted avg	0.85	0.86	0.85	780

الخاتمة

SVM. وأظهرت النتائج تفوق نموذج NLP، تم تطبيق عدة نماذج لتصنيف النصوص باستخدام تقنيات في تحسين أداء النماذج من خلال معالجة عدم توازن البيانات، مما أدى SMOTE كما ساهم استخدام إلى نتائج أكثر دقة وتوازنًا.

In []: