

```

# =====
# PROJECT 2-2
# NLP CLASSIFICATION
# SMS SPAM DETECTION
# =====

# =====
# 1. IMPORT LIBRARIES
# =====

import pandas as pd
import numpy as np
import os
import re
import string

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB

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

# =====
# 2. FIND DATASET IN KAGGLE
# =====

print("Files available in Kaggle:")

for dirname, _, filenames in os.walk("/kaggle/input"):
    for filename in filenames:
        print(
            os.path.join(dirname, filename)
        )

# =====
# 3. FIND CSV FILE
# =====

csv_files = []

for dirname, _, filenames in os.walk("/kaggle/input"):
    for filename in filenames:
        if filename.lower().endswith(".csv"):
            csv_files.append(
                os.path.join(dirname, filename)
            )

print("\nCSV files found:")

for file in csv_files:
    print(file)

```

```
if len(csv_files) == 0:

    raise FileNotFoundError(
        "No CSV file found in /kaggle/input/"
    )

# =====
# 4. FIND spam.csv
# =====

dataset_path = None

for file in csv_files:

    if "spam" in os.path.basename(file).lower():

        dataset_path = file

        break

# If spam.csv was not found, use first CSV

if dataset_path is None:

    dataset_path = csv_files[0]

print("\nUsing dataset:")

print(dataset_path)

# =====
# 5. LOAD DATASET
# =====

df = pd.read_csv(

    dataset_path,

    encoding="latin-1"

)

# =====
# 6. DISPLAY DATASET
# =====

print("\nDataset Shape:")

print(
    df.shape
)

print("\nColumn Names:")

print(
    df.columns.tolist()
)

print("\nFirst 5 Rows:")

display(
    df.head()
)

# =====
# 7. DATASET INFORMATION
# =====

print("\nDataset Information:")
```

```

df.info()

print("\nMissing Values:")

print(
    df.isnull().sum()
)

print("\nDuplicate Rows:")

print(
    df.duplicated().sum()
)

# =====
# 8. SELECT REQUIRED COLUMNS
# =====

# The original SMS Spam dataset usually contains:
#
# v1 = label
# v2 = message
#
# Other columns are unnecessary.

if "v1" in df.columns and "v2" in df.columns:

    df = df[
        [
            "v1",
            "v2"
        ]
    ].copy()

    df.columns = [
        "Label",
        "Message"
    ]

else:

    # Try to automatically identify columns

    print("\nAvailable columns:")

    print(
        df.columns.tolist()
    )

    raise ValueError(
        "The expected v1 and v2 columns were not found."
    )

# =====
# 9. REMOVE MISSING VALUES
# =====

df = df.dropna()

print(
    "\nShape after removing missing values:"
)

print(
    df.shape
)

# =====
# 10. REMOVE DUPLICATES
# =====

```

```
df = df.drop_duplicates()

print(
    "\nShape after removing duplicates:"
)

print(
    df.shape
)

# =====
# 11. CHECK CLASS DISTRIBUTION
# =====

print(
    "\nSpam / Ham Distribution:"
)

print(
    df["Label"].value_counts()
)

# =====
# 12. VISUALIZE CLASS DISTRIBUTION
# =====

plt.figure(
    figsize=(7, 5)
)

sns.countplot(
    data=df,
    x="Label"
)

plt.title(
    "SMS Spam and Ham Distribution"
)

plt.xlabel(
    "Message Type"
)

plt.ylabel(
    "Number of Messages"
)

plt.show()

# =====
# 13. TEXT PREPROCESSING
# =====

def clean_text(text):

    # Convert to string

    text = str(text)

    # Convert to lowercase

    text = text.lower()

    # Remove URLs
```

```
text = re.sub(
    r"http\S+|www\S+|https\S+",
    " ",
    text
)

# Remove email addresses
text = re.sub(
    r"\S+@\S+",
    " ",
    text
)

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

# Remove punctuation
text = text.translate(
    str.maketrans(
        "",
        "",
        string.punctuation
    )
)

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

return text

# Apply preprocessing
df["Clean_Message"] = df["Message"].apply(
    clean_text
)
```

```
# =====
# 14. DISPLAY PREPROCESSING EXAMPLE
# =====

print(
    "\nOriginal Message:"
)

print(
    df["Message"].iloc[0]
)

print(
    "\nCleaned Message:"
)

print(
    df["Clean_Message"].iloc[0]
)

# =====
# 15. ENCODE LABELS
# =====

# ham = 0
# spam = 1

df["Encoded_Label"] = df["Label"].map({

    "ham": 0,

    "spam": 1

})

print(
    "\nEncoded Labels:"
)

print(
    df[
        [
            "Label",
            "Encoded_Label"
        ]
    ].drop_duplicates()
)

# =====
# 16. PREPARE FEATURES AND TARGET
# =====

X = df["Clean_Message"]

y = df["Encoded_Label"]

# =====
# 17. SPLIT DATASET
# =====

X_train, X_test, y_train, y_test = train_test_split(

    X,

    y,

    test_size=0.20,

    random_state=42,
```

```
stratify=y
)

print(
    "\nTraining Samples:"
)

print(
    len(X_train)
)

print(
    "\nTesting Samples:"
)

print(
    len(X_test)
)

# =====
# 18. TF-IDF FEATURE REPRESENTATION
# =====

tfidf = TfidfVectorizer(

    max_features=8000,

    ngram_range=(1, 2),

    min_df=1,

    max_df=0.95,

    sublinear_tf=True
)

# Fit on training data

X_train_tfidf = tfidf.fit_transform(

    X_train
)

# Transform testing data

X_test_tfidf = tfidf.transform(

    X_test
)

print(
    "\nTF-IDF Training Shape:"
)

print(
    X_train_tfidf.shape
)

print(
    "\nTF-IDF Testing Shape:"
)

print(
    X_test_tfidf.shape
)
```

```
# =====
# 19. MODEL 1 - LOGISTIC REGRESSION
# =====

lr_model = LogisticRegression(

    max_iter=1000,

    random_state=42

)

lr_model.fit(

    X_train_tfidf,

    y_train

)

lr_pred = lr_model.predict(

    X_test_tfidf

)

# =====
# 20. LOGISTIC REGRESSION EVALUATION
# =====

lr_accuracy = accuracy_score(

    y_test,

    lr_pred

)

lr_precision = precision_score(

    y_test,

    lr_pred,

    zero_division=0

)

lr_recall = recall_score(

    y_test,

    lr_pred,

    zero_division=0

)

lr_f1 = f1_score(

    y_test,

    lr_pred,

    zero_division=0

)

print(

    "\n=====
```

```
)

print(
    "LOGISTIC REGRESSION RESULTS"
)

print(
    "=====
"
)

print(
    "Accuracy :",
    round(lr_accuracy, 4)
)

print(
    "Precision:",
    round(lr_precision, 4)
)

print(
    "Recall  :",
    round(lr_recall, 4)
)

print(
    "F1 Score :",
    round(lr_f1, 4)
)

print(
    "\nClassification Report:"
)

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

# =====
# 21. CONFUSION MATRIX
# =====

cm_lr = confusion_matrix(
    y_test,
    lr_pred
)

plt.figure(
    figsize=(7, 5)
)
```

```
sns.heatmap(

    cm_lr,

    annot=True,

    fmt="d",

    cmap="Blues",

    xticklabels=[
        "Ham",
        "Spam"
    ],

    yticklabels=[
        "Ham",
        "Spam"
    ]

)

plt.title(
    "Confusion Matrix - Logistic Regression"
)

plt.xlabel(
    "Predicted Label"
)

plt.ylabel(
    "Actual Label"
)

plt.show()

# =====
# 22. MODEL 2 - NAIVE BAYES
# =====

nb_model = MultinomialNB()

nb_model.fit(

    X_train_tfidf,

    y_train

)

nb_pred = nb_model.predict(

    X_test_tfidf

)

# =====
# 23. NAIVE BAYES EVALUATION
# =====

nb_accuracy = accuracy_score(

    y_test,

    nb_pred

)
```

```
nb_precision = precision_score(
    y_test,
    nb_pred,
    zero_division=0
)

nb_recall = recall_score(
    y_test,
    nb_pred,
    zero_division=0
)

nb_f1 = f1_score(
    y_test,
    nb_pred,
    zero_division=0
)

print(
    "\n======"
)

print(
    "NAIVE BAYES RESULTS"
)

print(
    "======"
)

print(
    "Accuracy :",
    round(nb_accuracy, 4)
)

print(
    "Precision:",
    round(nb_precision, 4)
)

print(
    "Recall  :",
    round(nb_recall, 4)
)

print(
    "F1 Score :",
    round(nb_f1, 4)
)

print(
    "\nClassification Report:"
)

print(
    classification_report(
```

```
        y_test,

        nb_pred,

        target_names=[
            "Ham",
            "Spam"
        ],

        zero_division=0

    )
)

# =====
# 24. NAIVE BAYES CONFUSION MATRIX
# =====

cm_nb = confusion_matrix(

    y_test,

    nb_pred

)

plt.figure(

    figsize=(7, 5)

)

sns.heatmap(

    cm_nb,

    annot=True,

    fmt="d",

    cmap="Greens",

    xticklabels=[
        "Ham",
        "Spam"
    ],

    yticklabels=[
        "Ham",
        "Spam"
    ]

)

plt.title(
    "Confusion Matrix - Naive Bayes"
)

plt.xlabel(
    "Predicted Label"
)

plt.ylabel(
    "Actual Label"
)

plt.show()

# =====
```

```

# 25. MODEL COMPARISON
# =====

results = pd.DataFrame({

    "Model": [

        "Logistic Regression",

        "Naive Bayes"

    ],

    "Accuracy": [

        lr_accuracy,

        nb_accuracy

    ],

    "Precision": [

        lr_precision,

        nb_precision

    ],

    "Recall": [

        lr_recall,

        nb_recall

    ],

    "F1 Score": [

        lr_f1,

        nb_f1

    ]

})

print(
    "\n=====
")

print(
    "MODEL COMPARISON"
)

print(
    "=====
")

display(
    results
)

# =====
# 26. PERFORMANCE VISUALIZATION
# =====

results_plot = results.set_index(

    "Model"

)

```

```
results_plot.plot(

    kind="bar",

    figsize=(10, 6)

)

plt.title(
    "SMS Spam Classification Model Comparison"
)

plt.ylabel(
    "Score"
)

plt.xlabel(
    "Model"
)

plt.ylim(
    0,
    1.05
)

plt.xticks(
    rotation=0
)

plt.grid(

    axis="y",

    alpha=0.3

)

plt.show()

# =====
# 27. TEST NEW SMS MESSAGES
# =====

new_messages = [

    "Congratulations! You have won a free prize. Click here now!",

    "Hey, are we still meeting for lunch today?",

    "URGENT! You have won $1000. Call this number immediately!",

    "Can you send me the lecture notes when you have time?",

    "Claim your free vacation now by clicking this link!"

]

# Clean messages

new_messages_clean = [

    clean_text(message)

    for message in new_messages

]

# Convert to TF-IDF
```

```
new_messages_tfidf = tfidf.transform(
    new_messages_clean
)

# Predict using Logistic Regression
predictions = lr_model.predict(
    new_messages_tfidf
)

# =====
# 28. DISPLAY PREDICTIONS
# =====

print(
    "\n=====
"
)

print(
    "NEW SMS PREDICTIONS"
)

print(
    "=====
"
)

for message, prediction in zip(
    new_messages,
    predictions
):
    if prediction == 1:
        result = "SPAM"
    else:
        result = "HAM"

    print(
        "\nMessage:"
    )

    print(
        message
    )

    print(
        "Predicted:",
        result
    )

# =====
# 29. SAVE RESULTS
# =====

results.to_csv(
    "/kaggle/working/sms_spam_model_results.csv",
    index=False
)
```

```
print(
    "\nResults saved successfully!"
)

print(
    "/kaggle/working/sms_spam_model_results.csv"
)
```

