Classification of Issues
Phase - 2 Report

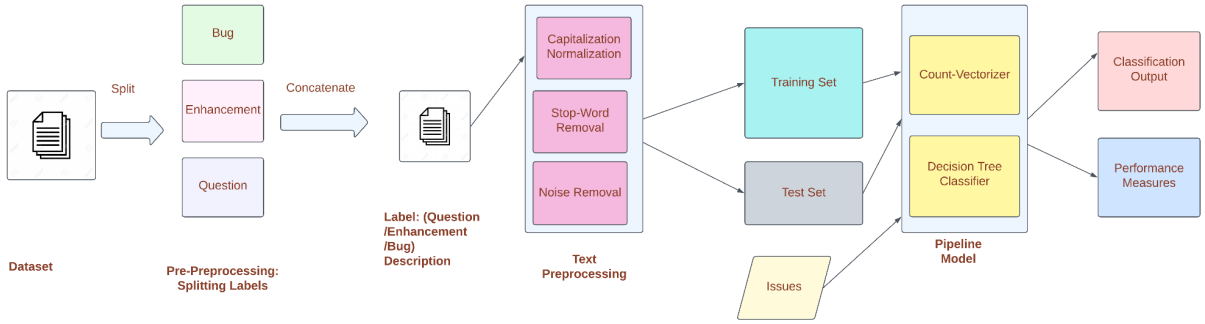
Issues on Github

Issues let you track your work on GitHub, where development happens. When you mention an issue in another issue or pull request, the issue's timeline reflects the cross-reference so that you can keep track of related work. To indicate that work is in progress, you can link an issue to a pull request. When the pull request merges, the linked issue automatically closes.

Here we aim at classifying the issues on github using Machine Learning, it involves analyzing the issues and assigning specific labels using various models. Github has issues labeled such as bug, help, enhancement, question. The model will learn from the dataset and will be able to make predictions. Using this prediction, the model will be able to classify a given issue and assign the label. If an issue is found out to be a bug it will be classified as a 'bug' label.

Study Design

Architectural Diagram



We have chosen the github issues dataset for training our model. It is a collection of various issues collected from numerous github repositories. After collecting the data, we perform the exploratory data analysis on the dataset.
It can be observed that the data is not well organized, so we need to format it before feeding it into the machine learning model. We begin with extracting the labels from each column. The columns: enhancement, bug and label are extracted. Now these extracted columns are concatenated and we come up with a single column for the label. This concatenation can be achieved by converting each of the columns into a dataframe. We then cut down the excess text in the label and make it into the required format. As the data still is noisy, we preprocess it by

removing the stop words and converting all the characters into lowercase letters. Stopwords are the commonly used words in a given language. These words being very common, they carry little information during model training. Removing the stopwords enables the model to focus on the most important words adding value to the model during training. As the data is ready, we split it into training and testing data to feed the model. Using the pipeline architecture, we automate the process of vectorization, converting the words into vectors and then training the model using various machine learning algorithms. Once the pipeline is fit, we can calculate the metrics like accuracy, precision, recall and the f-score to evaluate the model performance. We can try testing the model performance by using a test issue and running it through the model. Therefore, we have achieved the desired results through the model built.

Dataset:

https://tickettagger.blob.core.windows.net/datasets/dataset-labels-top3-30k-real.txt

Link to the Code

https://colab.research.google.com/drive/13wwRt26ZiXcm6qYS3Emflxc9eSe6Ok-9?usp=sharing

Experiments:

RQ1- How well do various simple ML-based algorithms perform?

RQ2- How well does Ticket Tagger perform in comparison to a simple ML-based algorithm?

Preliminary results:

RQ-1

How well does Ticket Tagger perform in comparison to a simple ML-based algorithm?

To answer this question we use github-labels-top3-34k dataset and compare the precision, recall and F-measure of 3 ML-based algorithms.The dataset has a total of 34,069 issues.A few steps to pre-process the data were required.We then split this data into a train and test data set in 7:3 ratio respectively.

Results:

For the first Ml-based model we take the Naive Bayes model.Figure no. 1 shows the precision, recall and F-measure metrics score of the Naive Bayes model when identifying the bug, enhancement and question issues.The second Ml-based model taken for comparison is the Decision Tree Classifier. Figure no. 2 shows the precision, recall and F-measure metrics score of the Decision Tree Classifier model when identifying the bug, enhancement and question issues. From these results we can analyze that the Decision Tree Classifier has a more consistent

precision, recall and F-measure than the Naive Bayes. For the third Ml-based model, the Logistic Regression model was taken. Figure no. 3 shows the precision, recall and F-measure metrics score of the Logistic Regression model when identifying the bug, enhancement and question issues. Upon analysis of the obtained metrics scores from the 3 different models we can see that the Logistic Regression model outperforms both the Naive Bayes and Decision Tree Classifier models. Figure no. 3 shows that the Logistic Regression model has an F-measure score above 0.70 for the bug and enhancement label in comparison to. The below 0.70 values of both Naive Bayes and Decision Tree Classifier models, thus indicating the Logistic Regression model to be the best.

	precision	recall	f1-score	support
bug	0.74	0.56	0.64	4872
enhancement	0.58	0.85	0.69	4301
question	0.47	0.09	0.15	1040
accuracy			0.63	10213
macro avg	0.60	0.50	0.49	10213
weighted avg	0.64	0.63	0.61	10213

Figure no. 1

	precision	recall	f1-score	support
bug	0.68	0.70	0.69	4872
enhancement	0.65	0.66	0.66	4301
question	0.30	0.21	0.25	1040
accuracy			0.64	10213
macro avg	0.54	0.53	0.53	10213
weighted avg	0.63	0.64	0.63	10213

Figure no. 2

	precision	recall	f1-score	support
bug	0.73	0.74	0.73	4872
enhancement	0.69	0.73	0.71	4301
question	0.41	0.29	0.34	1040
accuracy			0.69	10213
macro avg	0.61	0.59	0.59	10213
weighted avg	0.68	0.69	0.68	10213

Figure no. 3

RQ2

How well does Ticket Tagger perform in comparison to a simple ML-based algorithm?

To answer this question we compare the precision, recall and F-measure matrics of our above seen best ML-based model to the precision, recall and F-measure matrics of Ticket Tagger. We tested both Ticket Tagger and Logistic Regression model on a similar unbalanced dataset as it is more representative of reality. The distribution of issue types in the unbalanced data is as follows: 16,355(48%) tickets labeled as bug, 14,228(41.8%) marked as an enhancement and 3458(10.2%) question issues.

Results:

Figure no. 4 shows the effectiveness of Ticket Tagger to automatically identify issues on an unbalanced data set using the precision, recall and F-measure metrics. Ticker Tagger obtained a F-measure value of 0.75 for the bug label, 0.74 for the enhancement label and 0.48 for the question label. When these values are compared to the values of the Logistic Regression model which has values of 0.73 for the bug label, 0.71 for the enhancement label and 0.34 for the question label it is observed that Ticket Tagger outperforms the baseline Logistic Regression model approach for all labels and in all precision, recall, and F-measure metrics.

Metrics	Bug	Enhancement	Question
Precision	0.79	0.73	0.44
Recall	0.72	0.74	0.53
F-measure	0.75	0.74	0.48

Figure no. 4

Future Research:

For future research we could compare the difference between the precision, recall and F-measure metrics of Ticket Tagger when it is tested on a data set where the stop words such as ‘why’, ‘how’ or ‘what’ are eliminated in pre-processing to the existing precision, recall and F-measure metrics of Ticket Tagger when the stop words are not eliminated.