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 analytics 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 "../input/" 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

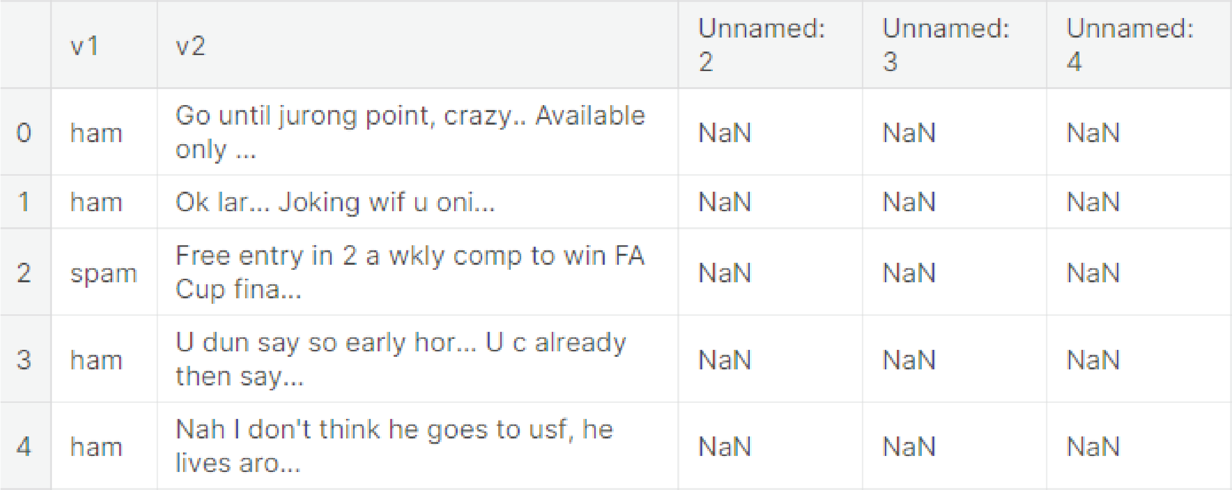
```
#necessary libraries

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-dataset/spam.csv', encoding='latin-1')
```

```
data.head()
```

OUTPUT

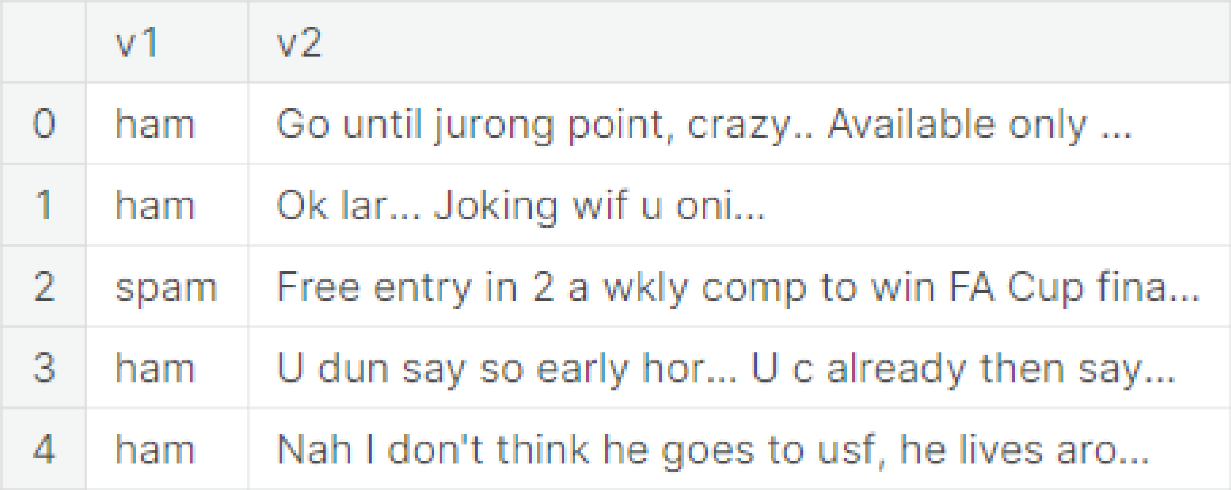


	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah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

```
columns_to_drop = ['Unnamed: 2', 'Unnamed: 3',
'Unnamed: 4']
data = data.drop(columns_to_drop, axis=1, errors=
'ignore')
```

```
data.head()
```

OUTPUT



	v1	v2
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...

Tokenization and cleaning

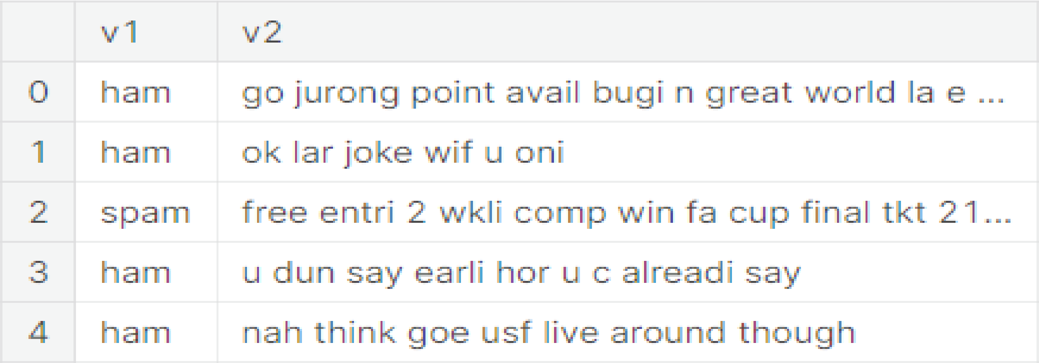
Tokenization is the process of splitting the input and output texts into smaller units that can be processed by the LLM AI models. Tokens can be words, characters, subwords, or symbols, depending on the type and the size of the model.

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



	v1	v2
0	ham	go jurong point avail bugi n great world la e ...
1	ham	ok lar joke wif u oni
2	spam	free entri 2 wkli comp win fa cup final tkt 21...
3	ham	u dun say earli hor u c alreadi say
4	ham	nah think goe usf live around though

```
from sklearn.feature_extraction.text import TfidfVectorizer
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(tfidf_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)
```

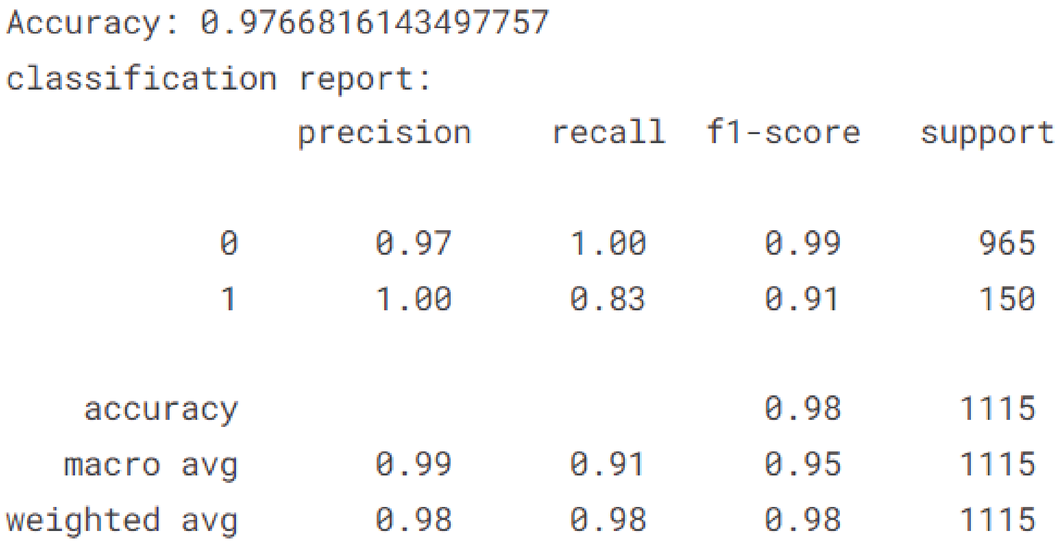
Random Forest Classifier

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

CODE

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_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_pred)

# Print the results
print("Accuracy:", accuracy)
print("classification report:\n", classification_rep)
```

OUTPUT



```
Accuracy: 0.9766816143497757
classification report:
              precision    recall  f1-score   support


           0       0.97      1.00      0.99       965
           1       1.00      0.83      0.91       150

    accuracy                           0.98      1115
   macro avg       0.99      0.91      0.95      1115
weighted avg       0.98      0.98      0.98      1115
```

Testing model

Test1

```
input_text = """\apple 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 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.
```