Recommending GitHub Projects for developer onboarding

Het Solanki*
het.solanki@lakeheadu.ca
Lakehead University
Student ID - 1150676

Nilam Bhosale
bhosalen@lakeheadu.ca
Lakehead University
Student ID - 1138964

Raj Salla
rsalla@lakeheadu.ca
Lakehead University
Student ID - 1155203

Abstract

GitHub is a software development platform that allows for project collaboration and participation. Developers typically look for relevant projects to reuse functions and find beneficial features. Recommending appropriate projects to developers can help them save time. Finding relevant projects from the numerous on GitHub is, however, tough. Furthermore, various users may have different needs. Developers would benefit from a recommendation system because it would cut down on the time it takes to locate relevant projects. Developers may be unable to contribute successfully due to a variety of social and technical constraints.
Failure to onboard developers frequently stymies not only individual advancement but also the evolution of open-source projects. In this work, we offer a method for recommending projects that takes into account developer habits and project features. We propose a learning-to-rank model, neural network for list-wise ranking (NNLRank), to select projects to which developers are likely to contribute, in order to reduce developers' costly onboarding activities. NNLRank recommends projects for onboarding based on project features and developer experience. We devise a method for optimising the neural network that employs a list-wise loss function to reduce the disparity between the predicted projects list and the ground-truth list chosen by developers.

Keywords: Neural networks, NNLRank, Dataset, GitHub, Projects, developers, GHTorrent, BigQuery

1 INTRODUCTION

GitHub, Inc. is an Internet hosting company that specialises on Git-based software development and version control. GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere [1]. It includes Git's distributed version control and source code management (SCM) features, as well as its own. Every project has access control and collaboration tools like bug tracking, feature requests, task management, continuous integration, and wikis. Developers are the founders of all software repositories in the open source software ecosystem. They don't just make a repository for their own project; they also look into other projects that interest them. They are not required to create anything from the ground up for their own initiatives. They frequently duplicate their relevant projects and then modify the characteristics of these projects to match their requirements [1]. Simultaneously, they are willing to contribute their abilities. They can offer an idea, solve a bug, or contribute code to the work of other developers. In the current situation, developers frequently expend considerable work (mainly manual) in locating appropriate open source projects to contribute to. They devote a lot of time to learning those projects. To learn about projects of interest, they frequently check various resources such as project documentation, mailing lists, forums, bug tracking systems, and so on. When developers begin contributing code after such manual effort, it is not guaranteed that their code will be accepted. Developers begin writing code and committing it in the hopes of their contributions being accepted. Before accepting contributions (e.g., commits), core project members frequently double-check that they are relevant to the project.
If the contributions continue to be unacceptable, developers must rewrite, evaluate, and test additional commits until they meet the project's standard and goal. Due to poor project selection at the start, this procedure can be quite time consuming. For beginners with little familiarity with the open source platform, the situation might be even worse, leading to dissatisfaction and final resignation. From the standpoint of the projects, a failed onboarding attempt impedes overall development progress often failed onboarding necessitates significant effort on the part of the core project members to examine unsatisfactory contributions provided during onboarding [3]. As a result, overall project growth may be slowed, causing significant unhappiness among developers, especially core members. Researchers are also developing tool support to assist newcomers with the onboarding process [4].
However, none of these tools directly assist developers in recommending relevant projects based on their prior experience and work history. GitHub recently began recommending projects to its users based on their ratings and the people they follow. So this recommendation is not tailored to the individual developer's needs, which not only wastes the developer's time to judge but also decreases the developer's trust in the recommendation. As a result, software project recommendations in the open source community must be individualised, as various engineers have different needs for the same project. We recommend projects that developers

care about in this article. In this paper, we make the following contributions:
[A] For the onboarding of GitHub projects, we built NNLRank, a recommender system.
[B] We also implemented Logistic Regression, neural network and decision tree classifier to get the better results and accuracy.
[C] Performed data preprocessing improve the overall accuracy.
[D] Created nine features to capture the onboarding pattern of developers.

1.1 Proposed Approach

We examine several project elements as well as developer activity and skill to capture the complex onboarding pattern. We present NNLRank, a learning-to-rank algorithm for recommending projects to which developers are likely to contribute. NNLRank scores a list of candidate projects using a ranking function (represented by a neural network), with the top-n projects being recommended to developers. We used a list-wise ranking loss function to repeatedly tune the network, with the goal of minimising the difference between the predicted list (project scores) and the ground-truth list selected by developers.
To allow fast convergence of model optimization, we use a learning rate decay method. NNLRank can also handle a list of candidate projects at the same time, allowing for faster onboarding. We evaluate the performance of various machine learning models such as Logistic Regression, Decision tree classifier and neural network to get the results with the highest accuracy.

2 PAPER ORGANIZATION

The following is a breakdown of the paper's structure. The background information and Methodology are described in the following parts. The experimental setting and Feature selection were then explained, followed by the results. Finally, we discussed our study's conclusion.

3 BACKGROUND

We address the onboarding problem in this work and present a recommendation tool to assist developers in finding appropriate projects to contribute to.
Researchers recommend using an onboarding tool to help developers find projects with fewer technical and social barriers. Some existing tools can assist developers in overcoming project obstacles by providing a search engine for API usages and high-quality examples, predicting code changing requests, and recommending code-related items such as closely related code, problem reports, and newsgroup articles [2].
These tools, on the other hand, assume that developers have already opted to join a project, although making such a decision is difficult [3], [4]. As a result, a recommender system must be built in the first place to recommend or filter projects (based on developers' search criteria).

3.1 Learning to Rank

Learning to rank is a method for converting a recommendation problem into a ranking problem by creating a scoring model in which the top-n rated projects are suggested. Learning to rank (LTR) refers to a group of algorithmic strategies that use supervised machine learning to solve search relevancy ranking difficulties.
In other words, it determines how query results are sorted. The training data for a learning to rank model consists of a list of results for a query and a relevance rating for each of those results with respect to the query. Pointwise, pairwise, and listwise are the three main ways to Learning to rank [6].

3.2 Point-Wise Ranking

Individual results are ranked using pointwise techniques, which look at a single document at a time and use classification or regression to find the optimal ranking. During these processes, we give each document points for how well they fit. We add those up and sort the result list. Note here that each document score in the result list for the query is independent of any other document score, i.e each document is considered a "point" for the ranking, independent of the other "points". However, pointwise models may fail to work as they ignore local relationships between projects so that projects with similar features are all assigned with the same scores [6].

3.3 Pair-Wise Ranking

Pairwise approaches look at two documents together. They also use classification or regression — to decide which of the pair ranks higher. We compare this higher-lower pair against the ground truth and adjust the ranking if it doesn't match. The goal is to minimize the number of cases where the pair of results are in the wrong order relative to the ground truth (also called inversions). pair-wise ranking has a high computation complexity since all pairs of projects are usually required to be processed.

3.4 List-Wise Ranking

Listwise approaches decide on the optimal ordering of an entire list of documents. Ground truth lists are identified, and the machine uses that data to rank its list. Listwise approaches use probability models to minimize the ordering error., They can get quite complex compared to the pointwise or pairwise approaches. Pointwise and pairwise approaches transform the ranking problem into a surrogate regression or classification task. Listwise methods, in contrast, solve the problem more directly by maximizing the evaluation metric. The loss is directly computed on the whole list of documents (hence listwise) with corresponding predicted ranks. In this way, ranking metrics can be more directly incorporated into the loss.

4 METHODOLOGY

4.1 Ranking Projects

To recommend suitable projects for a developer to onboard, we built a list-wise ranking model using a probabilistic function. There are certain features which affect the probability of a project being onboarded by new developers. Successful onboarding decision feature of a project is considered a major aspect which contributes to the onboarding decision probability of new developers. Apart from this, number of commits which represents how active the project is; is also contributing factor. Based on these two important features we decided to artificially define the onboarding probability function as follows:

$$\begin{aligned} df['onbProbability'] = 1 - df['onbDecision'] * 0.8 \\ - df['commits'] * 0.2 \end{aligned}$$

As observed from the equation, more weight is given to the successful onboarding decision as compared to number of commits as the prior is an important deciding feature. With this probabilistic function we achieved perfect ranking of our projects. Figure 1 shows the graph which is plotted against the successful onboarding decision against the onboarding probability obtained from the function. From this graph it is inferred that, the projects with the more number of successful onboarding decisions has higher probability of being onboarded. Conversely, the projects with less number of successful onboarding decision has the least probability of being onboarded as a result, these projects should not be recommended to new developers.

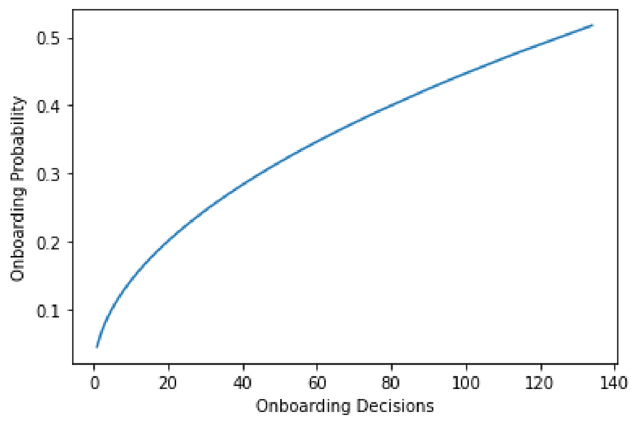

Figure 1. Ranking Projects

Converting a classification task to a ranking task was the initial step for this project. Different approaches have been carried out to learn this probabilistic ranking function to observe better outcome and performance. The following section will discuss NNLRank method in depth.

4.2 Neural Network

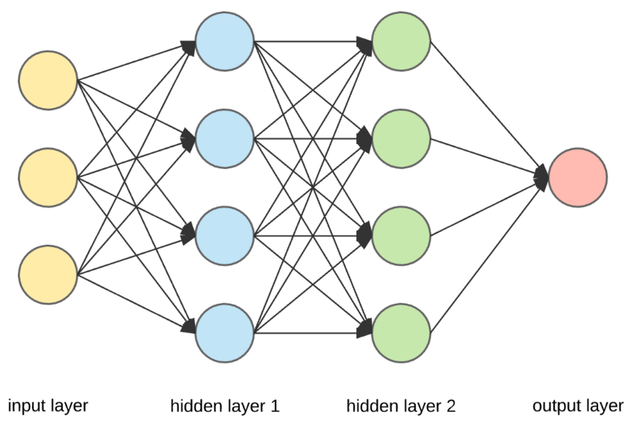

Figure 2. Neural Network

Figure 2 shows the network structure of our model contains 4 layers with two hidden layers, one input layer and one output layer. The input layer has 8 nodes corresponding to 8 designed features. Each hidden layer contains 4 hidden nodes and the output layer contains only 1 node for the rank.
Precisely, to predict the rank of n projects represented by 8 features we are using forward propagation and back propagation techniques to train our model. The network takes all these projects 8 features for its input layer, and passes them through each layer from left to right. A dot product of input from previous layer and corresponding weights and bias with current layer is calculated. Sigmoid is used as an activation function between input layer and hidden layers.The final step in a forward pass is to evaluate the predicted output(rank) against an expected output.
Note that, as the weights and biases are unknown at first, we initialize them to uniform random values. To optimize weights and biases in network, we use a loss function. Log loss function is used which aims to minimize the difference between the predicted output and the expected output. gradients of this loss function is calculated with respect to weights and biases which is then back propagated to adjust the weights and biases in each layer based on these gradient values.
We perform a gradient descent method to optimize these weights and biases. The optimization stops when the loss function converges to a small value. The important aspect of optimizing the network parameters using gradient descent is that, it calculates the gradients of loss function first and then updates the weights and biases in each layer. Learning rate

is a very significant value, which decides how long the next step will be for the gradient descent approach. If the learning rate is not proper, the model will converge very slowly. It is a trial and error method to find a perfect learning rate value.

5 EXPERIMENTAL SETUP

We'll walk through our project's life cycle. In this part, we'll discuss about the dataset, how data was obtained for initial setup, how we pre-processed the dataset to get sampled projects, how we implemented feature selection, and how we recommended the projects. You can refer to the Figure 3 as shown below:

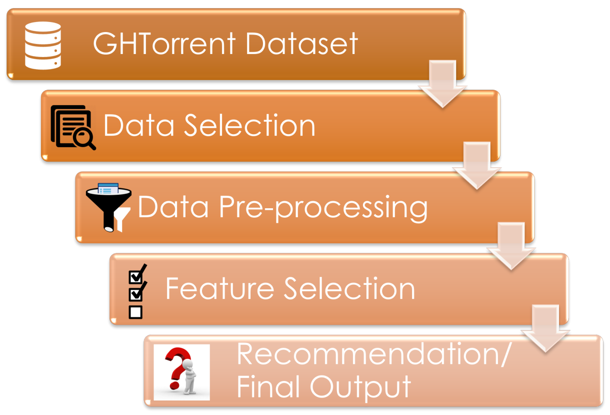

Figure 3. Proposed Methodology

5.1 GHTorrent Dataset

The gathering and curation of data from software repositories is a typical prerequisite of many empirical software engineering research. GitHub has been a popular project hosting, mirroring, and collaboration tool in recent years. Researchers can utilise the REST API provided by GitHub to access both commits to project repositories and events created by user actions on project resources. GHTorrent seeks to build a scalable off-line copy of GitHub's event streams and persistent data and make it available as a service to the research community. The project's design and first implementation are presented in this document, as well as how the offered datasets may be accessed and analysed.

5.2 Data Selection

One of the most challenging tasks in any machine learning project is data preparation. The process of cleaning and altering raw data prior to processing and analysis is known as data preparation. It's a crucial stage before processing that frequently include reformatting data, making data changes, and integrating data sets to enrich data. For data experts or business users, data preparation might be time consuming, but it is necessary to put data in context in order to transform it into insights and reduce bias caused by poor data quality. Standardizing data formats, enhancing source data, and/or reducing outliers are all common steps in the data preparation process.

In this section, we'll go through the dataset as well as how we got the data and analysed it. We selected the database from the GHTorrent data dump which consists GitHub data between the years 2012 and 2016. GHTorrent monitors the Github public event timeline. It then stores the raw JSON responses to a MongoDB database, while also extracting their structure in a MySQL database. BigQuery was used to extract data from Ghtorrent Dump stored on Google cloud platform. In the database schema available on Google cloud platform consisted of 20 different tables comprising of various information regarding projects, users and repositories. For our project, we selected projects, project_members, Commit, Watchers and Users.

5.3 Data Pre-processing

The process of converting raw data into a comprehensible format is known as data preprocessing. We can't deal with raw data, thus this is a key stage in data mining. Before using machine learning or data mining methods, we need to make sure the data is of good quality.

5.3.1 Sampling Projects.

We investigated 4 years of onboarding decisions (2012 - 2016) from GHTorrent data dump whose projects are written in 2 popular programming languages Java and Python on GitHub. It is commonly known that GitHub has a large number of dormant and personal projects that cannot be joined by other developers, potentially resulting in biased analysis. As a result, deleted, private, and forked repositories were eliminated, while projects with fewer than 2 watchers and 5 developers were excluded. Furthermore, non-English projects with insufficient descriptions are more likely to be personal or toy repositories, and thus will struggle to attract global developers. In total, we sampled 2974 active projects from Github.

5.3.2 Sampling Developers.

The sampled 2974 projects contains 54304 developers and among these developers its was required to extract only active developers. To focus on active developers, we excluded deleted users, fake users and users without geographical information. The fake users are accounts that use GitHub authentication to spam other websites, and they are discovered by GitHub and labelled as fake in the collected dataset. In this way, we extracted 20883 active developers.

5.3.3 Successful Onboarding Decisions.

As developers join and leave projects frequently, a successful onboarding process should contain major commits after a developer joins. In this study, we defined successful onboarding as a choice with at least 6 commits (median of the sampled active projects). Furthermore, because developers cannot join projects they start themselves, we deleted such decisions.

5.4 Feature Selection

The act of picking the most significant features to input in machine learning algorithms is known as feature selection, and it is one of the primary components of feature engineering. Feature selection strategies are used to minimise the number of input variables by removing redundant or unnecessary features and restricting the collection of features down to the ones that are most useful to the machine learning model [8].
The following are the primary advantages of completing feature selection in advance rather than relying on the machine learning model to determine which features are most important:

1. Simpler models are easier to describe; a model that is overly complicated and difficult to understand is not valuable.
2. training durations are reduced: a more exact collection of characteristics reduces the amount of time required to train a model.
3. Increase the precision of the estimations that may be derived for a particular simulation by reducing variance.
4. escape the curse of high dimensionality: the dimensionally cursed phenomenon claims that as dimensionality and the number of features rise, the volume of space expands so quickly that the amount of data accessible decreases - To minimise dimensionality, feature selection can be employed.

The selected features should contain the important information from the input data, allowing the intended job to be completed using this reduced representation rather of the entire initial data. For our project we were able to extract a total of 8 features. Figure 4 below shows a list of them, with comprehensive explanations thereafter:

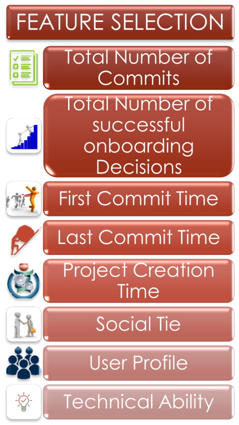

Figure 4. Selected Features

5.4.1 Total Number of Commits.

The commit command on GitHub is used by developers to put their modifications into a project. The total number of commits in a project indicates its quality. The less commits a project has, the less likely it is to be suggested, and vice versa. A commit will only count as a contribution if one of the following is true [9]:

1. You are a collaborator on the repository or are a member of the organization that owns the repository.
2. You have forked the repository.
3. You have opened a pull request or issue in the repository.
4. You have starred the repository.

If a project had 1290 commits, but only 1098 were accepted, we only examined those commits and attributed the total number of commits for that project to 1098. The commits table was utilised to assess the total number of commits. We gathered all of the accepted commits for a project and totaled them to get the overall number of commits.

5.4.2 Total Number of successful onboarding Decisions.

The total number of successful onboarding decisions refers to the number of developers successfully onboarded/ joined to a project. The greater the number of developers on board, the better the project will be. Because developers join and leave projects often [10], an effective onboarding process should contain major commits once a developer joins.
If there were three developers on a project, we calculated the absolute difference between the time they joined and the time the project was created, and if the difference was greater than 0, we designated that as a successful onboarding decision, and added the count for each successful onboarding decision. The projects table was used to determine the overall number of successful onboarding decisions. In this study, we defined successful onboarding as a choice with at least 6 commits (median of the sampled active projects).

5.4.3 First Commit Time.

A commit is an activity in version control systems that transmits the most recent changes to the repository, making these changes part of the repository's head revision. Unlike data management commits, commits in version control systems are stored indefinitely in the repository. Some projects' lifecycles begin with the first commit, rather than with the participation of the developers. As a result, we calculate how long a project has been accepting contributors since its initial contribution to determine its onboarding potential.
If a project has 23 commits, and the commit with the commit id 150 has the least time or the first commit, that commit id was chosen. The commits table was used to determine the time of the first commit. We determined the first commit time for each project by calculating the lowest commit time for that project.

5.4.4 Last Commit Time.

The last commit time is the last pushed event, which reflects the most recent commit made (on any branch) and submitted to any repository by a user. Even if no newcomer has joined in, a project may be updated often by developers. We assume that this scenario indicates that the project's personnel is insufficient, and that more developers are needed to share the workload. As a result, we use the last commit time to calculate the time since the project's last commit.
If a project has 5 commits, and the commit with the commit id 123 has the most recent time, that commit id was chosen. The commits table was utilised to determine the last commit time. By determining the most recent commit time for each project, we were able to determine the last commit time.

5.4.5 Project Creation Time.

It has been established that a project in its early stages attracts more developers because when a project is steady, developers confront greater technical and social hurdles [11], [12]. The moment when a project is successfully created is referred to as the project creation time.
We found the absolute difference between onboarding time and project creation time to assess the development phase of a prospective project. The projects table was utilised to determine the project creation time. By initially determining the project creation time from the projects table, we were able to calculate the difference in time for the first developer when he joined the project.

5.4.6 Social Tie.

Developers are eager to join a project whose members have previously cooperated with them [11], and the project owner is the major factor for attracting developers' participation [12]. As a result, we use the number of collaborative projects with the owner to determine a developer's social ties to a project. For bigger projects with numerous participants, however, individual participation may be constrained since developers are less inclined to communicate with the project owner [11].
If a developer in a project has previously worked with the developer who is seeking to onboard, the Social Tie is 1. Similarly, we assessed each developer's Social Ties and added them together to determine the overall Social Ties for a given project. The Social connection was discovered using the project_members table form database. We initially choose a project, then searched for projects that all of the members had previously worked on together and totaled the results.

5.4.7 User Profile.

We'll suppose that a developer wants to work on a project with others from the same firm. As a result, the developer will be able to swiftly adjust to the new project[13].
If a developer in a certain project works for the same business as the developer wanting to onboard that project, User Profile is equal to 1. Similarly, we totaled the user profiles of all developers who worked for the same business. The users table was used to locate the user profile. We tallied the number of users who shared the same corporate profile during onboarding (Usercompany).

5.4.8 Technical Ability.

Developers are more inclined to code in languages they are familiar with [11], [14]. To do so, we analyse how well developers' skills fit the requirements of a target project using past language experience.
If a developer has worked on two Java projects and three PHP projects, and a potential project is written in Java, then Technical ability is equal to 2. We utilised the projects table and the project_members table to discover the technical abilities. First, we picked a project from the projects database, and then we looked up developers for that project in the project_members table. We discovered all the projects where they worked on the same language from the projects table after finding two developers from the project_members table.

6 TRAINING AND TESTING

We have a total of 2937 candidate project lists, with each list including 257 to 2794 projects. To reduce the impact of outliers, we use the min-max normalisation approach to normalise 8 feature values across each list. The data was divided into two sets: training and testing, with one set (25 percent of the lists) being utilised for testing and the other for training. Furthermore, there was a class imbalance observed which resulted in inefficient results.
As seen in the Figure 5 the samples for the positive class are less compared to the negative class. To address this issue we performed random oversampling which randomly duplicates samples from the minority class.

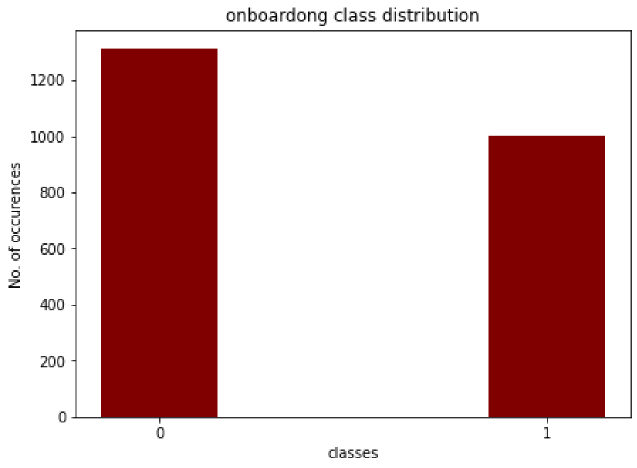

Figure 5. Class Distribution

7 RESULTS

This section investigates whether the proposed model can recommend the right projects to developers for their successful onboarding. Using NNLRank model we achieved a good testing accuarcy of 83.59%. The Stochactic Gradient Descent algorithm along with backtracking resulted in gradual convergence towards optimum results. The model is learning gradually with each epoch as observed in the Figure 6 where the accuracy increases with each epoch. Conversely, the Figure 7 shows that error between predicted and expected output gradually decreases with each epoch.

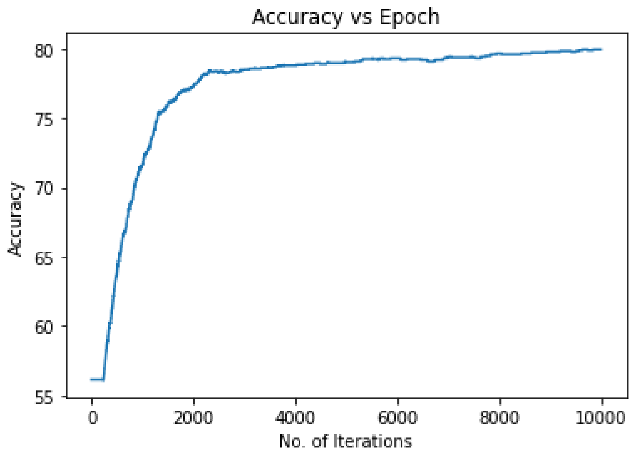

Figure 6. Accuracy

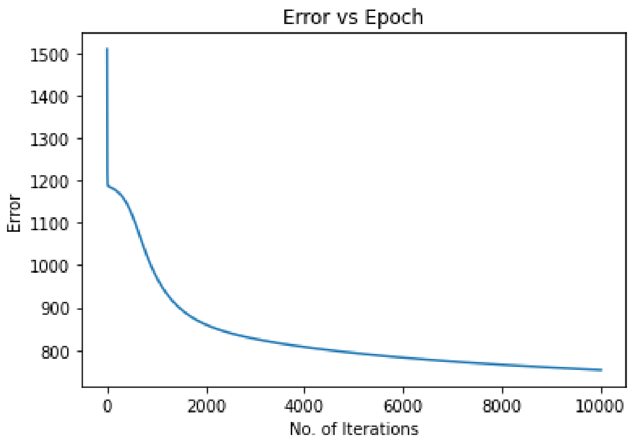

Figure 7. Error

NNLRank model shows the precision of 83.79% and recall of 75%. Which means that our model has 75% probability of correctly predicting a given sample. The confusion matrix shown in the figure 8 gives more idea about the performance of our model. 60 samples for class '1' (Recommend to Developer) are incorrectly classified whereas only 35 samples of class '0' (Do not Recommend to Developer) are incorrectly classified.

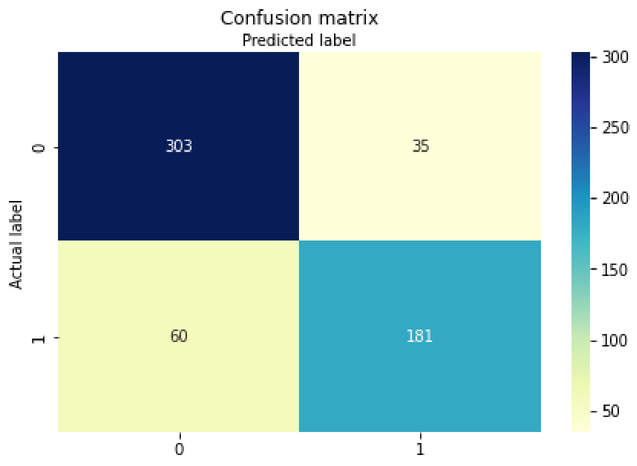

Figure 8. Confusion Matrix for NNLRank

To Compare our model's performance we performed three more algorithms namely Logistic Regression, MLPClassifier and Decision Tree Classifier. Table represents the accuracy, precision and recall for NNLRank model, Logistic Regression and MLP Classifier. The results from Decision Tree Classifier is not included as it resulted in overfitting.

Model	**Accuracy**	**Precision**	**Recall**
NNLRank	83.59	83.79	75.10
Logistic Regression	78.92	87.89	57.26
Mlp Classifier	97.40	96.31	97.10

As observed from the table, Logistic Regression gives the least effecient performance with 78.92% accuracy. Whereas, MLP Classifier provides the highest accuracy with 97.40%. Compared to other two approaches, NNLRank model gives a comparatively good performance but it can be improved and can give correct predictions with more accuracy.

8 CHALLENGES

There are many features which involves time and time differences which lead to confusion in selection of developer's onboarding time. We selected first commit time of developer's instead of project joined time to avoid discrepancy. Also, the 8 features represents a particular project which involves time differences between project creation time and Developer's commit time but there are multiple developers in a project and it leads to another challenge as which Developer's data should be selected to represent the project. Furthermore, we extracted only 8 features but there can be more features which affect the developer's onboarding decision. We plan to explore more viable features in the future.

9 CONCLUSION

Despite the fact that onboarding is critical to the success of open source ecosystems, developers frequently experience huge hurdles in finding acceptable projects. In this research, we offer a probabilistic list-wise ranking mechanism that assists engineers by selecting appropriate projects for onboarding. To anticipate relevant projects, NNLRank Rank uses project attributes and developers' experience as features to represent projects.
Probabilistic function based on important features proved to be an effective ranking mechanism. Various methods such as Logistic Regression, MLPClassifier and DecisionTree Classifier were implemented for comparision with our NNLRank model. We proved the usefulness and efficiency of the List-wise model by experimenting with 2974 successful onboarding decisions. It significantly outperforms Logistic Regression while MLPClassifier proved to be most effecient. However, with more data and optimization NNLRank model can further improve its performance.

Acknowledgments

We would like to recognize the writers from different associations we have perused papers from and the individuals who have helped us in directing this research. Similarly, we would like to express our gratitude to Dr. Muhammad Asaduzzaman for directing us through his talks for giving important inputs and feedback that assisted us in completing this project.

References

[1] Hello world. GitHub Docs. (n.d.). Retrieved April 20, 2022, from https://docs.github.com/en/get-started/quickstart/hello-world

[2] Sun, Xu, W., Xia, X., Chen, X., & Li, B. (2018). Personalized project recommendation on GitHub. Science China. Information Sciences, 61(5), 1–14. https://doi.org/10.1007/s11432-017-9419-x

[3] I. Steinmacher, M. A. G. Silva, M. A. Gerosa, and D. F. Redmiles, "A systematic literature review on the barriers faced by newcomers to open source software projects," Inf. Softw. Technol., vol. 59, pp. 67–85, Mar. 2015.

[4] I. Steinmacher, T. Conte, M. A. Gerosa, and D. Redmiles, "Social barriers faced by newcomers placing their first contribution in open source software projects," in Proc. 18th ACM Conf. Comput. Supported Cooperat. Work Social Comput., 2015, pp. 1379–1392.

[5] Sun, X., Li, B., Duan, Y., Shi, W., amp; Liu, X. (2016, June 9). Mining Software Repositories for Automatic interface recommendation. Scientific Programming. Retrieved April 20, 2022, from https://www.hindawi.com/journals/sp/2016/5475964/

[6] Casalegno, F. (2022, February 28). Learning to rank: A complete guide to ranking using Machine Learning. Medium. Retrieved April 20, 2022, from https://towardsdatascience.com/learning-to-rank-a-complete-guide-to-ranking-using-machine-learning-4c9688d370d4

[7] Jiang, Wu, Q., Cao, J., Xia, X., & Zhang, L. (2021). Recommending tags for pull requests in GitHub. Information and Software Technology, 129, 106394–. https://doi.org/10.1016/j.infsof.2020.106394

[8] What is Feature Selection? Definition and FAQs | HEAVY.AI: 2022. https://www.heavy.ai/technical-glossary/feature-selection. Accessed: 2022- 04- 20.

[9] Getting all your commits in your contributions graph: 2022. https://github.community/t/getting-all-your-commits-in-your-contributions-graph/10186. Accessed: 2022- 04- 20.

[10] G. Robles and J. M. Gonzalez-Barahona, "Contributor turnover in libre software projects," in IFIP International Conference on Open Source Systems. Springer, 2006, pp. 273–286.

[11] K. Crowston, Q. Li, K. Wei, U. Y. Eseryel, and J. Howison, "Self-organization of teams for free/libre open source software development,"Information and software technology, vol. 49, no. 6, pp. 564–575, 2007

[12] J. Hahn, J. Y. Moon, and C. Zhang, "Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties," Information Systems Research, vol. 19, no. 3, pp. 369–391, 2008.

[13] Liu, Yang, D., Zhang, X., Ray, B., Rahman, M. M. (2018). Recommending GitHub Projects for Developer Onboarding. IEEE Access, 6, 52082–52094. https://doi.org/10.1109/ACCESS.2018.2869207

[14] Montandon, Valente, M. T., Silva, L. L. (2021). Mining the Technical Roles of GitHub Users. Information and Software Technology, 131, 106485–. https://doi.org/10.1016/j.infsof.2020.106485