Building and testing the SMS spam classification model

Preparations

### First we have to create the python environment for the model to run , get dataset from

“/kaggle/input/sms-spam-collection-dataset/spam.csv “

### Then clean and preprocess it, Import the necessary libraries in the Notebook and then load the dataset and run the head() function and the drop() function to ignore the Nan and undefined columns .

CODE

*# This Python 3 environment comes with many helpful analy tics libraries installed*

*# It is defined by the kaggle/python Docker image: https:*

*//github.com/kaggle/docker-python*

*# For example, here's several helpful packages to load*

import numpy as np *# linear algebra*

import pandas as pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*

*# Input data files are available in the read-only "../inp ut/" directory*

*# For example, running this (by clicking run or pressing*

*Shift+Enter) will list all files under the input directory*

import os

for dirname, \_, filenames **in** os.walk('/kaggle/input'): for filename **in** filenames:

print(os.path.join(dirname, filename))

*# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"*

*# You can also write temporary files to /kaggle/temp/,*

*but they won't be saved outside of the current session*

### Importing necessary libraries

*#necessary libraries*

import numpy as np *# linear algebra* import pandas as pd *# data processing* import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize from nltk.stem import PorterStemmer

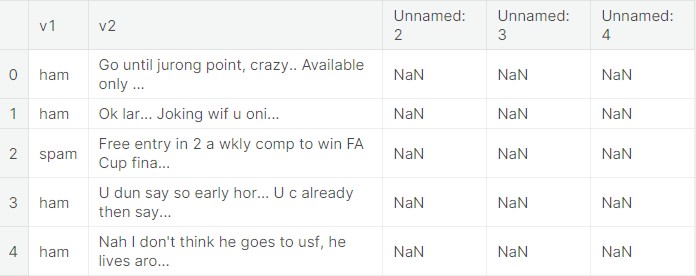
Loading the dataset

*# Load the dataset*

data = pd.read\_csv('/kaggle/input/sms-spam-collection-da taset/spam.csv', encoding='latin-1')

data.head()

# OUTPUT



columns\_to\_drop = ['Unnamed: 2', 'Unnamed: 3',

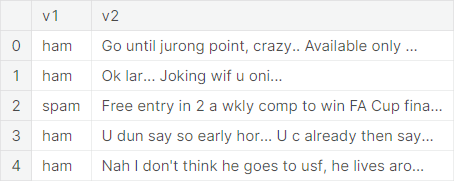
'Unnamed: 4']

data = data.drop(columns\_to\_drop, axis=1, errors=

'ignore')

data.head()

# OUTPUT



**Tokenization and cleaning**



Tokenization is the process of splitting the input and output texts into smaller



units that can be processed by the LLM AI models. Tokens can be words,



characters, subwords, or symbols, depending on the type and the size of the



model.



CODE

*# Clean the "v2" column*

data['v2'] = data['v2'].str.lower()

*# Tokenization and cleaning of data*

def preprocess\_text(text): tokens = word\_tokenize(text)

stop\_words = set(stopwords.words('english')) filtered\_tokens = [word for word **in** tokens if word.isa

lnum() **and** word **n ot in** stop\_words] stemmer = PorterStemmer()

stemmed\_tokens = [stemmer.stem(word) for word **in** fil tered\_tokens]

return ' '.join(stemmed\_tokens)

data['v2'] = data['v2'].apply(preprocess\_text)

data.head()

## OUTPUT

from sklearn.feature\_extraction.text import TfidfVectori zer

from sklearn.model\_selection import train\_test\_split

*# Load the dataset*

tfidf\_vectorizer = TfidfVectorizer()

tfidf\_matrix = tfidf\_vectorizer.fit\_transform(data['v2']

)

*# Label Encoding*

data['v1'] = data['v1'].map({'ham': 0, 'spam': 1})

*# Split Data*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(tfid f\_matrix,

data['v1'], test\_size=0.2, random\_state=42)

*# Check the shape of the TF-IDF matrix and the split data* print("TF-IDF Matrix Shape:", tfidf\_matrix.shape) print("Training Data Shape:", X\_train.shape) print("Testing Data Shape:", X\_test.shape)

# OUTPUT

TF-IDF Matrix Shape: (5572, 8672)

Training Data Shape: (4457, 8672)

Testing Data Shape: (1115, 8672)

Random Forest Classifier

Random forest is a commonly-used machine learning algorithm which combines the output of multiple decision trees to reach a single result.

### CODE

from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import accuracy\_score, classificati on\_report

*# Create a Random Forest classifier*

rf\_classifier = RandomForestClassifier(random\_state=42)

*# Train the classifier on the training data*

rf\_classifier.fit(X\_train, y\_train)

*# Make predictions on the testing data*

y\_pred = rf\_classifier.predict(X\_test)

*# Evaluate the model's performance*

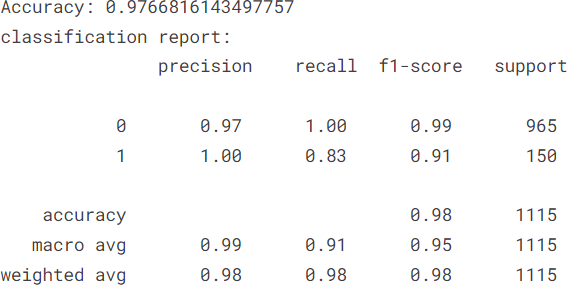
accuracy = accuracy\_score(y\_test, y\_pred) classification\_rep = classification\_report(y\_test, y\_pre d)

*# Print the results*

print("Accuracy:", accuracy)

print("classification report:**\n**", classification\_rep)

**OUTPUT**



# Testing model

**Test1**

input\_text = """**\a**pple Inc.Your iPhone 6 linked top\*\*\*zm ".edu) has

been used a few minutes

ago. To localize it,login now to your apple account ."""

*# Apply the same preprocessing as in your previous code*

input\_text = input\_text.lower()

*# Add more preprocessing steps if needed*

*# Transform the input text into a TF-IDF vector*

input\_tfidf = tfidf\_vectorizer.transform([input\_text])

*# Make a prediction using the trained Random Forest model*

prediction = rf\_classifier.predict(input\_tfidf)

*# predictions*

if prediction[0] == 1:

print("This message is predicted to be SPAM by trained model.")

else:

print("This message is predicted to be NOT SPAM by

trained model.")

# OUTPUT

This message is predicted to be NOT SPAM by trained model.

## Test2

|  |  |  |
| --- | --- | --- |
| input\_text1 = "Hey, I'm mark. How are you?."  *# Apply the same preprocessing as in your previous code*  input\_text1 = input\_text1.lower()  *# Transform the input text into a TF-IDF vector*  input\_tfidf = tfidf\_vectorizer.transform([input\_text1])  *# Make a prediction using the trained Random Forest model*  prediction = rf\_classifier.predict(input\_tfidf)  *# perdictions*  if prediction[0] == 1:  print("This message is predicted to be SPAM by traine d mode  l.")  else: | | |
| print("This message is ained model.") | predicted | to be NOT SPAM by tr |
| **OUTPUT** |  |  |
| This message is predicted model. | to be NOT | SPAM by trained |