<Title of project placed here>

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 times pull-requests from contributors go unmerged. The undertaken study attempts to determine if a contributors pull-request has the potential to get merged or not; if unmerged, the bad actors that triggered the decision. Furthermore, an attempt has been made to put forth some possible changes a contributor can make if their pull-request in fact go unmerged.

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 advices 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 1

Chapter 2 Related Work 2

Chapter 3 Method 3

3.1 Data 3
3.1.1 Contributor Characteristics 3
3.1.2 Project Characteristics 4
3.1.3 Pull-request Characteristics 5

3.2 Primary Analysis & Training 5
3.2.1 Data Preparation 6
3.2.2 eXtreme Gradient Boosting 7
3.2.3 Feature Selection 8
3.2.3.1 Strategy 1 9
3.2.3.2 Strategy 2 10
3.2.3.3 Strategy 3 11
3.2.3.4 Strategy 4 12

3.3 Evaluation 14
3.3.1 Mean of Feature Importance 15
3.3.2 Information Gain and Entropy 16

3.4 Recommendation 17
3.4.1 Caveats of Recommendation 17
3.4.2 Recommendation 18
3.4.2.1 Recommendation with Strategy 1 18
3.4.2.2 Recommendation with Strategy 4 19

Chapter 4 Conclusion 20

Appendix A 1

References 3

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 the previous years [2].

Pull-based model is quite different from the classic way of contributing. In pull-based development, contributors are able to fork or locally duplicate the main repository of a project they want to contribute to, make their changes independently, then create a pull request (PR) to ask that their changes be merged into the main repository [2]. This is vastly different from the traditional way of contributing which included sending change sets to development mailing list to issue tracking systems or through direct access to the version control system [2].

Websites such as GitHub, GitLab, 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 works is:

RQ1: What factors drive the integrators' decision to accept a pull-request?

RQ2: How can the factors contributing to rejection of pull-request be changed to increase the changes of 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]. Newcomers to the OSS community, thus, face unforeseen challenges [2]. For example,

- 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 [15 - 20].
- It is more likely that an individuals pull-request is accepted if they are part of the core team. [21 - 26]
- Gender of the contributor plays a part as well. Contributors identified as females decrease their chances of pull-request acceptance.
- Country of the contributor influences pull-request acceptance rate differently for different countries. It is important to note that if the contributor and integrator are from the same country, the chances of acceptance of pull-request increases.
- Programming languages seem to differ in their pull-request acceptance percentage.
- Popularity of a project as measured through watcher count, star count and fork count influence the fate of pull-request.
- Complexity of a pull-request as inferred from the length of description is seen to negatively influence pull-request acceptance.
- The nature of the pull-request, for example, if a pull-request is a bug fix, increases the chance of pull-request acceptance.
- Continuous Integration of a pull-request (its existence or not), latency, build count, all test passed, percentage of tests passed/failed, first and last build status are all seen to influence pull request decision making.

As recognized from multiple exploratory researches, an array of features that contribute negatively towards the acceptance of pull-request exist already. In a real-life setting, 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 decision 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 developers (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].
first_pr	Whether it is the first pull request of a contributor [3].
core_member	Whether the contributor is a core member or not [3].
contrib_gender	The gender of the contributor [3].
same_country	Whether contributor and integrator belong to the same affiliation [3].
same_affiliation	Whether the contributor and the integrator belong to the same affiliation [3].
contrib_X/inte_X	The Big Five personality traits of contributor/integrator (open: openness; cons: conscientious; extra: extraversion; agree: agreeableness; neur: neuroticism) [3].
X_diff	The absolute difference of Big Five personality traits between contributor and integrator [3].
social_strength	The fraction of team members that interacted with the contributor in the last three months [3].

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

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

3.1.2 Project Characteristics

A plethora of features are 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].
project_age	Time interval in months from the project creation to the pull request creation [3].
pushed_delta	The time interval in seconds between the opening time of the two latest pull requests [3].
pr_succ_rate	Acceptance rate of pull-requests in the project [3].
open_issue_num	Number of opened issues when submitting the pull request [3].
open_pr_num	Number of opened pull requests when submitting the pull request [3].
fork_num	Number of forks of project when submitting the pull request [3].

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].
bug_fix	Whether pull-request fixes a bug [3].
test_inclusion	Whether test code exists in a pull request [3].
hash_tag/ at_tag	Whether #/@ tag exists in comments or description [3].
part_num_X	Number of participants in comment (issue: issue comment; pr: pull request comment; commit: commit comment) [3].
ci_exists	Whether a pull request uses continuous integration tools [3].
ci_latency	Time interval in minutes from pull request creation to the first build finish time of CI tools [3].
ci_test_passed	Whether passed all the CI builds [3].
ci_failed_perc	Percentage of failed CI builds [3].
churn_deletion	Number of deleted lines of code [3].
description_length	Word count of pull request description [3].
comment_conflict	Whether the keyword 'conflict' exists in comments [3].
pr_comment_num	Number of pull request comments [3].
part_num_code	Number of participants in both pull request comment and commit comment [3].
ci_build_num	Number of CI builds [3].
perc_neg/pos/neu_emotion	Percentage of negative/positive/neutral emotion in comments [3].
ci_first_build_status	First build result of CI tool [3].
ci_last_build_status	Last build result of CI tool [3].

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. Upon converting the csv file into a pandas dataframe, it was observed that the *"merged_or_not"* feature indicated if a pull-request was successfully integrated or not. This particular feature was a binary feature- meaning it only had 1 or 0 values; 1 being merged successfully and 0 being 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**. Moreover, most of the features from the initial 120 features were stripped off from the data. From this point onwards, only 49 features [3] will be considered, i.e., features explained in the "Data" section and its subsections will be paid importance to. As an added measure, the class imbalance was checked upon once again and it was observed that the 80-20 split remains. Further ensuring that the dataset had 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 a little exploration, it was evident that a fairly large number of fields were missing data. In order to tackle the problem, it was decided that the fields with missing values would be replaced with the values that occur the most for their respective feature. Appendix A.1 and A.2 provides a visual representation 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 from 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 type 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[1] class. However, later on in the experiment, it became apparent that in order to provide recommendations, knowledge of labels respective of their values is important. Therefore, annotations made through LabelEncoder were removed and manual annotation was performed. This

[1] sklearn.preprocessing.LabelEncoder

allowed for harvesting the labels and value pair 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 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 in these cases since XGBoost can be relied upon to deal with it [7]. 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 is used for training purpose and was aptly named **train**, whereas, the remaining 40% is used for testing and was aptly names **test**.

Prior to the training and testing being performed, XGBoost has an array of hyperparameters that could be tuned to leverage the advantages of the XGBoost algorithm. A dictionary containing various parameters and list of values respective of each of the parameters were created. This constituted to about 100 different combinations. Upon tuning the hyperparameters on the train and test dataset, the best hyperparameters were found to be:

- *colsample_bytree:* 0.50604
- *gamma:* 1.97312
- *max_depth:* 13.0
- *min_child_weight:* 1.0
- *reg_alpha:* 40.0
- *reg_lambda:* 0.5644

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
```

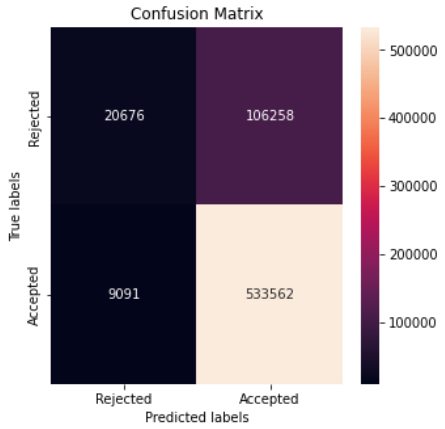


Figure 3.1: Confusion Matrix from the initial train-test

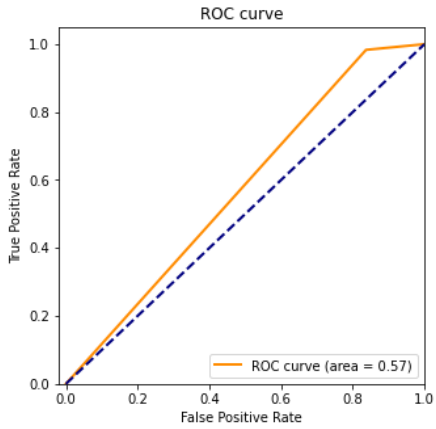


Figure 3.2: ROC-AUC curve representing the True Positives and False Positives

3.2.3 Feature Selection

Data development increases dimensions and computational costs are overcome by feature selection and extraction [9]. Selection of feature 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 [10]. 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, usage of the in-built *feature_importance_* function of XGBoost could be used for determining the importance of features. The default *importance_type* value provided by XGBoost is *"gain"*. This resulted in determination of the threshold(cutoff) value relatively easier. According to the *feature_importances_* function, gain from each feature in the dataset is noted in the subsequent table.

'ci_last_build_status'	0.23996039	'core_member'	0.23593336
'part_num_issue'	0.16061245	'ci_build_num'	0.054171726
'ci_first_build_status'	0.041376572	'first_pr'	0.04090331
'ci_exists'	0.02614957	'part_num_commit'	0.025500331
'acc_commit_num'	0.022627039	'perc_inte_pos_emo'	0.021209408
'contrib_follow_integrator'	0.021032399	'contrib_first_emo'	0.014595332
'same_country'	0.012426201	'language'	0.010328107
'ci_test_passed'	0.009435813	'open_pr_num'	0.0075981733
'description_length'	0.00701664	'open_issue_num'	0.006905698
'account_creation_days'	0.0054122065	'pr_succ_rate'	0.004028126
'first_response_time'	0.0037981686	'fork_num'	0.0033574612

'ci_latency'	0.0029261701	'contrib_gender'	0.0025455211
'project_age'	0.0021149688	'comment_conflict'	0.002082463
'churn_deletion'	0.0020254715	'contrib_affiliation'	0.00202366
'pushed_delta'	0.0018857354	'perc_inte_neu_emo'	0.0018369576
'part_num_pr'	0.0017614204	'perc_external_contribs'	0.0014349588
'part_num_code'	0.0010713502	'perc_inte_neg_emo'	0.00083616417
'churn_addition'	0.0008317455	'social_strength'	0.00082640274
'test_inclusion'	0.0006254929	'contrib_country'	0.0005026878
'bug_fix'	0.00029038585	'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 3.4: 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 modelling 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, 9 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', 'acc_commit_num']
```

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      0.595     0.168     0.262    126934
    Accepted      0.833     0.973     0.898    542653

    accuracy                          0.821    669587
   macro avg      0.714     0.570     0.580    669587
weighted avg      0.788     0.821     0.777    669587
```

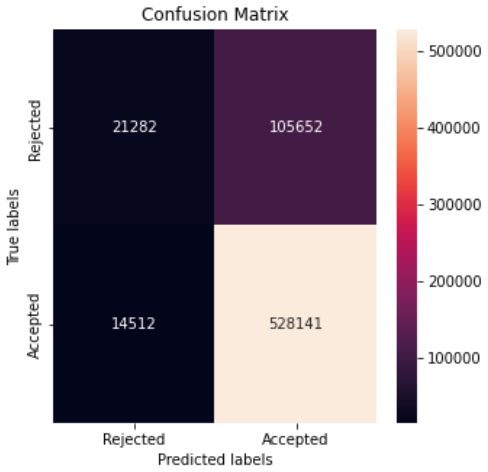


Figure 3.3: Confusion Matrix for experiment with features from Strategy 1

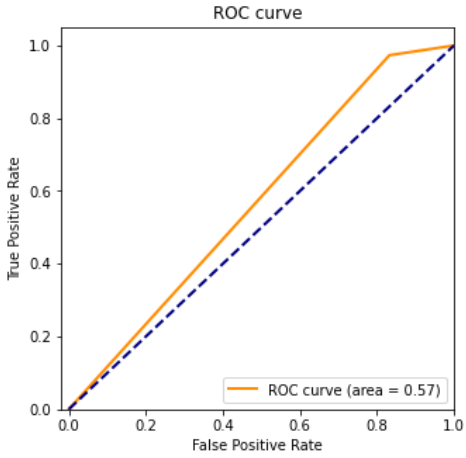


Figure 3.4: ROC-AUC curve representing the True Positives and False Positives

With almost no change in the ROC-AUC curve and very minimal change in the f-1 score, it could be safely said that these 9 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 of higher gain value than the threshold is considered to be the most important features driving the prediction the mechanism 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
```

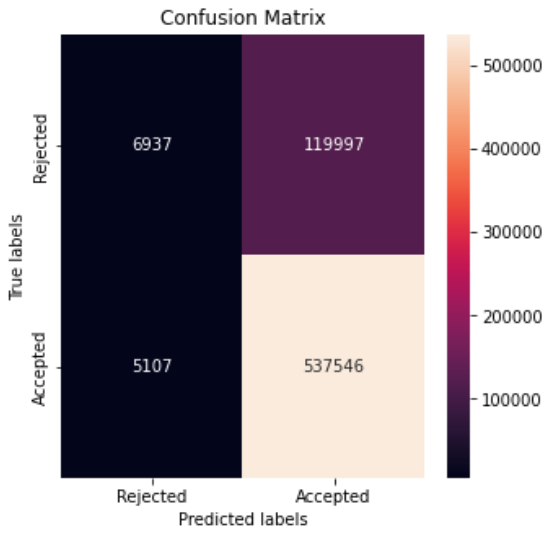


Figure 3.5: Confusion Matrix for experiment with features from Strategy 2

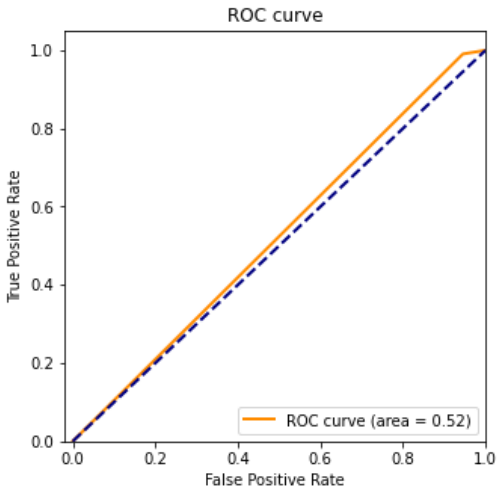


Figure 3.6: ROC-AUC curve representing the True Positives and False Positives

The f1-score reduced significantly and the area under the curve decreased to 0.52, indicating how important the other 5 features from *imp_features_1* is. In this case, the remaining 5 features could be termed as 'spurious features'.

Machine learning models are susceptible to learning irrelevant patterns [11]. They rely on some spurious features that we humans know to avoid [11]. 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

Derivative is primarily used when there is some varying quantity and the rate of change is not constant [12]. 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 off of. 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 (`-0.001467584463005716`), 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      0.706     0.159     0.260    126934
    Accepted      0.833     0.985     0.903    542653

    accuracy                          0.828    669587
   macro avg      0.770     0.572     0.581    669587
weighted avg      0.809     0.828     0.781    669587
```

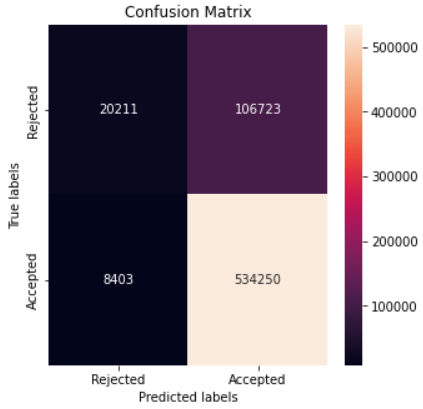


Figure 3.7: Confusion Matrix for experiment with features from Strategy 3

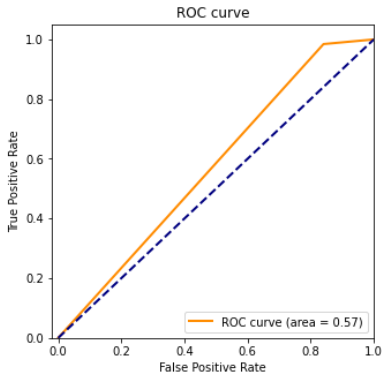


Figure 3.8: ROC-AUC curve representing the True Positives and False Positives

The f1-score and ROC-AUC curve has the same values as the baseline model which used all 45 features from the data and the model produced from sampling data in Strategy 1 using only 9 features. Hence, it is safe to say that even though there is zero negative impact from this model, this strategy is not fruitful for usage in evaluation process.

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,

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

The value (A) is the possible rates of attribute A, with Yc being the subset of y, where A possesses the sum of c [9]. Furthermore, the rule of *Eq. 1* is that the total entropy of y, followed by data segregation, based on feature A [9].

The information gain from the features, using the above method is present in the following image extracted from Colab+ Notebook[2] :

	feature	gain
44	same_country	0.000016
45	same_affiliation	0.000026
14	bug_fix	0.000065
12	test_inclusion	0.000069
10	hash_tag	0.000151
34	contrib_gender	0.000196
9	comment_conflict	0.000211
39	perc_inte_pos_emo	0.000506
4	pr_comment_num	0.000519
38	perc_inte_neg_emo	0.000571
7	part_num_pr	0.000617
8	part_num_code	0.000651
37	contrib_first_emo	0.000892
6	part_num_commit	0.001073
18	ci_test_passed	0.001271
41	contrib_follow_integrator	0.001331
40	perc_inte_neu_emo	0.001357
24	project_age	0.001925
20	ci_first_build_status	0.002163
22	language	0.002444
15	ci_exists	0.002533
35	contrib_country	0.002751
11	at_tag	0.002809
17	ci_build_num	0.004172
21	ci_last_build_status	0.012212
13	description_length	0.012738
31	first_pr	0.013166
19	ci_failed_perc	0.014183
3	churn_deletion	0.015566
36	contrib_affiliation	0.017104
2	churn_addition	0.019403
32	account_creation_days	0.019848
25	open_issue_num	0.020882
33	core_member	0.021656
27	open_pr_num	0.023592
5	part_num_issue	0.027620
42	prior_interaction	0.039590
16	ci_latency	0.040479
23	fork_num	0.046445
30	acc_commit_num	0.051246
28	first_response_time	0.056442
43	social_strength	0.057280
26	pr_succ_rate	0.135751
1	perc_external_contribs	0.233978
29	pushed_delta	0.310288
0	id	0.706550

Table 3.5: Information Gain for every feature.

The threshold was then determined by calculating the mean of the information gain values. Features that have 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_streng
th', '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 less features
              precision    recall  f1-score   support

    Rejected      0.620     0.163     0.258    126934
    Accepted      0.833     0.977     0.899    542653

    accuracy                          0.822    669587
   macro avg      0.727     0.570     0.579    669587
weighted avg      0.793     0.822     0.778    669587
```

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

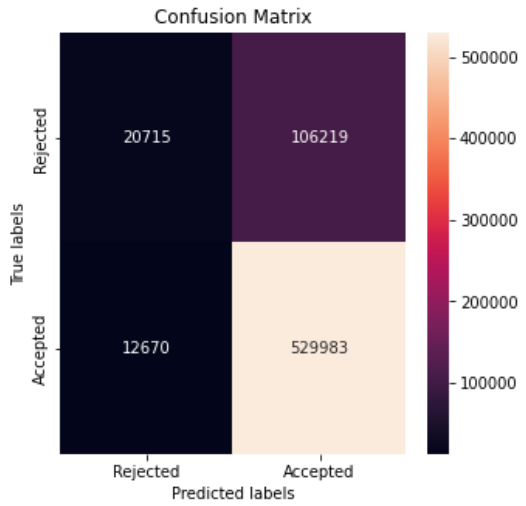


Figure 3.9: Confusion Matrix for experiment with features from Strategy 4

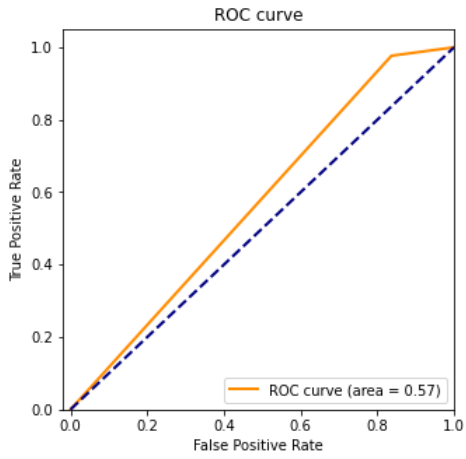


Figure 3.10: ROC-AUC curve representing the True Positives and False Positives

The f1-score and ROC-AUC area value remained the same as the baseline model, model from Strategy 1 and model from Strategy 3. However, a significant difference between this model and model from Strategy 1 is that the features involved in the prediction are different, except for 2 features. *imp_features_1* harvests a different set of features than *imp_features_4*. The only common features between the two lists being *'part_num_issue'* and *'acc_commit_num'*. This provides an interesting insight into the data.

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 selection of a subset of features. Thus, both the 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 **train_test** 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 pre-processing phase of **train_test** dataset were used for annotating the same values in **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 9 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      0.598     0.170     0.264    319295
    Accepted      0.833     0.973     0.897   1354674

    accuracy                          0.820   1673969
   macro avg      0.715     0.571     0.581   1673969
weighted avg      0.788     0.820     0.777   1673969
```

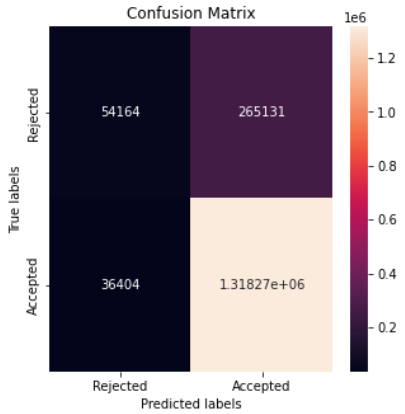


Figure 3.11: Confusion Matrix for experiment with features from Strategy 1 on evaluation dataset

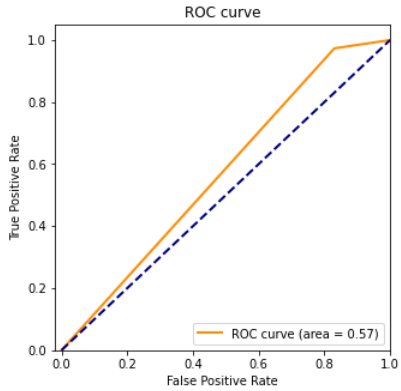


Figure 3.12: ROC-AUC curve representing the True Positives and False Positives

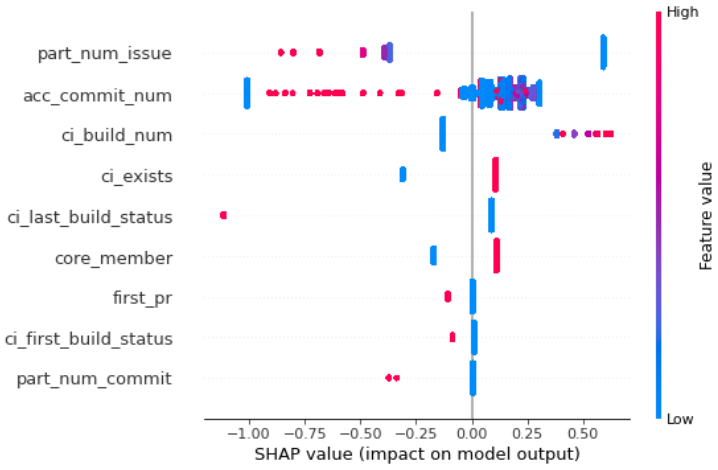


Figure 3.13: Shapley value representing directionality impact of features from Strategy 1

A further step was taken to determine the under-performing features among these 9 features through the use of SHAP (Shapley Additive exPlanations) [3] values. The *summary_plot* method was used to plot the directionality impact of each feature towards prediction. As evident, higher values of features *'acc_commit_num'* and *'part_num_issue'* leads to a pull-request being rejected. Even though, features *'ci_last_build_status'*, *'ci_first_build_status'*, *'first_pr'*

[3] An introduction to explainable AI with Shapley Values

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.

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	0.639	0.138	0.228	319295
Accepted	0.829	0.982	0.899	1354674
accuracy			0.821	1673969
macro avg	0.734	0.560	0.563	1673969
weighted avg	0.792	0.821	0.771	1673969

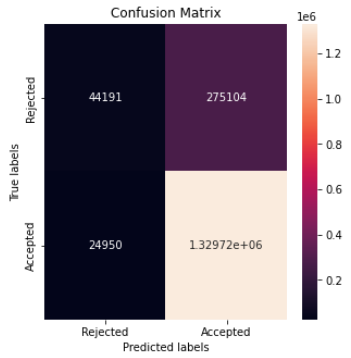


Figure 3.14: Confusion Matrix for experiment with features from Strategy 4 on evaluation dataset

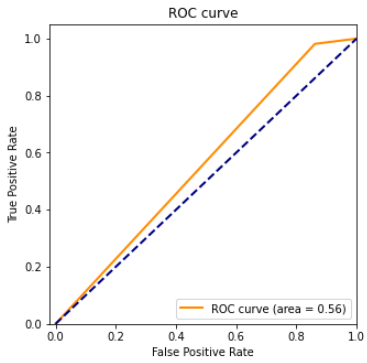


Figure 3.15: ROC-AUC curve representing the True Positives and False Positives

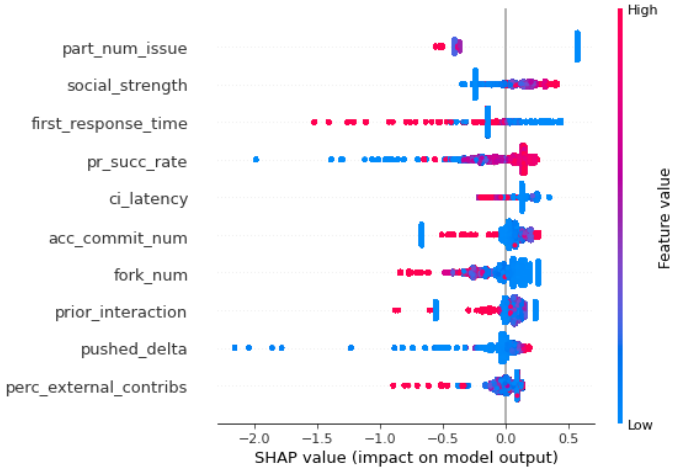


Figure 3.16: Shapley value representing directionality impact of features from Strategy 4

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.

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

- 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 upon after considering a couple of factors.
    - 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 BEAM search decoder in this experiment only consists of integer or float values as shown below.

	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

Figure 3.17: 3 rows representing a pull-request with multiple manually annotated features

    - The numerical values in a sequence provide little to no meaningful information.
    - Finally, some features have the value 0 or 0.00 in its 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 BEAM search decoder still had to deal with sequence of value not providing meaningful information.

  No logical idea to eliminate this problem could be produced at this point.
- The 45 features used to carry out the experiment are categorized in three groups- Contributor Characteristics, Project Characteristics and Pull-request Characteristics. Recommendation for most of the features are sometimes not practical. For example:
    - first_pr: It may be possible that the pull-request that is being recommended for is the first pull-request from the contributor.
    - core_member: Core members of a projects are more likely to have their pull-requests accepted but in open-source development, anybody could create a pull-request. However, not all individual

making the pull-request is a core-member and thus the acceptance rate is diminished.
- 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.
- same_affiliation: It is not possible for individuals to be employed by an affiliation to increase their chances of pull-request acceptance.
- language: It is not always possible to rewrite codes into a different language to increase chances of pull-request acceptance.
- account_creation_day: Individuals cannot increase or decrease the number of days they have been present of the GitHub platform.
- open_issue_num: Individuals cannot change the number of issues opened impulsively.
- project_age: Individuals do not have any control over the age of a project.

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 off of those two sets of features.

The 'recommendation' was done based off of 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:

```
Merged requests mostly had ' 1 ', whereas unmerged requests mostly had ' 1 ' in  ci_last_build_status
Merged requests mostly had ' 1.0 ', whereas unmerged requests mostly had ' 1.0 ' in  ci_build_num
Merged requests mostly had ' 1 ', whereas unmerged requests mostly had ' 1 ' in  ci_first_build_status
Merged requests mostly had ' 1.0 ', whereas unmerged requests mostly had ' 1.0 ' in  ci_exists
```

Figure 3.18: Recommendation output for pull-request features 'ci_last_build_status', 'ci_first_build_status', 'ci_build_num' and *'ci_exists'.*

Label 1 relates to "success" in case of all 4 features. However, suggestions could not be made in this case and no difference in value is observed.

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:

```
Merged requests mostly had ' 120.0 ', whereas unmerged requests mostly had ' 125.0 ' in  ci_latency
```

Figure 3.19: Recommendation output for pull-request features 'ci_latency'.

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

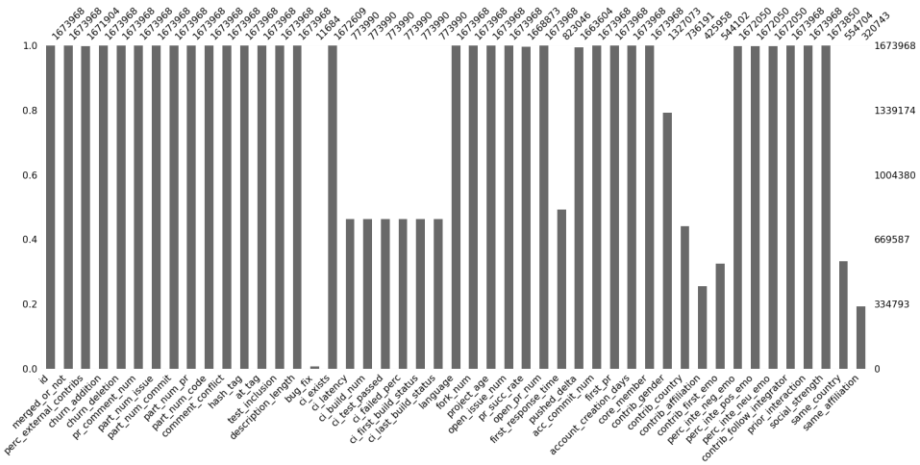
Chapter 4 Conclusion

The goal of the experiment was to determine the bad actors in a pull-request and provide recommendations to overcome the limitations of a bad pull-request to enhance the chances of merging.

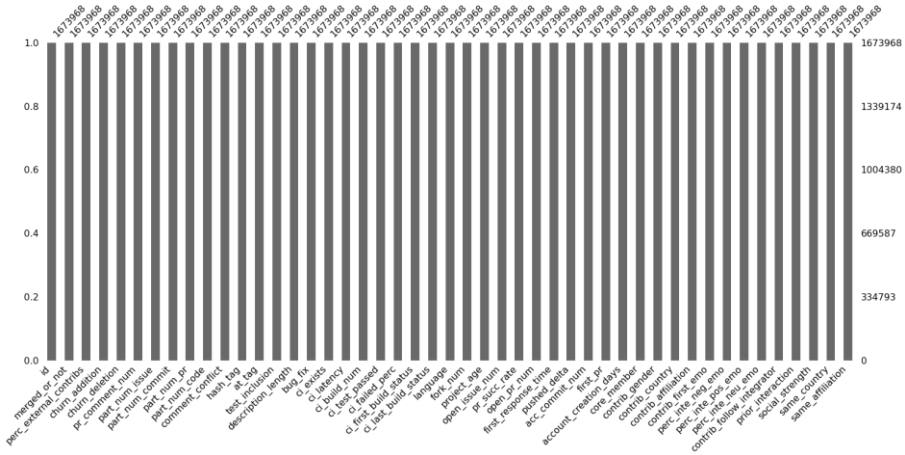
The exploratory research not only managed to find the bad actors related to a pull-request as expected for RS1, it provided deeper understanding of each of the feature involved and how they affect the merge decision; even though the actual merge decision depends on the integrator in a real-life scenario. The experiment discovered limitations in the recommendation phase. Although the limitations were not eliminated and recommendations provided as anticipated in RS2, it provided useful intuition towards the features involved.

The model provided some relatively good perceptions of the data and provided opportunities for further research. One of the opportunities being, synthesizing logical ways of overcoming the caveats discussed in the recommendation section. Another one being, 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 finally check if it is predicted to be accepted.

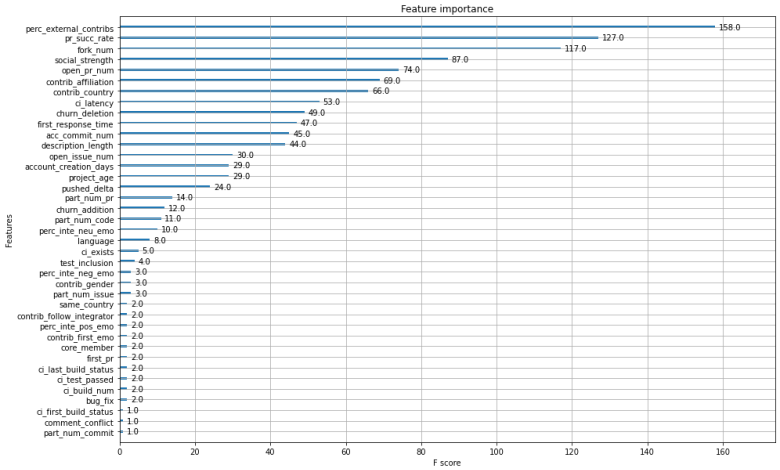
Appendix A



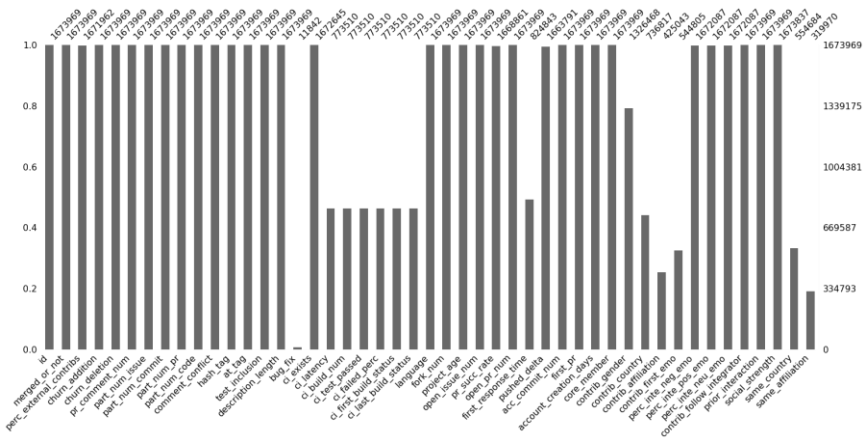
Appendix A.1: Missing values from **train_test** dataset with 1,673,968 pull requests with 45 features.



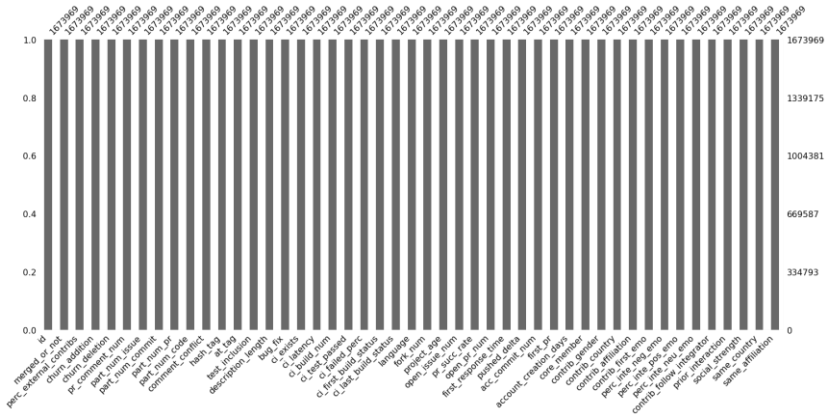
Appendix A.2: **train_test** dataset with 1,673,968 pull- requests with 45 features after pre-processing.



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.

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]