Low Level Design (LLD)

**Spam Ham Classifier**

Document Version: 2.0

Last Date of Revision: 23/12/2022

# **Document Version Control**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date Issued** | **Version** | **Description** | **Author** |
| 22/12/2022 | 1 | Initial LLD – V1.0 | Himanshu Goswami |
| 23/12/2022 | 2 | Unit Test Cases defined and appended | Himanshu Goswami |
|  |  |  |  |
|  |  |  |  |

# **Contents**

**Document Version Control**……….……………………………………….……………………….……………………………….……2

1. **Introduction**…………………….……………………………………………………………………..………………………………….4
   1. Why this Low-Level Design Document? .……………………………………..……………..……………………….4
   2. Scope……………….……………………………………………………………………….….…………………………..…………4
2. **Architecture**………………………………….………………………………………………….…………………………………………5
3. **Architecture Description**…………………………………………………………………….………………………………..…….6
   1. Data Description…..……………………………………………………………………….…………………………………….6
   2. Data Insertion into Database..…………………………………………………………………………………….……….6
   3. Data Ingestion………………………………………………………………………………………………….………………….6
   4. Splitting Data…………………………………………………………………………………………………….…………………6
   5. Data Validation…………………………………………………………………………..……………….………………………6
   6. Data Pre-Processing…………………………………………………………………………………………………………….6
   7. Transforming Text Data……………………………………………………………………………………………………….6
   8. Model Building…………………………………………………………………………………………………………………….7
   9. Model Evaluation…………………………………………………………………………………………………………………7
   10. Model Pusher…………………………………………………………………………………………………………..7
   11. Cloud Setup……………………………………………………………..……….……………………………………..7
   12. Pushing App to Cloud……………………………………………………………………………………………….7
   13. Data From User………………………………………………………………………………………………………..7
   14. Data Transformation………………………………………………………………………………………………..7
   15. Model Call for Prediction………………………………………………………………………………………….7
   16. Spam Classification…………………………………………………………………………………………………..7
4. **Unit Test Cases**……………………………………………………………………………………………………………………………8

# **Introduction**

## **Why this High-Level Design Document?**

The goal of LLD or a Low-Level Design Document (LLD) is to give the internal logical design of the actual program code for Spam Ham Classifier. LLD describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer can directly code the program from the document.

## **Scope**

Low-Level design (LLD) is a component-level design process that follows a step-by-step refinement process. This process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work.

# **Architecture**

Data Insertion into Database

Data Ingestion

Start

Data for Recommendation

Transforming Text Data

Data Preprocessing

Data Validation

Splitting Data

Model Pusher

Model Evaluation

Model Building

NLKT Techniques

Data From User

Application Start

Pushing app to Cloud

Cloud Setup

Data Transformation

Model Call for Prediction

End

Spam Classification

# **Architecture Description**

## **Data Description**

This corpus has been collected from free or free for research sources at the Internet:

* A collection of 425 SMS spam messages was manually extracted from the Grumbletext Web site.
* A subset of 3,375 SMS randomly chosen ham messages of the NUS SMS Corpus (NSC), which is a dataset of about 10,000 legitimate messages collected for research at the Department of Computer Science at the National University of Singapore.
* A list of 450 SMS ham messages collected from Caroline Tag's PhD Thesis.

This dataset contains a total of 5574 texts classified as ham or spam.

## **Data Insertion into Database**

1. Database Creation and connection – Create a database with name passed.
2. Inserting data in JSON format in MongoDB.

## **Data Ingestion**

Imported data directly from database and created a DataFrame (tabular form) out of the JSON formatted data.

## **Splitting Data**

Divided data into three parts – Train set, Test set and Validation set.

## **Data Validation**

Creating perimeters to ensure any upcoming data is of same format as the current one. If conditions aren’t met then program must stop here, else proceed to next steps.

## **Data Pre-processing**

Data Pre-processing or Data Transformation steps includes Null value handling, removing duplicated values etc.

## **Transforming Text Data**

Transforming the text so as to make it ready for model building. Lower casing the text, stop words removal, punctuation removal, Tokenization, stemming.

## **Model Building**

We will use TFIDF first to vectorize the text data. Then we will find the best model. We will calculate the Accuracy Score, Precision Score & False Positive section of Confusion Matrix. Whichever algorithm gives us max accuracy & precision score as well as least False Positive value in confusion matrix table (preferably 0), we will pick that model.

## **Model Evaluation**

We will evaluate the model using Test Data we segregated in Data Ingestion. If model new model outperforms ongoing model, then we will replace it.

## **Model Pusher**

After picking best model, we push the best performing model, transformer & encoder files into other directory to be easily picked during operation.

## **Cloud Setup**

We will be deploying the model on Streamlit Cloud service.

## **Pushing App to Cloud**

After cloud setup, we push the app to cloud and get it configured as per the use case to get text data from the user easily.

## **Data From User**

A text field provided on the app allows user to input text.

## **Data Transformation**

Received data will get transformed using NLTK techniques.

## **Model Call for Prediction**

Data gets through the model which instantly classifies whether the text is a ham or spam.

## **Spam Classification**

As soon the user inputs text, within a second user gets the display of result on the app front page that whether the text is spam or ham.

# **Unit Test Cases**

|  |  |  |
| --- | --- | --- |
| Test Case Description | Pre-Requisite | Expected Result |
| Verify whether the application URL is accessible to the user. | 1. Application URL should be defined. | Application URL should be accessible to the user. |
| Verify whether the application loads completely for the user when the URL is accessed. | 1. Application URL is accessible  2. Application is deployed | The application should load completely for the user when the URL is accessed. |
| Verify whether the user is able to see input field. | 1. Application is accessible. | User should be able to see input field. |
| Verify whether user is able to edit input field. | 1. Application is accessible. | User should be able to edit input field. |
| Verify whether the user get submit button to submit the input. | 1. Application is accessible. | User should get submit button to submit the inputs. |
| Verify whether the user is presented with recommended results on clicking submit. | 1. Application is accessible.  2. Input from the user. | User should be presented with recommended results on clicking submit. |
| Verify whether the recommended result is in accordance to the selections user made. | 1. Application is accessible.  2. Input from the user. | The recommended results should be in accordance to the selection user made. |