PROJECT 1: AI-POWERED SPAM CLASSIFIER

PHASE 1: PROBLEM DEFINITION AND DESIGN THINKING

PROBLEM DEFINITION:

The problem is to build an AI-powered spam classifier that can accurately distinguish between spam and non-spam messages in emails or text messages. The goal is to reduce the number of false positives (classifying legitimate messages as spam) and false negatives (missing actual spam messages) while achieving a high level of accuracy.

DESIGN THINKING:

STEP 1: DATA COLLECTION

- For the purpose of data collection, we can explore the availability of a dataset containing labelled samples of both spam and non-spam messages, which may be sourced from platforms like Kaggle.
- Ensure the dataset is diverse.

STEP 2: DATA PRE-PROCESSING

The text data needs to be cleaned and pre-processed. This involves removing special characters, converting text to lowercase, and tokenizing the text into individual words.

Text Data Cleaning:

This involves the removal of any unwanted or irrelevant characters from the text data. Special characters, symbols, and formatting that do not contribute to the understanding of the text are typically eliminated. For example, removing punctuation marks, HTML tags, or non-alphanumeric characters.

Converting Text to Lowercase:

Text data is often converted to lowercase to ensure uniformity in text analysis. This step helps in treating words in different cases (e.g., "Hello" and "hello") as the same, preventing potential duplication and improving the consistency of text features.

Tokenization:

Tokenization is the process of splitting the text into individual words or tokens. This step is crucial for natural language processing (NLP) tasks as it breaks down the text into manageable units for analysis. For example, the sentence "I love machine learning" would be tokenized into ["I", "love", "machine", "learning"].

STEP 3: FEATURE EXTRACTION

- Convert the tokenized words into numerical features that the neural network can understand.
- Common techniques include TF-IDF (Term Frequency-Inverse Document Frequency), which quantifies the importance of words in the documents.
- This step transforms raw text into a format suitable for deep learning models to process.

STEP 4: MODEL SELECTION

- Choose a deep learning architecture for the spam classifier. Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) units are suitable for sequential data like text.
- Design the neural network with embedding layers to represent words, LSTM layers to capture sequential patterns, and output layers with sigmoid activation for binary classification (spam or non-spam).

STEP 5: EVALUATION

We will measure the model's performance using metrics like accuracy, precision, recall, and F1-score.

Accuracy:

This metric tells us the percentage of messages the model correctly classified as either spam or non-spam. It's a measure of overall correctness.

Precision:

Precision tells us the proportion of messages the model classified as spam that were actually spam. In other words, it measures how many of the predicted spam messages were truly spam.

Recall:

Recall, also known as sensitivity, tells us the proportion of actual spam messages that the model correctly identified as spam. It measures the model's ability to find all the spam messages.

F1-Score:

The F1-score is a single number that combines both precision and recall. It provides a balanced measure of a model's performance, especially when you want to avoid either too many false positives (precision) or too many false negatives (recall).

STEP 6: ITERATIVE IMPROVEMENT

- We will fine-tune the model and experiment with hyperparameters to improve its accuracy.
- Fine-tuning the model involves making adjustments to the model's internal settings or architecture to improve its accuracy and effectiveness in classifying spam and non-spam messages.
- Hyperparameter Tuning: Fine-tuning hyperparameters, which are configuration settings that affect the model's behaviour, to find the best combination for optimal performance.
- Additionally, consider techniques like early stopping to prevent overfitting and optimize the model's performance continually.
- Iterate on the model design and hyperparameters until the desired accuracy and balance between false positives and false negatives are achieved.