

Project 2:NLP Text Classification: SMS Spam Detection

Student Name: Marya Mohammed Bawazir .

Student ID: 253000692

Course:Practical_Image_Processing_and_Natural_Language_Processing_Second Trimester 2026

Project Workflow:

1. Dataset Download and Exploration
2. Data Preprocessing
3. Train-Test Split
4. Feature Representation using BoW and TF-IDF
5. Model Training
6. Model Evaluation and Comparison
7. Best Model Evaluation
8. Prediction on New SMS Messages

italicized text

```
import os
import re
import zipfile
import urllib.request
import warnings

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

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    confusion_matrix,
    ConfusionMatrixDisplay
)

warnings.filterwarnings("ignore")
```

```

RANDOM_STATE = 42
print("Libraries imported successfully.")

Libraries imported successfully.

# 2. Download and Load the Dataset

# Official UCI SMS Spam Collection Dataset
dataset_url =
"https://archive.ics.uci.edu/static/public/228/sms+spam+collection.zip
"

zip_path = "sms_spam_collection.zip"
extract_dir = "sms_spam_data"

# Download the dataset
if not os.path.exists(zip_path):
    urllib.request.urlretrieve(dataset_url, zip_path)
    print("Dataset downloaded successfully.")
else:
    print("Dataset ZIP file already exists.")

# Extract the ZIP file
os.makedirs(extract_dir, exist_ok=True)

with zipfile.ZipFile(zip_path, "r") as zip_ref:
    zip_ref.extractall(extract_dir)

# Read the dataset
data_path = os.path.join(extract_dir, "SMSSpamCollection")

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

# Display dataset information
print("Dataset shape:", df.shape)

df.head()

Dataset downloaded successfully.
Dataset shape: (5572, 2)

{"summary": "{\n  \"name\": \"df\", \n  \"rows\": 5572, \n  \"fields\": [\n    {\n      \"column\": \"label\", \n      \"properties\": {\n        \"dtype\": \"category\", \n        \"num_unique_values\": 2, \n        \"samples\": [\n          \"spam\", \n          \"ham\" \n        ], \n

```

```

{"semantic_type": "\n", "description": "\n", "column": "message", "properties": {"dtype": "string", "num_unique_values": 5169, "samples": ["K, makes sense, btw carlos is being difficult so you guys are gonna smoke while I go pick up the second batch and get gas", "URGENT! Your mobile No ***** WON a \u00a32,000 Bonus Caller Prize on 02/06/03! This is the 2nd attempt to reach YOU! Call 09066362220 ASAP! BOX97N7QP, 150ppm"]}, "type": "dataframe", "variable_name": "df"}

```

3. Explore the Dataset

Display general information about the dataset

```

print("Dataset Information:")
df.info()

```

Check for missing values

```

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

```

Check for duplicate rows

```

print("\nDuplicate Rows:")
print(df.duplicated().sum())

```

Display class distribution

```

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

```

```

Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  -
 0   label      5572 non-null   object
 1   message    5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB

```

Missing Values:

```

label      0
message    0
dtype: int64

```

Duplicate Rows:

```

403

```

Class Distribution:

```

label
ham      4825
spam      747
Name: count, dtype: int64

# 4. Data Preprocessing

# Remove duplicate rows
df = df.drop_duplicates().reset_index(drop=True)

# Convert labels to numerical values
# ham = 0, spam = 1
df["target"] = df["label"].map({
    "ham": 0,
    "spam": 1
})

# Function for cleaning text
def clean_text(text):
    # Convert text to lowercase
    text = text.lower()

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

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

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

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

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

    return text

# Apply preprocessing
df["clean_message"] = df["message"].apply(clean_text)

print("Dataset shape after removing duplicates:", df.shape)

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

# Display original and cleaned messages
df[["label", "message", "clean_message", "target"]].head(10)

```

Dataset shape after removing duplicates: (5169, 4)

Class Distribution After Preprocessing:

label

ham 4516

spam 653

Name: count, dtype: int64

```
{"summary":{"name": "df[[\\\\"label\\\\"], \\\\\"message\\\\",\n\\\\"clean_message\\\\"], \\\\\"target\\\\"]]\\",\n  "rows": 10,\n  "fields": [\n    {\n      "column": "label",\n      "properties": {\n        "dtype": "category",\n        "num_unique_values": 2,\n        "samples": [\n          "spam",\n          "ham"\n        ],\n        "semantic_type": "\n",\n        "description": "\n" }\n      },\n    {\n      "column": "message",\n      "properties": {\n        "dtype": "string",\n        "num_unique_values": 10,\n        "samples": [\n          "WINNER!! As a valued network customer you have been\nselected to receivea \u00a3900 prize reward! To claim call\n09061701461. Claim code KL341. Valid 12 hours only.\n",\n          "Ok\nlar... Joking wif u oni...\n"],\n        "semantic_type": "\n",\n        "description": "\n" }\n      },\n    {\n      "column": "clean_message",\n      "properties": {\n        "dtype": "string",\n        "num_unique_values": 10,\n        "samples": [\n          "winner as a valued network customer you\nhave been selected to receivea prize reward to claim call claim code\nkl valid hours only\n",\n          "ok lar joking wif u oni\n"],\n        "semantic_type": "\n",\n        "description": "\n" }\n      },\n    {\n      "column": "target",\n      "properties": {\n        "dtype": "number",\n        "std": 0,\n        "min": 0,\n        "max": 1,\n        "num_unique_values": 2,\n        "samples": [\n          1,\n          0\n        ],\n        "semantic_type": "\n",\n        "description": "\n" }\n      }\n    ]\n  },\n  "type": "dataframe"}
```

5. Split the Dataset

Features and target

X = df["clean_message"]

y = df["target"]

Split the dataset into 80% training and 20% testing

```
X_train, X_test, y_train, y_test = train_test_split(\n    X,\n    y,\n    test_size=0.20,\n    random_state=42,\n    stratify=y\n)
```

```

# Display the sizes
print("Total samples:", len(df))
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())

```

```

Total samples: 5169
Training samples: 4135
Testing samples: 1034

```

```

Training Class Distribution:
target
0      3613
1       522
Name: count, dtype: int64

```

```

Testing Class Distribution:
target
0       903
1       131
Name: count, dtype: int64

```

6. Feature Representation Methods

```

# -----
# Method 1: Bag of Words (BoW)
# -----

```

```

bow_vectorizer = CountVectorizer(
    stop_words="english",
    ngram_range=(1, 2),
    min_df=2
)

X_train_bow = bow_vectorizer.fit_transform(X_train)
X_test_bow = bow_vectorizer.transform(X_test)

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

```

```

# -----
# Method 2: TF-IDF
# -----

```

```

tfidf_vectorizer = TfidfVectorizer(
    stop_words="english",
    ngram_range=(1, 2),
    min_df=2,
    sublinear_tf=True
)

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

print("\nTF-IDF:")
print("Training shape:", X_train_tfidf.shape)
print("Testing shape:", X_test_tfidf.shape)

Bag of Words:
Training shape: (4135, 5240)
Testing shape: (1034, 5240)

TF-IDF:
Training shape: (4135, 5240)
Testing shape: (1034, 5240)

# 7. Train and Evaluate Models

# Define the models
models = {
    "Multinomial Naive Bayes": MultinomialNB(),

    "Logistic Regression": LogisticRegression(
        max_iter=1000,
        class_weight="balanced",
        random_state=42
    ),

    "Linear SVM": LinearSVC(
        class_weight="balanced",
        random_state=42
    )
}

# Feature representation methods
feature_sets = {
    "Bag of Words": (X_train_bow, X_test_bow),
    "TF-IDF": (X_train_tfidf, X_test_tfidf)
}

# Store results
results = []

# Train and evaluate each model
for feature_name, (X_tr, X_te) in feature_sets.items():

```

```

for model_name, model in models.items():

    # Train the model
    model.fit(X_tr, y_train)

    # Make predictions
    y_pred = model.predict(X_te)

    # Calculate evaluation metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, zero_division=0)
    recall = recall_score(y_test, y_pred, zero_division=0)
    f1 = f1_score(y_test, y_pred, zero_division=0)

    # Save results
    results.append({
        "Feature Representation": feature_name,
        "Model": model_name,
        "Accuracy": accuracy,
        "Precision": precision,
        "Recall": recall,
        "F1-Score": f1
    })

# Convert results to DataFrame
results_df = pd.DataFrame(results)

# Sort according to F1-score
results_df = results_df.sort_values(
    by="F1-Score",
    ascending=False
).reset_index(drop=True)

# Display results as percentages
display(
    results_df.style.format({
        "Accuracy": "{:.2%}",
        "Precision": "{:.2%}",
        "Recall": "{:.2%}",
        "F1-Score": "{:.2%}"
    })
)

<pandas.io.formats.style.Styler at 0x782bd691ed20>

# 8. Evaluate the Best Model

# Best model: Bag of Words + Multinomial Naive Bayes
best_model = MultinomialNB()

```

```

# Train the best model
best_model.fit(X_train_bow, y_train)

# Make predictions
y_pred_best = best_model.predict(X_test_bow)

# Display the classification report
print("Best Model: Bag of Words + Multinomial Naive Bayes")
print("\nClassification Report:\n")

print(
    classification_report(
        y_test,
        y_pred_best,
        target_names=["Ham", "Spam"],
        digits=4
    )
)

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred_best)

disp = ConfusionMatrixDisplay(
    confusion_matrix=cm,
    display_labels=["Ham", "Spam"]
)

disp.plot(values_format="d")

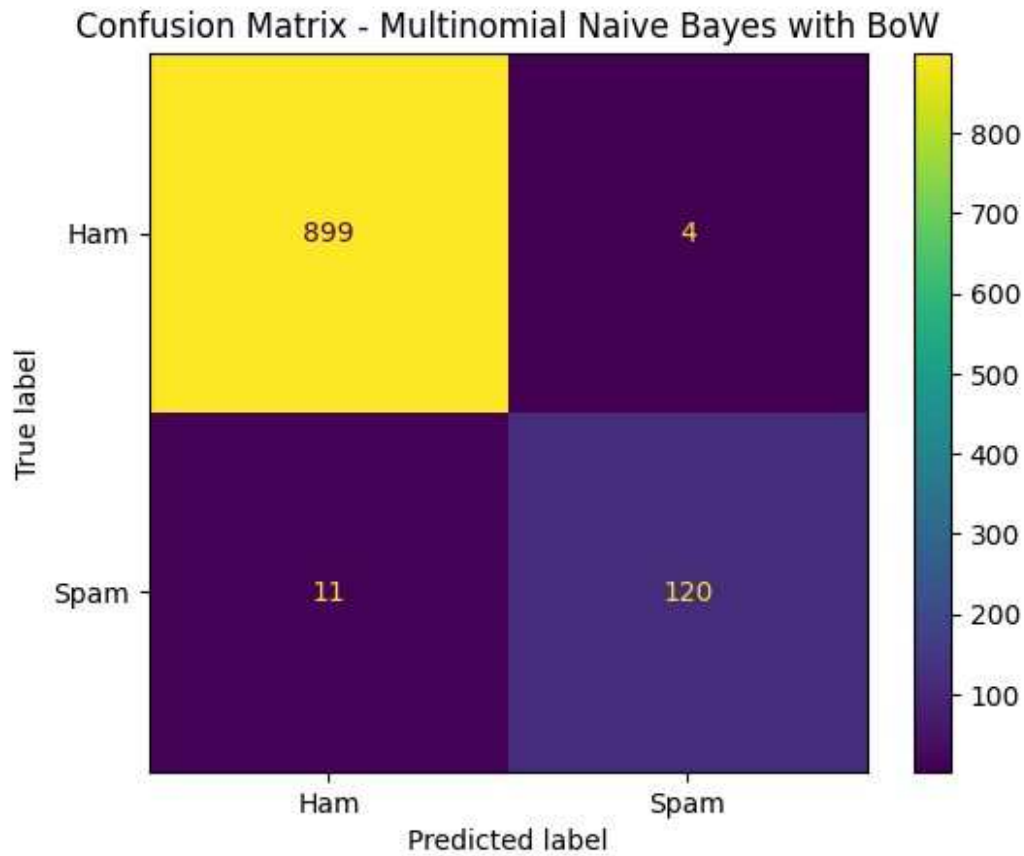
plt.title("Confusion Matrix - Multinomial Naive Bayes with BoW")
plt.show()

```

Best Model: Bag of Words + Multinomial Naive Bayes

Classification Report:

	precision	recall	f1-score	support
Ham	0.9879	0.9956	0.9917	903
Spam	0.9677	0.9160	0.9412	131
accuracy			0.9855	1034
macro avg	0.9778	0.9558	0.9665	1034
weighted avg	0.9854	0.9855	0.9853	1034



9. Model Performance Comparison

Create a label combining feature method and model

```
plot_df = results_df.copy()
```

```
plot_df["Method + Model"] = (
    plot_df["Feature Representation"] +
    " + " +
    plot_df["Model"]
)
```

Plot F1-scores

```
plt.figure(figsize=(12, 6))
```

```
bars = plt.bar(
    plot_df["Method + Model"],
    plot_df["F1-Score"]
)
```

```
plt.title("F1-Score Comparison of NLP Classification Models")
```

```
plt.xlabel("Feature Representation + Model")
```

```
plt.ylabel("F1-Score")
```

```

plt.ylim(0.80, 1.0)

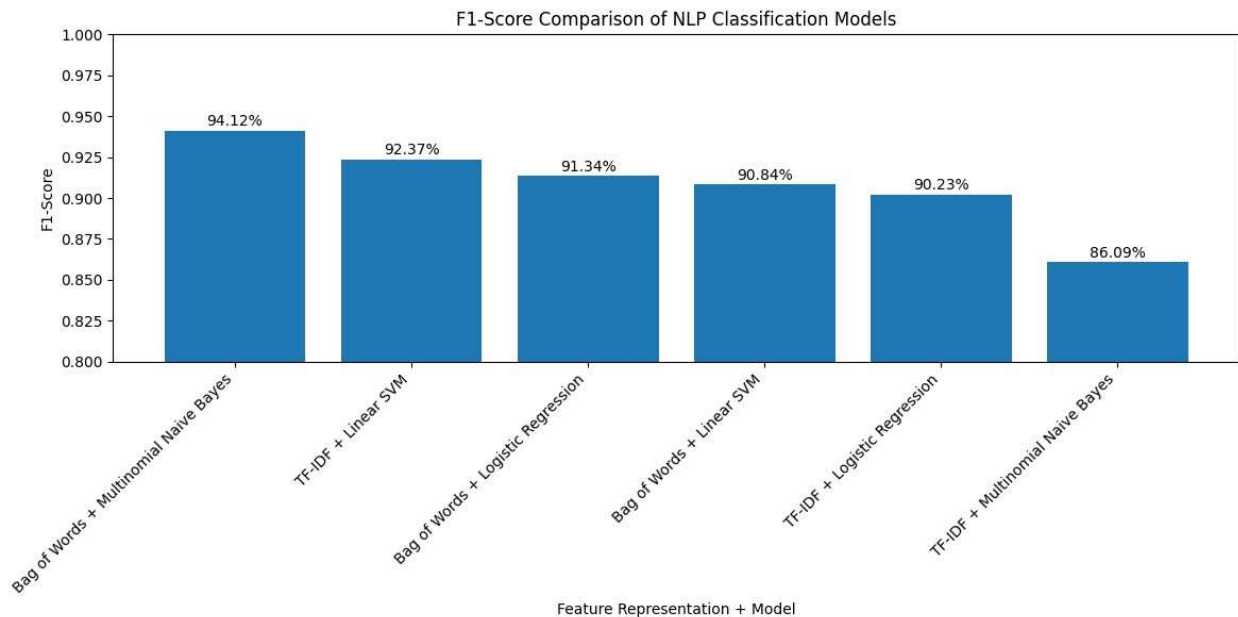
plt.xticks(
    rotation=45,
    ha="right"
)

# Add values above bars
for bar in bars:
    height = bar.get_height()

    plt.text(
        bar.get_x() + bar.get_width()/2,
        height + 0.003,
        f"{height:.2%}",
        ha="center"
    )

plt.tight_layout()
plt.show()

```



10. Test the Best Model on New Messages

```

new_messages = [
    "Congratulations! You have won a free prize. Call now to claim your reward.",
    "Hi, are we still meeting at the library at 4 pm?",
    "URGENT! You have won $1000. Click here to claim your prize.",
    "Please send me the homework file when you finish."
]

```

```

# Apply the same preprocessing
clean_new_messages = [
    clean_text(message)
    for message in new_messages
]

# Convert text using the trained Bag of Words vectorizer
new_messages_bow = bow_vectorizer.transform(clean_new_messages)

# Predict
predictions = best_model.predict(new_messages_bow)

# Convert numerical predictions to labels
prediction_labels = [
    "Spam" if prediction == 1 else "Ham"
    for prediction in predictions
]

# Display results
prediction_df = pd.DataFrame({
    "Message": new_messages,
    "Prediction": prediction_labels
})

display(prediction_df)

{"summary":{"name": "prediction_df", "rows": 4, "fields": [{"column": "Message", "properties": {"dtype": "string", "num_unique_values": 4, "samples": [{"Hi, are we still meeting at the library at 4 pm?", "Please send me the homework file when you finish.", "Congratulations! You have won a free prize. Call now to claim your reward."}], "semantic_type": ""}, {"column": "Prediction", "properties": {"dtype": "string", "num_unique_values": 2, "samples": [{"Ham", "Spam"}], "semantic_type": "", "description": ""}], "type": "dataframe", "variable_name": "prediction_df"}

```

Conclusion

This project developed an NLP-based SMS spam classification model using the SMS Spam Collection dataset. The dataset was preprocessed by removing duplicate records, converting text to lowercase, and removing URLs, email addresses, numbers, punctuation, and special characters. The cleaned dataset contained 5,169 messages and was divided into 80% training and 20% testing sets using stratified sampling.

Two feature representation methods, Bag of Words (BoW) and TF-IDF, were applied to convert the textual data into numerical features. Three machine learning classifiers were evaluated: Multinomial Naive Bayes, Logistic Regression, and Linear SVM. The models were compared using accuracy, precision, recall, and F1-score.

The experimental results showed that Multinomial Naive Bayes with Bag of Words achieved the best overall performance, with 98.55% accuracy, 96.77% precision, 91.60% recall, and 94.12% F1-score for the Spam class. The confusion matrix also showed that the model correctly identified 120 of 131 spam messages. Finally, testing the model on new messages demonstrated its ability to distinguish between Spam and Ham messages.