

```
In [55]: print ('بسم الله الرحمن الرحيم')
print ('الاختبار الثاني')
print ('Practical_Image_Processing_and_Natural_Language_Processing_Second Trimester')
print ('تقديم الطالب / حسن محمد احمد الامير خليفة')
print ('الرقم الجامعي', '253002941')
print ('اعداد الدكتور / هاجر صالح')
print ('الداتا ست التي تم استخدامها https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset')
```

```
بسم الله الرحمن الرحيم
الاختبار الثاني
Practical_Image_Processing_and_Natural_Language_Processing_Second Trimester
تقديم الطالب / حسن محمد احمد الامير خليفة
الرقم الجامعي 253002941
اعداد الدكتور / هاجر صالح
الداتا ست التي تم استخدامها https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset
```

```
In [2]: print ('Import Libraries')
import os
import torch
import torchvision
import numpy as np
import matplotlib.pyplot as plt
from torchvision import datasets, transforms
from torch.utils.data import DataLoader, random_split
import torch.nn as nn
import torch.optim as optim
```

Import Libraries

```
In [3]: import pandas as pd

df = pd.read_csv(
    "spam.csv",
    encoding="latin-1"
)

print(df.columns)
print(df.head())
```

```
Index(['v1', 'v2', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], dtype='object')
```

	v1	v2	Unnamed: 2	\
0	ham	Go until jurong point, crazy.. Available only	...	NaN
1	ham	Ok lar... Joking wif u oni...		NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...		NaN
3	ham	U dun say so early hor... U c already then say...		NaN
4	ham	Nah I don't think he goes to usf, he lives aro...		NaN

	Unnamed: 3	Unnamed: 4
0	NaN	NaN
1	NaN	NaN
2	NaN	NaN
3	NaN	NaN
4	NaN	NaN

In [5]: `print('الخطوة 1: تجهيز البيانات')`

الخطوة 1: تجهيز البيانات

In [6]: `import pandas as pd`

```
# قراءة البيانات
df = pd.read_csv(
    "spam.csv",
    encoding="latin-1"
)

# الاحتفاظ بالأعمدة المطلوبة فقط
df = df[["v1", "v2"]]

# إعادة تسمية الأعمدة
df.columns = ["label", "text"]

# عرض المعلومات
print("Dataset Shape:", df.shape)

print("\nColumns:")
print(df.columns)

print("\nFirst 5 Rows:")
print(df.head())

print("\nClass Distribution:")
print(df["label"].value_counts())
```

Dataset Shape: (5572, 2)

Columns:

Index(['label', 'text'], dtype='object')

First 5 Rows:

	label	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Class Distribution:

label

ham 4825

spam 747

Name: count, dtype: int64

In [12]: `print('الخطوة 2: فحص البيانات وتنظيفها')`
`print('Stopwords إزالة الروابط إزالة الأرقام إزالة علامات الترقيم إزالة lowercase سنقوم ب: تحويل النص إلى')`

الخطوة 2: فحص البيانات وتنظيفها

Stopwords Lemmatizati
on
إزالة الروابط إزالة الأرقام إزالة علامات الترقيم إزالة lowercase سنقوم ب: تحويل النص إلى

```
In [8]: # فحص القيم المفقودة
print("Missing Values:")
print(df.isnull().sum())

# فحص الصفوف المكررة
print("\nDuplicate Rows:")
print(df.duplicated().sum())

# حذف الصفوف المكررة
df = df.drop_duplicates()

# إعادة ترتيب Index
df = df.reset_index(drop=True)

print("\nDataset Shape After Removing Duplicates:")
print(df.shape)
```

Missing Values:

```
label    0
text     0
dtype: int64
```

Duplicate Rows:

```
403
```

Dataset Shape After Removing Duplicates:

```
(5169, 2)
```

```
In [9]: print ('3 الخطوة: Data Preprocessing')
```

3 الخطوة: Data Preprocessing

```
In [13]: print ('إلى رقم Label الخطوة 4: تحويل')
```

إلى رقم Label الخطوة 4: تحويل

```
In [14]: print ('إلى رقم Label الخطوة 4: تحويل')
```

إلى رقم Label الخطوة 4: تحويل

```
In [15]: df["label_encoded"] = df["label"].map({
    "ham": 0,
    "spam": 1
})

print(df[["label", "label_encoded"]].head())
```

```
label  label_encoded
0    ham             0
1    ham             0
2  spam             1
3    ham             0
4    ham             0
```

```
In [17]: print('Dataset 80 %سنستخدم Training 20% Testing: تقسيم')
```

Dataset 80 %سنستخدم Training 20% Testing: تقسيم

```
In [ ]: print('')
```

```
In [22]: !pip install nltk
```


[illegible]

Successfully installed nltk-3.10.3 regex-2026.7.19

```
In [23]: import pandas as pd
import re
import nltk

from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# =====
# 1. Load Dataset
# =====

df = pd.read_csv(
    "spam.csv",
    encoding="latin-1"
)

# Keep only required columns
df = df[["v1", "v2"]]

# Rename columns
```

```

df.columns = ["label", "text"]

# Remove duplicate rows
df = df.drop_duplicates()

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

# =====
# 2. Download NLP Resources
# =====

nltk.download("stopwords")
nltk.download("wordnet")

# =====
# 3. Text Preprocessing
# =====

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

lemmatizer = WordNetLemmatizer()

def preprocess_text(text):

    # Convert to Lowercase
    text = text.lower()

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

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

    # Remove punctuation and special characters
    text = re.sub(r"[^a-zA-Z\s]", " ", text)

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

    # Tokenization
    words = text.split()

    # Remove stopwords
    words = [
        word for word in words
        if word not in stop_words
    ]

    # Lemmatization
    words = [
        lemmatizer.lemmatize(word)
        for word in words
    ]

```

```

    ]

    return " ".join(words)

# Apply preprocessing
df["clean_text"] = df["text"].apply(preprocess_text)

# =====
# 4. Encode Labels
# =====

df["label_encoded"] = df["label"].map({
    "ham": 0,
    "spam": 1
})

# =====
# 5. Check Results
# =====

print("Dataset shape:", df.shape)

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

print("\nFirst 5 rows:")
print(df.head())

print("\nClass distribution:")
print(df["label"].value_counts())

print("\nEncoded labels:")
print(df["label_encoded"].value_counts())

```

```

[nltk_data] Downloading package stopwords to
[nltk_data]   C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
[nltk_data] Downloading package wordnet to
[nltk_data]   C:\Users\user\AppData\Roaming\nltk_data...

```

Dataset shape: (5169, 4)

Columns:

```
['label', 'text', 'clean_text', 'label_encoded']
```

First 5 rows:

	label	text \	clean_text	label_encoded
0	ham	Go until jurong point, crazy.. Available only ...	go jurong point crazy available bugis n great ...	0
1	ham	Ok lar... Joking wif u oni...	ok lar joking wif u oni	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	free entry wkly comp win fa cup final tkts st ...	1
3	ham	U dun say so early hor... U c already then say...	u dun say early hor u c already say	0
4	ham	Nah I don't think he goes to usf, he lives aro...	nah think go usf life around though	0

Class distribution:

```
label
ham      4516
spam      653
Name: count, dtype: int64
```

Encoded labels:

```
label_encoded
0      4516
1       653
Name: count, dtype: int64
```

```
In [24]: print ('الآن: Split the Dataset')
```

الآن: Split the Dataset

```
In [25]: from sklearn.model_selection import train_test_split
```

```
# Features
X = df["clean_text"]

# Target
y = df["label_encoded"]

# Split: 80% Training / 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("Training samples:", len(X_train))
print("Testing samples:", len(X_test))
```

```
print("\nTraining class distribution:")
print(y_train.value_counts())

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

Training samples: 4135

Testing samples: 1034

Training class distribution:

label_encoded

0 3613

1 522

Name: count, dtype: int64

Testing class distribution:

label_encoded

0 903

1 131

Name: count, dtype: int64

In [26]: `print ('الخطوة التالية: TF-IDF')`

الخطوة التالية: TF-IDF

In [27]: `from sklearn.feature_extraction.text import TfidfVectorizer`

```
# Create TF-IDF Vectorizer
tfidf = TfidfVectorizer(
    max_features=5000,
    ngram_range=(1, 2)
)

# Fit and transform training data
X_train_tfidf = tfidf.fit_transform(X_train)

# Transform testing data
X_test_tfidf = tfidf.transform(X_test)

print("TF-IDF Training Shape:", X_train_tfidf.shape)
print("TF-IDF Testing Shape:", X_test_tfidf.shape)
```

TF-IDF Training Shape: (4135, 5000)

TF-IDF Testing Shape: (1034, 5000)

In [28]: `print ('الخطوة 1 - تدريب Naive Bayes')`

الخطوة 1 - تدريب Naive Bayes

In [29]: `from sklearn.naive_bayes import MultinomialNB`

```
# Create the model
nb_model = MultinomialNB()

# Train the model
nb_model.fit(
    X_train_tfidf,
    y_train
```

```

)

# Make predictions
y_pred_nb = nb_model.predict(
    X_test_tfidf
)

print("Naive Bayes model trained successfully!")

```

Naive Bayes model trained successfully!

In [30]: `print ('الخطوة 2 - تقييم النموذج')`

الخطوة 2 - تقييم النموذج

```

In [31]: from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report
)

# Calculate metrics
accuracy_nb = accuracy_score(
    y_test,
    y_pred_nb
)

precision_nb = precision_score(
    y_test,
    y_pred_nb,
    zero_division=0
)

recall_nb = recall_score(
    y_test,
    y_pred_nb,
    zero_division=0
)

f1_nb = f1_score(
    y_test,
    y_pred_nb,
    zero_division=0
)

print("=====")
print("Naive Bayes - TF-IDF")
print("=====")

print("Accuracy :", round(accuracy_nb, 4))
print("Precision:", round(precision_nb, 4))
print("Recall   :", round(recall_nb, 4))
print("F1-Score :", round(f1_nb, 4))

print("\nClassification Report:")

```

```
print(
    classification_report(
        y_test,
        y_pred_nb,
        target_names=["Ham", "Spam"],
        zero_division=0
    )
)
```

```
=====
Naive Bayes - TF-IDF
=====
Accuracy : 0.9691
Precision: 0.9901
Recall    : 0.7634
F1-Score  : 0.8621
```

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

      Ham           0.97         1.00         0.98         903
      Spam           0.99         0.76         0.86         131

   accuracy                   0.97         1034
  macro avg           0.98         0.88         0.92         1034
 weighted avg           0.97         0.97         0.97         1034
```

In [32]: `print ('الآن نعمل Confusion Matrix')`

الآن نعمل Confusion Matrix

```
In [33]: from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Create confusion matrix
cm_nb = confusion_matrix(
    y_test,
    y_pred_nb
)

print("Confusion Matrix:")
print(cm_nb)

# Display confusion matrix
disp = ConfusionMatrixDisplay(
    confusion_matrix=cm_nb,
    display_labels=["Ham", "Spam"]
)

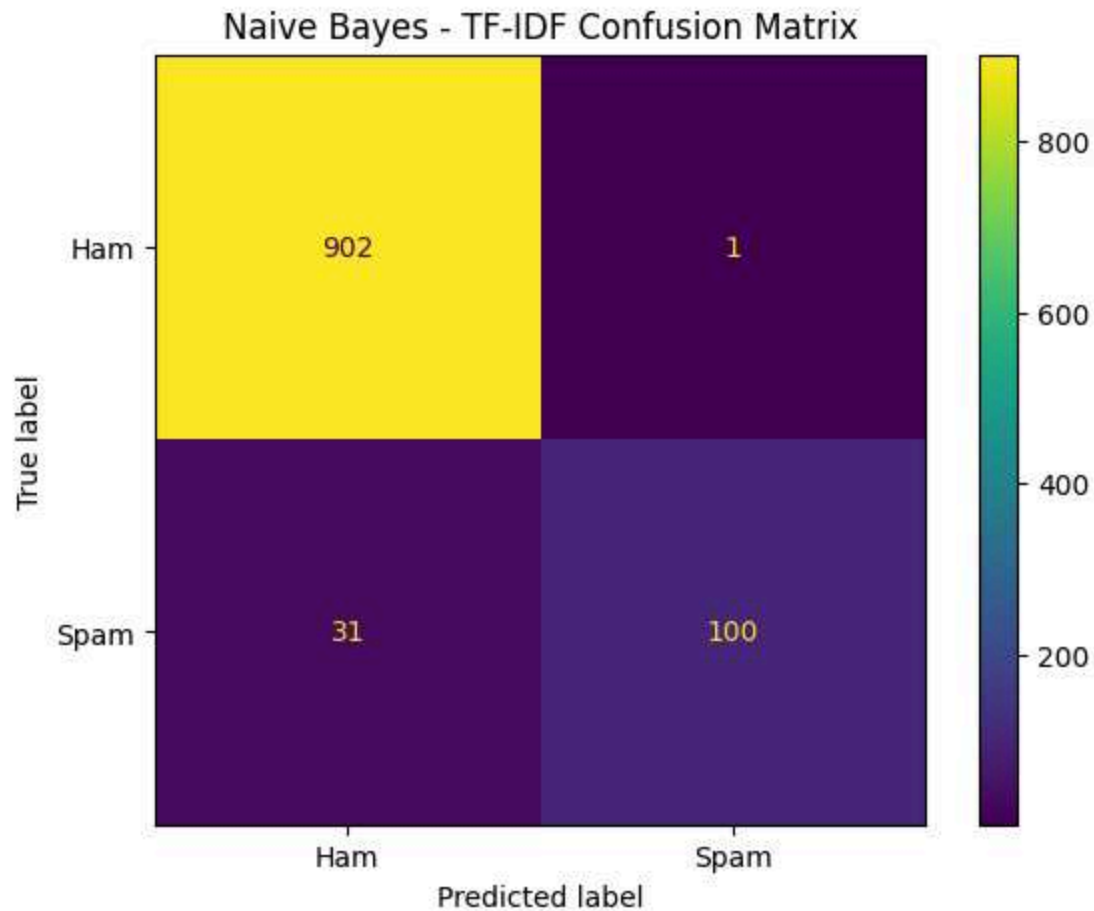
fig, ax = plt.subplots(figsize=(7, 5))

disp.plot(
    ax=ax
)
```

```
plt.title("Naive Bayes - TF-IDF Confusion Matrix")
plt.show()
```

Confusion Matrix:

```
[[902  1]
 [ 31 100]]
```



```
In [34]: print ('ثم النموذج الثاني: Logistic Regression')
```

ثم النموذج الثاني: Logistic Regression

```
In [35]: from sklearn.linear_model import LogisticRegression
```

```
# Create Logistic Regression model
lr_model = LogisticRegression(
    max_iter=1000,
    random_state=42
)

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

# Make predictions
y_pred_lr = lr_model.predict(
    X_test_tfidf
)
```

```
print("Logistic Regression model trained successfully!")
```

Logistic Regression model trained successfully!

```
In [36]: print ('ثم التقييم')
```

ثم التقييم:

```
In [37]: from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report
)

# Calculate metrics
accuracy_lr = accuracy_score(
    y_test,
    y_pred_lr
)

precision_lr = precision_score(
    y_test,
    y_pred_lr,
    zero_division=0
)

recall_lr = recall_score(
    y_test,
    y_pred_lr,
    zero_division=0
)

f1_lr = f1_score(
    y_test,
    y_pred_lr,
    zero_division=0
)

print("=====")
print("Logistic Regression - TF-IDF")
print("=====")

print("Accuracy :", round(accuracy_lr, 4))
print("Precision:", round(precision_lr, 4))
print("Recall   :", round(recall_lr, 4))
print("F1-Score :", round(f1_lr, 4))

print("\nClassification Report:")

print(
    classification_report(
        y_test,
        y_pred_lr,
        target_names=["Ham", "Spam"],
```

```

        zero_division=0
    )
)

```

```

=====
Logistic Regression - TF-IDF
=====

```

```

Accuracy : 0.9555
Precision: 0.9885
Recall    : 0.6565
F1-Score  : 0.789

```

Classification Report:

	precision	recall	f1-score	support
Ham	0.95	1.00	0.98	903
Spam	0.99	0.66	0.79	131
accuracy			0.96	1034
macro avg	0.97	0.83	0.88	1034
weighted avg	0.96	0.96	0.95	1034

```
In [38]: print ('الخطوة التالية: Bag of Words')
```

الخطوة التالية: Bag of Words

```
In [39]: from sklearn.feature_extraction.text import CountVectorizer
```

```

# Create Bag of Words vectorizer
bow = CountVectorizer(
    max_features=5000,
    ngram_range=(1, 2)
)

# Fit and transform training data
X_train_bow = bow.fit_transform(X_train)

# Transform testing data
X_test_bow = bow.transform(X_test)

print("Bag of Words Training Shape:", X_train_bow.shape)
print("Bag of Words Testing Shape:", X_test_bow.shape)

```

```

Bag of Words Training Shape: (4135, 5000)
Bag of Words Testing Shape: (1034, 5000)

```

```
In [40]: print ('1. تدريب Naive Bayes + BoW')
```

1. تدريب Naive Bayes + BoW

```
In [41]: from sklearn.naive_bayes import MultinomialNB
```

```

# Create model
nb_bow = MultinomialNB()

# Train model
nb_bow.fit(

```

```

        X_train_bow,
        y_train
    )

    # Predictions
    y_pred_nb_bow = nb_bow.predict(
        X_test_bow
    )

    print("Naive Bayes + Bag of Words trained successfully!")

```

Naive Bayes + Bag of Words trained successfully!

In [42]: `print ('2. تقييم النموذج')`

2. تقييم النموذج

```

In [43]: from sklearn.metrics import (
        accuracy_score,
        precision_score,
        recall_score,
        f1_score,
        classification_report
    )

    accuracy_nb_bow = accuracy_score(
        y_test,
        y_pred_nb_bow
    )

    precision_nb_bow = precision_score(
        y_test,
        y_pred_nb_bow,
        zero_division=0
    )

    recall_nb_bow = recall_score(
        y_test,
        y_pred_nb_bow,
        zero_division=0
    )

    f1_nb_bow = f1_score(
        y_test,
        y_pred_nb_bow,
        zero_division=0
    )

    print("=====")
    print("Naive Bayes - Bag of Words")
    print("=====")

    print("Accuracy :", round(accuracy_nb_bow, 4))
    print("Precision:", round(precision_nb_bow, 4))
    print("Recall   :", round(recall_nb_bow, 4))
    print("F1-Score :", round(f1_nb_bow, 4))

```

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

print(
    classification_report(
        y_test,
        y_pred_nb_bow,
        target_names=["Ham", "Spam"],
        zero_division=0
    )
)
```

```
=====
Naive Bayes - Bag of Words
=====
```

```
Accuracy : 0.9845
Precision: 0.96
Recall    : 0.916
F1-Score  : 0.9375
```

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

     Ham       0.99         0.99         0.99         903
     Spam       0.96         0.92         0.94         131

 accuracy          0.98         0.98         0.98         1034
  macro avg       0.97         0.96         0.96         1034
 weighted avg     0.98         0.98         0.98         1034
```

```
In [44]: print ('Logistic Regression + Bag of Words')
```

```
Logistic Regression + Bag of Words
```

```
In [45]: from sklearn.linear_model import LogisticRegression
```

```
# Create Logistic Regression model
lr_bow = LogisticRegression(
    max_iter=1000,
    random_state=42
)

# Train the model
lr_bow.fit(
    X_train_bow,
    y_train
)

# Predictions
y_pred_lr_bow = lr_bow.predict(
    X_test_bow
)

print("Logistic Regression + Bag of Words trained successfully!")
```

```
Logistic Regression + Bag of Words trained successfully!
```

```
In [46]: print ('ثم التقييم')
```

ثم التقييم:

```
In [47]: accuracy_lr_bow = accuracy_score(
    y_test,
    y_pred_lr_bow
)

precision_lr_bow = precision_score(
    y_test,
    y_pred_lr_bow,
    zero_division=0
)

recall_lr_bow = recall_score(
    y_test,
    y_pred_lr_bow,
    zero_division=0
)

f1_lr_bow = f1_score(
    y_test,
    y_pred_lr_bow,
    zero_division=0
)

print("=====")
print("Logistic Regression - Bag of Words")
print("=====")

print("Accuracy :", round(accuracy_lr_bow, 4))
print("Precision:", round(precision_lr_bow, 4))
print("Recall   :", round(recall_lr_bow, 4))
print("F1-Score :", round(f1_lr_bow, 4))

print("\nClassification Report:")

print(
    classification_report(
        y_test,
        y_pred_lr_bow,
        target_names=["Ham", "Spam"],
        zero_division=0
    )
)
```

=====

Logistic Regression - Bag of Words

=====

Accuracy : 0.9778
Precision: 0.9909
Recall : 0.8321
F1-Score : 0.9046

Classification Report:

	precision	recall	f1-score	support
Ham	0.98	1.00	0.99	903
Spam	0.99	0.83	0.90	131
accuracy			0.98	1034
macro avg	0.98	0.92	0.95	1034
weighted avg	0.98	0.98	0.98	1034

```
In [48]: print ('2. للنموذج الأفضل Confusion Matrix الآن نعمل')
```

2. للنموذج الأفضل Confusion Matrix الآن نعمل

```
In [49]: print ('Naive Bayes + Bag of Words')
```

Naive Bayes + Bag of Words

```
In [50]: from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Confusion Matrix
cm_best = confusion_matrix(
    y_test,
    y_pred_nb_bow
)

print("Confusion Matrix:")
print(cm_best)

# Display
disp = ConfusionMatrixDisplay(
    confusion_matrix=cm_best,
    display_labels=["Ham", "Spam"]
)

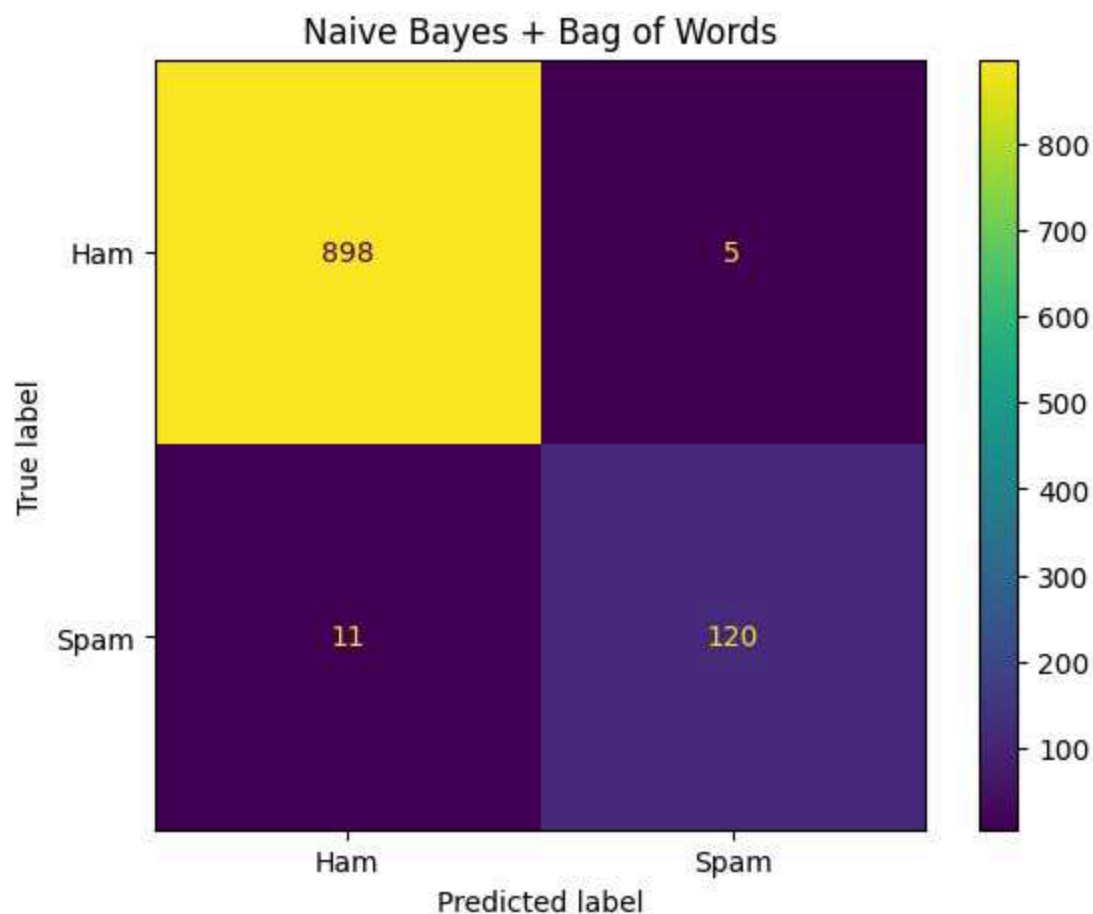
fig, ax = plt.subplots(figsize=(7, 5))

disp.plot(
    ax=ax
)

plt.title("Naive Bayes + Bag of Words")
plt.show()
```

Confusion Matrix:

```
[[898  5]
 [ 11 120]]
```



In [51]: `print ('3. حفظ النتائج النهائية')`

3. حفظ النتائج النهائية

```
In [52]: results = pd.DataFrame({
    "Model": [
        "Naive Bayes",
        "Logistic Regression",
        "Naive Bayes",
        "Logistic Regression"
    ],
    "Feature Representation": [
        "TF-IDF",
        "TF-IDF",
        "Bag of Words",
        "Bag of Words"
    ],
    "Accuracy": [
        accuracy_nb,
        accuracy_lr,
        accuracy_nb_bow,
        accuracy_lr_bow
    ],
    "Precision": [
        precision_nb,
```

```

        precision_lr,
        precision_nb_bow,
        precision_lr_bow
    ],

    "Recall": [
        recall_nb,
        recall_lr,
        recall_nb_bow,
        recall_lr_bow
    ],

    "F1-Score": [
        f1_nb,
        f1_lr,
        f1_nb_bow,
        f1_lr_bow
    ]
]
}))

# Round values
results[
    ["Accuracy", "Precision", "Recall", "F1-Score"]
] = results[
    ["Accuracy", "Precision", "Recall", "F1-Score"]
].round(4)

print(results)

# Save results
results.to_csv(
    "NLP_Model_Comparison.csv",
    index=False
)

print("\nResults saved as NLP_Model_Comparison.csv")

```

	Model	Feature Representation	Accuracy	Precision	Recall	\
0	Naive Bayes	TF-IDF	0.9691	0.9901	0.7634	
1	Logistic Regression	TF-IDF	0.9555	0.9885	0.6565	
2	Naive Bayes	Bag of Words	0.9845	0.9600	0.9160	
3	Logistic Regression	Bag of Words	0.9778	0.9909	0.8321	

	F1-Score
0	0.8621
1	0.7890
2	0.9375
3	0.9046

Results saved as NLP_Model_Comparison.csv

In [53]: `print('The Naive Bayes classifier using Bag of Words achieved the best overall perf`

The Naive Bayes classifier using Bag of Words achieved the best overall performance, with an accuracy of 98.45%, precision of 96.00%, recall of 91.60%, and F1-score of 93.75%.

```
In [54]: print ('تم انتهاء الجزء الثاني من المشروع الحمد لله')
```

تم انتهاء الجزء الثاني من المشروع الحمد لله

```
In [ ]:
```