

SMS Spam Classification Using NLP and Machine Learning

Dataset
(https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset)

Cell 2: Upload Dataset

from google.colab import files

uploaded = files.upload()

Choose Files

spam.csv

spam.csv(text/csv) - 503663 bytes, last modified: 8/18/2026 - 100% done

Saving spam.csv to spam (1).csv

Cell 3: Import Libraries

import pandas as pd

import numpy as np

import re

import nltk

import matplotlib.pyplot as plt

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

from nltk.stem import PorterStemmer

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

from sklearn.linear_model import LogisticRegression

from sklearn.naive_bayes import MultinomialNB

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import (

accuracy_score,

precision_score,

recall_score,

f1_score,

classification_report,

ConfusionMatrixDisplay

)

nltk.download('punkt')

nltk.download('punkt_tab')

nltk.download('stopwords')

[nltk_data] Downloading package punkt to /root/nltk_data...

[nltk_data] Package punkt is already up-to-date!

[nltk_data] Downloading package punkt_tab to /root/nltk_data...

[nltk_data] Package punkt_tab is already up-to-date!

[nltk_data] Downloading package stopwords to /root/nltk_data...

[nltk_data] Package stopwords is already up-to-date!

True

Cell 4: Load Dataset

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

df.head()

v1

v2

Unnamed: 2

Unnamed: 3

Unnamed: 4

0

ham

Go until jurong point, crazy.. Available only ...

NaN

NaN

NaN

1

ham

Ok lar... Joking wif u oni...

NaN

NaN

NaN

2

spam

Free entry in 2 a wkly comp to win FA Cup fina...

NaN

NaN

NaN

3

ham

U dun say so early hor... U c already then say...

NaN

NaN

NaN

4

ham

Nah I don't think he goes to usf, he lives aro...

NaN

NaN

NaN

Next steps:

Generate code with df

New interactive sheet

Cell 5: Check Dataset Shape

print("Number of rows:", df.shape[0])

print("Number of columns:", df.shape[1])

Number of rows: 5572

Number of columns: 5

Cell 6: Display Dataset Information

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 5572 entries, 0 to 5571

Data columns (total 5 columns):

Column Non-Null Count Dtype

0 v1 5572 non-null object

1 v2 5572 non-null object

2 Unnamed: 2 50 non-null object

3 Unnamed: 3 12 non-null object

4 Unnamed: 4 6 non-null object

dtypes: object(5)

memory usage: 217.8+ KB

Cell 7: Check Class Distribution

df['v1'].value_counts()

count

v1

ham 4825

spam 747

dtype: int64

Cell 8: Plot Class Distribution

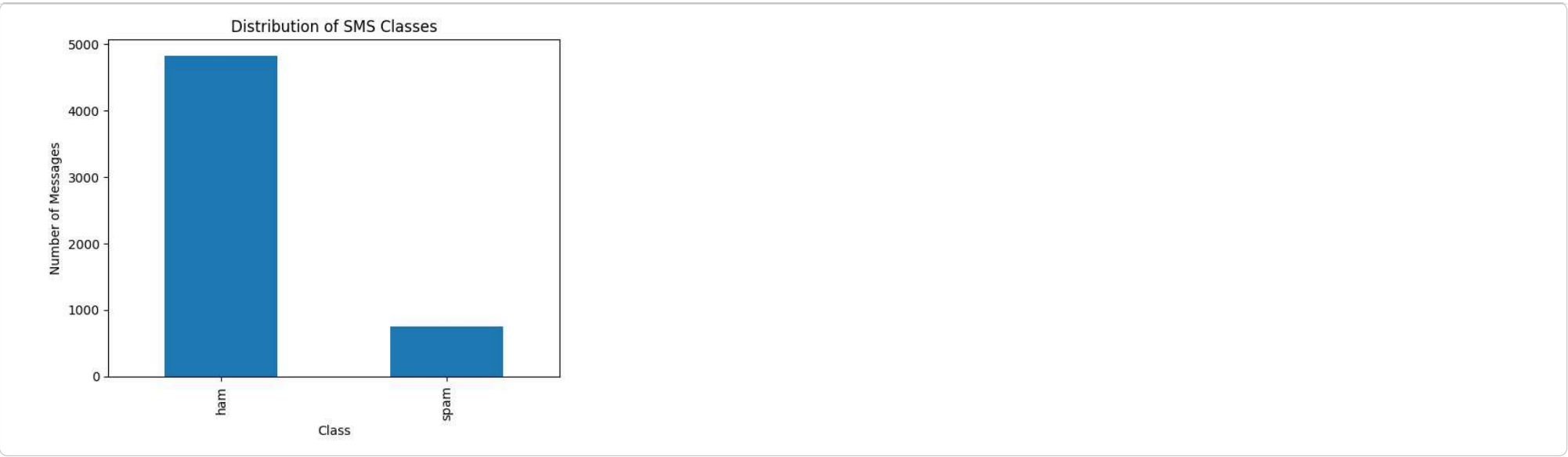
df['v1'].value_counts().plot(kind='bar')

plt.title("Distribution of SMS Classes")

plt.xlabel("Class")

plt.ylabel("Number of Messages")

plt.show()



```
# Cell 9: Remove Unnecessary Columns

df = df[['v1', 'v2']]

df.head()
```

	v1	v2
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...

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

```
# Cell 10: Rename Dataset Columns

df.columns = ['label', 'message']

df.head()
```

	label	message
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...

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

```
# Cell 11: Check Missing Values

df.isnull().sum()
```

	0
label	0
message	0

dtype: int64

```
# Cell 12: Remove Missing Values and Duplicates

df = df.dropna()
df = df.drop_duplicates()

print("Dataset shape after cleaning:", df.shape)
```

Dataset shape after cleaning: (5169, 2)

```
# Cell 13: Prepare NLP Preprocessing Tools

stemmer = PorterStemmer()

stop_words = set(stopwords.words('english'))
```

```
# Cell 14: Create Text Preprocessing Function

def preprocess_text(text):

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

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

    # Remove HTML tags
    text = re.sub(r'<.*?>', '', text)

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

    # Tokenization
    tokens = word_tokenize(text)

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

    # Apply stemming
    tokens = [stemmer.stem(word) for word in tokens]

    # Reconstruct cleaned text
    clean_text = ' '.join(tokens)

    return clean_text
```

```
# Cell 15: Apply Text Preprocessing

df['clean_message'] = df['message'].apply(preprocess_text)

df[['message', 'clean_message']].head()
```



```
lr_tfidf.fit(X_train_tfidf, y_train)

y_pred_lr_tfidf = lr_tfidf.predict(X_test_tfidf)

print("Logistic Regression + TF-IDF")
print(classification_report(y_test, y_pred_lr_tfidf))
```

Logistic Regression + TF-IDF				
	precision	recall	f1-score	support
ham	0.96	1.00	0.98	903
spam	1.00	0.70	0.83	131
accuracy			0.96	1034
macro avg	0.98	0.85	0.90	1034
weighted avg	0.96	0.96	0.96	1034

```
# Cell 24: Naive Bayes with TF-IDF

nb_tfidf = MultinomialNB()

nb_tfidf.fit(X_train_tfidf, y_train)

y_pred_nb_tfidf = nb_tfidf.predict(X_test_tfidf)

print("Naive Bayes + TF-IDF")
print(classification_report(y_test, y_pred_nb_tfidf))
```

Naive Bayes + TF-IDF				
	precision	recall	f1-score	support
ham	0.96	1.00	0.98	903
spam	0.99	0.73	0.84	131
accuracy			0.97	1034
macro avg	0.98	0.87	0.91	1034
weighted avg	0.97	0.97	0.96	1034

```
# Cell 25: Random Forest with TF-IDF

rf_tfidf = RandomForestClassifier(
    n_estimators=200,
    random_state=42
)

rf_tfidf.fit(X_train_tfidf, y_train)

y_pred_rf_tfidf = rf_tfidf.predict(X_test_tfidf)

print("Random Forest + TF-IDF")
print(classification_report(y_test, y_pred_rf_tfidf))
```

Random Forest + TF-IDF				
	precision	recall	f1-score	support
ham	0.97	1.00	0.99	903
spam	0.98	0.82	0.89	131
accuracy			0.97	1034
macro avg	0.98	0.91	0.94	1034
weighted avg	0.98	0.97	0.97	1034

```
# Cell 26: Create Model Evaluation Function
```

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

    accuracy = accuracy_score(y_true, y_pred)

    precision = precision_score(
        y_true,
        y_pred,
        pos_label='spam'
    )

    recall = recall_score(
        y_true,
        y_pred,
        pos_label='spam'
    )

    f1 = f1_score(
        y_true,
        y_pred,
        pos_label='spam'
    )

    return {
        'Model': model_name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1
    }
```

```
# Cell 27: Compare All Models
```

```
results = []

results.append(
    evaluate_model(
        "Logistic Regression + BOW",
        y_test,
        y_pred_lr_bow
    )
)

results.append(
    evaluate_model(
        "Naive Bayes + BOW",
        y_test,
        y_pred_nb_bow
    )
)

results.append(
    evaluate_model(
        "Random Forest + BOW",
        y_test,
        y_pred_rf_bow
    )
)

results.append(
    evaluate_model(
        "Logistic Regression + TF-IDF",
        y_test,
        y_pred_lr_tfidf
    )
)

results.append(
    evaluate_model(
        "Naive Bayes + TF-IDF",
```

```
        y_test,
        y_pred_nb_tfidf
    )
)

results.append(
    evaluate_model(
        "Random Forest + TF-IDF",
        y_test,
        y_pred_rf_tfidf
    )
)

results_df = pd.DataFrame(results)

results_df
```

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression + BOW	0.976789	1.000000	0.816794	0.899160
1	Naive Bayes + BOW	0.979691	0.936508	0.900763	0.918288
2	Random Forest + BOW	0.972921	0.981308	0.801527	0.882353
3	Logistic Regression + TF-IDF	0.962282	1.000000	0.702290	0.825112
4	Naive Bayes + TF-IDF	0.965184	0.989691	0.732824	0.842105
5	Random Forest + TF-IDF	0.974855	0.981651	0.816794	0.891667

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

Cell 28: Display Best Model Based on F1 Score

```
best_model = results_df.loc[
    results_df['F1 Score'].idxmax()
]

print("Best Model:")
print(best_model)
```

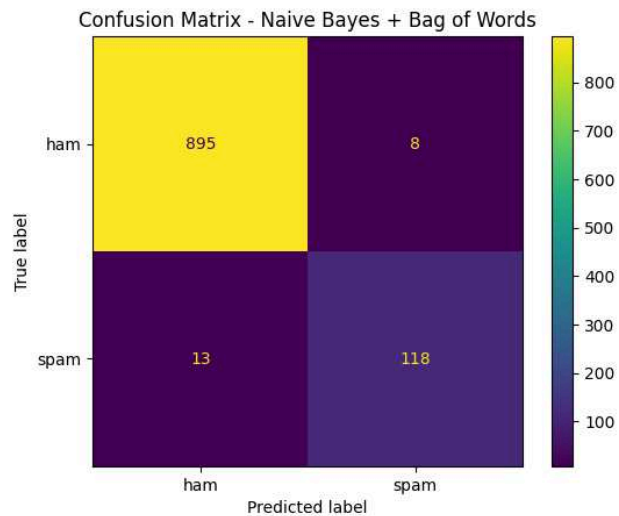
Best Model:
Model Naive Bayes + BOW
Accuracy 0.979691
Precision 0.936508
Recall 0.900763
F1 Score 0.918288
Name: 1, dtype: object

Cell 29: Plot Confusion Matrix

```
ConfusionMatrixDisplay.from_predictions(
    y_test,
    y_pred_nb_bow
)

plt.title("Confusion Matrix - Naive Bayes + Bag of Words")

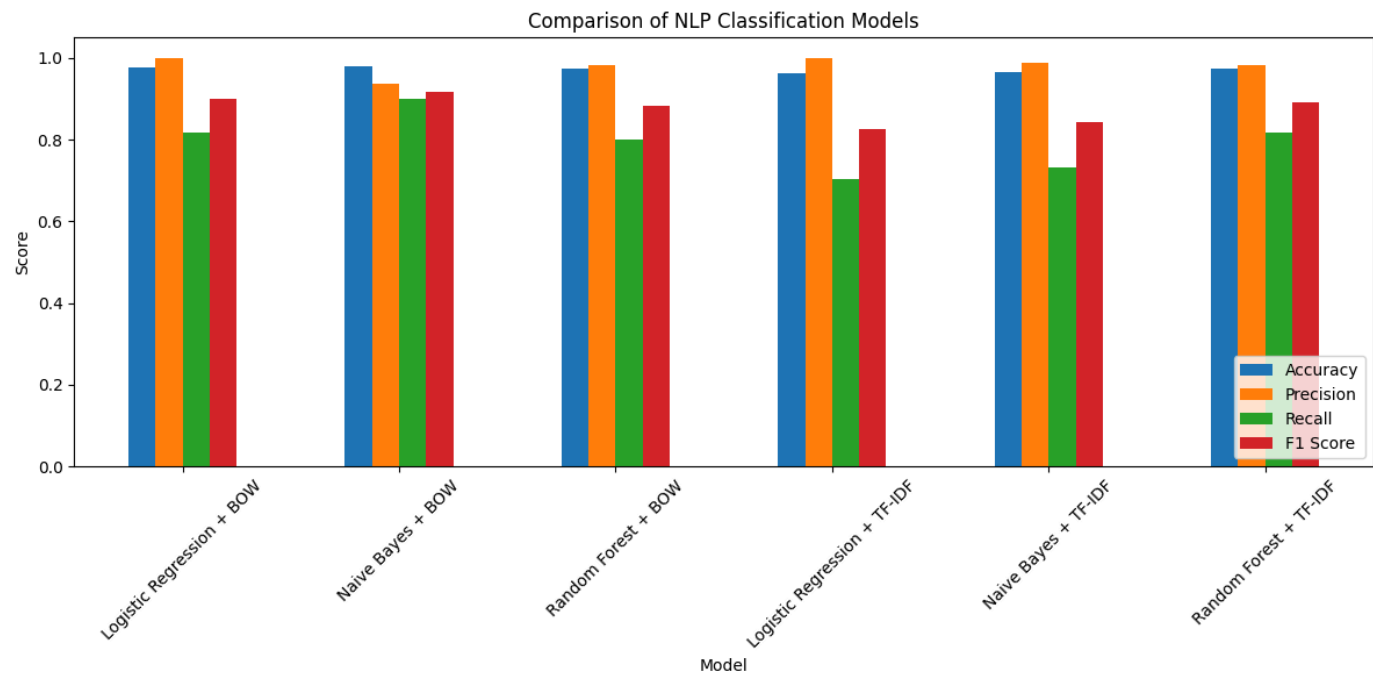
plt.show()
```



Cell 30: Plot Model Performance Comparison

```
results_df.set_index('Model')[
    ['Accuracy', 'Precision', 'Recall', 'F1 Score']
].plot(
    kind='bar',
    figsize=(12, 6)
)

plt.title("Comparison of NLP Classification Models")
plt.ylabel("Score")
plt.xlabel("Model")
plt.xticks(rotation=45)
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()
```



Conclusion

In this project, an SMS Spam Classification system was developed using Natural Language Processing and Machine Learning techniques.

The dataset was preprocessed by converting the messages to lowercase, removing URLs, HTML tags, special characters, and stopwords, followed by tokenization and stemming.

Two feature representation methods were applied: Bag of Words and TF-IDF.

Three machine learning algorithms were trained and evaluated: Logistic Regression, Multinomial Naive Bayes, and Random Forest.

The models were evaluated using Accuracy, Precision, Recall, F1 Score, and Classification Report. The model with the highest F1 Score was selected as the best model for spam detection.