

## Project 2: NLP Text Classification — SMS Spam Detection Hala Hajjar

```
# Project 2: NLP Text Classification - SMS Spam Detection

# 1. Import Required Libraries

import pandas as pd
import numpy as np
import re
import string
import matplotlib.pyplot as plt

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.svm import LinearSVC

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

print("Libraries imported successfully!")
```

Libraries imported successfully!

```
# 2. Download the NLP Dataset

url = "https://raw.githubusercontent.com/justmarkham/pycon-2016-tutorial/master/data,"

df = pd.read_csv(
    url,
    sep="\t",
    header=None,
    names=["label", "message"]
)

print("\nDataset loaded successfully!")
print("Dataset Shape:", df.shape)

print("\nFirst 5 Rows:")
```

```
print(df.head())

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

print("\nMissing Values:")
print(df.isnull().sum())
```

Dataset loaded successfully!  
Dataset Shape: (5572, 2)

First 5 Rows:

|   | 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... |

Class Distribution:

| label |      |
|-------|------|
| ham   | 4825 |
| spam  | 747  |

Name: count, dtype: int64

Missing Values:

| label   | 0 |
|---------|---|
| message | 0 |

dtype: int64

# 3. Data Preprocessing

# Convert labels:

```
# ham = 0
# spam = 1
```

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

# Function to clean text

```
def clean_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
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)

print("\nData Preprocessing Completed!")

print("\nOriginal and Cleaned Messages:")
print(df[["message", "clean_message"]].head())

```

Data Preprocessing Completed!

Original and Cleaned Messages:

|   | message \                                         | clean_message                                     |
|---|---------------------------------------------------|---------------------------------------------------|
| 0 | Go until jurong point, crazy.. Available only ... | go until jurong point crazy available only in ... |
| 1 | Ok lar... Joking wif u oni...                     | ok lar joking wif u oni                           |
| 2 | Free entry in 2 a wkly comp to win FA Cup fina... | free entry in a wkly comp to win fa cup final ... |
| 3 | U dun say so early hor... U c already then say... | u dun say so early hor u c already then say       |
| 4 | Nah I don't think he goes to usf, he lives aro... | nah i dont think he goes to usf he lives aroun... |

```

# 4. Split the Dataset
# 80% Training
# 10% Validation
# 10% Testing

```

```

X = df["clean_message"]
y = df["label"]

```

```

# First split:
# 80% Training
# 20% Temporary
X_train, X_temp, y_train, y_temp = train_test_split(
    X,
    y,
    test_size=0.20,

```

```
        random_state=42,  
        stratify=y  
    )  
  
    # Divide remaining 20% into:  
    # 10% Validation  
    # 10% Testing  
    X_val, X_test, y_val, y_test = train_test_split(  
        X_temp,  
        y_temp,  
        test_size=0.50,  
        random_state=42,  
        stratify=y_temp  
    )  
  
    print("\nDataset Split Completed Successfully!")  
  
    print("Training Samples:", len(X_train))  
    print("Validation Samples:", len(X_val))  
    print("Testing Samples:", len(X_test))
```

```
Dataset Split Completed Successfully!  
Training Samples: 4457  
Validation Samples: 557  
Testing Samples: 558
```

## ✓ 5. Feature Representation Method 1

### Bag of Words (BoW)

```
bow_vectorizer = CountVectorizer(  
    max_features=5000,  
    stop_words="english"  
)  
  
X_train_bow = bow_vectorizer.fit_transform(X_train)  
X_val_bow = bow_vectorizer.transform(X_val)  
X_test_bow = bow_vectorizer.transform(X_test)  
  
print("\nBag of Words Completed!")  
print("BoW Training Shape:", X_train_bow.shape)
```

```
Bag of Words Completed!  
BoW Training Shape: (4457, 5000)
```

```
# 6. Feature Representation Method 2
# TF-IDF

tfidf_vectorizer = TfidfVectorizer(
    max_features=5000,
    stop_words="english"
)

X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_val_tfidf = tfidf_vectorizer.transform(X_val)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

print("\nTF-IDF Completed!")
print("TF-IDF Training Shape:", X_train_tfidf.shape)
```

```
TF-IDF Completed!
TF-IDF Training Shape: (4457, 5000)
```

```
# 7. Function for Training and Evaluation

def evaluate_model(model, X_train_data, X_test_data,
                  y_train_data, y_test_data, model_name):

    # Train the model
    model.fit(X_train_data, y_train_data)

    # Predict testing data
    y_pred = model.predict(X_test_data)

    # Calculate evaluation metrics
    accuracy = accuracy_score(y_test_data, y_pred)
    precision = precision_score(y_test_data, y_pred)
    recall = recall_score(y_test_data, y_pred)
    f1 = f1_score(y_test_data, y_pred)

    print("\n" + "=" * 60)
    print(model_name)
    print("=" * 60)

    print("Accuracy :", round(accuracy, 4))
    print("Precision:", round(precision, 4))
    print("Recall   :", round(recall, 4))
    print("F1 Score :", round(f1, 4))
```

```

print("\nClassification Report:")
print(
    classification_report(
        y_test_data,
        y_pred,
        target_names=["Ham", "Spam"]
    )
)

return accuracy, precision, recall, f1, y_pred

```

#### # 8. Train Logistic Regression

```

lr_model = LogisticRegression(
    max_iter=1000
)

lr_results = evaluate_model(
    lr_model,
    X_train_tfidf,
    X_test_tfidf,
    y_train,
    y_test,
    "Logistic Regression - TF-IDF"
)

```

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

Accuracy : 0.9677  
 Precision: 1.0  
 Recall : 0.76  
 F1 Score : 0.8636

#### Classification Report:

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| Ham          | 0.96      | 1.00   | 0.98     | 483     |
| Spam         | 1.00      | 0.76   | 0.86     | 75      |
| accuracy     |           |        | 0.97     | 558     |
| macro avg    | 0.98      | 0.88   | 0.92     | 558     |
| weighted avg | 0.97      | 0.97   | 0.97     | 558     |

#### # 9. Train Naive Bayes

```
nb_model = MultinomialNB()

nb_results = evaluate_model(
    nb_model,
    X_train_tfidf,
    X_test_tfidf,
    y_train,
    y_test,
    "Naive Bayes - TF-IDF"
)
```

```
=====
Naive Bayes - TF-IDF
=====
Accuracy : 0.9695
Precision: 1.0
Recall   : 0.7733
F1 Score : 0.8722

Classification Report:
              precision    recall  f1-score   support

     Ham       0.97         1.00         0.98         483
     Spam       1.00         0.77         0.87          75

 accuracy                0.97         558
 macro avg              0.98         0.89         0.93         558
 weighted avg           0.97         0.97         0.97         558
```

# 10. Train Support Vector Machine (SVM)

```
svm_model = LinearSVC()

svm_results = evaluate_model(
    svm_model,
    X_train_tfidf,
    X_test_tfidf,
    y_train,
    y_test,
    "SVM - TF-IDF"
)
```

```
=====
SVM - TF-IDF
=====
```

Accuracy : 0.9839  
 Precision: 0.9714  
 Recall : 0.9067  
 F1 Score : 0.9379

#### Classification Report:

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| Ham          | 0.99      | 1.00   | 0.99     | 483     |
| Spam         | 0.97      | 0.91   | 0.94     | 75      |
| accuracy     |           |        | 0.98     | 558     |
| macro avg    | 0.98      | 0.95   | 0.96     | 558     |
| weighted avg | 0.98      | 0.98   | 0.98     | 558     |

#### # 11. Compare Model Performance

```
results = pd.DataFrame({
    "Model": [
        "Logistic Regression",
        "Naive Bayes",
        "SVM"
    ],
    "Accuracy": [
        lr_results[0],
        nb_results[0],
        svm_results[0]
    ],
    "Precision": [
        lr_results[1],
        nb_results[1],
        svm_results[1]
    ],
    "Recall": [
        lr_results[2],
        nb_results[2],
        svm_results[2]
    ],
    "F1 Score": [
        lr_results[3],
        nb_results[3],
        svm_results[3]
    ]
})
```

```
print("\n" + "-" * 60)
```

```
print("\n" + "=" * 60)
print("MODEL COMPARISON")
print("=" * 60)

print(results.round(4))

# 12. Find the Best Model

best_index = results["Accuracy"].idxmax()
best_model_name = results.loc[best_index, "Model"]
best_accuracy = results.loc[best_index, "Accuracy"]

print("\nBest Model:", best_model_name)
print("Best Accuracy:", round(best_accuracy, 4))

# 13. Confusion Matrix for SVM

ConfusionMatrixDisplay.from_predictions(
    y_test,
    svm_results[4],
    display_labels=["Ham", "Spam"]
)

plt.title("SVM Confusion Matrix")
plt.show()

# 14. Test the Model Using a New Message

new_message = [
    "Congratulations! You won a free prize. Call now to claim your reward."
]

clean_new_message = [
    clean_text(new_message[0])
]

new_message_vector = tfidf_vectorizer.transform(
    clean_new_message
)

prediction = svm_model.predict(
    new_message_vector
)

print("\n" + "=" * 60)
print("NEW MESSAGE PREDICTION")
```

```
print( = " 60")

print("Message:")
print(new_message[0])

if prediction[0] == 1:
    print("\nPrediction: SPAM")
else:
    print("\nPrediction: HAM")

print("\nProject 2 Completed Successfully!")
```

## =====

## MODEL COMPARISON

|   | Model               | Accuracy | Precision | Recall | F1 Score |
|---|---------------------|----------|-----------|--------|----------|
| 0 | Logistic Regression | 0.9677   | 1.0000    | 0.7600 | 0.8636   |
| 1 | Naive Bayes         | 0.9695   | 1.0000    | 0.7733 | 0.8722   |
| 2 | SVM                 | 0.9839   | 0.9714    | 0.9067 | 0.9379   |

Best Model: SVM

Best Accuracy: 0.9839

