

```
# ربط Google Drive
from google.colab import drive
drive.mount('/content/drive')

# استيراد pandas
import pandas as pd

# مسار الملف الصحيح
file_path = "/content/drive/MyDrive/Musab Alsalamah - Project 2/unbalanceddataset.csv"

# قراءة الملف
df = pd.read_csv(file_path)

# عرض البيانات
df.head()
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

	text	sentiment
0	Java Concurrency in Practice is probably the b...	positive
1	haha aww hun i bet you are more creative tha...	positive
2	_pickle lol, thank you very much Hope you're h...	positive
3	Out for an evening on the town with jeremy. Sa...	negative
4	- just took over the #1 Most Endorsed spot on...	positive

Next steps: [Generate code with df](#) [New interactive sheet](#)

```
# فحص البيانات

# عرض أسماء الأعمدة
print("Columns:")
print(df.columns)

# عرض حجم البيانات (عدد الصفوف والأعمدة)
print("\nShape:")
print(df.shape)

# عرض توزيع الفئات
print("\nClass distribution:")
print(df["sentiment"].value_counts())
```

```
Columns:
Index(['text', 'sentiment'], dtype='object')

Shape:
(2600, 2)

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

```
# تجهيز أدوات تنظيف النصوص

import re
import nltk

# تحميل الموارد المطلوبة
nltk.download("punkt")
nltk.download("stopwords")

# استيراد الأدوات
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer

# إنشاء قائمة الكلمات الشائعة (مثل: the, is, in)
stop_words = set(stopwords.words("english"))

# تحويل الكلمات لأصلها (Stemming) إنشاء أداة
stemmer = PorterStemmer()
```

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

```
# دالة تنظيف النصوص

def preprocess_text(text):
    # تحويل النص إلى حروف صغيرة
    text = str(text).lower()

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

    # إذا وجدت HTML حذف وسوم
    text = re.sub(r'<.*?>', '', text)

    # حذف الرموز والأرقام وعلامات الترقيم
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # تقسيم النص إلى كلمات
    tokens = word_tokenize(text)

    # حذف الكلمات الشائعة
    tokens = [word for word in tokens if word not in stop_words]

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

    # إعادة تجميع الكلمات في نص واحد
    clean_text = " ".join(stemmed_tokens)

    return clean_text
```

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

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt_tab.zip.
True
```

```
# تطبيق التنظيف على البيانات

# إنشاء عمود جديد للنص بعد التنظيف
df["clean_text"] = df["text"].apply(preprocess_text)

# حذف الصفوف الفارغة (إن وجدت)
df = df.dropna()

# عرض النتائج للتحقق
print(df[["text", "clean_text", "sentiment"]].head())
```

```
text \
0  Java Concurrency in Practice is probably the b...
1  haha aww hun i bet you are more creative tha...
2  _pickle lol, thank you very much Hope you`re h...
3  Out for an evening on the town with jeremy. Sa...
4  - just took over the #1 Most Endorsed spot on...

clean_text sentiment
0  java concurr practic probabl best java book iv... positive
1              haha aww hun bet creativ positive
2  pickl lol thank much hope your great day positive
3  even town jeremi sad carri cant come negative
4  took endors spot twindexxcom thank endors positive
```

```
# تقسيم البيانات

from sklearn.model_selection import train_test_split

# X = النصوص بعد التنظيف
X = df["clean_text"]

# y = التصنيفات
y = df["sentiment"]

# تقسيم البيانات (70% تدريب - 30% اختبار)
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.30,
    random_state=42,
    stratify=y # للحفاظ على نفس نسبة الفئات
)
```

```
# عرض الأحجام للتأكد
print("Training size:", len(X_train))
print("Testing size:", len(X_test))
```

```
Training size: 1820
Testing size: 780
```

```
# Feature Representation باستخدام Bag of Words
```

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
# إنشاء المحول
bow_vectorizer = CountVectorizer()
```

```
# تدريب المحول على بيانات التدريب ثم تحويلها إلى أرقام
X_train_bow = bow_vectorizer.fit_transform(X_train)
```

```
# تحويل بيانات الاختبار باستخدام نفس المحول
X_test_bow = bow_vectorizer.transform(X_test)
```

```
# عرض أبعاد البيانات بعد التحويل
print("X_train_bow shape:", X_train_bow.shape)
print("X_test_bow shape:", X_test_bow.shape)
```

```
X_train_bow shape: (1820, 3659)
X_test_bow shape: (780, 3659)
```

```
# Feature Representation باستخدام TF-IDF
```

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

```
# إنشاء المحول
tfidf_vectorizer = TfidfVectorizer()
```

```
# تدريب المحول على بيانات التدريب ثم تحويلها إلى أرقام
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
```

```
# تحويل بيانات الاختبار باستخدام نفس المحول
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

```
# عرض أبعاد البيانات بعد التحويل
print("X_train_tfidf shape:", X_train_tfidf.shape)
print("X_test_tfidf shape:", X_test_tfidf.shape)
```

```
X_train_tfidf shape: (1820, 3659)
X_test_tfidf shape: (780, 3659)
```

```
# دالة تقييم النموذج
```

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

```
def evaluate_model(model_name, y_true, y_pred):
```

```
    # حساب المقاييس الأساسية
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average="weighted")
    recall = recall_score(y_true, y_pred, average="weighted")
    f1 = f1_score(y_true, y_pred, average="weighted")
```

```
    # مصفوفة الالتباس
    cm = confusion_matrix(y_true, y_pred)
```

```
    # تقرير التصنيف
    report = classification_report(y_true, y_pred)
```

```
    # طباعة النتائج
    print("Model:", model_name)
    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall:", recall)
    print("F1 Score:", f1)
```

```
    print("\nConfusion Matrix:")
    print(cm)
```

```
print("\nClassification Report:")
print(report)
```

```
# إنشاء النموذج

from sklearn.ensemble import RandomForestClassifier

# إنشاء النموذج
rf_model = RandomForestClassifier(random_state=42)
```

```
# تدريب النموذج على Bag of Words

# تدريب النموذج على بيانات التدريب
rf_model.fit(X_train_bow, y_train)

# التنبؤ على بيانات الاختبار
y_pred_bow = rf_model.predict(X_test_bow)

# عرض أول 10 توقعات
print(y_pred_bow[:10])
```

```
['positive' 'positive' 'positive' 'positive' 'positive' 'negative'
 'positive' 'positive' 'positive' 'negative']
```

```
# تقييم النموذج باستخدام Bag of Words

# تشغيل دالة التقييم
evaluate_model("Random Forest (Bow)", y_test, y_pred_bow)
```

```
Model: Random Forest (Bow)
Accuracy: 0.8538461538461538
Precision: 0.8517887642311605
Recall: 0.8538461538461538
F1 Score: 0.8369652771722379
```

```
Confusion Matrix:
[[ 82  98]
 [ 16 584]]
```

```
Classification Report:
              precision    recall  f1-score   support

   negative         0.84         0.46         0.59         180
   positive         0.86         0.97         0.91         600

   accuracy                    0.85         780
  macro avg         0.85         0.71         0.75         780
 weighted avg         0.85         0.85         0.84         780
```

```
# تدريب النموذج على TF-IDF

# إنشاء نموذج جديد (مهم عشان ما يتأثر بالنموذج السابق)
rf_model_tfidf = RandomForestClassifier(random_state=42)

# تدريب النموذج على بيانات
rf_model_tfidf.fit(X_train_tfidf, y_train)

# التنبؤ على بيانات الاختبار
y_pred_tfidf = rf_model_tfidf.predict(X_test_tfidf)

# عرض أول 10 توقعات
print(y_pred_tfidf[:10])
```

```
['positive' 'positive' 'positive' 'positive' 'positive' 'negative'
 'positive' 'positive' 'positive' 'positive']
```

```
# تقييم النموذج باستخدام TF-IDF

evaluate_model("Random Forest (TF-IDF)", y_test, y_pred_tfidf)
```

```
Model: Random Forest (TF-IDF)
Accuracy: 0.8448717948717949
Precision: 0.8406099471024558
Recall: 0.8448717948717949
F1 Score: 0.8260281352888357
```

```
Confusion Matrix:
[[ 77 103]
 [ 18 582]]
```

```
Classification Report:
              precision    recall  f1-score   support

   negative       0.81       0.43       0.56       180
   positive       0.85       0.97       0.91       600

 accuracy              0.84       780
 macro avg       0.83       0.70       0.73       780
 weighted avg    0.84       0.84       0.83       780
```

مقارنة النتائج

```
print("Comparison between models:")
print("-----")
print("Model 1: Random Forest (Bag of Words)")
print("Model 2: Random Forest (TF-IDF)")
print("-----")
print("Note: TF-IDF usually gives better results because it considers word importance.")
```

Comparison between models:

```
-----
Model 1: Random Forest (Bag of Words)
Model 2: Random Forest (TF-IDF)
-----
```

Note: TF-IDF usually gives better results because it considers word importance.

رسم مقارنة بين النماذج

```
import matplotlib.pyplot as plt

# لكل نموذج حساب Accuracy
acc_bow = accuracy_score(y_test, y_pred_bow)
acc_tfidf = accuracy_score(y_test, y_pred_tfidf)

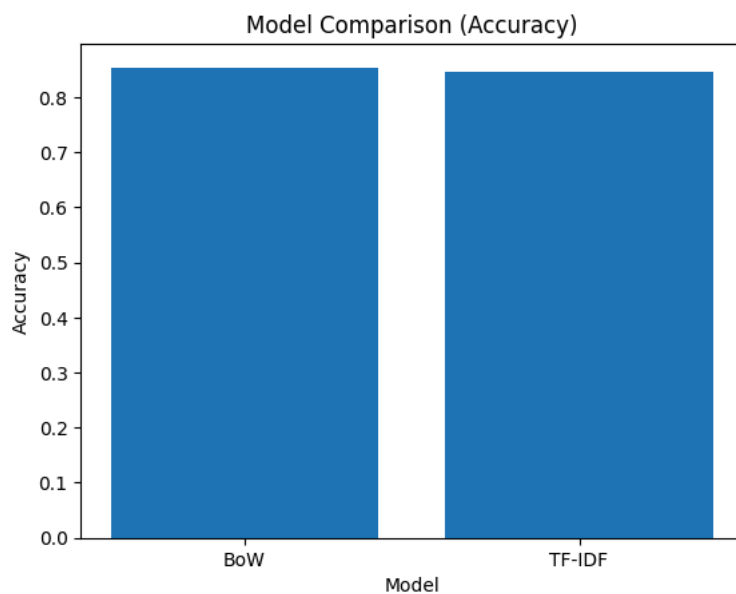
# أسماء النماذج
models = ["BoW", "TF-IDF"]

# القيم
accuracies = [acc_bow, acc_tfidf]

# رسم Bar Chart
plt.figure()
plt.bar(models, accuracies)

# العناوين
plt.title("Model Comparison (Accuracy)")
plt.xlabel("Model")
plt.ylabel("Accuracy")

# عرض الرسم
plt.show()
```



#رسم Confusion Matrix

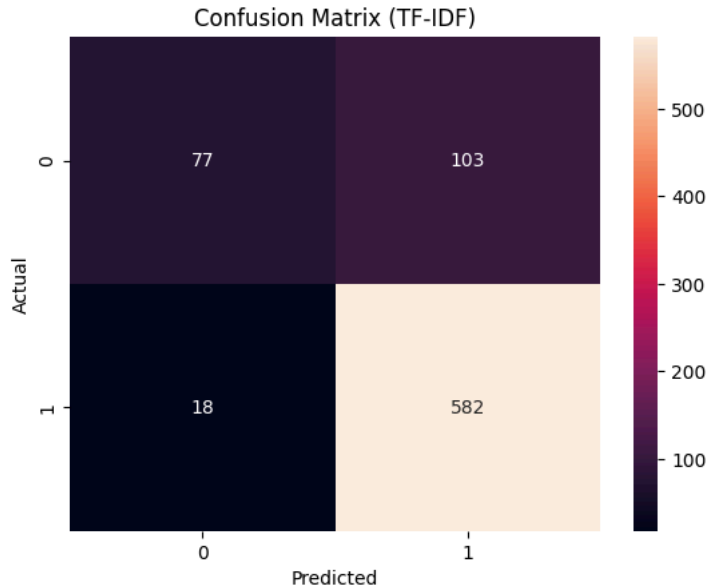
```
import seaborn as sns
```

```
# مصفوفة الالتباس TF-IDF
cm = confusion_matrix(y_test, y_pred_tfidf)

plt.figure()
sns.heatmap(cm, annot=True, fmt='d')

plt.title("Confusion Matrix (TF-IDF)")
plt.xlabel("Predicted")
plt.ylabel("Actual")

plt.show()
```



```
# رسم مقارنة Precision / Recall / F1

import matplotlib.pyplot as plt

# حساب المقاييس لنموذج Bag of Words
precision_bow = precision_score(y_test, y_pred_bow, average='weighted')
recall_bow = recall_score(y_test, y_pred_bow, average='weighted')
f1_bow = f1_score(y_test, y_pred_bow, average='weighted')

# حساب المقاييس لنموذج TF-IDF
precision_tfidf = precision_score(y_test, y_pred_tfidf, average='weighted')
recall_tfidf = recall_score(y_test, y_pred_tfidf, average='weighted')
f1_tfidf = f1_score(y_test, y_pred_tfidf, average='weighted')

# أسماء المقاييس
metrics = ["Precision", "Recall", "F1-score"]

# القيم لكل نموذج
bow_values = [precision_bow, recall_bow, f1_bow]
tfidf_values = [precision_tfidf, recall_tfidf, f1_tfidf]

# تحديد أماكن الأعمدة
x = range(len(metrics))

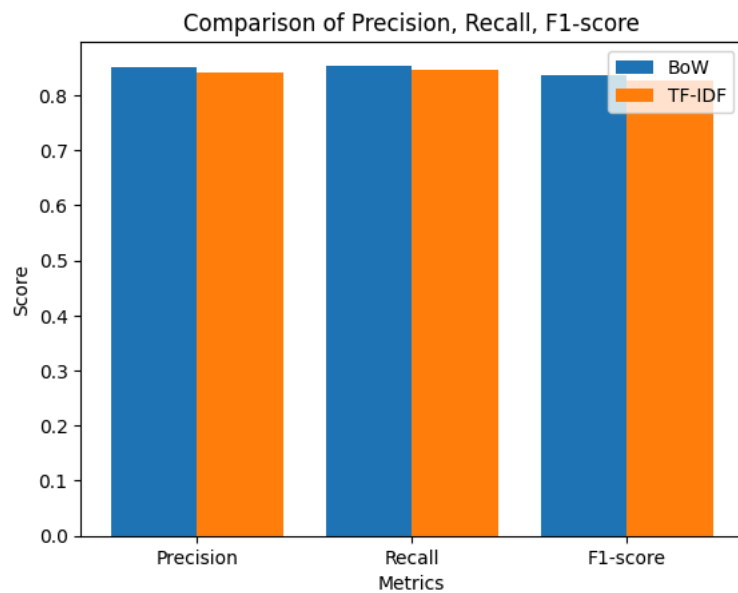
# رسم الأعمدة
plt.figure()

plt.bar(x, bow_values, width=0.4, label="Bow")
plt.bar([i + 0.4 for i in x], tfidf_values, width=0.4, label="TF-IDF")

# العناوين
plt.xticks([i + 0.2 for i in x], metrics)
plt.xlabel("Metrics")
plt.ylabel("Score")
plt.title("Comparison of Precision, Recall, F1-score")

# عرض الـ legend
plt.legend()

# عرض الرسم
plt.show()
```



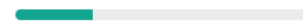
Start coding or [generate](#) with AI.

	Share
1.2 Thesis Objectives	
1.3 Thesis Contributions	
1.4 Thesis Organization	
Chapter 2 Background and Related Work	
2.1 Introduction	
2.2 Classical Machine Learning Methods	
2.3 Deep Learning Approaches	
2.4 Hybrid and Optimized Models	
2.6 Research Gaps and Opportunities	
2.8 Previous Studies on Traffic Congestion Forecasting	
2.9 Comparative Summary Table	
2.10 Summary	
Chapter 3 Proposed Approach	
3.1 Overview	

AI Detector

24 parts of your text contain patte
AI text. These can also occur in yc
use generative AI, you can cite it.

Possible AI text detected



Does it seem accurate?

Citations added