
# Factors Affecting Pull-Request Acceptance and their Potential to Guide Contributors of Open-Source Software Repository

Pritha Sarkar

School of Computing Science

Sir Alwyn Williams Building

University of Glasgow

G12 8RZ

A dissertation presented in part fulfillment of the requirements of the Degree of Master of Science at the University of Glasgow

01 September 2022

# Abstract

Pull-based development has brought about a new era within the open-source software (OSS) world. As GitHub remains the most sought-after platform for code-hosting, code reviewing, and code integration, software developers flock to the platform to offer their takes on projects. However, most of the time pull requests from contributors go unmerged. The undertaken study attempts to determine if a contributor's pull-request has the potential to get merged or not. The experiment tries to determine the most impactful features related to a pull-request. For example, researchers have observed that the gender of the contributor, number of open issues by the contributor, programming language used in the development, etc. plays a part in the acceptance decision. Furthermore, an attempt has been made to put forth some possible changes a contributor can make so that their future pull-requests have a higher chance of acceptance.

The dataset used for this experiment is "the most comprehensive and largest one toward a complete picture for pull-based development research" [3] and contains a total of 3,347,937 pull-requests and 120 features related to those pull-requests.

# Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic form.

Name: Pritha Sakar

Signature: 

# Acknowledgements

I would like to thank Dr. Handan Gul Calikli for all her advice and encouraging words throughout the past two months. I would also like to thank Kenneth Obando Rodríguez (MSc), Interim professor at the Technological Institute of Costa Rica and researcher for the "State of the Nation Program" for lending me a remote workspace on his personal computer.

# Contents

# Chapter 1 Introduction

Distributed software development projects leverage collaboration models and patterns to streamline the process of integrating incoming contributions [2]. The pull-based development model is a form of distributed software development that has gained tremendous popularity in previous years [2].

Pull-based model is quite different from the classic way of contributing. Pull-request development has two main components- contributor and integrator. The integrators are part of the core team in charge of creating the project. The contributors are often, independent developers trying to contribute to the project. Contributors fork or locally duplicate the main repository of a project [1] they would like to contribute towards. Subsequently, they make changes locally and independently, finally creating a pull request to ask that their changes be merged into the main repository [1]. This is vastly different from the traditional way of contributing which included sending change sets to the development mailing list to issue tracking systems or through direct access to the version control system [2].

Internet hosting platforms for software development and version control, such as GitHub, GitLab, and Bitbucket, employ pull-based development for easy collaboration among individuals all around the world. Not only that, they allow users to subscribe to and/or visualize information about activities of projects and users and offer threaded asynchronous communication within PRs [2]. However complex, this method has soared the popularity OSSs to newer heights. Much needed is multiple extensive research through pull-requests determining the factors at play regarding the acceptance of it. The overall goal of this work is:

**RQ1:** What factors facilitate or deteriorate pull-request acceptance?

**RQ2:** To what extent can recommendations based on these factors guide contributors for pull-request acceptance?

# Chapter 2 Related Work

Open-Source Software projects create communities where developers wishing to participate submit contributions, usually as source code patches [2]. The onion model is a widely accepted way of organizing OSS communities by contributions [14]: a core team receives contributions and determines whether to accept it or reject it based on technical merits or trust [2].

Popularity of OSS projects and GitHub has skyrocketed over the decade. As of January 2020, GitHub reports having over 40 million users and more than 190 million repositories (including at least 28 million public repositories), making it the largest host of source code in the world [41]. Due to the massive collection of repositories available on GitHub, researchers have harvested and put together multiple datasets that could be used for multiple purposes. One of the first of such datasets produced was constructed from almost 900 projects and 350,000 pull-requests [15]. The feature selection for the dataset was largely based upon pull-requests related to submission of patch, or bug fixing [15]. Moreover, some features were based on semi-structured interviews of GitHub developers [15]. Another massive dataset curated from pull-requests was the GHTorent Dataset [42]. The dataset was able to filter through duplicate pull-requests and harvested 14 features related to each dataset [42]. Finally, the largest dataset available related to pull-request and that was used in this experiment has been harvested from 11,230 projects and contains 96 features (unrestricted version) [3]. The sheer volume of the dataset and the distinctive features associated with each pull-request were thought to be able to provide a broadened understanding of pull-request acceptance factors and thus have been used for this experiment[1].

Due to the availability of the above-mentioned datasets, multiple pieces of exploratory research were able to be carried out. These previously done researches recognized arrays of features that contribute positively/ negatively towards the acceptance of pull-request. For example, the experience of developers, conceptualized as the count of previous pull-requests, previous pull request acceptance rate, accepted commit count, as well as days since account creation, influences the pull request acceptance [3] [15 - 20]. It is more likely that an individual pull-request is accepted if they are part of the core team [3] [21 - 26]. Gender of the contributor plays a part as well. Contributors identified as females decrease their chances of pull-request acceptance [3] [27]. Country of the contributor influences pull-request acceptance rate differently for different countries [3] [28]. It is important to note that if the contributor and integrator are from the same country, the chances of acceptance of pull-request increases [3] [28]. Programming languages seem to differ in their pull-request acceptance percentage [3] [29 – 31]. The popularity of a project as measured through watcher count [15], star count [15] and fork count [30] [32] influence the fate of pull-request [3] [32]. The complexity of a pull-request as inferred from the length of description is seen to negatively influence pull-request acceptance [3] [33]. The nature of the pull-request, for example, if a pull-request is a bug fix, increases the chance of pull-request acceptance [3] [16] [29]. Continuous Integration of a pull-request (its existence or not), latency, build count, all test passed, percentage

[1] https://github.com/PrithaSarkar/Dissertation

of tests passed/failed, first, and last build status are all seen to influence pull request decision making [3] [33 - 37].

In a real-life scenario, the decision depends solely on the integrator, and machine learning algorithms could merely predict the outcome but cannot guarantee the decision outcome.

# Chapter 3 Method

## 3.1 Data

Data used to carry out the research has been acquired from another research "On the Shoulders of Giants: A New Dataset for Pull-based Development Research" [3] after special permission was granted by authors of the paper for accession of the extended version. The data, aptly named **new_pullreq**, came in the form of a CSV (comma separated vales) file and had 3,347,937 pull requests and 120 metrics. This dataset is the largest dataset for pull-based development research [3].

In addition, the data [3] came with a set of features that influence merge decisions the most [4]. In this research, we only focus on the features present in [3]. The following sections provide a brief overview of the different classes and types of features.

### 3.1.1 Contributor Characteristics

An array of features has been grouped as contributor characteristics. These features are related to the contributors (pull-request submitters) and integrators. This also includes features regarding interactions between two contributors or a contributor and a project [3]. The features grouped as contributor characteristics are explained in brief in the section below.

| ***Feature*** | ***Description*** |
|---|---|
| *acc_commit_num* | The number of accepted commits of a contributor before the creation of a pull request [3] [16]. |
| *first_pr* | Whether it is the first pull request of a contributor [3] [18] [19]. |
| *core_member* | Whether the contributor is a core member or not [3] [21] [22] [24]. |
| *contrib_gender* | The gender of the contributor [3] [44]. |
| *same_country* | Whether contributor and integrator belong to the same affiliation [3] [28]. |
| *same_affiliation* | Whether the contributor and the integrator belong to the same affiliation [3] [20]. |
| *contrib_X/inte_X* | The Big Five personality traits of contributor/integrator (open: openness; cons: conscientious; extra: extraversion; agree: agreeableness; neur: neuroticism) [3] [45]. |
| *X_diff* | The absolute difference of Big Five personality traits between contributor and integrator [3] [46]. |
| *social_strength* | The fraction of team members that interacted |

| | |
|---|---|
| | with the contributor in the last three months [3] [33]. |
| *account_creation_days* | The time interval in days from the contributor's account creation to the pull-request creation [3] [17]. |
| *prior_review_num* | The number of prior reviews of an integrator [3] [20]. |
| *first_response_time* | The time interval in minutes from pull request creation to the first response by a reviewer [3] [33]. |
| *contrib_country/inte_country* | Country of residence of contributor/ integrator [3] [28]. |
| *prior_interaction* | Number of times that the contributor interacted with the project in the last three months [3] [32]. |
| *contrib_affiliation/inte_affiliation* | The affiliation that the contributor/ integrator belongs to [3] [20]. |
| *perc_contrib/inte_X* | The percentage of contributor/ integrator's emotion in comments (ne: negative, pos: positive, neu: neutral) [3] [46]. |
| *contrib_first_emo/inte_first_emo* | The emotion of the contributor/ integrator's first comment [3] [46]. |
| *contrib_follow_integrator* | Whether the contributor follows the integrator when submitting a pull request [3] [32]. |

*Table 3.1: Feature names and feature descriptions categorized as contributor characteristics.*

### 3.1.2 Project Characteristics

A plethora of features is categorized to be exclusively about the project characteristic. The following table gives an overview of the same.

| ***Features*** | ***Description*** |
|---|---|
| *language* | Programming language of the project [3] [29] [17] [19]. |
| *project_age* | Time interval in months from the project creation to the pull request creation [3] [32] [33]. |
| *pushed_delta* | The time interval in seconds between the opening time of the two latest pull requests [3] [47]. |
| *pr_succ_rate* | The acceptance rate of pull requests in the project [3] [47]. |
| *open_issue_num* | The number of opened issues when submitting the pull request [3] |

| | |
|---|---|
| | [47]. |
| *open_pr_num* | The number of opened pull requests when submitting the pull request [3] [20] [33]. |
| *fork_num* | Number of forks of project when submitting the pull request [3] [47]. |

*Table 3.2: Feature names and feature descriptions categorized as project characteristics.*

### 3.1.3 Pull-request Characteristics

A wide variety of features were decided to be related to the actual pull-request. These features range from continuous integration to tags and comments included during the pull-request.

| ***Features*** | ***Description*** |
|---|---|
| *churn_addition* | Number of added lines of code [3] [33]. |
| *bug_fix* | Whether pull-request fixes a bug [3] [16] [29]. |
| *test_inclusion* | Whether test code exists in a pull request [3] [24] [32] [33]. |
| *hash_tag/ at_tag* | Whether #/@ tag exists in comments or description [3] [48] [33]. |
| *part_num_X* | Number of participants in comment (issue: issue comment; pr: pull request comment; commit: commit comment) [3] [15]. |
| *ci_exists* | Whether a pull request uses continuous integration tools [3] [36]. |
| *ci_latency* | Time interval in minutes from pull request creation to the first build finish time of CI tools [3] [33]. |
| *ci_test_passed* | Whether passed all the CI builds [3] [34] [35]. |
| *ci_failed_perc* | Percentage of failed CI builds [3] [37]. |
| *churn_deletion* | Number of deleted lines of code [3] [33]. |
| *description_length* | Word count of pull request description [3] [33]. |
| *comment_conflict* | Whether the keyword 'conflict' exists in comments [3] [1]. |
| *pr_comment_num* | Number of pull request comments [3] [15]. |
| *part_num_code* | Number of participants in both pull request comment and commit comment [3] [15]. |
| *ci_build_num* | Number of CI builds [3] [37]. |

| | |
|---|---|
| *perc_neg/pos/neu_emotion* | Percentage of negative/positive/neutral emotion in comments [3] [46]. |
| *ci_first_build_status* | First build result of CI tool [3] [37]. |
| *ci_last_build_status* | Last build result of CI tool [3] [37]. |

*Table 3.3: Feature names and feature descriptions categorized as pull-request characteristics.*

## 3.2 Primary Analysis & Training

On trend with every other data intensive project, the data on hand had to be setup strategically for evaluation and finally, recommendation. The following sections and sub-sections dive deep into the methodical approach that was adapted for pre-processing and training the data in order to reach the ultimate goal of the project.

### 3.2.1 Data Preparation

The **new_pullreq** dataset has 3,347,937 rows and 120 columns of data. The dataset pertains of a feature named *"merged_or_not"*. This feature indicates if a pull-request was successfully integrated or not and is a binary feature- meaning it only had 1 or 0 values; 1 corresponds to merged successfully, whereas, 0 indicates an unsuccessful merge request. It was paramount at this stage, to understand the distribution of the two classes (merged or unmerged). A quick look through pointed out that among all the pull requests, 80.94% were successful merge requests and the remaining 19.05% were unsuccessful. This indicated a huge imbalance in class.

The decision to use half of the original dataset for initial training and testing was adopted and henceforth only 1,673,968 rows of pull-requests will be used and is named **train_test**. However, most of the features from the initial 120 features were stripped off from the data as the study is limited to the 49 features already recognized to be important [3]. From this point onwards, features explained in the "Data" section and its subsections will be paid importance. As an added measure, the class imbalance was checked once again and it was observed that the 80-20 split remains. Further ensuring that the dataset had an even distribution of merged and unmerged pull-requests.

It was important to check the number of missing values in the dataset nominated for testing. Upon exploration, it was evident that a fairly large number of fields were missing data. For instance, the feature that has the greatest number of missing data is *'bug_fixes'* with only 0.69% of data present. And, the feature that has the least number of missing data is *'contrib_gender'* with 79.28% of data present. A visual representation exhibiting the amount of data present respective to each feature has been provided in Appendix A.1.

It became apparent that preprocessing measures should be employed on the dataset in order to avoid inaccurate predictions and a biased model [38]. A few numbers of strategies exist that could be applied during this phase of the experiment, i.e., removal or deletion of missing values, and imputing missing values with mean/median/mode [38]. However, using any of those strategies

would not be able to produce fruitful results for the problem at hand. For example, since the absence of value does not exist across a single row and occurs for different columns or features, column-wise or row-wise deletion of pull-request would warrant a significant amount of data loss. Similarly, replacing the missing or NaN values with mean/median/mode would be non-sensical for features such as *'contrib_affiliation'*, *'contrib_country'*, *'contrib_gender'* etc. Therefore, to tackle the problem, the fields with missing values were replaced with the values that occur the most for their respective feature. Appendix A.1 and A.2 provide visual representations of the before and aftermath of the dataset.

Some of the features among the 49 features had string values. *'ci_first_build_status'* and *'ci_last_build_status'* were populated by values *'success'* and *'failure'*. The feature *'language'* was populated by values representative of different programming languages, i.e., R, Python, Java, etc. The field had 6 unique values. Features *'gender'* had 2 unique values *'female'* and *'male'*. *'contrib_first_emo'* feature dealt with the emotion of the contributor and had three unique values *'positive'*, *'negative',* and *'neutral'*. Finally, features *'contrib_country'* and *'contrib_affiliation'* had 175 and 1202 unique values, respectively, and were all string types as they were representative of the countries and affiliations the contributors hailed from. Since, the classification model to be used for the experiment was XGBoost, the string values present in the features had to be translated into integer or float values. This is because, internally, XGBoost models represent all problems as a regression predictive modeling problem that only takes numerical values as input [5]. Initially, annotation was done using sklearn's LabelEncoder[2] class. However, later on, in the experiment, it became apparent that in order to provide recommendations, knowledge of labels respective to their values is important. Therefore, annotations made through LabelEncoder were removed and manual annotation was performed. This allowed for harvesting the labels and value pairs as key-value pair in dictionary format.

### 3.2.2 eXtreme Gradient Boosting

Tree boosting is a highly effective and widely used machine learning method [6]. As described and demonstrated earlier, the data used to undertake this experiment had a significant number of fields with missing data. Even though the imputation phase was included as an added measure in the preprocessing stage, it incorporated some bias in it [7]. Thus, the data was considered to be sparse data. XGBoost's Sparsity-aware Split Finding algorithm shows its importance [7] in these cases since XGBoost can be relied upon to deal with it. Due to this reason, XGBoost was nominated to be the model to be used for the research.

The **train_test** is further divided and the first 60% of the rows are used for training purposes and was aptly named **train**, whereas, the remaining 40% is used for testing and was aptly named **test**.

Prior to the training and testing is performed, XGBoost has an array of hyperparameters that could be tuned to leverage the advantages of the XGBoost

[2] sklearn.preprocessing.LabelEncoder

algorithm. A dictionary containing various parameters and a list of values respective to each of the parameters were created[3]. This constituted about 100

different combinations. Upon tuning the hyperparameters on the train and test dataset, the best hyperparameters were found to be:

| ***Hyperparameter Name*** | ***Hyperparameter Value*** |
|---|---|
| colsample_bytree | 0.60111 |
| gamma | 1.23569 |
| max_depth | 12.0 |
| min_child_weight | 2.0 |
| reg_alpha | 40.0 |
| reg_lambda | 0.33834 |

*Table 3.4: Hyperparameters of XGBoost method and their values as calculated to provide the least loss for the experiment.*

The initial trail with fit was performed on all columns of the train dataset sans the *'id'* and '*merged_or_not'* columns, and the prediction was done on the same set of columns but from the rows of the test dataset. The experiment produced the following measures of accuracy metrics, confusion matrix, and ROC-AUC curve:

| Evaluation for: Baseline Model | | | | |
|---|---|---|---|---|
| | precision | recall | f1-score | support |
| Rejected | 0.695 | 0.163 | 0.264 | 126934 |
| Accepted | 0.834 | 0.983 | 0.902 | 542653 |
| accuracy | | | 0.828 | 669587 |
| macro avg | 0.764 | 0.573 | 0.583 | 669587 |
| weighted avg | 0.808 | 0.828 | 0.781 | 669587 |

*Table 3.5: Accuracy metrics of the baseline model*

[3] "A Guide on XGBoost hyperparameters tuning" Kaggle Notebook was used as inspiration.

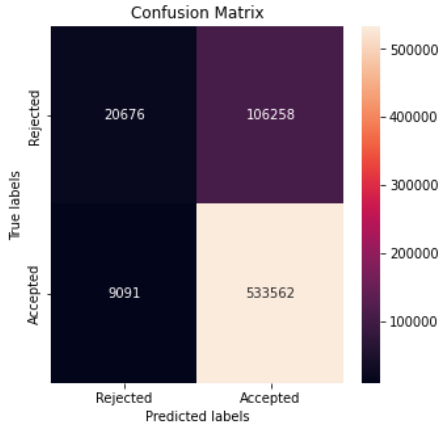



*Figure 3.1: Confusion Matrix from the initial train-test*

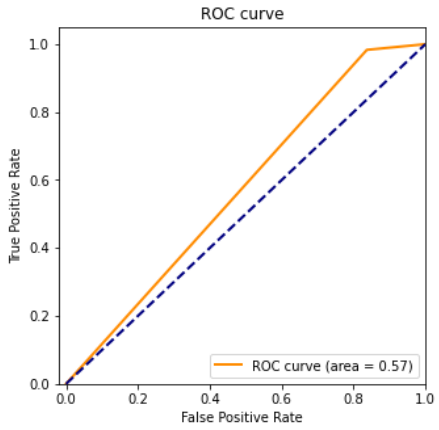



*Figure 3.2: ROC-AUC curve representing the True Positives and False Positives*

### 3.2.3 Feature Selection

Data development increases dimensions and computational costs are overcome by feature selection and extraction [9]. Feature selection involves nominating a subset of the original feature set [9]. There are several means of performing feature selection, broadly grouped into three main types: Filter, Wrapper and Embedded [9]. The filter method evaluates feature subsets with predefined criteria independent of any grouping [9] [10]. Wrapper methods of feature selection train models using a set of features. Features are then either added or removed from this set based on the inference [39]. Embedded techniques, on the other hand, combine the qualities of both filter and wrapper methods [39]. In this research, we mainly employ strategies for filtering out the features performing the best.

A benefit of using ensembles of decision tree methods like gradient boosting is that they can automatically provide estimates of feature importance from a trained predictive model [8]. XGBoost model class has functions to calculate feature importance on the modeling problem. One way, XGBoost does it is by using the f1-score that each feature produces. The bar chart demonstrating the importance of each attribute by their f1-score can be found in Appendix A.3. The scores are calculated explicitly for each attribute in the dataset, allowing attributes to be ranked and compared to each other [8]. Additionally, using the built-in *feature_importance_* function of XGBoost could be used for determining the importance of features. The default *importance_type* value provided by XGBoost is *"gain"*. "Gain" is the improvement in accuracy brought by a feature to the branches it is on [40]. "Gain" should not be confused with "Information Gain" as information gain makes use of entropy. Using *feature_importances_* resulted in the determination of the threshold (cut-off) value relatively easier. According to the *feature_importances_* function, gain from each feature in the dataset is noted in the subsequent table.

| *Feature Name* | *Gain* | *Feature Name* | *Gain* |
|---|---|---|---|
| 'ci_last_build_status' | 0.24167833 | 'core_member' | 0.23765129 |
| 'part_num_issue' | 0.16189131 | 'ci_build_num' | 0.054558802 |
| 'ci_first_build_status' | 0.041673012 | 'first_pr' | 0.041293345 |
| 'ci_exists' | 0.02635729 | 'part_num_commit' | 0.025682155 |
| 'acc_commit_num' | 0.019347258 | 'perc_inte_pos_emo' | 0.021371162 |
| 'contrib_follow_integrator' | 0.021180367 | 'contrib_first_emo' | 0.014700067 |
| 'same_country' | 0.0125083355 | 'language' | 0.010392028 |
| 'ci_test_passed' | 0.009497247 | 'open_pr_num' | 0.0071660695 |
| 'description_length' | 0.0070639974 | 'open_issue_num' | 0.0063185664 |
| 'account_creation_days' | 0.0049437135 | 'pr_succ_rate' | 0.0038450176 |
| 'first_response_time' | 0.0037446804 | 'fork_num' | 0.002792273 |
| 'ci_latency' | 0.0027003235 | 'contrib_gender' | 0.0025613788 |
| 'project_age' | 0.0018670487 | 'comment_conflict' | 0.0020959827 |
| 'churn_deletion' | 0.0016174928 | 'contrib_affiliation' | 0.00216667 |
| 'pushed_delta' | 0.0016308214 | 'perc_inte_neu_emo' | 0.0018514622 |
| 'part_num_pr' | 0.0016569793 | 'perc_external_contribs' | 0.0013071023 |
| 'part_num_code' | 0.0010782555 | 'perc_inte_neg_emo' | 0.0008452241 |
| 'churn_addition' | 0.00077492266 | 'social_strength' | 0.0008083276 |
| 'test_inclusion' | 0.00062936294 | 'contrib_country' | 0.00045985624 |
| 'bug_fix' | 0.00029252985 | 'at_tag' | 0.0 |
| 'hash_tag' | 0.0 | 'ci_failed_perc' | 0.0 |
| 'prior_interaction' | 0.0 | 'pr_comment_num' | 0.0 |
| 'same_affiliation' | 0.0 | | |

*Table 3.6: Gain of each feature according to XGBoost.*

The following subsections provide an extensive explanation of the strategies involved in determining the best and worst actors of the predictive modeling task.

#### 3.2.3.1 Strategy 1

The threshold(cutoff) value in this strategy is the mean of the gain determined by the *feature_importances_* function of XGBoost. Any feature that has the same or higher gain than the threshold value is considered to be important for prediction and the subset of features is called *imp_features_1*. Among 45 features, 8 features are determined to be in this subset.

```
imp_features_1 = ['ci_last_build_status', 'core_member', 'part_num_is
sue', 'ci_build_num', 'ci_first_build_status', 'first_pr', 'ci_exists
', 'part_num_commit']
```

As recognized earlier, pull-request from core member of a development team has a higher chance of acceptance as opposed to a stranger. This justifies the occurrence of *'core_member'* feature in *imp_features_1* justifies. Likewise, the existence and success of continuous integration, the practice of automating the integration of code changes from multiple contributors into a single software

project [43], in a pull-request is valuable information to be looked into. From the initial look, it could be said that this subset of features displays great potential to be the factor facilitating the merge decision.

Pull-request were sampled from the **train** dataset and included only the above-mentioned features. Fit and prediction from the newly sampled data produced the following accuracy measures.

| Evaluation for: Performance of new model with fewer features | | | | |
|---|---|---|---|---|
| | precision | recall | f1-score | support |
| Rejected | 0.582 | 0.059 | 0.107 | 126934 |
| Accepted | 0.818 | 0.990 | 0.896 | 542653 |
| accuracy | | | 0.814 | 669587 |
| macro avg | 0.700 | 0.525 | 0.502 | 669587 |
| weighted avg | 0.773 | 0.814 | 0.746 | 669587 |

*Table 3.7: Accuracy metrics of new model with features determined through the i mplementation of Strategy 1*

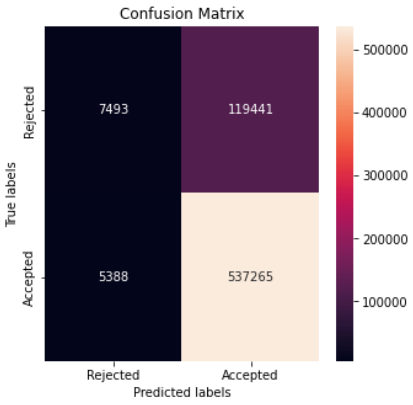



*Figure 3.3: Confusion Matrix for experiment with features from Strategy 1*

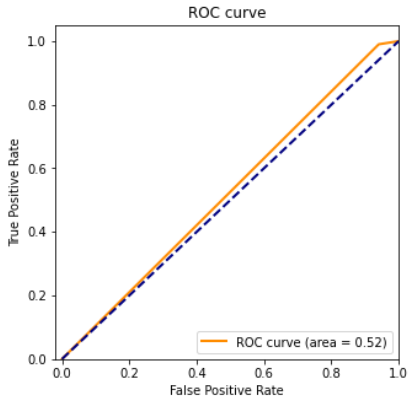



*Figure 3.4: ROC-AUC curve representing the True Positives and False Positives*

We can observe that there is a significant change in the ROC-AUC curve area but minimal change in the f-1 score. It could be said that these 8 features are the most representative of the data.

### 3.2.3.2 Strategy 2

The threshold value in this strategy is the standard deviation of gain values from the *feature_importances_* function [9]. Any feature that has the same higher gain value than the threshold is considered to be the most important feature driving the prediction of the model. The new subset of features is as follows:

```
imp_features_2 = ['ci_last_build_status', 'core_member', 'part_num_is
sue', 'ci_build_num']
```

*imp_features_1* and *imp_features_2* has 4 common attributes. It could be said that *imp_features_2* is a subset of *imp_features_1*.

When data is sampled from the **train** dataset using the features present in *imp_features_2* and fitted for prediction, the following accuracy measures are produced:

| Evaluation for: Performance of new model with fewer features | | | | |
|---|---|---|---|---|
| | precision | recall | f1-score | support |
| Rejected | 0.576 | 0.055 | 0.100 | 126934 |
| Accepted | 0.818 | 0.991 | 0.896 | 542653 |
| accuracy | | | 0.813 | 669587 |
| macro avg | 0.697 | 0.523 | 0.498 | 669587 |
| weighted avg | 0.772 | 0.813 | 0.745 | 669587 |

*Table 3.8: Accuracy metrics of new model with features determined through the implementation of Strategy 2*

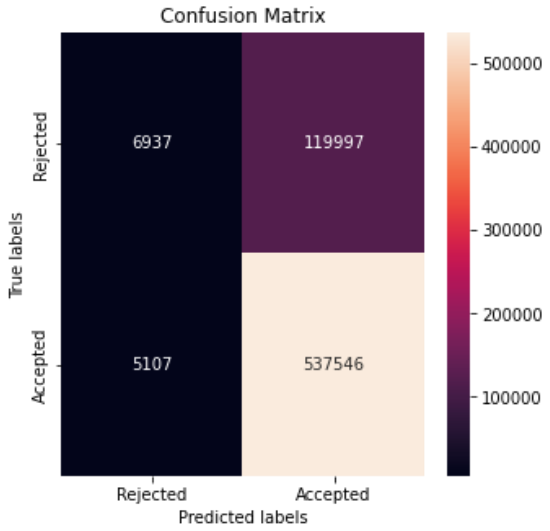



*Figure 3.5: Confusion Matrix for experiment with features from Strategy 2*

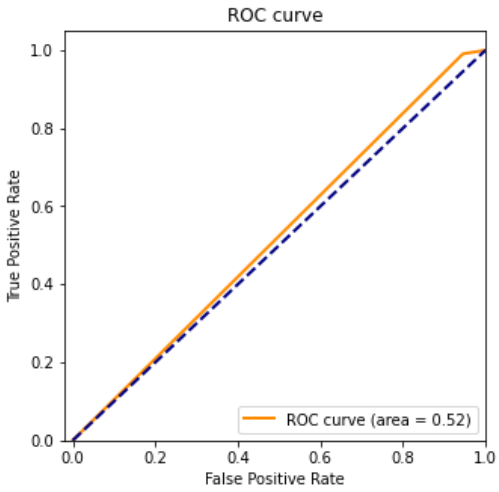



*Figure 3.6: ROC-AUC curve representing the True Positives and False Positives*

The f1-score changed but the ROC-AUC curve area remained the same as produced by features of Strategy 1, indicating how important the other 4 features from *imp_features_1* are. In this case, the 4 features, *'ci_exists'*, *'ci_build_num'*, '*part_num_commit*', and *'first_pr'* could be considered to be 'spurious features'. Whereas, *'ci_first_build_status'*, *'ci_last_build_status'*, *'core_member'*, and *'part_num_issue'* could be considered to be the core features that provide the most information to the model.

Machine learning models are susceptible to learning irrelevant patterns [11]. They rely on some spurious features that we humans know to avoid [11]. Spurious features provide seemingly unimportant details to models that turn out to be important in overall predictions. Thus, it could be said that even though the features present in *imp_features_2* are the top-performing features for the task of

prediction and has the highest gains, using them alone would not produce fruitful results.

### 3.2.3.3 Strategy 3

The derivative is primarily used when there is some varying quantity and the rate of change is not constant [12]. The derivative is used to measure the sensitivity of one variable with respect to another variable [12]. As seen earlier, the rate of change in gain for each feature was varying. This observation prompted the decision of using derivatives of gain acquired from *feature_importances_* function for feature extraction. In this strategy, the threshold value from Strategy 1 was also leveraged. The threshold value in this strategy was determined in two steps.

Step 1: Determine the derivative of gain by using gain as *dy* and mean of the gains (threshold value from Strategy 1) as *dx*. This resulted in a list of 45 values; each one a derivative of a feature.
Step 2: Determine threshold for Strategy 3 by calculating the mean of the derivatives acquired from Step 1.

However, since the threshold is a negative value `(-0.0013368133540180597)` all 45 features were selected for *imp_features_3*. Prediction using *imp_features_*3 resulted in the following accuracy measures.

| Evaluation for: Performance of new model with fewer features | | | | |
|---|---|---|---|---|
| | precision | recall | f1-score | support |
| Rejected | 0.693 | 0.182 | 0.288 | 126934 |
| Accepted | 0.837 | 0.981 | 0.903 | 542653 |
| accuracy | | | 0.830 | 669587 |
| macro avg | 0.765 | 0.581 | 0.596 | 669587 |
| weighted avg | 0.809 | 0.830 | 0.787 | 669587 |

*Table 3.9: Accuracy metrics of new model with features determined through implementation of Strategy 3*

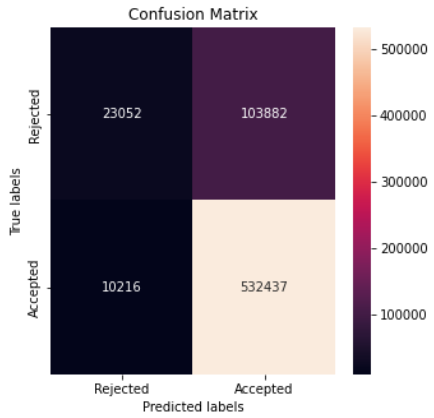



*Figure 3.7: Confusion Matrix for experiment with features from Strategy 3*

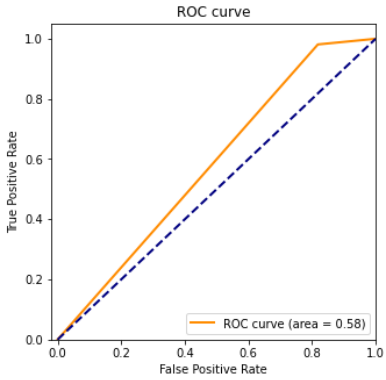



*Figure 3.8: ROC-AUC curve representing the True Positives and False Positives*

The area under ROC-AUC curve has a higher value than the baseline model which used all 45 features from the dataset. Hence, it is safe to say that even though there is the slightly higher impact from this model, this strategy is not fruitful for feature selection.

#### 3.2.3.4 Strategy 4

The fourth and final strategy uses information gain and entropy for determining the best performing features.

Information gain (IG) is an entropy-based selection method, which involves calculation from the output data grouped by feature A, denoted as gain (y, A) [9]. The Information Gain (y, A) is represented as,

$$gain(y, A) = entropy(y) - \sum_{C \in vals(A)} \frac{y_c}{y} entropy(yc) \qquad (Eq.\ 1)$$

The value (A) can be defined as the possible rates of attribute A when $y_c$ is the subset of y [9]. Moreover, the gain(y,A) can be defined as entropy of data segregation based on feature A removed from the total entropy of y [9].

The information gain from the features, using the above method is present in the following image extracted from Colab+ Notebook[4] :

| ***Feature*** | ***Information Gain*** | ***Feature*** | ***Information Gain*** |
|---|---|---|---|
| 'same_country' | 0.000016 | 'same_affiliation' | 0.000026 |
| 'bug_fix' | 0.000065 | 'test_inclusion' | 0.000069 |
| 'hash_tag' | 0.000151 | 'contrib_gender' | 0.000196 |
| 'comment_conflict' | 0.000211 | 'perc_inte_pos_emo' | 0.000506 |
| 'pr_comment_num' | 0.000519 | 'perc_inte_neg_emo' | 0.000571 |
| 'part_num_pr' | 0.000617 | 'part_num_code' | 0.000651 |
| 'contrib_first_emo' | 0.000892 | 'part_num_commit' | 0.001073 |
| 'ci_test_passed' | 0.001271 | 'contrib_follow_integrator' | 0.001331 |
| 'perc_inte_neu_emo' | 0.001357 | 'project_age' | 0.001925 |
| 'ci_first_build_status' | 0.002163 | 'language' | 0.002444 |
| 'ci_exists' | 0.002533 | 'contrib_country' | 0.002751 |
| 'at_tag' | 0.002809 | 'ci_build_num' | 0.004172 |
| 'ci_last_build_status' | 0.012212 | 'description_length' | 0.012738 |
| 'first_pr' | 0.013166 | 'ci_failed_perc' | 0.014183 |
| 'churn_deletion' | 0.015566 | 'contrib_affiliation' | 0.017104 |
| 'churn_addition' | 0.019403 | 'account_creation_days' | 0.019848 |
| 'open_issue_num' | 0.020882 | 'core_member' | 0.021656 |
| 'open_pr_num' | 0.023592 | 'part_num_issue' | 0.027620 |
| 'prior_interaction' | 0.039590 | 'ci_latency' | 0.040479 |
| 'fork_num' | 0.046445 | 'acc_commit_num' | 0.051246 |
| 'first_response_time' | 0.56442 | 'social_strength' | 0.057280 |
| 'pr_succ_rate' | 0.135751 | 'perc_external_contribs' | 0.233978 |
| 'pushed_delta' | 0.310288 | 'id' | 0.706550 |

*Table 3.10: Information Gain value of every feature.*

[4] Due to resource restriction, the calculation was performed on one-fourth of the original dataset.

The threshold was then determined by calculating the mean of the information gain values. Features that have the same or higher information gain value than the threshold is considered to be the important actors of the prediction procedure and are considered to be *imp_features_4*.

```
imp_features_4 = ['part_num_issue', 'prior_interaction',
'ci_latency', 'fork_num', 'acc_commit_num', 'first_response_time',
'social_strength', 'pr_succ_rate', 'perc_external_contribs',
'pushed_delta']
```

*imp_feature_4* contains 10 of the original 45 features. Data sampled from the **train** dataset using the features present in *imp_features_4* were used for prediction and the following accuracy measures were produced as a result.

| Evaluation for: Performance of new model with fewer features | | | | |
|---|---|---|---|---|
| | precision | recall | f1-score | support |
| Rejected | 0.622 | 0.164 | 0.259 | 126934 |
| Accepted | 0.833 | 0.977 | 0.899 | 542653 |
| accuracy | | | 0.823 | 669587 |
| macro avg | 0.727 | 0.570 | 0.579 | 669587 |
| weighted avg | 0.793 | 0.823 | 0.778 | 669587 |

*Table 3.11: Accuracy metrics of new model with features determined through implementation of Strategy 4*

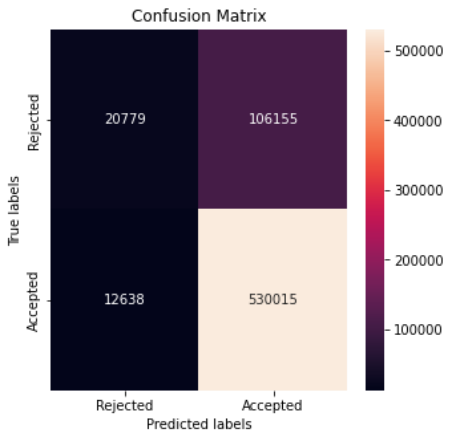



*Figure 3.9: Confusion Matrix for experiment with features from Strategy 4*

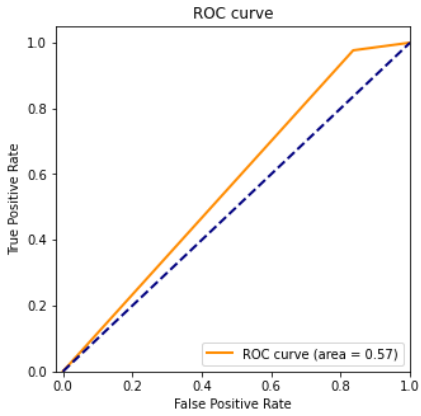



*Figure 3.10: ROC-AUC curve representing the True Positives and False Positives*

The f1-score and ROC-AUC area value remained the same as the baseline model. However, a significant difference between this model and model from Strategy 1 is that the features involved in the prediction are different, except for 1 feature. *imp_features_1* harvests a different set of features than *imp_features_4*. The only common feature between the two lists being *'part_num_issue'*. This provides an interesting insight into the data. It could be determined that a relatively

different set of features provide same, if not more, information. Subsequently, this set of feature could also be termed as core features.

## 3.3 Evaluation

Out of the 3,347,937 pull-requests present in the original dataset **new_pullreq**, 1,673,968 pull-requests were used in the initial train and test phase. The data helped determine the best model and features for prediction. The remaining pull-requests were reserved for the evaluation phase. The model and features acquired from the train and test phase would be applied on this remaining set of data.

From the feature selection section above, it is evident that Strategy 1 and Strategy 4 were the most feasible strategies for the selection of a subset of features. Thus, both strategies would be deployed on the evaluation dataset called **rec**. However, since this half of the original dataset remained unprocessed, the **rec** dataset needed to go through the stages of pre-processing similar to the training and testing dataset. To re-iterate, the pre-processing steps involved,

- Dropping the features not present in the tables from *Contributor Characteristics, Project Characteristics,* and *Pull-request Characteristics*.
- Replacing the NaN or null fields with the most common value present in a column. The before and aftermath of the process can be found in Appendix A.4 and A.5, respectively.
- Transforming the string type values into integer, float, or boolean type values. The dictionaries created from manual annotation during the pre-processing phase of **train_test** dataset was used for annotating the same values in the **rec** dataset.

### 3.3.1 Mean of Feature Importance

Recalling the *imp_feature_1* list from Strategy 1 and dropping features from the **rec** dataset that are not present in the *imp_feature_1* list would result in a dataset of 1,673,969 rows of pull-requests and 8 features.

The prediction of the hyperparameter tuned XGBoost model on this dataset produces the following metrics:

| Evaluation for: Model Performance on Evaluation Data | | | | |
|---|---|---|---|---|
| | precision | recall | f1-score | support |
| Rejected | 0.586 | 0.060 | 0.109 | 319295 |
| Accepted | 0.817 | 0.990 | 0.895 | 1354674 |
| accuracy | | | 0.813 | 1673969 |
| macro avg | 0.702 | 0.525 | 0.502 | 1673969 |
| weighted avg | 0.773 | 0.813 | 0.745 | 1673969 |

*Table 3.12: Accuracy metrics of on evaluation dataset from features determined in Strategy 1*

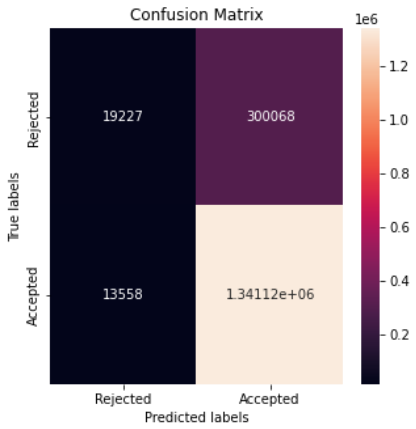



*Figure 3.11: Confusion Matrix for experiment with features from Strategy 1 on evaluation dataset*

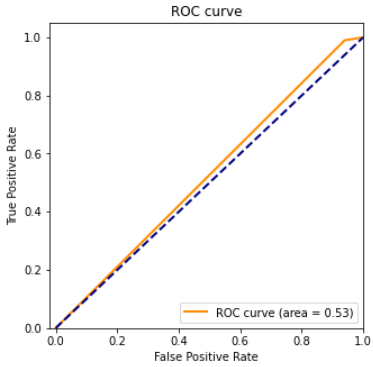



*Figure 3.12: ROC-AUC curve representing the True Positives and False Positives*

A further step was taken to determine the under-performing features among these 8 features through the use of SHAP (Shapley Additive exPlanations)[5] values. The *summary_plot* method was used to plot the directionality impact of each feature towards prediction.

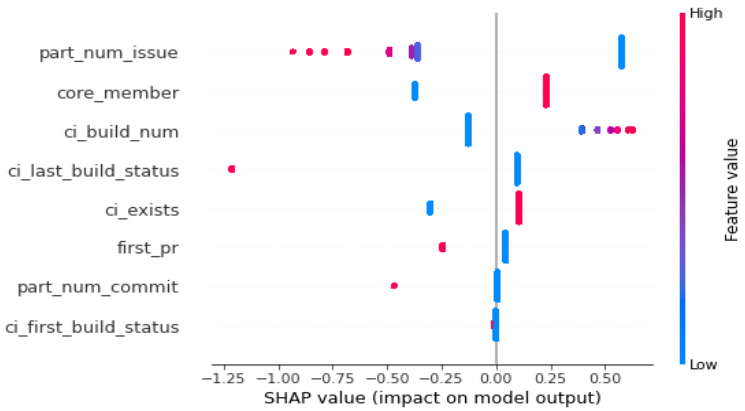



*Figure 3.13: Shapley value representing directionality impact of features from Strategy 1*

As evident, higher values of features *'part_num_issue'* leads to a pull-request being rejected. Even though, features *'ci_last_build_status'* , *'ci_first_build_status'*, *'first_pr'* and *'part_num_commit'* contribute towards rejection, they have comparatively less impact on the prediction.

### 3.3.2 Information Gain and Entropy

Recalling the *imp_feature_4* list from Strategy 4 and dropping features from the **rec** dataset that are not present in the *imp_feature_4* list would result in a dataset of 1,673,969 rows of pull-requests and 10 features.

[5] An introduction to explainable AI with Shapley Values

The prediction of the hyperparameter tuned XGBoost model on this dataset produces the following metrics:

| Evaluation for: Model Performance on Evaluation Data | | | | |
|---|---|---|---|---|
| | precision | recall | f1-score | support |
| Rejected | 0.638 | 0.141 | 0.230 | 319295 |
| Accepted | 0.829 | 0.981 | 0.899 | 1354674 |
| accuracy | | | 0.821 | 1673969 |
| macro avg | 0.733 | 0.561 | 0.564 | 1673969 |
| weighted avg | 0.792 | 0.821 | 0.771 | 1673969 |

*Table 3.13: Accuracy metrics of on evaluation dataset from features determined in Strategy 4*

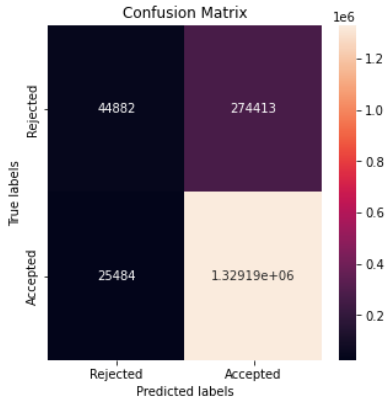



*Figure 3.14: Confusion Matrix for experiment with features from Strategy 4 on evaluation dataset*

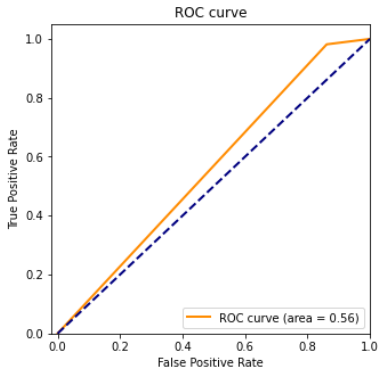



*Figure 3.15: ROC-AUC curve representing the True Positives and False Positives*

As evident, from the summary plot showcasing directionality impact, lower values of features *'pr_succ_rate'*, *'pushed_delta'* result in rejection of pull-requests. The same is true for higher values of features *'first_response_time'*, *'perc_external_contribs'*, *'acc_commit_num'* and *'fork_num'*. Even though, features like *'prior_interaction'* and *'ci_latency'* act the same, their contribution is comparatively less significant.

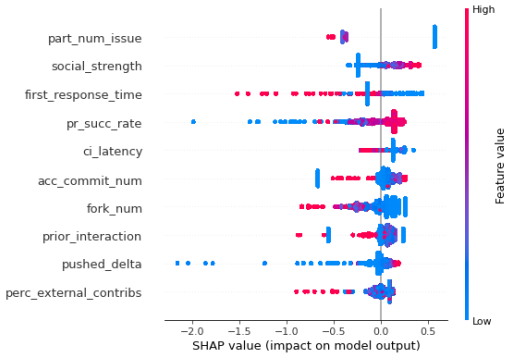



*Figure 3.16: Shapley value representing directionality impact of features from Strategy 4*

## 3.4 Recommendation

The goal of this research has been to determine the bad actors of a particular pull-request on GitHub and provide recommendations for the poorly acting features in order to increase the chances of the acceptance of that particular pull-request. However, in the course of the experiment, a significant number of caveats were recognized that possessed an unforeseen threat to the task of recommendation. The following sub-section details the caveats.

### 3.4.1 Caveats of Recommendation

1) Initially, the goal was to deploy BEAM search, a heuristic search algorithm that explores a graph by expanding the most promising node in a limited set [13]. However, it became evident quite early in the recommendation process that BEAM search, in this case, has a higher rate of producing erroneous results. This conclusion was reached after considering a couple of factors.
    i. BEAM search decoder would require a dictionary to train on. To disambiguate the term "dictionary" for BEAM search decoder with the term "dictionary" used for manual annotation, the term "lexicon" will be used while referring to dictionaries with respect to BEAM search. The lexicon that could be provided to the BEAM search decoder in this experiment only consists of integer or float values as depicted in Appendix A.6.
    ii. The numerical values in a sequence provide little to no meaningful information.
    iii. Finally, some features have the value 0 or 0.00 in their rows. BEAM search decoder uses logarithmic functions in its implementation and log0 is mathematically undefined. An attempt was made to re-write the dictionary and start the labels from 1 while performing manual annotations instead of 0, the entire prospect of deploying the BEAM search decoder still had to deal with the sequence of values not providing meaningful information.

    No logical idea to eliminate this problem could be produced at this point.

2) The 45 features used to carry out the experiment are categorized into three groups- Contributor Characteristics, Project Characteristics, and Pull-request Characteristics. Recommendations for most of the features are sometimes not practical. For example:
    i. *first_pr*: It may be possible that the pull-request that is being recommended is the first pull-request from the contributor.
    ii. *core_member*: Core members of projects are more likely to have their pull-requests accepted but in open-source development, anybody could create a pull-request. However, not all individuals making the pull-request is a core member and thus the acceptance rate is diminished.
    iii. *same_country*: It is not possible for individuals creating the pull-request to relocate from one country to another to increase the chances of pull-request acceptance.
    iv. *same_affiliation*: It is not possible for individuals to be employed by affiliation to increase their chances of pull-request acceptance.
    v. *language*: It is not always possible to rewrite codes into a different language to increase the chances of pull-request acceptance.

vi. *account_creation_day*: Individuals cannot increase or decrease the number of days they have been present on the GitHub platform.

vii. *open_issue_num*: Individuals cannot change the number of issues opened impulsively. Alternatively, if contributors would like to increase the chances of their pull-request being accepted, they could decrease the number of issues opened at the time of the creation of pull-request. However, that too depends on a person and the importance of the issues opened. Hence, the recommendation of this feature becomes complicated.

viii. *project_age*: Individuals do not have any control over the age of a project. One possibility to eliminate this limitation remains with the contributors looking up fairly new projects to contribute towards. However, that possibility might not provide fruitful results as the nature and language of the project and the expertise of the contributor are also at play.

The above-mentioned features provide a mere glimpse into the hardships of the recommendation tasks. These issues only allowed for the recommendations to be done of features categorized as Pull Request Characteristics.

### 3.4.2 Recommendations

Pull-request characteristics that could be used for recommendation were noted in a list named *pull_request_characteristics*. Since two sets of features were used for evaluation of the **rec** dataset, attempts of recommendations were done based on those two sets of features.

The 'recommendation' was done based on the most commonly occurring value in a feature to be recommended from the pull-requests that were accepted versus the most commonly occurring value in a feature to be recommended from the pull-requests that were rejected.

#### 3.4.2.1 Recommendation with Strategy 1

Strategy 1 produced a subset of features called *imp_features_1*. The list in question, when compared with the list *pull_request_characteristics*, results in common features present in both of them. The common features, in this case, were *'ci_last_build_status'*, *'ci_first_build_status'*, *'ci_build_num'*, and *'ci_exists'*. All 4 features that could be recommended were related to the task of continuous integration. The observed difference between accepted pull-requests and rejected pull-requests for these 4 features is as follows:

| ***Features*** | ***Most frequent value observed in merged pull requests*** | ***Most frequent value observed in unmerged pull requests*** |
|---|---|---|
| ci_first_build_status | 1 | 2 |
| ci_last_build_status | 1 | 2 |
| ci_exists | 1.0 | 1.0 |

| *Features* | *Most frequent value observed in merged pull requests* | *Most frequent value observed in unmerged pull requests* |
|---|---|---|
| ci_build_num | 1.0 | 1.0 |

*Table 3.15: Values frequently found in merged pull-requests and unmerged pull-requests for recommendation features.*

A quick look into the dictionaries *'dict_ci_last_build_status'*, *'dict_ci_exists'*, *'dict_ci_build_num'* and *'dict_ci_first_build_status'* created earlier during the experiment, shows that Label 1 relates to *'success'* and Label 2 relates to *'failure'*. Thus, it could be said that successful continuous integration for both the first build and the last build is important for pull-request acceptance.

### 3.4.2.2 Recommendation with Strategy 4

Strategy 4 produced a subset of features called *imp_features_4*. The list in question, when compared with the list *pull_request_characteristics*, results in common features present in both of them. The common feature, in this case, was *'ci_latency'*. The feature that could be recommended were related to the time needed for continuous integration. The observed difference between accepted pull-requests and rejected pull-requests for *'ci_latency'* is as follows:

| *Features* | *Most frequent value observed in merged pull requests* | *Most frequent value observed in unmerged pull requests* |
|---|---|---|
| ci_latency | 120.0 | 125.0 |

*Table 3.16: Values frequently found in merged pull-requests and unmerged pull-requests for recommendation features.*

Suggesting that latency in continuous integration should be decreased (in minutes).

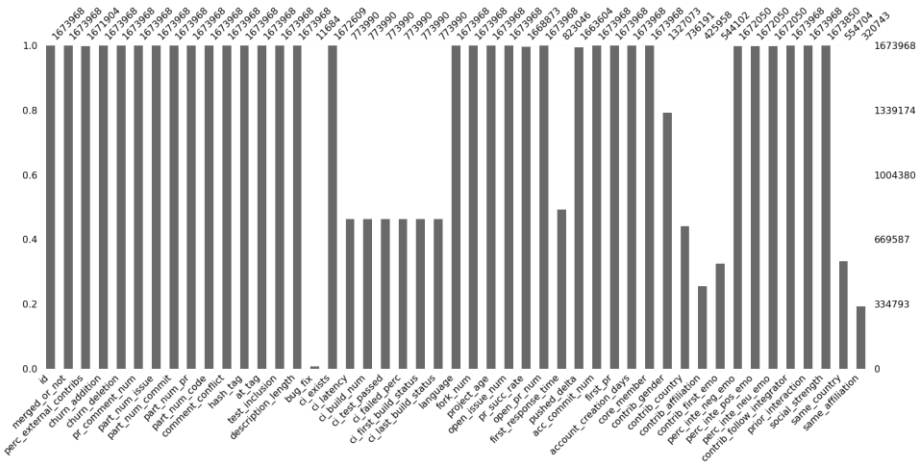
# Chapter 4 Conclusion

The goal of the experiment was to determine the bad factors in a pull-request and provide recommendations to overcome the limitations of a bad pull-request to enhance the chances of merging. An extensive analysis of the dataset at hand was done in order to achieve these goals. From devising 4 different types of strategies to understanding how they contribute to the prediction of the nominated eXtreme Gradient Boosting model to critical commentary regarding the task of recommendation was provided.

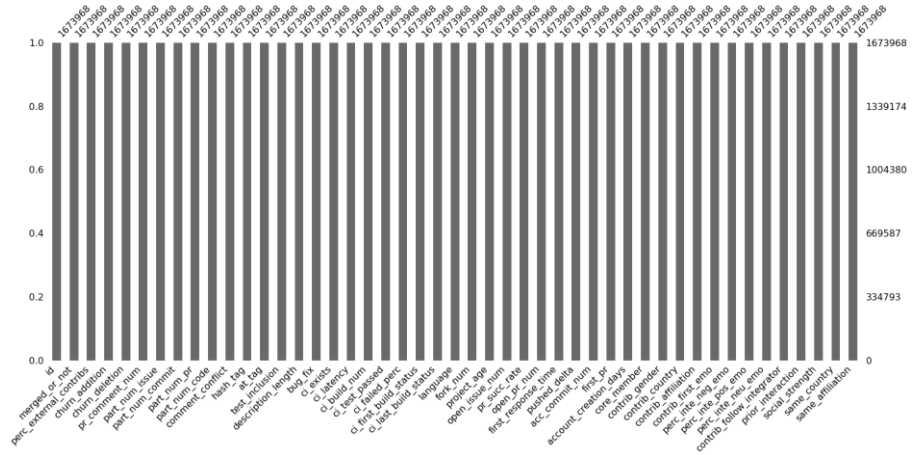
The exploratory research not only managed to find the bad factors related to a pull-request as expected for RQ1, but it also provided a deeper understanding of each of the features involved, i.e., whether they facilitate or deteriorate the fate of pull-request acceptance. The experiment discovered limitations during the implementation of the recommendation phase. Although the limitations were not eliminated and recommendations were provided as anticipated in RQ2.

The model provided some relatively good perceptions of the data and provide opportunities for further research. One of the opportunities is, synthesizing logical ways of overcoming the caveats discussed in the recommendation section. Another one is- the creation of a recommender system that would be able to recommend feasible options for transforming pull-requests and increasing their merge possibilities. Finally, the new and transformed pull-requests could be put through the model, devised in this experiment, to check if it is predicted to be accepted. It is important to note in this case that the actual decision lies with the integrators in real-life scenarios and the model could only provide checking facilities but will not guarantee acceptance of a PR. Moreover, since predictive models are based on statistical methodologies and determine the outcome based on likelihood estimation, it is important to be mindful of the false-positives and true-negatives values the model would produce.

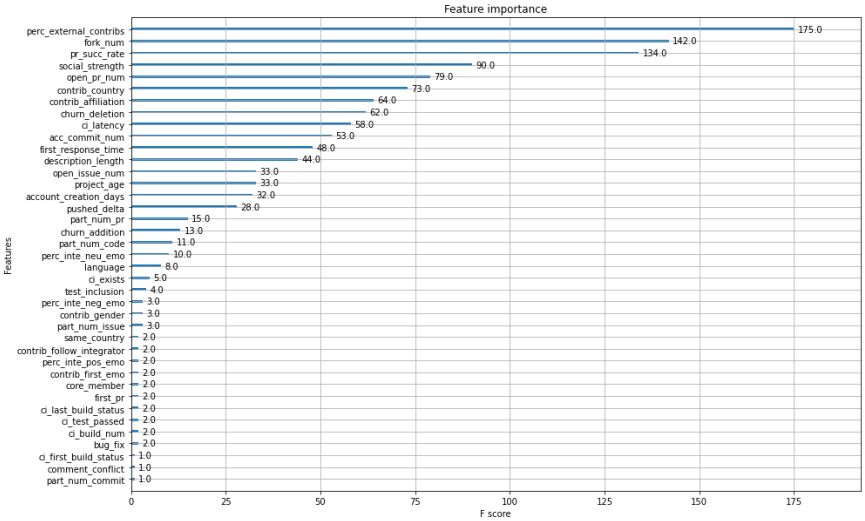
# Appendix A



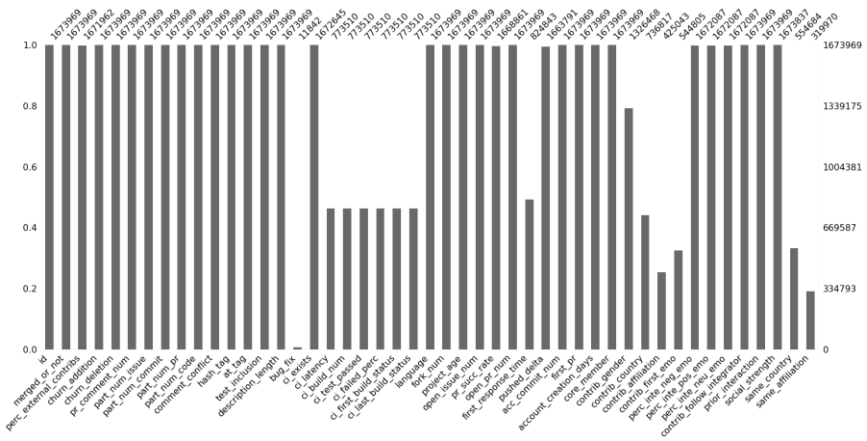

Appendix A.1: Missing values from **train_test** dataset with 1,673,968 pull requests with 45 features.



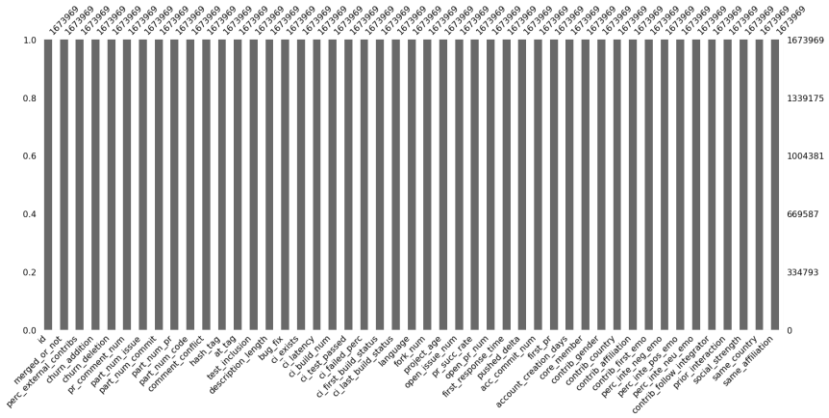

Appendix A.2: **train_test** dataset with 1,673,968 pull- requests with 45 features after pre-processing.




Appendix A.3: Feature importance as calculated by the XGBoost model.



Appendix A.4: Missing values from the **rec** dataset with 45 features.



Appendix A.5: **rec** dataset with 45 features after pre-processing.

| | part_num_issue | prior_interaction | ci_latency | fork_num | acc_commit_num | first_response_time | social_strength | pr_succ_rate | perc_external_contribs | pushed_delta |
|---|---|---|---|---|---|---|---|---|---|---|
| **1673968** | 3 | 193 | 125.0 | 132 | 797 | 135.0 | 0.333333 | 0.970787 | 0.010695 | 304038.0 |
| **1673969** | 0 | 128 | 2336.0 | 205 | 843 | 0.0 | 0.210526 | 0.984974 | 0.190860 | 8146.0 |
| **1673970** | 0 | 41 | 125.0 | 1 | 69 | 0.0 | 0.222222 | 0.958042 | 0.083893 | 1206410.0 |

Appendix A.6: 3 rows representing a pull-request with multiple manually annotated features

# References


[1] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In Proceedings of the 36th International Conference on Software Engineering (ICSE 2014). Association for Computing Machinery, New York, NY, USA, 345–355. https://doi.org/10.1145/2568225.2568260

[2] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. 2016. Work practices and challenges in pull-based development: the contributor's perspective. In Proceedings of the 38th International Conference on Software Engineering (ICSE '16). Association for Computing Machinery, New York, NY, USA, 285–296. https://doi.org/10.1145/2884781.2884826

[3] Xunhui Zhang, Ayushi Rastogi, and Yue Yu. 2020. On the Shoulders of Giants: A New Dataset for Pull-based Development Research. Proceedings of the 17th International Conference on Mining Software Repositories. Association for Computing Machinery, New York, NY, USA, 543–547. https://doi.org/10.1145/3379597.3387489

[4] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In Proceedings of the 36th International Conference on Software Engineering (ICSE 2014). Association for Computing Machinery, New York, NY, USA, 345–355. https://doi.org/10.1145/2568225.2568260

[5] Machine Learning Mastery. 2022. Data Preparation for Gradient Boosting with XGBoost in Python by Jason Bownlee. Retrieved July 10th, 2022 from https://machinelearningmastery.com/data-preparation-gradient-boosting-xgboost-python/

[6] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). Association for Computing Machinery, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[7] Medium. 2022. How XGBoost Handles Sparsities Arising From of Missing Data? (With an Example) by Cansu Ergün. Retrieved July 11th, 2022 from https://medium.com/hypatai/how-xgboost-handles-sparsities-arising-from-of-missing-data-with-an-example-90ce8e4ba9ca

[8] Notebook Community. 2022. Feature Importance and Feature Selection With XGBoost. Retrieved July 12th, 2022 from https://notebook.community/minesh1291/MachineLearning/xgboost/feature_importance_v1

[9] Prasetiyowati, M.I., Maulidevi, N.U. & Surendro, K. Determining threshold value on information gain feature selection to increase speed and prediction accuracy of random forest. *J Big Data* 8, 84 (2021). https://doi.org/10.1186/s40537-021-00472-4

[10] Ping Zhang, Wanfu Gao, Feature selection considering Uncertainty Change Ratio of the class label, Applied Soft Computing, Volume 95, 2020, 106537, ISSN 1568-4946, https://doi.org/10.1016/j.asoc.2020.106537.

[11] The Stanford AI Blog. 2022. Removing Spurious Features Can Hurt Accuracy and Affect Groups Disproportionately by Fereshte Khani. Retrieved on July 13th, 2022 from https://ai.stanford.edu/blog/removing-spuriousfeature/

[12] Byju's. 2022. Derivatives Meaning | First and Second order Derivatives. Retrieved on July 14th, 2022 from https://byjus.com/maths/derivatives/#:~:text=The%20derivative%20is%20primarily%20used%20when%20there%20is,variable%29%20with%20respect%20to%20another%20variable%20%28independent%20variable%29.

[13] Wikipedia, The Free Encyclopedia. 2022. Beam Search. Retrieved on July 25th, 2022 from https://en.wikipedia.org/wiki/Beam_search

[14] Ye, Y., & Kishida, K. (2003). Toward an understanding of the motivation of open source software developers. *25th International Conference on Software Engineering, 2003. Proceedings.*, 419-429.

[15] Georgios Gousios and Andy Zaidman. 2014. A dataset for pull-based development research. In Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014). Association for Computing Machinery, New York, NY, USA, 368–371. https://doi.org/10.1145/2597073.2597122

[16] Y. Jiang, B. Adams and D. M. German, "Will my patch make it? And how fast? Case study on the Linux kernel," 2013 10th Working Conference on Mining Software Repositories (MSR), 2013, pp. 101-110, doi: 10.1109/MSR.2013.6624016.

[17] Mohammad Masudur Rahman and Chanchal K. Roy. 2014. An insight into the pull requests of GitHub. In Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014). Association for Computing Machinery, New York, NY, USA, 364–367. https://doi.org/10.1145/2597073.2597121

[18] D. M. Soares, M. L. De Lima Junior, L. Murta and A. Plastino, "Rejection Factors of Pull Requests Filed by Core Team Developers in Software Projects with High Acceptance Rates," 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), 2015, pp. 960-965, doi: 10.1109/ICMLA.2015.41.

[19] Daricélio Moreira Soares, Manoel Limeira de Lima Júnior, Leonardo Murta, and Alexandre Plastino. 2015. Acceptance factors of pull requests in open-source projects. In Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC '15). Association for Computing Machinery, New York, NY, USA, 1541–1546. https://doi.org/10.1145/2695664.2695856

[20] O. Baysal, O. Kononenko, R. Holmes and M. W. Godfrey, "The influence of non-technical factors on code review," 2013 20th Working Conference on Reverse Engineering (WCRE), 2013, pp. 122-131, doi: 10.1109/WCRE.2013.6671287.

[21] O. Baysal, O. Kononenko, R. Holmes and M. W. Godfrey, "The Secret Life of Patches: A Firefox Case Study," 2012 19th Working Conference on Reverse Engineering, 2012, pp. 447-455, doi: 10.1109/WCRE.2012.54.

[22] Amiangshu Bosu and Jeffrey C. Carver. 2014. Impact of developer reputation on code review outcomes in OSS projects: an empirical investigation. In Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '14). Association for Computing Machinery, New York, NY, USA, Article 33, 1–10. https://doi.org/10.1145/2652524.2652544

[23] Amiangshu Bosu and Jeffrey C. Carver. 2014. Impact of developer reputation on code review outcomes in OSS projects: an empirical investigation. In Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '14). Association for Computing Machinery, New York, NY, USA, Article 33, 1–10. https://doi.org/10.1145/2652524.2652544

[24] Gustavo Pinto, Luiz Felipe Dias, and Igor Steinmacher. 2018. Who gets a patch accepted first? comparing the contributions of employees and volunteers. In Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '18). Association for Computing Machinery, New York, NY, USA, 110–113. https://doi.org/10.1145/3195836.3195858

[25] Daricélio Moreira Soares, Manoel Limeira de Lima Júnior, Leonardo Murta, and Alexandre Plastino. 2015. Acceptance factors of pull requests in open-source projects. In Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC '15). Association for Computing Machinery, New York, NY, USA, 1541–1546. https://doi.org/10.1145/2695664.2695856

[26] Daricélio Moreira Soares, Manoel Limeira de Lima Júnior, Leonardo Murta, and Alexandre Plastino. 2015. Acceptance factors of pull requests in open-source projects. In Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC '15). Association for Computing Machinery, New York, NY, USA, 1541–1546. https://doi.org/10.1145/2695664.2695856

[27] Terrell, Josh, Andrew Kofink, Justin Middleton, Clarissa Rainear, Emerson R. Murphy-Hill and Chris Parnin. "Gender bias in open source: Pull request acceptance of women versus men." *PeerJ Prepr.* 4 (2016): e1733.

[28] Ayushi Rastogi, Nachiappan Nagappan, Georgios Gousios, and André van der Hoek. 2018. Relationship between geographical location and evaluation of developer contributions in github. In Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '18). Association for Computing Machinery, New York, NY, USA, Article 22, 1–8. https://doi.org/10.1145/3239235.3240504

[29] Rohan Padhye, Senthil Mani, and Vibha Singhal Sinha. 2014. A study of external community contribution to open-source projects on GitHub. In Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014). Association for Computing Machinery, New York, NY, USA, 332–335. https://doi.org/10.1145/2597073.2597113

[30] Mohammad Masudur Rahman and Chanchal K. Roy. 2014. An insight into the pull requests of GitHub. In Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014). Association for Computing Machinery, New York, NY, USA, 364–367. https://doi.org/10.1145/2597073.2597121

[31] Daricélio Moreira Soares, Manoel Limeira de Lima Júnior, Leonardo Murta, and Alexandre Plastino. 2015. Acceptance factors of pull requests in open-source projects. In Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC '15). Association for Computing Machinery, New York, NY, USA, 1541–1546. https://doi.org/10.1145/2695664.2695856

[32] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of social and technical factors for evaluating contribution in GitHub. In Proceedings of the 36th International Conference on Software Engineering (ICSE 2014). Association for Computing Machinery, New York, NY, USA, 356–366. https://doi.org/10.1145/2568225.2568315

[33] Yu, Y., Yin, G., Wang, T. *et al.* Determinants of pull-based development in the context of continuous integration. *Sci. China Inf. Sci.* **59**, 080104 (2016). https://doi.org/10.1007/s11432-016-5595-8

[34] G. Gousios, A. Zaidman, M. -A. Storey and A. v. Deursen, "Work Practices and Challenges in Pull-Based Development: The Integrator's Perspective," 2015 IEEE/ACM

37th IEEE International Conference on Software Engineering, 2015, pp. 358-368, doi: 10.1109/ICSE.2015.55.

[35] Y. Tao, D. Han and S. Kim, "Writing Acceptable Patches: An Empirical Study of Open Source Project Patches," 2014 IEEE International Conference on Software Maintenance and Evolution, 2014, pp. 271-280, doi: 10.1109/ICSME.2014.49.

[36] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. 2015. Quality and productivity outcomes relating to continuous integration in GitHub. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015). Association for Computing Machinery, New York, NY, USA, 805–816. https://doi.org/10.1145/2786805.2786850

[37] F. Zampetti, G. Bavota, G. Canfora and M. D. Penta, "A Study on the Interplay between Pull Request Review and Continuous Integration Builds," 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2019, pp. 38-48, doi: 10.1109/SANER.2019.8667996.

[38] Medium. 2022. How to Handle Missing Data in Data Preprocessing by Pallavi Srivastava. Retrieved on 24th August from https://python.plainenglish.io/handling-missing-data-b92044bc2066

[39] Analytics Vidya. 2022. Introduction to Feature Selection methods with an example (or how to select the right variables?) by Sauravkaushik8 Kaushik. Retrieved on 26th august, 2022 from https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variables/

[40] Towards Data Science. 2022. The Multiple faces of “Feature importance” in XGBoost by Amjad Abu-Rmileh. Retrieved on 26th August, 2022 from https://towardsdatascience.com/be-careful-when-interpreting-your-features-importance-in-xgboost-6e16132588e7

[41] Wikipedia, The Free Encyclopedia. 2022. GitHub. Retrieved on 27th August, 2022 from https://en.wikipedia.org/wiki/GitHub#:~:text=As%20of%20January%202020%2C%20GitHub%20reports%20having%20over,largest%20host%20of%20source%20code%20in%20the%20world.

[42] Georgios Gousios. 2013. The GHTorent dataset and tool suite. In Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13). IEEE Press, 233–236. doi: doi/10.5555/2487085.2487132

[43] Atlassian. 2022. Software Development. Retrieved on 27th August from https://www.atlassian.com/continuous-delivery/continuous-integration#:~:text=Continuous%20integration%20(CI)%20is%20the,builds%20and%20tests%20then%20run.

[44] Josh Terrell, Andrew Kofink, Justin Middleton, Clarissa Rainear, Emerson Murphy Hill, Chris Parnin, and Jon Stallings. 2017. Gender differences and bias in open source: Pull request acceptance of women versus men. PeerJ Computer Science 3 (2017), e111.

[45] R. N. Iyer, S. A. Yun, M. Nagappan and J. Hoey, "Effects of Personality Traits on Pull Request Acceptance," in IEEE Transactions on Software Engineering, vol. 47, no. 11, pp. 2632-2643, 1 Nov. 2021, doi: 10.1109/TSE.2019.2960357.

[46] Iyer, Rahul. 2019. Effects of Personality Traits and Emotional Factors in Pull Request Acceptance. http://hdl.handle.net/10012/14952

[47] Nikhil Khadke, Ming Han Teh, and Minghan Shen. [n.d.]. Predicting Acceptance of GitHub Pull Requests. ([n. d.]).

[48] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. 2017. A preliminary analysis on the effects of propensity to trust in distributed software development. In 2017 IEEE 12th international conference on global software engineering (ICGSE). IEEE, 56–60