Low-Level Design (LLD) Document

Spam Ham Classifier

**1. Introduction**

**1.1 Purpose**

The purpose of this document is to provide a detailed description of the components, classes, functions, and data flow within the Spam Ham Classifier project. This document serves as a guide for developers and engineers involved in the project, ensuring consistency and clarity in the implementation.

**1.2 Scope**

This document covers the implementation details of the Spam Ham Classifier, including text preprocessing, model prediction, and the user interface. It also includes details about deployment on Streamlit Cloud.

**2. Detailed Component Design**

**2.1 Text Preprocessing Modul**

**2.1.1 Description**

This module handles the preprocessing of the input SMS text to prepare it for classification. The preprocessing steps include:

- Lowercasing the text

- Tokenization

- Removal of stopwords and punctuation

- Stemming

**2.1.2 Functions**

**- `transform\_text(text: str) -> str`:**

**- Purpose:** Preprocesses the input text.

**- Inputs:** `text` - A string containing the SMS message.

**- Outputs:** A string of processed text, ready for vectorization.

**- Steps:**

1. Convert the text to lowercase.

2. Tokenize the text using `nltk.word\_tokenize()`.

3. Remove non-alphanumeric tokens.

4. Remove stopwords and punctuation.

5. Stem the remaining tokens using `PorterStemmer`.

6. Return the preprocessed text as a single string.

**2.1.3 Dependencies**

**- NLTK:** For tokenization and stopword removal.

**- String:** For punctuation removal.

**- PorterStemmer:** For stemming words.

**2.2 Vectorization Module**

**2.2.1 Description**

This module transforms the preprocessed text into a numerical format using the TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer, which is essential for model prediction.

**2.2.2 Functions**

**- `tfidf.transform(text: List[str]) -> scipy.sparse.csr.csr\_matrix`:**

**- Purpose:** Converts the preprocessed text into a TF-IDF vector.

**- Inputs:** A list of preprocessed text strings.

**- Outputs:** A sparse matrix of TF-IDF features.

**- Details:** This vector is used as input to the machine learning model.

**2.2.3 Dependencies**

**- Scikit-learn:** For the TF-IDF vectorizer.

**2.3 Model Prediction Module**

**2.3.1 Description**

This module uses the Naive Bayes Multinomial model to predict whether the SMS is spam or not based on the TF-IDF vector.

**2.3.2 Functions**

**- `model.predict(vector: scipy.sparse.csr.csr\_matrix) -> int`:**

**- Purpose:** Predicts the class label (spam or not spam) for the input vector.

**- Inputs:** A TF-IDF vector.

**- Outputs:** An integer (1 for spam, 0 for not spam).

**- Details:** The prediction is then displayed in the user interface.

**2.3.3 Dependencies**

**- Scikit-learn:** For the Naive Bayes Multinomial model.

**2.4 User Interface Module**

**2.4.1 Description**

This module is responsible for the interactive user interface that allows users to input text and view classification results.

**2.4.2 Functions**

**- `st.text\_area(label: str) -> str`:**

**- Purpose:** Provides a text area for user input.

**- Inputs:** `label` - A string that labels the text area.

**- Outputs:** The user input as a string.

**- `st.button(label: str) -> bool`:**

**- Purpose:** Provides a button that triggers the prediction when clicked.

**- Inputs:** `label` - A string that labels the button.

**- Outputs:** A boolean indicating whether the button was clicked.

- **`st.header(text: str)`:**

**- Purpose:** Displays the classification result to the user.

**- Inputs:** `text` - A string containing the result ("Spam" or "Not Spam").

**2.4.3 Flow**

1. The user enters SMS text into the text area.

2. The user clicks the "Predict" button.

3. The text is preprocessed, vectorized, and passed through the model.

4. The result is displayed in the header.

**2.4.4 Dependencies**

**- Streamlit:** For creating and managing the web interface.

**2.5 Deployment Module**

**2.5.1 Description**

Handles the deployment of the application on Streamlit Cloud.

**2.5.2 Components**

**- Streamlit Cloud:** Hosts the application and serves it to users.

**- Requirements File:** Lists all dependencies needed for the application to run on Streamlit Cloud.

**2.5.3 Steps**

**1. Prepare the environment:** Ensure all dependencies are listed in `requirements.txt`.

**2. Deploy the app:** Upload the application files (including the model and vectorizer) to Streamlit Cloud.

**3. Monitor:** Use Streamlit Cloud’s dashboard to monitor app performance and logs.

**3. Data Flow Diagram (DFD)**

**3.1 Level 0 DFD**

**- User Input:** User enters SMS text.

**- Processing:** Text is preprocessed, vectorized, and classified.

**- Output:** Display classification result (Spam or Not Spam).

**3.2 Level 1 DFD**

**1. Input:**

- User inputs SMS text through Streamlit UI.

**2. Text Preprocessing:**

- Convert text to lowercase.

- Tokenize text.

- Remove stopwords and punctuation.

- Stem the words.

**3. Vectorization:**

- Convert processed text into a TF-IDF vector.

**4. Model Prediction:**

- Input TF-IDF vector into Naive Bayes model.

- Generate prediction (spam or not spam).

**5. Output:**

- Display result on Streamlit UI.

**4. Error Handling**

**4.1 User Input Errors**

**- Empty Input:** Display an error message if no text is entered.

**- Non-Text Input:** Validate that the input is text before processing.

**4.2 System Errors**

**- Model Loading Failure:** Ensure that the model is loaded correctly; handle exceptions and log errors.

**- Vectorization Issues:** Validate that the text is correctly vectorized before prediction.

**5. Testing Strategy**

**5.1 Unit Testing**

**- Text Preprocessing:** Test with various inputs to ensure proper tokenization, stopword removal, and stemming.

**- Model Prediction:** Test with known spam and ham examples to validate prediction accuracy.

**5.2 Integration Testing**

- Test the entire flow from text input to prediction output to ensure all modules work together as expected.

**5.3 User Interface Testing**

- Test the responsiveness and usability of the Streamlit interface on different devices and screen sizes.

**6. Conclusion**

This Low-Level Design document provides a detailed guide for the implementation of the Spam Ham Classifier project. By following this LLD, developers can ensure that the project is implemented consistently and meets the desired functionality and performance criteria.