**NLP based Duplicate Bug Report Detection using Supervised Machine Learning Algorithms**

**Design Document**

**Version 1.0**



**Group Id: F22024F062 (BC190400681)**

**Supervisor Name: Saad Ahmed**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date (dd/mm/yyyy)** | **Version** | **Description** | **Author** |
| 18/02/2023 | 1.0 | In this project, we aim to classify bug reports using machine learning models and natural language processing techniques. The project involves preprocessing the textual data in the bug reports using techniques such as tokenization, stop word removal, and lemmatization.  The processed data is then used to train and test machine learning algorithms, such as Naive Bayes, Support Vector Machines, and Random Forests, to classify the bug reports. The goal is to effectively triage and fix bugs by clearly identifying important features in the bug reports. | BC190400681 |

**Table of Contents**

1. [Introduction of Design Document -------------------------------------- Page No# 4](#Introduction)
2. [Literature Review (Optional) -------------------------------------------- Page No# 4](#LiteratureReview)
3. [Methodology --------------------------------------------------------------- Page No# 4](#Methodology)
4. [Material and Methods ----------------------------------------------------- Page No# 5](#Materialandmethods)
5. [Interface Design (Optional) ---------------------------------------------- Page No# 5](#interfacedesign)
6. [Test Cases ------------------------------------------------------------------ Page No# 5](#testcases)
7. **Introduction of Design Document**

The purpose of this design document is to provide outlines of the system's design, features, and functionalities, and will serve as a guide for me to build and implement the system efficiently.

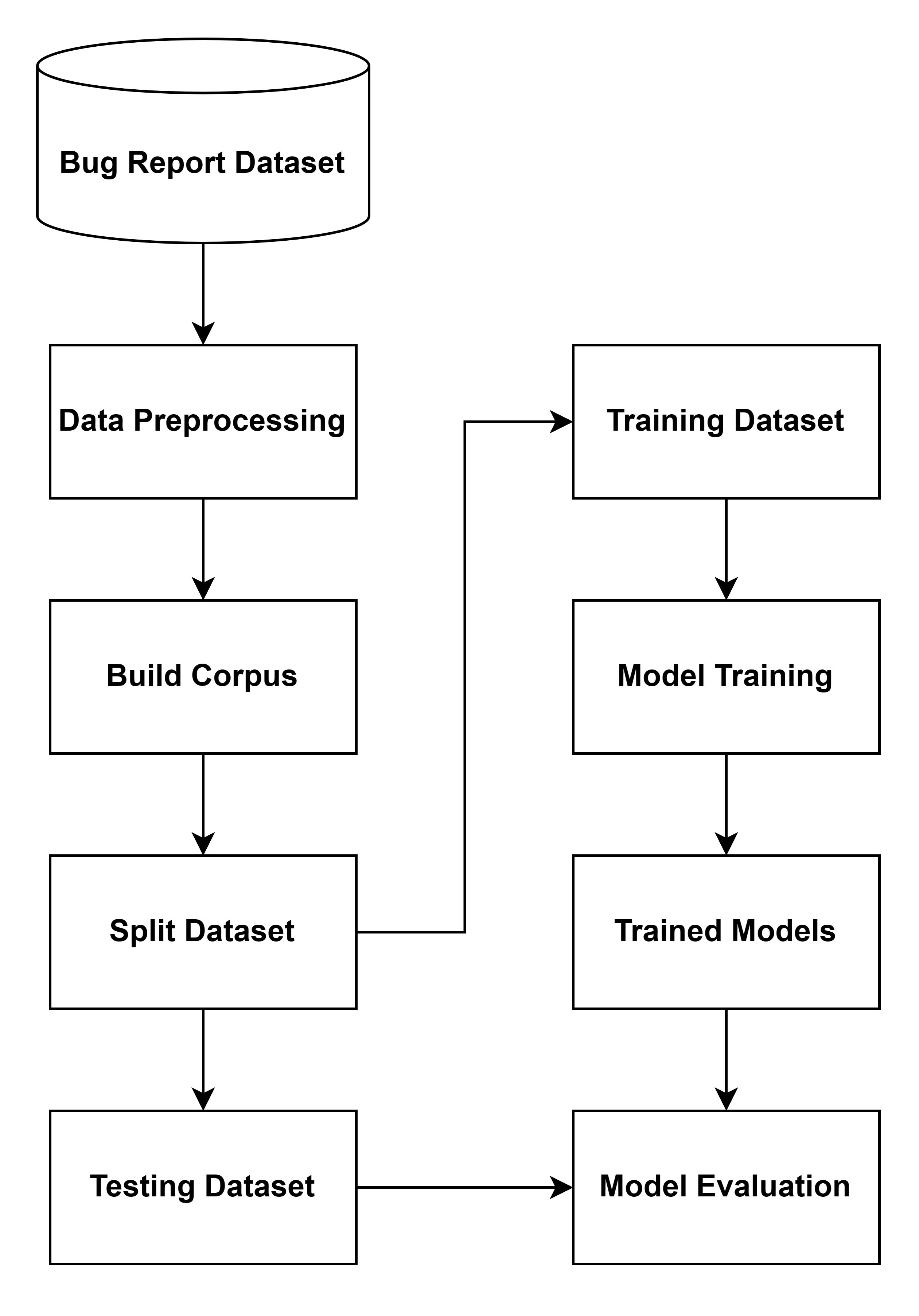
The design document will also benefit me by enabling me to identify potential design flaws or issues early on in the development process, ensuring the system's final version meets the requirements.

The system will operate by processing textual data through different NLP techniques such as tokenization, stop-word removal, and lemmatization, and then classifying the bug reports using machine learning algorithms.

1. **Literature Review (Optional)**

Entity Relationship Diagram (ERD) are most often used to design or debug relational databases. Because in this project there isn’t any database, therefore ERD is not required.

1. **Methodology**



1. **Material and Methods**

For this project, the primary materials and tools used for development will include programming languages such as Python and its libraries such as Scikit-Learn, Pandas, NumPy, and NLTK. These tools will be used to preprocess the data, implement the machine learning algorithms, and perform statistical analysis on the results obtained.

In addition, a dataset containing real-world text data will be used to train and test the machine learning models. The dataset will be acquired from a reliable source and will be preprocessed to remove any irrelevant information, outliers, or errors that might affect the performance of the models.

The methods used in this project will include various NLP techniques such as tokenization, stemming, and lemmatization, as well as machine learning algorithms such as Support Vector Machines (SVM), Naive Bayes, and Random Forests. The performance of these algorithms will be evaluated using metrics such as accuracy, precision, recall, and F1-score.

1. **Interface Design (Optional)**

Interface design/GUI is not required for this project.

1. **Test Cases**

Here are the test cases for each use case scenario:

**Test cases for use case scenario #1:**

|  |  |
| --- | --- |
| Test Case Title | Valid Data Processing |
| Test Case ID | TC-1.1 |
| Description | Test to verify that the system processes the data correctly with valid NLP techniques. |
| Pre-Conditions | The system environment must be configured. |
| Test Steps | 1. Launch the system. 2. Select the option to apply data processing techniques. 3. Choose the NLP techniques to apply. 4. Apply the selected techniques on the data. |
| Expected Results | The system should apply the selected NLP techniques on the data and return the processed data without errors. |
| Post-Conditions | The data has been processed using the selected NLP techniques. |
| Actual Results | The system successfully applies the selected NLP techniques on the data and returns the processed data without errors. |

|  |  |
| --- | --- |
| Test Case Title | Invalid Data Processing |
| Test Case ID | TC-1.2 |
| Description | Test to verify that the system fails if invalid NLP techniques are selected. |
| Pre-Conditions | The system environment must be configured. |
| Test Steps | 1. Launch the system. 2. Select the option to apply data processing techniques. 3. Choose invalid NLP techniques to apply. 4. Apply the selected techniques on the data. |
| Expected Results | The system should fail to apply the selected invalid NLP techniques on the data and return an error message. |
| Post-Conditions | The data should not be processed with the invalid NLP techniques. |
| Actual Results | The system fails to apply the selected invalid NLP techniques on the data and returns an error message. |

**Test cases for use case scenario #2:**

|  |  |
| --- | --- |
| Test Case Title | Successful Corpus Building |
| Test Case ID | TC-2.1 |
| Description | Test to verify that the system successfully builds the corpus using processed data. |
| Pre-Conditions | The data must be processed using NLP techniques. |
| Test Steps | 1. Launch the system. 2. Select the option to build the corpus. 3. Verify that the system successfully builds the corpus. |
| Expected Results | The system should successfully build the corpus using the processed data. |
| Post-Conditions | The corpus has been built and is ready to use for training machine learning models. |
| Actual Results | The system successfully builds the corpus using the processed data. |

|  |  |
| --- | --- |
| Test Case Title | Invalid Data Format |
| Test Case ID | TC-2.2 |
| Description | Test to verify that the system fails to build the corpus if the data is not in the correct format. |
| Pre-Conditions | The data must be processed using NLP techniques. |
| Test Steps | 1. Launch the system. 2. Select the option to build the corpus. 3. Provide the system with data that is not in the correct format. 4. Verify that the system fails to build the corpus. |
| Expected Results | The system should fail to build the corpus if the data is not in the correct format. |
| Post-Conditions | The corpus has not been built and cannot be used for training machine learning models. |
| Actual Results | The system failed to build the corpus because the data is not in the correct format. |

**Test cases for use case scenario #3:**

|  |  |
| --- | --- |
| Test Case Title | Data Splitting - Valid Percentage |
| Test Case ID | TC-3.1 |
| Description | Test to verify that the system correctly splits the data into training and testing sets when a valid percentage of data is provided for testing. |
| Pre-Conditions | The corpus must be built. |
| Test Steps | 1. Launch the system. 2. Select the option to split the data. 3. Input a valid percentage of data to be used for testing. 4. Click on the "Split" button. |
| Expected Results | The system should split the data into training and testing sets according to the specified percentage of data for testing. |
| Post-Conditions | The data has been split into training and testing sets. Actual Results: |
| Actual Results | The system split the data into training and testing sets according to the specified percentage of data for testing. |

|  |  |
| --- | --- |
| Test Case Title | Data Splitting - Invalid Percentage |
| Test Case ID | TC-3.2 |
| Description | Test to verify that the system fails if the percentage of data for testing is set to a value that is too low or too high. |
| Pre-Conditions | The corpus must be built. |
| Test Steps | 1. Launch the system. 2. Select the option to split the data. 3. Input an invalid percentage of data to be used for testing (e.g., less than 1% or more than 99%). 4. Click on the "Split" button. |
| Expected Results | The system should display an error message indicating that the percentage of data for testing is invalid and cannot be used. |
| Post-Conditions | The data has not been split into training and testing sets. Actual Results: |
| Actual Results | The system displayed an error message indicating that the percentage of data for testing is invalid and can’t be used. |

**Test cases for use case scenario #4:**

|  |  |
| --- | --- |
| Test Case Title | Successful Training with Default Parameters |
| Test Case ID | TC-4.1 |
| Description | Test the successful training of a model with default parameters. |
| Pre-Conditions | The dataset must be split into training and testing sets, and the selected model must be installed. |
| Test Steps | 1. Load the training and testing datasets into the system. 2. Select the model to be trained and initiate the training process. 3. Wait for the training process to complete. 4. Save the trained model. |
| Expected Results | The system should successfully train the model with default parameters and save the trained model without any errors. |
| Post-Conditions | The trained model is saved and ready to use for the evaluation process. |
| Actual Results | The system successfully trained the model with default parameters and saved the trained model without any errors. |

|  |  |
| --- | --- |
| Test Case Title | Model Training with Custom Parameters |
| Test Case ID | TC-4.2 |
| Description | Test the successful training of a model with custom parameters. |
| Pre-Conditions | The dataset must be split into training and testing sets, and the selected model must be installed. |
| Test Steps | 1. Load the training and testing datasets into the system. 2. Select the model to be trained and set custom training parameters. 3. Initiate the training process. 4. Wait for the training process to complete. 5. Save the trained model. |
| Expected Results | The system should successfully train the model with custom parameters and save the trained model without any errors. |
| Post-Conditions | The trained model is saved and ready to use for the evaluation process. |
| Actual Results | The system successfully trained the model with custom parameters and saved the trained model without any errors. |

**Test cases for use case scenario #5:**

|  |  |
| --- | --- |
| Test Case Title | Model Evaluation with Single Algorithm |
| Test Case ID | TC-5.1 |
| Description | Test the evaluation of a single algorithm on a dataset and compare its results with the expected results. |
| Pre-Conditions | The dataset must be split into training and testing sets, and the model must be trained. |
| Test Steps | 1. The user selects the algorithm to be evaluated. 2. The system evaluates the algorithm and presents the evaluation results. 3. The user compares the evaluation results with the expected results. |
| Expected Results | The evaluation results for the selected algorithm should match the expected results, including the confusion matrix, accuracy, precision, and recall. |
| Post-Conditions | The results of the evaluation are documented for future reference. |
| Actual Results | The evaluation results match the expected results, including the confusion matrix, accuracy, precision, and recall. |

|  |  |
| --- | --- |
| Test Case Title | Model Evaluation with Multiple Algorithms |
| Test Case ID | TC-5.2 |
| Description | Test the evaluation of multiple algorithms on a dataset and compare their results with each other. |
| Pre-Conditions | The dataset must be split into training and testing sets, and the models must be trained. |
| Test Steps | 1. The user selects the algorithms to be evaluated. 2. The system evaluates the algorithms and presents the evaluation results for each algorithm. 3. The user compares the evaluation results for each algorithm and decides on which algorithm to use. |
| Expected Results | The evaluation results for each algorithm should be presented accurately, including the confusion matrix, accuracy, precision, and recall. The user should be able to make a decision on which algorithm to use based on the comparison of the results. |
| Post-Conditions | The results of the evaluation are documented for future reference, and the decision on which algorithm to use is recorded. |
| Actual Results | The evaluation results for each algorithm are presented accurately, and the user can decide on which algorithm to use based on the comparison of the results. The decision on which algorithm to use is recorded. |

**Test cases for use case scenario #6:**

|  |  |
| --- | --- |
| Test Case Title | Successful Retraining of Model |
| Test Case ID | TC-6.1 |
| Description | Test the system's ability to successfully retrain the model with custom parameters and save the trained model without any errors. |
| Pre-Conditions | The current model's accuracy is less than 60%. |
| Test Steps | 1. Check the accuracy of the current model and find it to be less than 60%. 2. Select the option to retrain the model. 3. Adjust the suggested training parameters. 4. Retrain the model with the updated training parameters. 5. Evaluate the new model. |
| Expected Results | The new model is trained with the updated parameters and its accuracy is improved to be greater than or equal to 60%. |
| Post-Conditions | The system has a new model with improved accuracy. The user can use the new model for bug report classification. |
| Actual Results | The system successfully retrains the model with custom parameters, and the new model's accuracy is improved to be greater than or equal to 60%. |

|  |  |
| --- | --- |
| Test Case Title | Failed Retraining of Model |
| Test Case ID | TC-6.2 |
| Description | Test the system's ability to handle failed retraining of the model due to unsatisfactory results. |
| Pre-Conditions | The current model's accuracy is less than 60%. Test Steps: |
| Test Steps | 1. Check the accuracy of the current model and find it to be less than 60%. 2. Select the option to retrain the model. 3. Adjust the suggested training parameters. 4. Retrain the model with the updated training parameters. 5. Evaluate the new model. 6. Check the accuracy of the new model and find it to be still less than 60%. |
| Expected Results | The system suggests different training parameters or a different algorithm to try again for improving the model's accuracy. |
| Post-Conditions | The system has either a new model with improved accuracy or continues with the existing model for bug report classification. |
| Actual Results | The system suggests different training parameters or a different algorithm to try again for improving the model's accuracy when the accuracy of the new model is still less than 60%. |