

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
import nltk
import pandas as pd
import string

# Download necessary text parsing datasets
nltk.download('stopwords')
nltk.download('punkt')

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```
# Load the CSV file from your Drive environment
# Adjust the path string if the file is located within a specific subfolder
file_path = '/content/drive/My Drive/spam.csv'

# Read the file and handle format anomalies
df = pd.read_csv(file_path, encoding='latin1')

# Clean columns up by dropping extraneous delimiters if present
df = df.iloc[:, [0, 1]]
df.columns = ['label', 'message']
df = df.dropna(subset=['message'])

# Display dataframe structure confirmation
print(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...

```
# Convert categorical classification target to binary values
df['label_encoded'] = df['label'].map({'ham': 0, 'spam': 1})
```

```
nltk.download('punkt_tab')
```

```

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

def preprocess_sms(text):
    # Convert text to uniform lowercase
    text = text.lower()

    # Strip common platform token noise
    text = text.replace('<#>', '')

    # Isolate independent word tokens
    tokens = word_tokenize(text)

    # Eliminate punctuation and filter standard stop words
    cleaned_tokens = [
        word for word in tokens
        if word not in string.punctuation and word not in stop_words
    ]

    return cleaned_tokens

# Map processing across dataset
df['cleaned_tokens'] = df['message'].apply(preprocess_sms)

# Review progress sample
print(df[['message', 'cleaned_tokens']].head())

```

```

[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.

```

```

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

                                cleaned_tokens
0  [go, jurong, point, crazy, .., available, bugi...
1  [ok, lar, ..., joking, wif, u, oni, ...]
2  [free, entry, 2, wkly, comp, win, fa, cup, fin...
3  [u, dun, say, early, hor, ..., u, c, already, ...
4  [nah, n't, think, goes, usf, lives, around, th...

```

```

from sklearn.feature_extraction.text import TfidfVectorizer

# Format tokens for scikit-learn ingestion
df['cleaned_text_str'] = df['cleaned_tokens'].apply(lambda tokens: ' '.join(tokens))

# Create the vectorization transformation matrix
tfidf_vectorizer = TfidfVectorizer()
X = tfidf_vectorizer.fit_transform(df['cleaned_text_str'])
y = df['label_encoded'].values

```

```
print(f"Data Matrix Preprocessing Complete. Matrix shape: {X.shape}")
```

```
Data Matrix Preprocessing Complete. Matrix shape: (5572, 8606)
```

```
+++++=====
=====
```

```
# Step 1: Separate 20% of the complete dataset for final testing.
# The remaining 80% is held temporarily for further partitioning.
X_temp, X_test, y_temp, y_test = train_test_split(
    X,
    y,
    test_size=0.20,
    random_state=42,
    stratify=y
)

# Step 2: Split the remaining temporary 80% into training and validation sets.
# Since 10% of the total dataset is required for validation, and 10% is exactly
# 12.5% of the temporary 80% subset (0.10 / 0.80 = 0.125), we set test_size=0.1
X_train, X_val, y_train, y_val = train_test_split(
    X_temp,
    y_temp,
    test_size=0.125,
    random_state=42,
    stratify=y_temp
)

# Print out the verified distribution matrices
print(f"Dataset Partitioning Summary:")
print(f"-----")
print(f"Training set features shape (70%): {X_train.shape}")
print(f"Validation set features shape (10%): {X_val.shape}")
print(f"Testing set features shape (20%): {X_test.shape}")
print(f"\nTraining labels count : {len(y_train)}")
print(f"Validation labels count : {len(y_val)}")
print(f"Testing labels count : {len(y_test)}")
```

```
Dataset Partitioning Summary:
```

```
-----
```

```
Training set features shape (70%): (3899, 8606)
```

```
Validation set features shape (10%): (558, 8606)
```

```
Testing set features shape (20%): (1115, 8606)
```

```
Training labels count : 3899
```

```
Validation labels count : 558
```

```
Testing labels count : 1115
```

```
# 1. Format raw token lists back into standard sentence strings for sklearn
df['cleaned_text_str'] = df['cleaned_tokens'].apply(lambda tokens: ' '.join(tokens))

# 2. Extract split partitions using text strings instead of previous matrix elements
# Re-run the exact 70/10/20 split on raw text strings
X_str_temp, X_str_test, y_temp, y_test = train_test_split(
    df['cleaned_text_str'], y, test_size=0.20, random_state=42, stratify=y
)
X_str_train, X_str_val, y_train, y_val = train_test_split(
    X_str_temp, y_temp, test_size=0.125, random_state=42, stratify=y_temp
)

# 3. Initialize and fit TF-IDF vectorizer strictly on training text data
tfidf_vectorizer = TfidfVectorizer(max_features=3000)
X_train_tfidf = tfidf_vectorizer.fit_transform(X_str_train)

# 4. Transform validation and test arrays using the learned training vocabulary
X_val_tfidf = tfidf_vectorizer.transform(X_str_val)
X_test_tfidf = tfidf_vectorizer.transform(X_str_test)

print("--- TF-IDF Matrix Dimensions ---")
print(f"Train shapes      : {X_train_tfidf.shape}")
print(f"Validation shapes : {X_val_tfidf.shape}")
print(f"Test shapes       : {X_test_tfidf.shape}")
```

```
--- TF-IDF Matrix Dimensions ---
Train shapes      : (3899, 3000)
Validation shapes : (558, 3000)
Test shapes       : (1115, 3000)
```

```
from sklearn.feature_extraction.text import CountVectorizer

# Initialize and fit Bag of Words vectorizer strictly on training text data
count_vectorizer = CountVectorizer(max_features=3000)
X_train_bow = count_vectorizer.fit_transform(X_str_train)

# Transform validation and test configurations using the training vocabulary
X_val_bow = count_vectorizer.transform(X_str_val)
X_test_bow = count_vectorizer.transform(X_str_test)

print("--- Bag of Words Matrix Dimensions ---")
print(f"Train shapes      : {X_train_bow.shape}")
print(f"Validation shapes : {X_val_bow.shape}")
print(f"Test shapes       : {X_test_bow.shape}")
```

```
--- Bag of Words Matrix Dimensions ---
Train shapes      : (3899, 3000)
Validation shapes : (558, 3000)
Test shapes       : (1115, 3000)
```

Double-click (or enter) to edit

```
# Install sentence-transformers locally in the Colab session
!pip install sentence-transformers -q

from sentence_transformers import SentenceTransformer
import numpy as np

# Load a lightweight, highly efficient sentence encoder model
embedding_model = SentenceTransformer('all-MiniLM-L6-v2')

# Generate contextual numerical arrays from input text elements
X_train_emb = embedding_model.encode(X_str_train.tolist(), show_progress_bar=False)
X_val_emb = embedding_model.encode(X_str_val.tolist(), show_progress_bar=False)
X_test_emb = embedding_model.encode(X_str_test.tolist(), show_progress_bar=False)

print("--- Dense Embedding Matrix Dimensions ---")
print(f"Train shapes      : {X_train_emb.shape}")
print(f"Validation shapes : {X_val_emb.shape}")
print(f"Test shapes       : {X_test_emb.shape}")
```

```
modules.json: 100% 349/349 [00:00<00:00, 7.10kB/s]
config_sentence_transformers.json: 100% 116/116 [00:00<00:00, 1.67kB/s]
Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF token.
WARNING:huggingface_hub.utils._http:Warning: You are sending unauthenticated requests to the HF Hub. Please set a HF token.
README.md: 100% 10.5k/10.5k [00:00<00:00, 562kB/s]
sentence_bert_config.json: 100% 53.0/53.0 [00:00<00:00, 2.10kB/s]
config.json: 100% 612/612 [00:00<00:00, 23.5kB/s]
model.safetensors: reconstructing file: 100% 90.9MB / 90.9MB, 8.84MB/s
model.safetensors: downloading bytes: 85.0MB, 8.16MB/s
Loading weights: 100% 103/103 [00:00<00:00, 2393.37it/s]
tokenizer_config.json: 100% 350/350 [00:00<00:00, 7.14kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 2.19MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 3.87MB/s]
special_tokens_map.json: 100% 112/112 [00:00<00:00, 2.32kB/s]
config.json: 100% 190/190 [00:00<00:00, 1.91kB/s]
--- Dense Embedding Matrix Dimensions ---
Train shapes      : (3899, 384)
Validation shapes : (558, 384)
Test shapes       : (1115, 384)
```

Feature Method Overview Representation Method Matrix Type Dimensionality Best Use
 Case Bag of Words Sparse Variable (Vocabulary Size) Basic baseline statistical checks.
 TF-IDF Vectorizer Sparse Variable (Vocabulary Size) Highly accurate for text
 classification algorithms (Naive Bayes, SVM). Dense Embeddings Dense Fixed Vector
 (e.g., 384) Semantic mapping, cluster structures, and deep learning neural net ingestion.

```
n.naive_bayes import MultinomialNB
n.linear_model import LogisticRegression
n.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
import pandas as pd

-----
Training
-----

# Fit the classification algorithms
nb_model = MultinomialNB()
lr_model = LogisticRegression(random_state=42)

# Evaluate both models using the TF-IDF training matrices
nb_train_acc = accuracy_score(y_train, nb_model.predict(X_train_tfidf))
lr_train_acc = accuracy_score(y_train, lr_model.predict(X_train_tfidf))

-----
Validation Set Performance Check
-----

# Predict on validation set
nb_val_preds = nb_model.predict(X_val_tfidf)
lr_val_preds = lr_model.predict(X_val_tfidf)

# Print Validation Set Accuracy
print("Validation Set Accuracy ===")
print("Naive Bayes Accuracy: {accuracy_score(y_val, val_preds_nb):.4f}")
print("Logistic Regression Accuracy: {accuracy_score(y_val, val_preds_lr):.4f}\n")

-----
Evaluation on Unseen Test Dataset (20%)
-----

# Predict on test set
nb_test_preds = nb_model.predict(X_test_tfidf)
lr_test_preds = lr_model.predict(X_test_tfidf)

# Print Test Set Metrics
def print_test_metrics(y_true, y_pred, model_name):
    {
        "model": model_name,
        "accuracy": accuracy_score(y_true, y_pred),
        "precision": precision_score(y_true, y_pred),
        "recall": recall_score(y_true, y_pred),
        "f1-Score": f1_score(y_true, y_pred)
    }
```

```

comparative evaluation dataframe
mary = pd.DataFrame([
    te_metrics(y_test, test_preds_nb, "Multinomial Naive Bayes"),
    te_metrics(y_test, test_preds_lr, "Logistic Regression")

-----
Summary Reports
-----
FINAL TEST SET BENCHMARK METRICS ===")
cs_summary.to_string(index=False, float_format=lambda x: f"{x:.4f}"))

= DETAILED CLASSIFICATION REPORT (LOGISTIC REGRESSION) ===")
ification_report(y_test, test_preds_lr, target_names=['Ham (0)', 'Spam (1)'])

```

```

=== VALIDATION SET ACCURACY ===
Naive Bayes Accuracy: 0.9767
Logistic Regression Accuracy: 0.9624

```

```

=== FINAL TEST SET BENCHMARK METRICS ===

```

	Model	Accuracy	Precision	Recall	F1-Score
	Multinomial Naive Bayes	0.9749	0.9919	0.8188	0.8971
	Logistic Regression	0.9713	0.9916	0.7919	0.8806

```

=== DETAILED CLASSIFICATION REPORT (LOGISTIC REGRESSION) ===

```

	precision	recall	f1-score	support
Ham (0)	0.97	1.00	0.98	966
Spam (1)	0.99	0.79	0.88	149
accuracy			0.97	1115
macro avg	0.98	0.90	0.93	1115
weighted avg	0.97	0.97	0.97	1115

