

✓ SMS Spam Classification with Kaggle Dataset

This project of SMS spam classification uses the **SMS Spam Collection** dataset downloaded from this link <https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset?resource=download>

(5,574 SMS messages labeled as ham or spam)

- Loading the dataset via Kaggle or manual upload.
- Text preprocessing (lowercase, remove punctuation, stopwords, lemmatization).
- Data splitting into training and testing sets.
- Feature representation with **Bag-of-Words** (CountVectorizer) and **TF-IDF** (including n-grams).
- Definition of a reusable evaluation function that computes accuracy, precision, recall, F1 score, confusion matrix and displays a heatmap.
- Training and hyperparameter tuning for multiple classifiers via **GridSearchCV**:
 - **Random Forest**
 - **Logistic Regression**
 - **Decision Tree**
 - **Support Vector Machine (SVM)**
- Comparison of model performance on the validation set using the evaluation function.

```
# Install necessary packages (uncomment if needed)
# !pip install kaggle nltk scikit-learn seaborn matplotlib

import os
import pandas as pd
import numpy as np
import nltk
import re
import matplotlib.pyplot as plt
import seaborn as sns
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, confusion_matrix, classification_report)

# Download NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
# Dataset loading
# -----
# Option 1: Download via Kaggle API (requires API credentials and kaggle.json)
# !pip install kaggle -q
# import json
# os.environ['KAGGLE_USERNAME'] = 'your_username'
# os.environ['KAGGLE_KEY'] = 'your_key'
# !kaggle datasets download -d uciml/sms-spam-collection-dataset -p /content/ --unzip --force

from pathlib import Path
from google.colab import files

# Look for existing dataset files
possible_files = ['spam.csv', 'SMSSpamCollection']
dataset_file = None
for fname in possible_files:
    if Path(fname).exists():
        dataset_file = Path(fname)
        break
```

```

if dataset_file is None:
    print("Dataset file not found locally. Please upload 'spam.csv' or 'SMSSpamCollection'.")
    uploaded = files.upload()
    fname = next(iter(uploaded))
    dataset_file = Path(fname)

# Load the CSV/text file
print(f"Loading dataset from: {dataset_file}")
df = pd.read_csv(dataset_file, encoding='latin-1')

# Standardize column names to 'label' and 'text'
if 'v1' in df.columns and 'v2' in df.columns:
    df = df[['v1', 'v2']]
    df.columns = ['label', 'text']
elif 'label' in df.columns and 'text' in df.columns:
    df = df[['label', 'text']]
else:
    df = df.iloc[:, :2]
    df.columns = ['label', 'text']

print(f"Dataset shape: {df.shape}")
print(df.head())

```

Dataset file not found locally. Please upload 'spam.csv' or 'SMSSpamCollection'.

[Choose Files](#) No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving archive (1).zip to archive (1).zip

Loading dataset from: archive (1).zip

Dataset shape: (5572, 2)

	label	text
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...

```

# Text preprocessing
# -----
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def clean_message(msg: str) -> str:
    msg = msg.lower()
    msg = re.sub(r'^a-z\s]', ' ', msg)
    tokens = msg.split()
    tokens = [lemmatizer.lemmatize(t) for t in tokens if t not in stop_words]
    return ' '.join(tokens)

# Apply cleaning to all messages
df['clean_text'] = df['text'].astype(str).apply(clean_message)

print(df[['label', 'clean_text']].head())

```

	label	clean_text
0	ham	go jurong point crazy available bugis n great ...
1	ham	ok lar joking wif u oni
2	spam	free entry wkly comp win fa cup final tkts st ...
3	ham	u dun say early hor u c already say
4	ham	nah think go usf life around though

```

# Encode labels to numeric
label_mapping = {label: idx for idx, label in enumerate(sorted(df['label'].unique()))}
df['target'] = df['label'].map(label_mapping)

# Train-test split (80/20)
X_train, X_test, y_train, y_test = train_test_split(
    df['clean_text'],
    df['target'],
    test_size=0.2,
    random_state=42,
    stratify=df['target']
)

print(f"Training samples: {len(X_train)}, Test samples: {len(X_test)}")

```

Training samples: 4457, Test samples: 1115

```

# 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, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    cm = confusion_matrix(y_true, y_pred)
    report = classification_report(y_true, y_pred, target_names=sorted(label_mapping, key=label_mapping.get))

    metrics = {
        'Model Name': model_name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1,
        'Classification Report': report
    }

    # Plot confusion matrix
    plt.figure(figsize=(4, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
                xticklabels=np.unique(y_true),
                yticklabels=np.unique(y_true))
    plt.title(f'Confusion Matrix for {model_name}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()

    return metrics

```

```

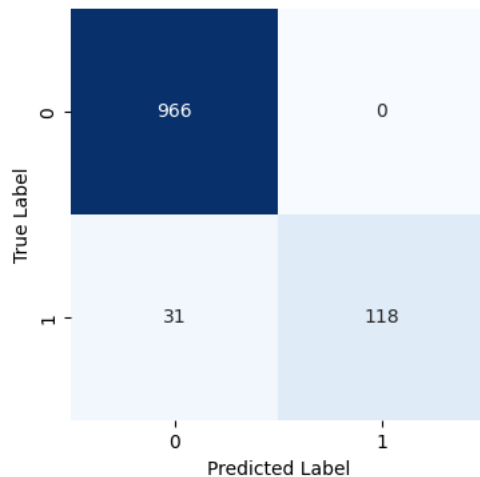
# Bag-of-Words feature representation
# -----
vectorizer_bow = CountVectorizer()
X_train_bow = vectorizer_bow.fit_transform(X_train)
X_test_bow = vectorizer_bow.transform(X_test)

# Random Forest with hyperparameter tuning
rf_classifier = RandomForestClassifier(random_state=42)
rf_param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20]
}
rf_grid = GridSearchCV(estimator=rf_classifier, param_grid=rf_param_grid, cv=5, scoring='accuracy', n_jobs=-1)
rf_grid.fit(X_train_bow, y_train)
best_rf = rf_grid.best_estimator_
y_pred_rf = best_rf.predict(X_test_bow)
metrics_rf = evaluate_model('RandomForestClassifier', y_test, y_pred_rf)
print("Best RandomForest params:", rf_grid.best_params_)
print(metrics_rf['Classification Report'])

# Logistic Regression with hyperparameter tuning (solver)
log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_param_grid = {
    'solver': ['liblinear', 'lbfgs']
}
log_grid = GridSearchCV(estimator=log_reg, param_grid=log_param_grid, cv=5, scoring='accuracy', n_jobs=-1)
log_grid.fit(X_train_bow, y_train)
best_log = log_grid.best_estimator_
y_pred_log = best_log.predict(X_test_bow)
metrics_log = evaluate_model('LogisticRegression', y_test, y_pred_log)
print("Best LogisticRegression params:", log_grid.best_params_)
print(metrics_log['Classification Report'])

```

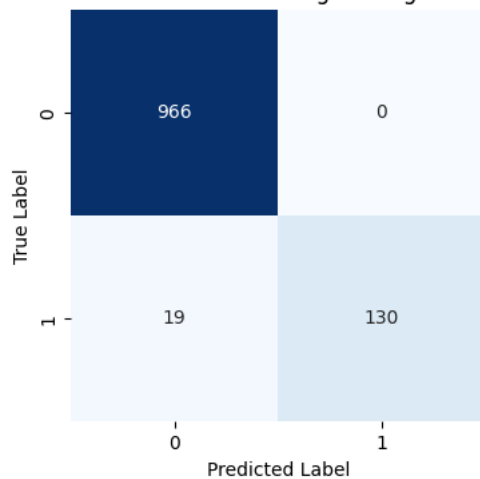
Confusion Matrix for RandomForestClassifier



Best RandomForest params: {'max_depth': None, 'n_estimators': 200}

	precision	recall	f1-score	support
ham	0.97	1.00	0.98	966
spam	1.00	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

Confusion Matrix for LogisticRegression



Best LogisticRegression params: {'solver': 'liblinear'}

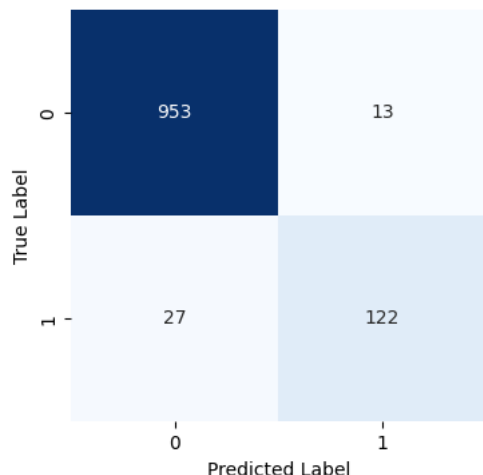
	precision	recall	f1-score	support
ham	0.98	1.00	0.99	966
spam	1.00	0.87	0.93	149
accuracy			0.98	1115
macro avg	0.99	0.94	0.96	1115
weighted avg	0.98	0.98	0.98	1115

```
# TF-IDF with unigrams and bigrams
# -----
vectorizer_tfidf = TfidfVectorizer(ngram_range=(1, 2), max_features=20000)
X_train_tfidf = vectorizer_tfidf.fit_transform(X_train)
X_test_tfidf = vectorizer_tfidf.transform(X_test)

# Decision Tree with hyperparameter tuning
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20]
}
dt_grid = GridSearchCV(estimator=dt_classifier, param_grid=dt_param_grid, cv=5, scoring='accuracy', n_jobs=-1)
dt_grid.fit(X_train_tfidf, y_train)
best_dt = dt_grid.best_estimator_
y_pred_dt = best_dt.predict(X_test_tfidf)
metrics_dt = evaluate_model('DecisionTreeClassifier', y_test, y_pred_dt)
print("Best DecisionTree params:", dt_grid.best_params_)
print(metrics_dt['Classification Report'])
```

```
# Support Vector Machine with hyperparameter tuning
svm_classifier = SVC(random_state=42)
svm_param_grid = {
    'kernel': ['linear', 'rbf', 'poly']
}
svm_grid = GridSearchCV(estimator=svm_classifier, param_grid=svm_param_grid, cv=5, scoring='accuracy', n_jobs=-1)
svm_grid.fit(X_train_tfidf, y_train)
best_svm = svm_grid.best_estimator_
y_pred_svm = best_svm.predict(X_test_tfidf)
metrics_svm = evaluate_model('SupportVectorMachine', y_test, y_pred_svm)
print("Best SVM params:", svm_grid.best_params_)
print(metrics_svm['Classification Report'])
```

Confusion Matrix for DecisionTreeClassifier

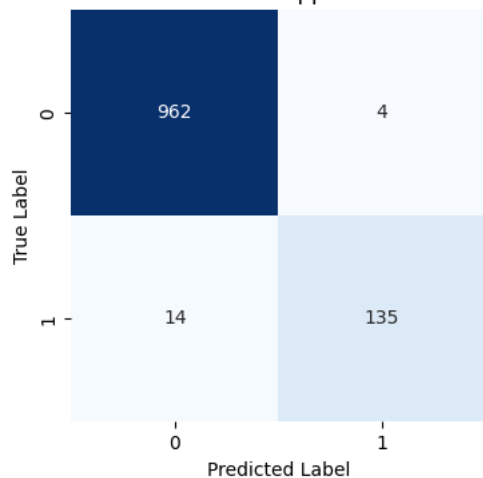


Best DecisionTree params: {'criterion': 'gini', 'max_depth': None}

	precision	recall	f1-score	support
ham	0.97	0.99	0.98	966
spam	0.90	0.82	0.86	149
accuracy			0.96	1115
macro avg	0.94	0.90	0.92	1115
weighted avg	0.96	0.96	0.96	1115

ham	0.97	0.99	0.98	966
spam	0.90	0.82	0.86	149
accuracy			0.96	1115
macro avg	0.94	0.90	0.92	1115
weighted avg	0.96	0.96	0.96	1115

Confusion Matrix for SupportVectorMachine



Best SVM params: {'kernel': 'linear'}

	precision	recall	f1-score	support
ham	0.99	1.00	0.99	966
spam	0.97	0.91	0.94	149
accuracy			0.98	1115
macro avg	0.98	0.95	0.96	1115
weighted avg	0.98	0.98	0.98	1115

ham	0.99	1.00	0.99	966
spam	0.97	0.91	0.94	149
accuracy			0.98	1115
macro avg	0.98	0.95	0.96	1115
weighted avg	0.98	0.98	0.98	1115

****Summary of the steps****

- 1. I Downloaded the NLP dataset** – The code includes instructions for downloading the SMS Spam Collection dataset via the Kaggle API and an option to upload the spam.csv file manually. Using the below link <https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset?resource=download>
- 2. Apply data preprocessing** – It cleans the text by lower-casing, removing punctuation, filtering stop-words and lemmatizing.

3. **Split the dataset** – It performs an 80/20 train–test split of the cleaned messages and labels.
4. **Apply feature representation methods** – It uses both Bag-of-Words (CountVectorizer) and TF-IDF (TfidfVectorizer with n-grams) to convert text into numerical feature vectors.
5. **Train and evaluate models** – It trains several classifiers (Random Forest, Logistic Regression, Decision Tree, SVM) with hyperparameter tuning via GridSearchCV, then evaluates them using accuracy, precision, recall, F1 scores and confusion-matrix plots.