

✓ Project 2 – NLP Classification

SMS Spam Classification

Name: Ahmad Alkahlout

Dataset: SMS Spam Collection Dataset

Task: Spam Message Classification

Step 1: Download Dataset from Kaggle

In this step, I downloaded the SMS Spam Collection dataset from Kaggle. The dataset is used to classify text messages as spam or ham.

```
# Download SMS Spam Collection dataset from Kaggle
!kaggle datasets download -d uciml/sms-spam-collection-dataset
```

Warning: Looks like you're using an outdated `kaggle` version (installed: 2.0.2), please consider upgrading to the latest
Dataset URL: <https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset>
License(s): unknown
Downloading sms-spam-collection-dataset.zip to /content
100% 211k/211k [00:00<00:00, 391kB/s]

```
# Unzip the downloaded dataset
!unzip -q sms-spam-collection-dataset.zip -d sms_spam

print("Dataset unzipped successfully")
```

Dataset unzipped successfully

```
# Check files inside the dataset folder
import os

for file in os.listdir("sms_spam"):
    print(file)
```

spam.csv

Step 2: Read Dataset

In this step, I read the SMS Spam dataset and displayed the first rows to check the data and its columns.

```
# Read dataset

import pandas as pd

df = pd.read_csv("sms_spam/spam.csv", encoding="ISO-8859-1")

df.head()
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4	
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN	
1	ham	OK lar... Joking wif u oni...	NaN	NaN	NaN	
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN	
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN	
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN	

Next steps:

[Generate code with df](#)

[New interactive sheet](#)

Step 3: Clean Dataset Columns

In this step, I removed the unused columns and renamed the required columns to make the dataset easier to use.

```
# Clean dataset columns

df = df[['v1', 'v2']]

df.columns = ['label', 'text']

df.head()
```

	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...

Next steps:

[Generate code with df](#)[New interactive sheet](#)

Step 4: Check Dataset Size and Class Distribution

In this step, I checked the number of samples and the distribution of ham and spam messages in the dataset.

```
# Check dataset size and class distribution

print("Number of samples:", len(df))

print("\nClass distribution:")
print(df['label'].value_counts())
```

Number of samples: 5572

Class distribution:

label

ham 4825

spam 747

Name: count, dtype: int64

Step 5: Data Preprocessing

In this step, I cleaned the text messages by converting them to lowercase, removing URLs, special characters, stopwords, and applying stemming.

```
# Import libraries for text preprocessing

import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
```

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

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data] Package punkt_tab is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Package wordnet is already up-to-date!
True
```

```
# Text preprocessing function

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

def preprocess_text(text):

    text = text.lower()

    text = re.sub(r'http\S+|www.\S+', '', text)

    text = re.sub(r'<.*?>', '', text)

    text = re.sub(r'^a-zA-Z\s', '', text)

    tokens = word_tokenize(text)

    tokens = [word for word in tokens if word not in stop_words]

    stemmed_tokens = [stemmer.stem(word) for word in tokens]

    clean_text = ' '.join(stemmed_tokens)

    return clean_text
```

```
# Apply text preprocessing

df["clean_text"] = df["text"].apply(preprocess_text)

df = df.dropna()

df[['text', 'clean_text']].head()
```

	text	clean_text
0	Go until jurong point, crazy.. Available only ...	go jurong point crazi avail bugi n great world...
1	Ok lar... Joking wif u oni...	ok lar joke wif u oni
2	Free entry in 2 a wkly comp to win FA Cup fina...	free entri wkli comp win fa cup final tkt st m...
3	U dun say so early hor... U c already then say...	u dun say earli hor u c already say
4	Nah I don't think he goes to usf, he lives aro...	nah dont think goe usf live around though

Step 6: Split the Dataset

In this step, I split the cleaned dataset into training and testing sets while preserving the class distribution.

```
# Split the dataset

from sklearn.model_selection import train_test_split

X = df['clean_text']
y = df['label']

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.30,
    random_state=42,
    stratify=y
)

print("Training samples:", len(X_train))
print("Testing samples:", len(X_test))
```

```
Training samples: 3900
Testing samples: 1672
```

Step 7: Feature Representation Using TF-IDF

In this step, I converted the cleaned text messages into numerical features using TF-IDF.

```
# TF-IDF feature representation

from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()

X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

print("Training features shape:", X_train_vec.shape)
print("Testing features shape:", X_test_vec.shape)

Training features shape: (3900, 5799)
Testing features shape: (1672, 5799)
```

Step 8: Model Evaluation Function

In this step, I created a function to evaluate the classification models using accuracy, precision, recall, F1-score, and confusion matrix.

```
# Function to evaluate model

from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

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)

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

    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
```

Step 9: Train Random Forest Classifier

In this step, I trained a Random Forest classifier on the original imbalanced dataset using TF-IDF features. GridSearchCV was used to select the best hyperparameters.

```
# Random Forest Classifier

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
```

```

rf_classifier = RandomForestClassifier(random_state=42)

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
}

grid_search = GridSearchCV(estimator=rf_classifier,
                           param_grid=param_grid,
                           cv=5,
                           scoring='accuracy',
                           n_jobs=-1)

grid_search.fit(X_train_vec, y_train)

best_rf_classifier = grid_search.best_estimator_

y_pred = best_rf_classifier.predict(X_test_vec)

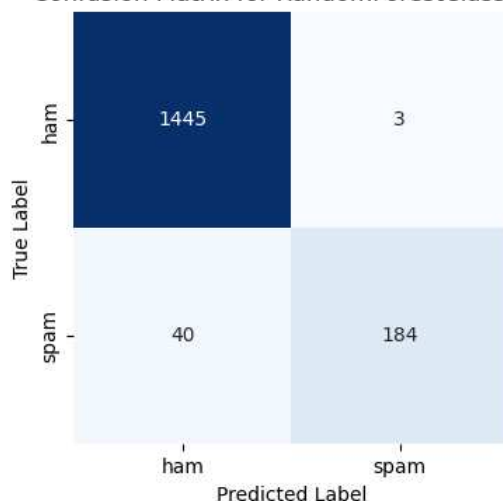
evaluation_results = evaluate_model(
    'RandomForestClassifier',
    y_test,
    y_pred
)

for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float)
              else f"{key}: \n{value}")

print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

Confusion Matrix for RandomForestClassifier



Model Name:

RandomForestClassifier

Accuracy: 0.9743

Precision: 0.9745

Recall: 0.9743

F1 Score: 0.9733

	precision	recall	f1-score	support
ham	0.97	1.00	0.99	1448
spam	0.98	0.82	0.90	224
accuracy			0.97	1672
macro avg	0.98	0.91	0.94	1672
weighted avg	0.97	0.97	0.97	1672

Best hyperparameters found by GridSearchCV:

{'max_depth': None, 'n_estimators': 100}

Step 10: Apply Random Oversampling

In this step, I applied RandomOverSampler to balance the training data by increasing the number of samples in the minority class.

```
# Apply Random Oversampling

from imblearn.over_sampling import RandomOverSampler
from collections import Counter

ros = RandomOverSampler(random_state=42)

X_ros, y_ros = ros.fit_resample(X_train_vec, y_train)

print("Before Random Oversampling:", Counter(y_train))
print("After Random Oversampling:", Counter(y_ros))
```

Before Random Oversampling: Counter({'ham': 3377, 'spam': 523})
After Random Oversampling: Counter({'spam': 3377, 'ham': 3377})

Step 11: Train Random Forest After Oversampling

In this step, I trained the Random Forest classifier again using the balanced training data after Random Oversampling.

```
# Random Forest after Random Oversampling

rf_classifier = RandomForestClassifier(random_state=42)

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
}

grid_search = GridSearchCV(estimator=rf_classifier,
                           param_grid=param_grid,
                           cv=5,
                           scoring='accuracy',
                           n_jobs=-1)

grid_search.fit(X_ros, y_ros)

best_rf_classifier = grid_search.best_estimator_

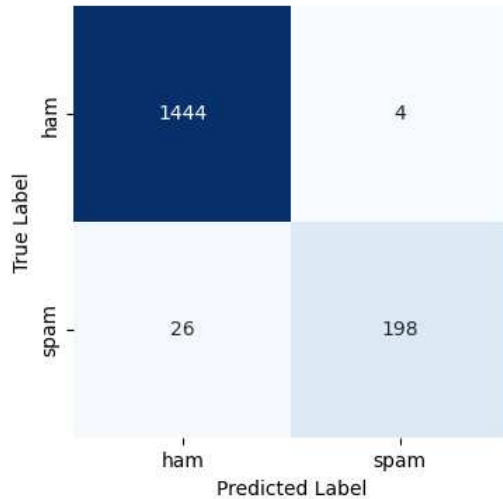
y_pred = best_rf_classifier.predict(X_test_vec)

evaluation_results = evaluate_model(
    'RandomForestClassifier',
    y_test,
    y_pred
)

for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float)
              else f"{key}: \n{value}")

print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```

Confusion Matrix for RandomForestClassifier



Model Name:

RandomForestClassifier

Accuracy: 0.9821

Precision: 0.9820

Recall: 0.9821

F1 Score: 0.9817

	precision	recall	f1-score	support
ham	0.98	1.00	0.99	1448
spam	0.98	0.88	0.93	224
accuracy			0.98	1672
macro avg	0.98	0.94	0.96	1672
weighted avg	0.98	0.98	0.98	1672

Best hyperparameters found by GridSearchCV:

{'max_depth': None, 'n_estimators': 200}

Step 12: Train Logistic Regression Model

In this step, I trained a Logistic Regression model using the balanced TF-IDF training data and evaluated its performance.

```
# Logistic Regression

from sklearn.linear_model import LogisticRegression

LR = LogisticRegression(random_state=42)

param_grid = {
    'solver': ['liblinear', 'lbfgs'],
}

grid_search = GridSearchCV(estimator=LR,
                           param_grid=param_grid,
                           cv=5,
                           scoring='accuracy',
                           n_jobs=-1)

grid_search.fit(X_ros, y_ros)

best_LR = grid_search.best_estimator_

y_pred = best_LR.predict(X_test_vec)

evaluation_results = evaluate_model(
    'LogisticRegression',
    y_test,
    y_pred
)

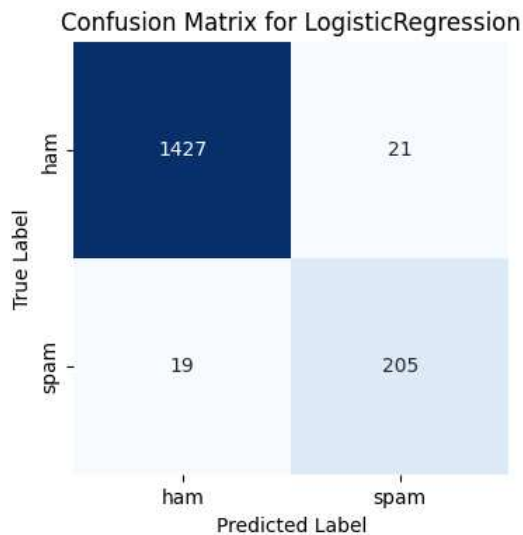
for key, value in evaluation_results.items():
```

```

if key == 'Classification Report':
    print(value)
else:
    print(f"{key}: {value:.4f}" if isinstance(value, float)
          else f"{key}: \n{value}")

print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```



Model Name:
 LogisticRegression
 Accuracy: 0.9761
 Precision: 0.9762
 Recall: 0.9761
 F1 Score: 0.9761

	precision	recall	f1-score	support
ham	0.99	0.99	0.99	1448
spam	0.91	0.92	0.91	224
accuracy			0.98	1672
macro avg	0.95	0.95	0.95	1672
weighted avg	0.98	0.98	0.98	1672

Best hyperparameters found by GridSearchCV:
 {'solver': 'liblinear'}

Step 13: Train Decision Tree Classifier

In this step, I trained a Decision Tree classifier using the balanced TF-IDF training data. GridSearchCV was used to select the best criterion and maximum tree depth.

```

# Decision Tree Classifier

from sklearn.tree import DecisionTreeClassifier

DT = DecisionTreeClassifier(random_state=42)

param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
}

grid_search = GridSearchCV(estimator=DT,
                           param_grid=param_grid,
                           cv=10,
                           scoring='accuracy',
                           n_jobs=-1)

grid_search.fit(X_ros, y_ros)

best_DT = grid_search.best_estimator_

```

```

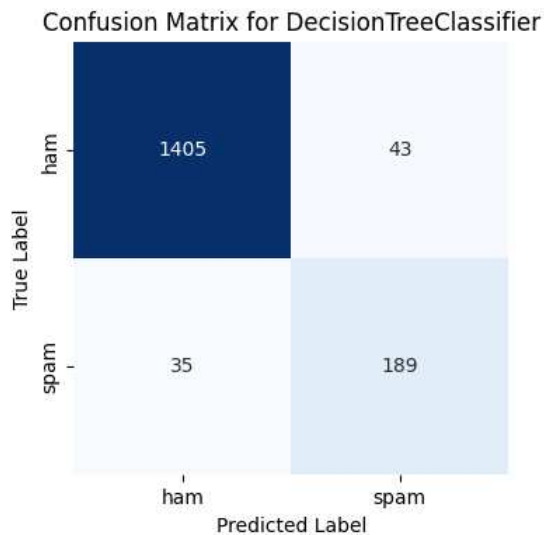
y_pred = best_DT.predict(X_test_vec)

evaluation_results = evaluate_model(
    'DecisionTreeClassifier',
    y_test,
    y_pred
)

for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float)
              else f"{key}: \n{value}")

print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```



Model Name:
DecisionTreeClassifier
Accuracy: 0.9533
Precision: 0.9541
Recall: 0.9533
F1 Score: 0.9537

	precision	recall	f1-score	support
ham	0.98	0.97	0.97	1448
spam	0.81	0.84	0.83	224
accuracy			0.95	1672
macro avg	0.90	0.91	0.90	1672
weighted avg	0.95	0.95	0.95	1672

Best hyperparameters found by GridSearchCV:
{'criterion': 'entropy', 'max_depth': None}

Step 14: Compare Model Results

In this step, I compared the performance of the trained classification models using accuracy, precision, recall, and F1-score.

```

# Compare model results

results = {
    'Model': [
        'Random Forest',
        'Logistic Regression',
        'Decision Tree'
    ],
    'Accuracy': [
        0.9821,
        0.9761,

```

```

        0.9533
    ],
    'Precision': [
        0.9820,
        0.9762,
        0.9541
    ],
    'Recall': [
        0.9821,
        0.9761,
        0.9533
    ],
    'F1 Score': [
        0.9817,
        0.9761,
        0.9537
    ]
}

results_df = pd.DataFrame(results)

results_df

```

	Model	Accuracy	Precision	Recall	F1 Score	
0	Random Forest	0.9821	0.9820	0.9821	0.9817	
1	Logistic Regression	0.9761	0.9762	0.9761	0.9761	
2	Decision Tree	0.9533	0.9541	0.9533	0.9537	

Next steps: [Generate code with results_df](#) [New interactive sheet](#)

Step 15: Conclusion

In this project, I applied NLP preprocessing and TF-IDF feature representation to classify SMS messages as ham or spam. The dataset was imbalanced, so Random Oversampling was applied to balance the training data. Three machine learning models were trained and evaluated: Random Forest, Logistic Regression, and Decision Tree. Random Forest achieved the best overall performance with an accuracy of 98.21% and an F1-score of 98.17%.