SMS Spam Collection Dataset and Preprocessing

In this step, we load the dataset for the spam detection project.The dataset is stored in a CSV file located at '/content/spam.csv'. We use the pandas library to read the CSV file and do some preprocessing to the dataset like text Cleaning, Stemming and etc.

To perform natural language processing tasks, we'll first install the Natural Language Toolkit (NLTK) library.

Analysis:

We import the pandas library using import pandas as pd.
We use pd.read_csv() to read the CSV file containing the dataset. The encoding='latin-1' argument is used to handle special characters.
We select only the relevant columns ('v1' for labels, 'v2' for email content) using data[['v1', 'v2']].
Finally, we display the resulting DataFrame to inspect the loaded data.

Code:

```
import pandas as pd
# Load the dataset
data = pd.read_csv('/kaggle/input/sms-spam-collection-dataset/spam.csv', encoding='latin-1')
data = data[['v1', 'v2']]  # Selecting only the relevant columns

data #printing
```

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...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will Ì_ b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

Data Preprocessing

In this step, we perform data preprocessing tasks, which include converting labels to binary values and removing duplicates from the dataset.

Explanation:

We use data['v1'].apply(lambda x: 1 if x == 'spam' else 0) to convert the labels. 'ham' is mapped to 0, and 'spam' is mapped to 1 in the 'v1' column.

We then remove duplicate rows from the dataset using data = data.drop_duplicates().

The resulting DataFrame is displayed to show the cleaned dataset.

Code:

```
# Convert 'ham' to 0 and 'spam' to 1 directly in the 'v1' column
data['v1'] = data['v1'].apply(lambda x: 1 if x == 'spam' else 0)

# removing duplicates
data = data.drop_duplicates()
data
```

Output

	v1	v2
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	1	This is the 2nd time we have tried 2 contact u...
5568	0	Will Ì_ b going to esplanade fr home?
5569	0	Pity, * was in mood for that. So...any other s...
5570	0	The guy did some bitching but I acted like i'd...
5571	0	Rofl. Its true to its name

5169 rows × 2 columns

Text Cleaning:

Text cleaning involves removing any unnecessary characters, symbols, or noise from the text data. This might include punctuation, special characters, and numbers.

Explanation:

We import the regular expression (re) module using import re.

The function clean_text() takes a string text as input and uses a regular expression to remove all characters except alphabetic characters .

The cleaned text is then returned.

We apply this function to the 'v2' column of the DataFrame using data['v2'].apply(lambda x: clean_text(x)). This cleans the text in each email.

Code:

```
import re
def clean_text(text):
    cleaned_text = re.sub(r'[^a-zA-Z]', ' ', text)
    return cleaned_text

data['v2'] = data['v2'].apply(lambda x: clean_text(x))
```

Lowercasing:

Converting all text to lowercase ensures that the model doesn't treat "Hello" and "hello" as different words.

Explanation:

We use the str.lower() method to convert all text in the 'v2' column to lowercase. This helps standardize the text data and ensure that the model is not case-sensitive.

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

Tokenization:

Tokenization involves splitting the text into individual words or tokens. The NLTK library can be used for this.

Explanation:

In this code cell, we use nltk.download('punkt') to download the necessary resources for tokenization from the Natural Language Toolkit (NLTK). This resource includes pre-trained models for tokenizing text into words or sentences. This step is essential for further text processing.

Code:

```
import nltk
nltk.download('punkt')
[nltk_data] Downloading package punkt to /usr/share/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Output

True

Stemming:

Stemming reduces words to their base forms. This can help in reducing the dimensionality of the feature space.

Explanation:
We import the PorterStemmer class from the NLTK library.

We initialize an instance of the PorterStemmer as stemmer.

We define a function stem_words(words) that takes a list of words and applies stemming to each word using the stemmer.stem() method.

We apply this function to the 'v2' column of the DataFrame, effectively reducing words to their base forms through stemming. This step can help improve the model's performance by reducing the feature space.

Code:

```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()

def stem_words(words):
    return [stemmer.stem(word) for word in words]

data['v2'] = data['v2'].apply(stem_words)
data
```

Output

	v1	v2
0	0	[go, until, jurong, point, crazi, avail, onli,...
1	0	[ok, lar, joke, wif, u, oni]
2	1	[free, entri, in, a, wkli, comp, to, win, fa, ...
3	0	[u, dun, say, so, earli, hor, u, c, alreadi, t...
4	0	[nah, i, don, t, think, he, goe, to, usf, he, ...
...	...	...
5567	1	[thi, is, the, nd, time, we, have, tri, contac...
5568	0	[will, b, go, to, esplanad, fr, home]
5569	0	[piti, wa, in, mood, for, that, so, ani, other...
5570	0	[the, guy, did, some, bitch, but, i, act, like...
5571	0	[rofl, it, true, to, it, name]

5169 rows × 2 columns