AI -Spam Classifier

Dataset and its detail explanation:

https://www.kaggle.com/datasets/team-ai/spam-text-message-classification

Building the AI Spam Classifier using dataset:

Create a Kaggle Account:

If you don't already have one, you'll need to create a Kaggle account. It's free to sign up.

AI spam Classifier progress

Download the Dataset:

Visit the dataset page on Kaggle ;

Click on the "Download" button on the right-hand side of the dataset page to download the dataset in the Excel format.

Load the Dataset in Excel:

Once the dataset is downloaded, locate the file on your local computer.

Double-click the Excel file to open it in Microsoft Excel or another compatible spreadsheet software.

Begin Your Analysis:

You can now start your messege/mail spam classification using Excel or any data analysis tool you prefer. You can perform tasks like data cleaning, data visualization, and statistical analysis depending on your project's goals.

If you plan to use a specific programming language like Python for your analysis, you can also load the dataset using libraries such as Pandas to work with the data programmatically. Here's an example of how to load an Excel dataset using Python and Pandas:

<u>Python code;</u>

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import
train_test_split
from tensorflow import keras
from
tensorflow.keras.preprocessing.text
import Tokenizer
from
tensorflow.keras.preprocessing.sequ
ence import pad_sequences

# Load the SMS spam dataset
data =
pd.read_csv('sms_spam_dataset.csv')
```

```
# Replace 'sms_spam_dataset.csv'
with your dataset


# Preprocessing the text data
max_words = 10000  # Maximum
number of words in the vocabulary
max_sequence_length = 100  #
Maximum sequence length for text


tokenizer =
Tokenizer(num_words=max_words,
filters='!"#$%&()*+,-
./:;<=>?@[\\]^_`{|}~\t\n',
lower=True)
tokenizer.fit_on_texts(data['text'])
```

```
X =
tokenizer.texts_to_sequences(data['te
xt'])
X = pad_sequences(X,
maxlen=max_sequence_length)

# Convert labels to binary format (0
for non-spam, 1 for spam)
y = data['label'].map({'ham': 0,
'spam': 1})

# Split the dataset into training and
testing sets
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
# Build an LSTM-based model
model = keras.Sequential()
model.add(keras.layers.Embedding(i
nput_dim=max_words,
output_dim=32,
input_length=max_sequence_length))
model.add(keras.layers.LSTM(64))
model.add(keras.layers.Dense(1,
activation='sigmoid'))

model.compile(loss='binary_crossent
ropy', optimizer='adam',
metrics=['accuracy'])

# Train the model
```

```
model.fit(X_train, y_train, epochs=5,
batch_size=64)

# Evaluate the model
loss, accuracy =
model.evaluate(X_test, y_test)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy:
{accuracy:.4f}")
```

In this code:

1. **We preprocess the text data by tokenizing it and converting it into sequences and padding the sequences to a fixed length.**

2. We split the dataset into training and testing sets.
3. We build a simple LSTM-based neural network model for text classification.
4. The model is compiled with binary cross-entropy loss and the Adam optimizer.
5. We train the model on the training data.
6. Finally, we evaluate the model on the test data.
7. Replace 'sms_spam_dataset.csv' with our dataset, and further hyperparameter tuning and more complex model architectures can be applied for

better performance. This example is a basic starting point for building an AI-based SMS spam classifier using deep learning techniques

.Preprocessing steps:

1.Data Collection and Understanding:

Acquire a labeled dataset of text messages, where each message is tagged as spam or not spam (ham).

Understand the structure of your dataset, such as the columns, labels, and data distribution.

2.Text Cleaning:

Remove any unnecessary characters, symbols, and formatting from the text data.

Convert text to lowercase to ensure uniformity.

Eliminate special characters, punctuation, and numbers that don't carry significant information.

3.Tokenization:

Split the text into individual words or tokens. Tokenization is essential for feature extraction.

4.Stopword Removal:

Remove common words (stopwords) like "and," "the," "is," etc. that don't contribute much to distinguishing spam from non-spam.

5.Stemming or Lemmatization:

Reduce words to their base or root form to reduce inflectional forms and improve feature extraction. Common methods include stemming and lemmatization.

6.Feature Extraction:

Convert text data into numerical features, such as using techniques like Count Vectorization, TF-IDF (Term Frequency-Inverse Document Frequency), or word embeddings like Word2Vec or GloVe.

7.Data Splitting:

Split the dataset into training and testing sets to evaluate the model's performance.

8.Balancing the Data (optional):

If your dataset is imbalanced (i.e., significantly more ham than spam or vice versa), consider

oversampling or undersampling to balance the classes.

9.Model Building:

Choose a suitable machine learning or deep learning algorithm for text classification. Common choices include Naive Bayes, Support Vector Machines (SVM), Random Forest, LSTM, etc.

10.Training:

Train the chosen model on the training data using the text features and corresponding labels.

11.Model Evaluation:

Evaluate the model's performance using metrics such as accuracy, precision, recall, F1-score, and AUC-ROC on the test dataset.

12.Hyperparameter Tuning:

Fine-tune the model by adjusting hyperparameters (e.g., regularization strength, learning rate) to optimize its performance.

13.Cross-Validation (optional):

Perform cross-validation to ensure the model's generalizability and reduce overfitting.

14.Model Deployment (if applicable):

Deploy the trained model to classify incoming messages in a real-world application.

15.Monitoring and Maintenance:

Continuously monitor and update the model to adapt to changes in spam patterns and improve its accuracy over time.