Intelligent Program Analysis and Program Indexing - III

A Thesis Submitted

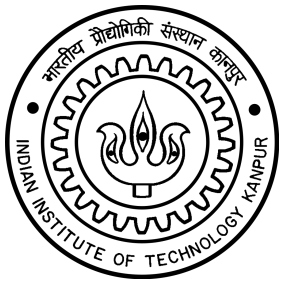
In Partial Fulfillment of the Requirements

For the Degree of M.Tech.

by

Mayank Bansal

20111032



to the

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur

January, 2023

Declaration

This is to certify that at the thesis titled **"INTELLIGENT PROGRAM ANALYSIS AND PROGRAM INDEXING - III"** has been authored by me. It presents the research conducted by me under the supervision of **PROF. PURUSHOTTAM KAR, PROF. AMEY KARKARE**.

To the best of my knowledge, it is an original work, both in terms of research content and narrative, and has not been submitted elsewhere, in part or in full, for a degree. Further, due credit has been attributed to the relevant state-of-the-art and collaborations (if any) with appropriate citations and acknowledgments, in line with established norms and practices.

Name: Mayank Bansal (20111032)
Program: M.Tech.
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur, Kanpur, 208016.
January, 2023

Page intentionally left blank

Certificate

It is certified that the work contained in the thesis titled **"INTELLIGENT PROGRAM ANALYSIS AND PROGRAM INDEXING - III"** by **MAYANK BANSAL** has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Prof. Purushottam Kar, Prof. Amey Karkare
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
Kanpur, 208016.
January, 2023

Abstract

Name of student: **Mayank Bansal** Roll no: **20111032**
Degree for which submitted: **M.Tech.**
Department: **Department of Computer Science and Engineering**
Thesis title: **Intelligent Program Analysis and Program Indexing - III**

Name of thesis supervisor: **Prof. Purushottam Kar, Prof. Amey Karkare**
Month and year of thesis submission: **January, 2023**

The use of artificial intelligence and machine learning in education has grown significantly in recent years. Numerous AI-assisted solutions for use in educational settings are constantly being developed. IIT Kanpur's ESC101, an introductory programming course, uses tools like PRIORITY to help the problem setters.

PRIORITY is a tool with AI support for labelling programming issues. The programme is designed to assist instructors of IIT Kanpur's ESC101 programming course. The enormous corpus of programming problems from earlier ESC101 offers are labelled by PRIORITY using semi-supervised approaches, making the issues searchable. Tutors' jobs could become considerably quicker and simpler as a result.

Users can look for programming questions from previous ESC101 course offers on PRIORITY's web portal. It makes use of machine learning algorithms to automatically label each programming question so that it may be searched for.

Priority has been designed with the ability to store feedback (both implicit and explicit) but is not trained on it. We incorporate the feedback by curating a new dataset. Based upon several carefully selected criteria, the feedback is classified into two categories, and then instance-wise weights are assigned. Due to the similarity in the design scheme between the old and the new dataset, we were also able to easily perform a comparative study between the previous version (refer here) and the new version of Prirority. We also used an Ensemble based technique for the label prediction task.

We propose further optimisation by moving away from the traditional static weight assignment to a more dynamic approach.

Acknowledgments

I would like to express gratitude to all the people who helped me during my stay at IIT Kanpur. I especially thank my advisors Purushottam Kar and Amey Karkare for guiding me throughout this thesis work and also to help me improve as a person. They were always there when I got stuck or need help with something.

I would also like to thank Preeti and Debanjan who has been my partner during this thesis. I also extend thanks to my batch mates, friends and family members.

This thesis was compiled using a template graciously made available by Olivier Commowick `http://olivier.commowick.org/thesis_template.php`. The template was suitably modified to adapt to the requirements of the Indian Institute of Technology Kanpur.

Mayank Bansal
January, 2023

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Contents

Interest in the topic of Artificial Intelligence (AI) has increased dramatically across a number of domains. Particularly intriguing is the increase in the use of AI tools and algorithms in the field of education. Massive open online courses (MOOCs) are quickly taking off as a well-liked method of instructing programming. The need to create technologies that can lessen the workload for the instructors and Teaching Assistants (TAs) grows as the number of students enrolled in these courses rises.

Compilation mistakes are one of the largest learning obstacles for a new programmer. [5] demonstrates how much time rookie programmers spend attempting to fix these mistakes. Additionally, the diagnostics offered by the compiler are frequently too complicated for beginning programmers, which may exacerbate the problem rather than resolve it. Because of this, there has been a lot of attention recently in the field of automatic compilation error repair. [3] suggests an AI-based pipeline that accepts an incorrect programme as input and outputs an accurate target program. Additionally, they divide this compiler error correction procedure into a number of manageable modules, which makes the pipeline a far better fit for instructional applications than other cutting-edge approaches. [6] [10] enhance their work by making changes to certain modules. Concretely, [6] [10] demonstrate how compiler diagnostics can offer some insightful input and how better label metadata management can greatly benefit the pipeline. They also suggest a brand-new approach to creating synthetic data that will directly help the pipeline.

Large MOOCs also struggle with the reuse of questions and problems. Consider IIT Kanpur's ESC101, a fundamental programming course that enrols hundreds of students each semester. The lecturers and tutors must provide programming issues for each session of the course, which the students then resolve in lab sessions. However, subsequent offerings cannot make use of this enormous corpus of issues that has been gathered because these problems are not indexed. In order to address this, PRIORITY [10] [6] (PRoblem IndicatOr ReposITorY) was created which is an AI-based labelling method to categorise these problems. A web app was also designed to enable this course's instructors to look for a problem from a previous offering.

We propose improvements which lead to optimisations in the working of PRIORITY. The most significant improvement being the incorporation of user feedback which enables PRIORITY to make suggestions that are more tailored to the needs of the tutors or the instructors or the Teaching Assistants (TAs).

We also propose few changes to base classifier of PRIORITY which leads to minor improvements in accuracy metrics as well as a considerable speedup. The speedup will allow PRIORITY to provide faster suggestions to the users, thus enhancing their experience.

1.1 Our Contributions

The key contributions of this thesis are enumerated below:

1. We Looked into the data generated by the repair class module of MACER. Besides the usual repair classes, MACER clubs the rest of the classes into miscellaneous class. We looked into the repairs for miscellaneous class to identify any patterns via which another repair class could be formed.

2. We propose accommodation of the user generated feedback which can be in the form of clicks, texts, comments, ratings etc. into the dataset for better suggestions to the user.

3. We propose use of threshold in the label propagation in PRIORITY [10] to ensure:

    - Concrete prediction by the classifier.
    - Speedup by reducing the number of cycles.
    - Improvement in precision and recall.

CHAPTER 2

Related Works

Other works related to MACER [3] and PRIORITY [10] [6] has been done and published.

[6] and [10] introduced MACER++ which was an improvement to MACER. The focus was on optimisations to each of the modules and repair classes. They also proposed atomic repair, a pipeline of Data generation, Feature Encoding, Training and then finally prediction. Atomic repair focuses on atomic operations or a single insert, single replacement, single deletion. It also works well with zero shot repairs, i.e., it performs well on unseen errors.

[2] and [11] suggests a number of modifications to the PRIORITY back-end machine learning architecture. [2] [11] suggest utilising a more substantial set of features. These characteristics were taken from the solution C programs' Abstract Syntax Trees and Function Call Graphs.

For feature selection, [11] experimented with a variety of correlation techniques. The test data points' f1-score significantly increased as a result of the Pearson correlation algorithm's efficient feature selection. Following supervised feature selection, one-vs-rest logistic regression is used to train the model. Because they significantly rely on programming language tokens, programming labels exist that demand different numbers of attributes to be detected. As a result, [11] experimented with various feature counts for each programming label and noticed a notable increase in the f1-score. By employing the ensemble technique during the feature selection phase, they proposed further optimization. This method greatly enhances accuracy and f1-score by combining Pearson correlation and normalised mutual information for different values of K.

The feature extraction as suggested by [2] has been carried out using a variety of tree traversal and graph techniques. [2] employ a filter-based supervised feature selection technique with normalised mutual information score to train a one-vs-all logistic regression model for the label prediction problem. To make sure that the model only predicts the pertinent labels during prediction, they use a ranking-style strategy. To complete the difficulty score prediction task, they use ensemble learning approaches. To integrate the predictions of the top-performing regressor and classifier models, they have employed a Mixture of Experts (MOE) model.

CHAPTER 3

PRIORITY

Contents

Every semester, ESC101, a course for beginning programmers, is provided. It teaches the fundamentals of programming. Every week in this course, labs are held where students are given programming questions to answer in a set amount of time. This quiz's questions are based on the programming ideas covered in this week's and prior weeks' lectures. These programming questions must be created each week by the course tutors (and TAs). Using the online coding platform PRUTOR [4], a sizable corpus of such questions from all of this course's offerings has been compiled. However, because these issues cannot be searched, every year the tutors must start by developing these questions from fresh. It takes a lot of time and work to do this.

The primary goal of PRIORITY is to enable searchable access to the extensive corpus of programmes that are accessible from prior ESC101 course offerings. The ability to search these questions depending on the knowledge needed to solve them, their difficulty, etc. should be available to tutors and teaching assistants (TAs). This would provide the tutors a solid place to start because they could see what kinds of issues were used in earlier presentations. As a result, solving problems becomes lot easier and quicker. There are 28 tags or labels in the PRIORITY that are

connected to programming ideas in order to make these issues searchable. To better understand the working of PRIORITY refer to [10] and [6].

In this section we discuss different optimisations like confidence score, data curation using feedback and static and dynamic weights for PRIORITY which we are able to propose after various experiments .

3.1 Confidence Score and Threshold

In the prior work [10] [6], PRIORITY used label propagation (which is a semi-supervised machine learning technique). The process involves using the already labelled instances to label the unlabelled instances. It is carried out to increase the number of labelled instances due to several of the well known algorithms performing poorly on the available data.

The process is carried out on a per cycle basis. The classifier provides confidence scores for all of the predictions that it does, and 50 top predictions are moved to the training set with the predicted labels. This cycle is repeated till all of the unlabelled instances are moved to the labelled set. A major drawback of this setup is that the classifier's predictions are magnified. Meaning, if the classifier performs poorly over the initially labelled data, then the labeling of the unlabelled data would follow the same.

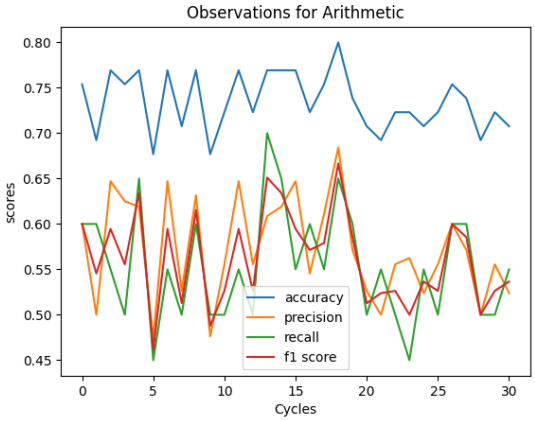
The precision and recall plots for each of the labels per cycle can be seen in the graphs included.

A better approach we present is to observe the probabilities for each of the labels presented by the algorithm (note that the labelling is done via a one vs rest approach where for each of the problem instances it is predicted by the algorithm whether it belongs to the given label or not) and put a threshold or a cut-off.

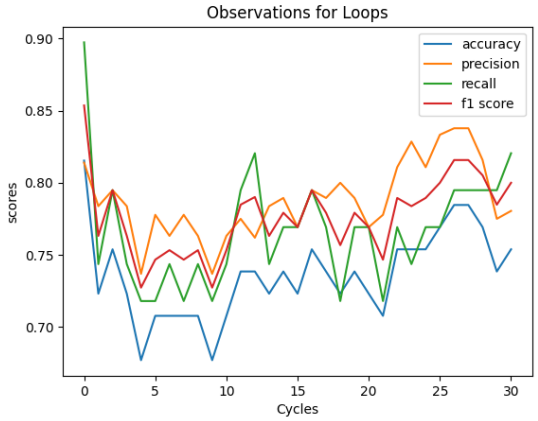
Placing a high threshold allows us to only pick or label those unlabelled instances about which we require high confidence from the algorithm itself. It also provides a benefit by restricting the label propagation to an early stop rather than exhausting all of the cycles. From the label wise precision and recall plots included for the threshold score of +0.85, we can see that the label propagation stops for several labels at an earlier point or cycle as compared to previously.

This increases the speed of the method as there is an early stopping now with the exhaustion of cycles no longer required. This results in a speedup in the training phase of PRIORITY. While a few of the labels showed no improvement, several of the labels showed minor improvements, as can be compared via the plots.

The threshold of +0.85 is obtained after thorough experimentation and fulfilment of certain criteria, which are:

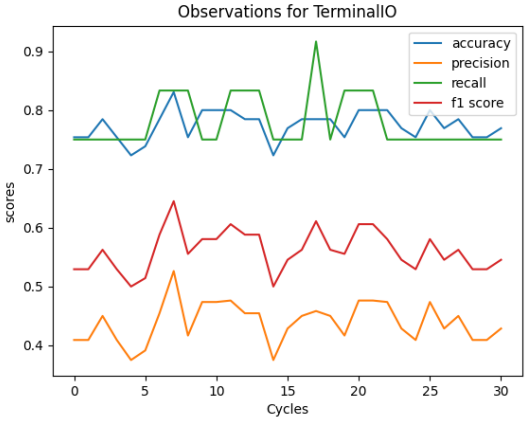


(a) Label Arithmetic

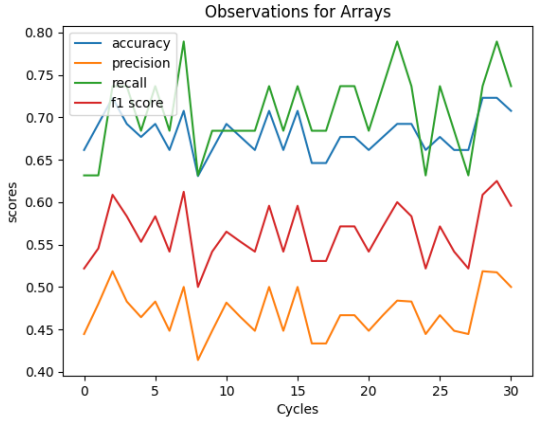


(b) Label Loops

Figure 3.1: Plots for labels Arithmetic and Loops without using confidence scores



(a) Label Terminal IO



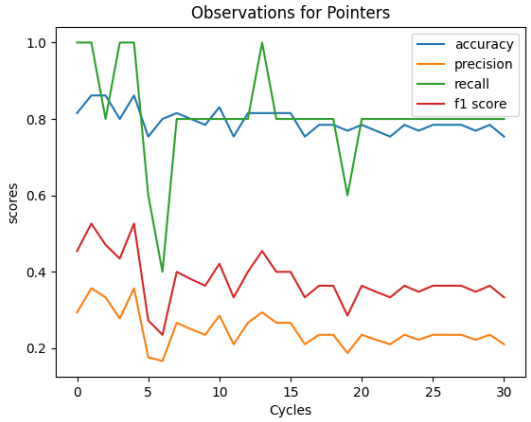
(b) Label Arrays

Figure 3.2: Plots for labels Terminal IO and Arrays without using confidence scores

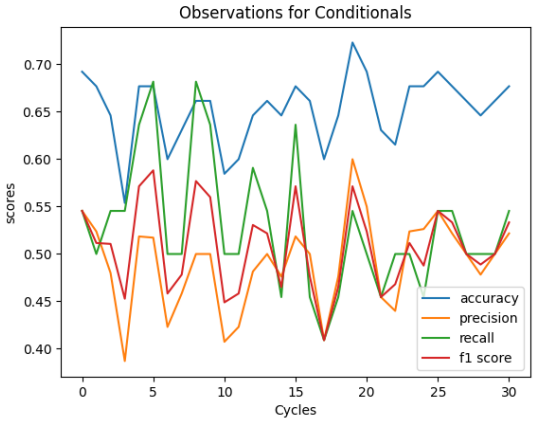
- High confidence score to ensure a concrete prediction by the classifier.
- Sufficiently low threshold value to observe a significant number of cycles.
- Improvement in precision.
- Improvement in recall.

3.2 Data Curation:

The task is to create a new dataset. We can put to use everything in the dataset that priority stores. Priority has been designed to gather/collect the feedback obtained from the users. Feedback can be of different forms and different types. The feedback is composed of two methods.

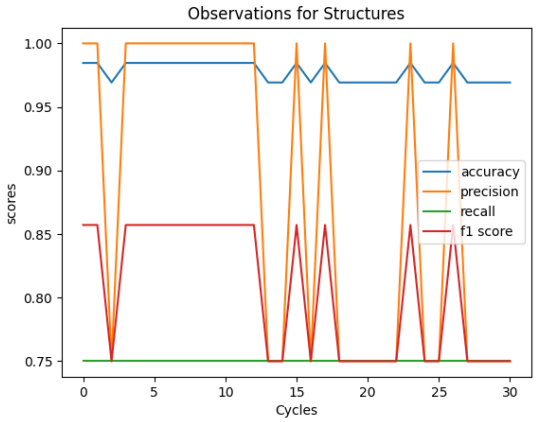


(a) Label Pointers

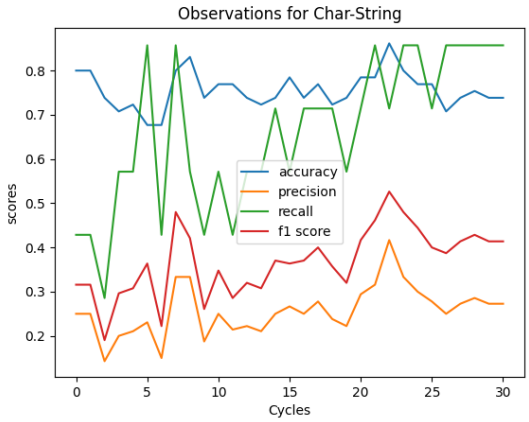


(b) Label Conditionals

Figure 3.3: Plots for labels Pointers and Conditionals without using confidence scores

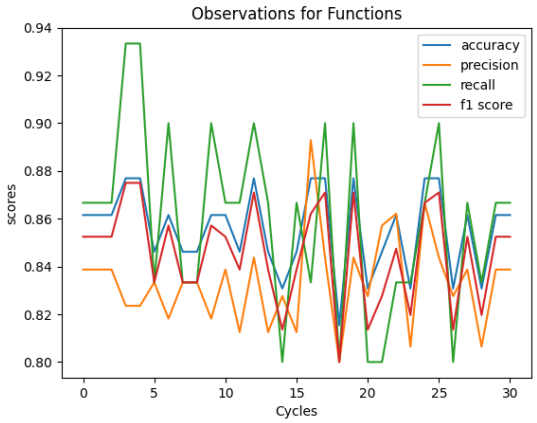


(a) Label Structures

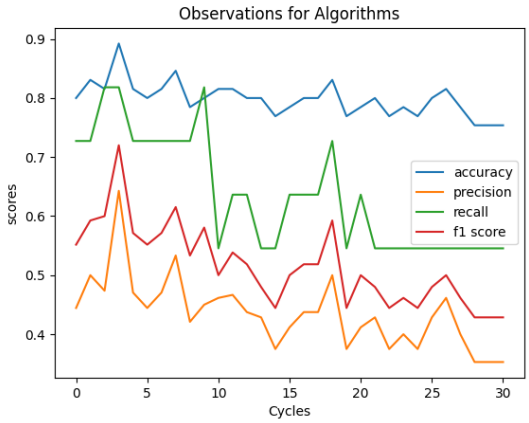


(b) Label Char-string

Figure 3.4: Plots for labels Structures and Char-string without using confidence scores

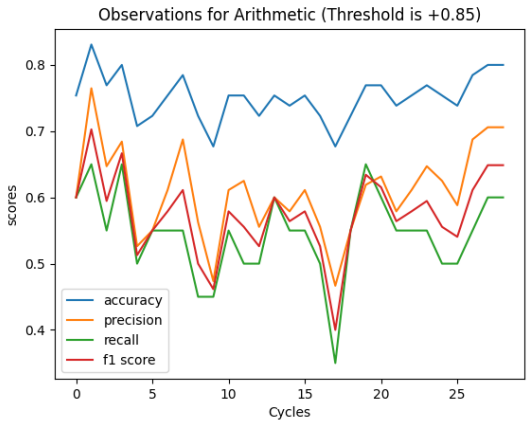


(a) Label Functions

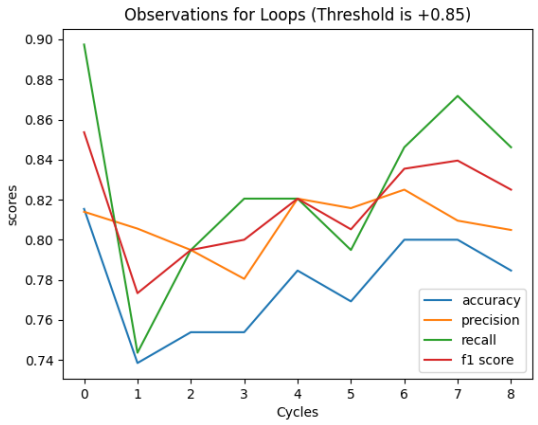


(b) Label Algorithms

Figure 3.5: Plots for labels Functions and Algorithms without using confidence scores

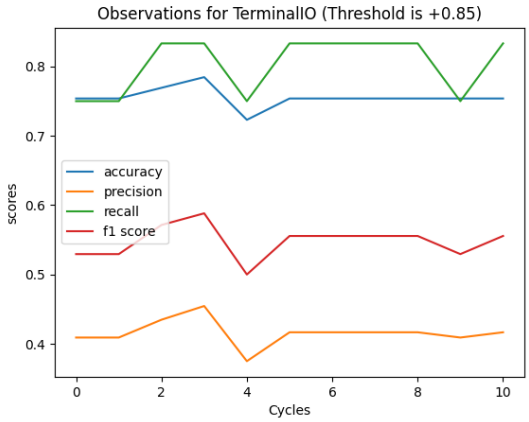


(a) Label Arithmetic

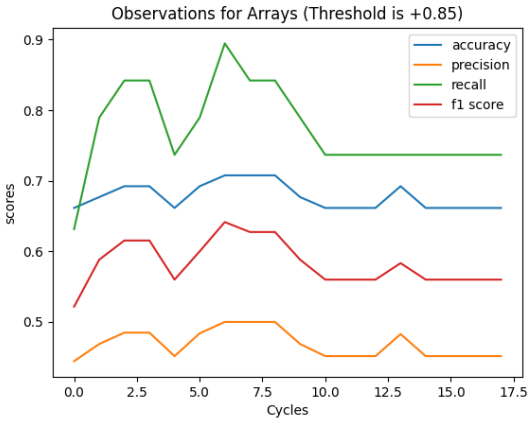


(b) Label Loops

Figure 3.6: Plots for labels Arithmetic and Loops using confidence scores

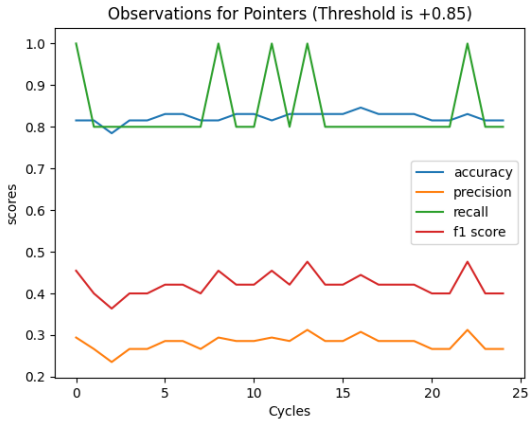


(a) Label Terminal IO

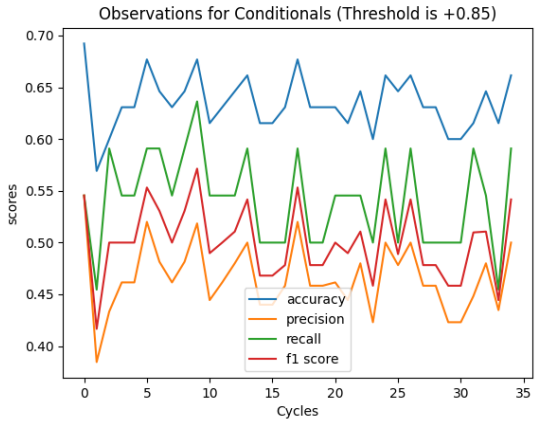


(b) Label Arrays

Figure 3.7: Plots for labels Terminal IO and Arrays using confidence scores

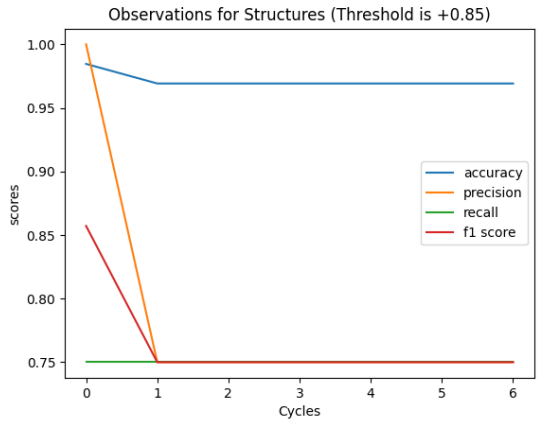


(a) Label Pointers

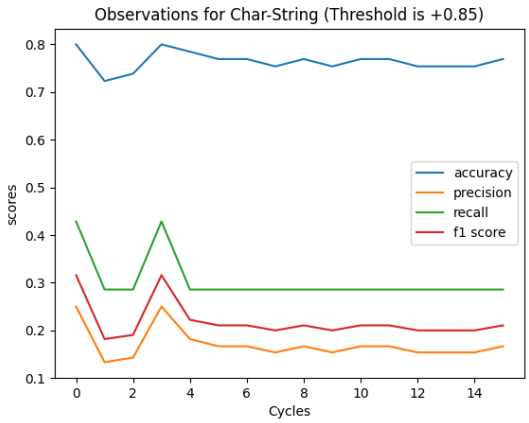


(b) Label Conditionals

Figure 3.8: Plots for labels Pointers and Conditionals using confidence scores

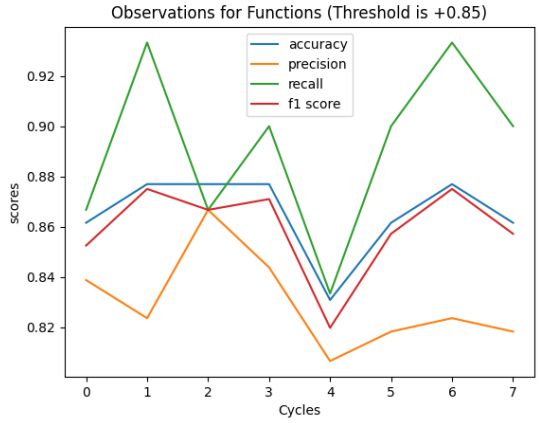


(a) Label Structures

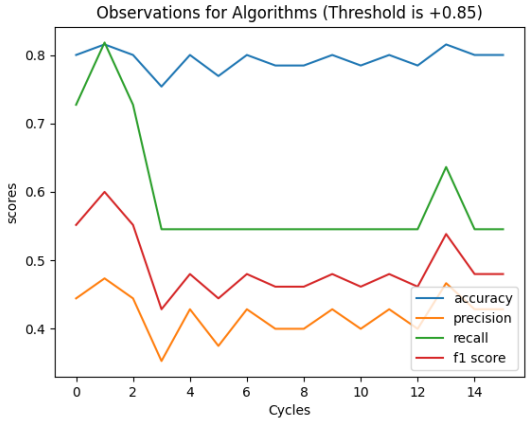


(b) Label Char-string

Figure 3.9: Plots for labels Structures and Char-string using confidence scores



(a) Label Functions



(b) Label Algorithms

Figure 3.10: Plots for labels Functions and Algorithms using confidence scores

- Implicit feedback (a weak form of feedback): Such feedback is not mentioned but taken by monitoring user activity. A lot of companies like Google, Facebook etc. employ this in their product nowadays.
- Explicit feedback: Includes mentioned feedback consisting of comments, re-annotations, user ratings or star ratings.

The purpose of creating a new dataset or modifying an existing dataset is to use the acquired information (the feedback) to improve the pre-existing data source. The newly formed dataset can also be used for the new classifiers to run and train on them. The modification or building of a new dataset also allows us to compare the model used in the previous work [6] [10] and the new models we designed and built.

As opposed to the previous approach [10] of performing label propagation (which is a semi-supervised machine learning approach), which yielded minimal benefits as the approach is highly

prone to mislabelling the unlabelled data and thus, in turn, affects the overall functioning and predictive power of the model, the approach tried here takes into account the concrete feedback provided by the users and then learns and predicts accordingly.

Let's look at the feedback stored by Priority for each of the forms.

3.2.1 Implicit Feedback

Priority stores implicit feedback in a table called 'clicks'. After a close inspection of the table, we discovered that there were around 500 entries in the table. Some of the important fields in the table are:

- Start time: The time the user opened the given problem and started looking at it.
- End time: The time at which the user finishes looking at the given problem and either switches to another problem or selects the problem for further use.
- Query: This states the query or the labels or the problem attributes the user specified for searching for a problem.
- Problem id: The id associated with the problem that is browsed by the user.
- Category: A categorical variable. Upon further inspection, it consists of three categories:
  - Navigate: The user navigated to some other problem or back to the main page and decided to not go with the current problem.
  - Template copied: The user copied the problem template. Further inspection is required to ensure whether the problem suggestion was found to be useful by the user.
  - Solution copied: The user decided to go with the currently selected problem and copied the problem's solution for further use.
  - Statement copied: The user decided to go with the currently selected problem but only copied the problem statement.

The problems have been classified into different categories to get a more thorough insight to the user actions based upon what is suggested to them.

The end_time and start_time parameters are used to calculate dwell time. Dwell time is the amount of time a user spends observing a given problem. It is a value regularly used in 'Search Engine Optimisation and can also be used here. A small dwell time implies that the user didn't find the given problem helpful and switched to other page, while an extended dwell time can be inconclusive, as it can mean that the user might be away from the keyboard, using their phone, or something else. We will have a more detailed look at dwell time in a later section.

3.2.2 Explicit Feedback

The different forms of explicit feedback are stored in different tables. The tables are as follows:

- Feedback
- Feedback_rating
- Text_feedback

3.2.2.1 Feedback

This table consists of the re-annotations or modifications to the problem labels as suggested by the user based on their judgement. The tables consist of 'problem_id', and all the labels that PRIORITY can assign to a given problem.

The row values that can be entered consist of binary values. i.e., 0 and 1. This can be thought of as a switch as well, with 0 indicating the switch is off and 1 indicating the switch is on. The labels with entry 1 have been suggested by the user as the labels suitable for the problem, while the label with entry 0 has been indicated by the user as unsuitable for the problem.

The suggestions made here can be used to modify the existing dataset for improvements.

3.2.2.2 Feedback Rating

This table consists of the user rating given by a user to the suggestions provided by PRIORITY as per the query specified by the user. The rating goes on a scale of 1 - 5, with 5 being the highest. The ratings reflect the goodness or badness of the suggestion by PRIORITY and thus can be considered while providing suggestions to the users. The table consisted of only a few entries and thus was not further accounted for while developing the new dataset.

3.2.2.3 Text Feedback

Feedback in the form of comments is stored within this table. Users are provided with a textbox where they can input any comment about the suggestion offered by PRIORITY webiste. Upon close inspection of the table, it was observed that it only consisted of a few entries, several of which were dummy entries to check the functioning of the feedback. Thus, this was also not pursued any further.

Only the 'Feedback' table was used to make changes to the pre-existing labels per the user's suggestions according to the 'feedback' table. The rest of the tables were ignored due to the small amount of data present in them (most of which was dummy data).

3.2.3 Template Copied

The category "template_copied" implies that the user copied the template of the provided suggestion for the problem type searched by the user. This does not ensure if the suggestion proved

useful for the user as the template can be generic and thus could have been copied by the user for some other purpose.

The first step was to check if there was any overlap between the “template_copied”, “statement_copied” and “solution_copied”. If any “template_copied” instance was also classified as one of the latter two, that would mean the suggestion would be useful. Overlappings were checked by checking for problem ID, as each problem is assigned a unique problem ID. We found out that there are no overlaps between any of the categories.

The next step was to inspect the templates of the “template_copied” instances. The number of “template_copied” instances was small, and thus we decided to go with manual inspection. Upon inspection, we discovered that the templates were not generic and were unique to the problem statement, implying that the suggested problem was useful to the user. Thus, we decided to classify the “template_copied” instances as instances that might have been found to be useful by the user. Further criteria were required to concretize if the provided suggestions were found to be useful or not.

3.2.4 Dwell Time

The time a user spends on a selected problem is called the dwell time. Dwell time helps us to get a concrete opinion on whether the problem suggestion provided by PRIORITY is helpful or not for cases where it has not been explicitly mentioned by the user. Thus, implicit feedback storage is used here.

As stated earlier, the "clicks" table contains a " category " field. The field specifies the action performed by the user. By combining the dwell time and the category field, we can obtain information on whether the provided suggestion was useful or not. Take a look again at the "category" column:

- Template_copied
- Statement_copied
- Solution_copied
- Navigate

By their definitions, “template_copied”, “statement_copied”, and “solution_copied” all imply that the user found the suggestion to be useful, and thus we classify such instances as “Hard Yes”. For instance, with the category “navigate”, dwell time comes into play. It is possible that some “navigate” instances are of use while others can simply be ignored. By fixing a time range or interval of the dwell time, the useful instances of the navigate categories can be picked out and thus can be classified as “Soft Yes” meaning that there are chances that the provided suggestion was useful for the user, but we cannot ensure that.

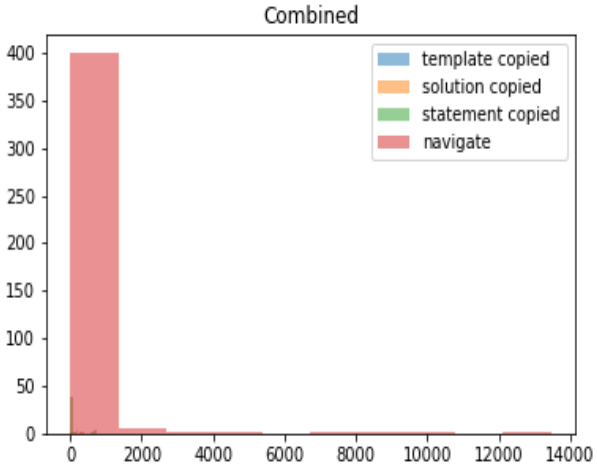
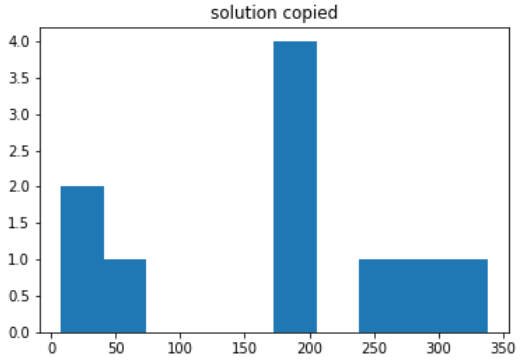
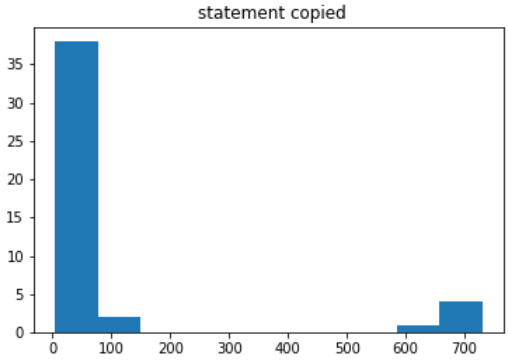


Figure 3.11: Combined plot of navigate, template_copied, solution_copied, statement_copied



(a) Histogram plot for solution_copied instances



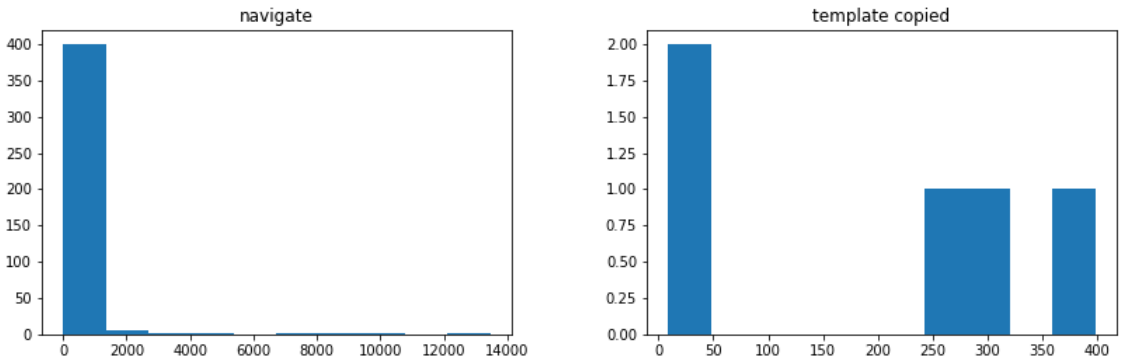
(b) Histogram plot for statement_copied instances

Figure 3.12: Histogram plot for solution_copied and statement_copied instances

We plotted histograms for each of the categories. The histogram was plotted for dwell time. The X-axis contained bins, while the Y-axis contained the number of instances for each dwell time. The purpose was to observe the overlapping between the plots of the different categories with the “navigate” category. The overlapping time intervals will give the time range for the dwell time in the “navigate” category. All of the instances thus falling within the given time interval for “navigate” can be classified as “Weak Yes”.

As we can observe from the plots, the number of instances with the category “navigate” outnumbers the rest of the categories. This results in difficulty inspecting the time intervals for overlapping between the two sets of categories as the overlapping histogram is challenging to interpret. Thus, a different methodology to check for overlapping intervals is required.

An alternative tried for this was to place different threshold limits over the dwell time for the “navigate” instances to take a reasonable time range and also to simmer down the number of cases and then re-try for the histograms. Different threshold values were tried, which can be seen in the



(a) Histogram plot for navigate instances

(b) Histogram plot for template_copied instances

Figure 3.13: Histogram plot for navigate and template_copied instances

table below:

Threshold value (in seconds)	Number of entries
No threshold	415
700	387
500	380
100	344
50	302
30	256
20	222
10	143
5	57

Table 3.1: Number of entries with category as "navigate" corresponding to different threshold values.

Even with a small threshold limit, “navigate” still has significantly more instances than the other three categories, which will still prove difficult to interpret from the combined histogram. Thus, an alternative method to plotting histograms and inspecting them was required.

3.2.5 Gamma Distribution

We moved on to the idea of finding a time interval via distribution after the drawbacks of the histogram method. The first step was to select the distribution, summarize, and explain the data corresponding to the dwell time for each category instance. In machine learning, instead of storing all of the points, we can explain the points via a distribution that is represented by the mean and standard deviation.

The Gaussian distribution [1], which is a very popular distribution, was our first choice. However, if the data can never have negative values, then Gaussian is not a good summary since the summary by a Gaussian will say that the data has some non-zero probability of being negative as well, whereas, in reality, we know that the data can never ever be negative. As we are trying to fix a distribution over the dwell time, there won’t be any negative values in the data. This resulted in

us searching for alternatives that suit our use case and criteria.

Gamma distribution proved to be the perfect candidate. It is used to explain/summarize quantities like time duration (dwell time in our case), which never take any negative values and thus, as per its structure, will show the probability of data being negative as zero.

The gamma distribution provides an interval function support [8] which allows us to find out the interval (time interval in our case) after specifying a “confidence” parameter value. The parameter enables us to specify a percentile value. For example, we used a confidence value of 0.9, implying that we are looking for an interval range containing the 90 percentile of the data. This allows us to eliminate instances corresponding to small and large dwell time values because, as stated earlier, these values do not help us draw valuable conclusions from the user activity.

For all four categories, first, the gamma distribution is fitted and then, using the interval function, a confidence interval or time interval is found containing about 90 percentile of the data values. The time interval enables us to perform classification of the “navigate” category data into “Weak Yes”.

We inspected the obtained interval values for each of them and observed that the interval for the category “template_copied” was entirely inside of “solution_copied” and “statement_copied”. Thus, “template_copied” was ignored moving forward. We proceeded next with what we called "Discrete Labelling."

A choice was in front of us with the confidence intervals of the “statement_copied” and “solution_copied” categories. We decided to proceed with the union of the intervals as it would give us a more extensive time range to work with. Any instances of the “navigate” category with a dwell time lying in the new unionized interval were classified as “Weak Yes”, while the other instances not satisfying the criteria were not classified into any category and were left unclassified.

A note here is that we could have taken the intersection of the intervals as well, but that would have resulted in a narrower interval and restricted us from incorporating as much user feedback as possible.

3.2.6 “Hard Yes” and “Weak Yes”

As stated before, the instances belonging to the categories: "template_copied", "statement_copied", "solution_copied" were all classified as "Hard Yes" due to them being useful for the user. The instances of the category "navigate" are classified as "Weak Yes" that have overlapping dwell time intervals with the above three categories (found with the help of gamma distribution).

The purpose of classifying the instances as “Hard Yes” and “Weak Yes” is to assign each instance a suitable weight. The weightage will act as importance or significance that is given to

each of the classified instances. For this, we used instance weightage instead of class weightage, as weighing by instances allows us to go instance by instance. We gave "Hard Yes" classified instances a weight of 1, indicative that such instances are of concrete usage to us, while we performed several experiments with different weights for the "Weak Yes" instances. The instances classified as neither "Hard Yes" nor "Weak Yes", were assigned a weight of 0.

The challenge was to create or split the dataset into training and testing sets. The training set needs to consist of labelled and unlabelled data instances, while the testing can only consist of labelled data instances. This meant avoiding the random splitting of the dataset and manually splitting it. From the previous iterations of PRIORIRTY [10] [6], we discovered that a ratio of 3:1 works pretty well, with 3 instances of each label going to the training set and 1 instance of the same label going to the test set.

Another reason for manual splitting is that we specify the sample weights for each instance in our training set. A random split would mean a random assignment of weights to the instances irrespective of them being classified as "Hard Yes" or "Weak Yes". Though manual splitting does reduce the randomness of the splitting, it is required for the reasons stated above.

3.2.7 Re-annotations

The re-annotations or modifications suggested or offered by the user to PRIORITY were also accommodated in the new dataset. The "Weak Yes" instances were not added to the test set. For the test set, we cannot have labelled data instances we are unsure of; thus, the "Weak Yes" instances are avoided.

They are added to the training set to improve the data and add to it over which the model learns. If the problem was previously found to be unlabelled, it is labelled using the suggested annotations or labels suggested by the user (recorded in the feedback table). And if the problem is already labelled, then a duplicate of the problem with the suggested labels by the user is added to the dataset. This results in the same problem having multiple entries in the dataset but with different problem labels assigned to them.

As "Hard Yes" instances are more about concrete feedback, they are accommodated in both the training and test set splits. If a problem classified as "Hard Yes" is found to be present in the test set, then it is duplicated with the user-suggested labels. While if a problem belonging to the training set has been classified as a "Hard Yes" instance by us, it will follow the same route as the "Weak Yes" classified instances, i.e., if it is already labelled, then we will duplicate it with the suggested labels by the user, and if it is not labelled, then the problem is labelled with the suggested labels.

This method also allowed us to increase the count of labelled instances while the number of unlabelled instances either remaining the same or decreasing.

A point to note is that, following the format of the previous iterations of Priority [10] [6], the labels have been added in the string format for ease of representation and use.

(labels for each of the instances (if they are labelled) are represented as a list. Whereas, the unlabelled instances, are either represented by “NaN”, an empty list, i.e., [] or by “” indicating the same)

3.3 Comparison : Old Dataset vs. New Dataset

As per[6] [10] Priority uses BalancedBaggingClassifier, part of the Imblearn library [7]. While the previous iteration focused on label propagation via both probability [10] and confidence score, the newly curated dataset focuses on accommodating user feedback, which in theory should prove to be a much more reliable method than the former ones. To test this out, the same model was trained and tested on both the older dataset and the new dataset. As we know from [10] Priority labels any of the problems in its dataset to 11 major labels. The metrics for comparison are the exact same which have been used by [6] [10] for reporting the results in their findings. This is done to ease comparison and to keep the comparison fair as well. The overall results as well as the label wise results have been shown below:

New Dataset -

Hamming own 0.79
Hamming np 0.20
Ones acc 0.44
TP 76 TN 451 FP 37 FN 96
Total Positives 172.0
Precision 0.67
Recall 0.44

Old Dataset -

Hamming own 0.79
Hamming np 0.20
First 0.80
Ones acc 0.66
TP 115 TN 412 FP 76 FN 57
Total Positives 172.0
Precision 0.60
Recall 0.66

TP : True Positive(s) TN : True Negatives(s) FP : False Positive(s) FN : False Negative(s)

Labels	Metric	New Dataset	Old Dataset
Arithmetic	Precision	0.58	0.64
	Recall	0.33	0.52
Loops	Precision	0.75	0.78
	Recall	0.53	0.66
TerminalIO	Precision	0.45	0.47
	Recall	0.41	0.66
Arrays	Precision	0.66	0.51
	Recall	0.63	0.89
Pointers	Precision	0.33	0.28
	Recall	0.4	0.8
Conditionals	Precision	0.6	0.625
	Recall	0.40	0.45
Structures	Precision	1.0	0.75
	Recall	0.4	0.6
Char-string	Precision	0.75	0.4
	Recall	0.42	0.57
Functions	Precision	0.92	0.80
	Recall	0.41	0.80
Algorithms	Precision	0.66	0.43
	Recall	0.18	0.63

From the results, we inferred that, the model produced a higher precision but a lower recall on the new dataset. This implies that the model is conservative in making the predictions as the recall is low. The positives classified are fewer in number, but a good chunk of them are correct.

For the old dataset, we observed a lower precision but a higher recall value, indicating that the model was much more liberal in making its predictions. It made more positives but got fewer of them correct.

The conservative nature of the model could be attributed to the elimination of the label propogation and focusing solely on user feedback. Label propagation introduced an element of learning for the model in which it labelled the unlabelled problems in the dataset. This allowed the model to be much more liberal with its predictions.

3.4 Bagging Classifier

As our methodology requires sample weights to be assigned to each of the instances based upon their classification, which we performed earlier, we required instance based weighing rather than class based. The Balanced-Bagging-Classifier does not provide such functionality, which means looking for alternatives but with the constraint of similar functionality to Balanced-Bagging-Classifier which could enable us to compare our results and findings.

Bagging-Classifier [9] is a classifier part of the sklearn library largely resembling the structure of Balanced-Bagging-Classifier. Decision Trees as base estimators were used for both of the classifiers. Unlike the Balanced-Bagging-Classifier, Bagging-Classifier has the fit function. The fit function allows us to specify an array called "sample_weights" which contains the weights instance-wise for the training input values.

We experimented with the different weights (assigned to the "Weak Yes") and observed the outputs in terms of precision, recall and f-score for each of the label and overall as well with the objective of comparing the performance with the previous works and also to find out the weights for which a balanced precision-recall value can be found.

3.5 Precision and Recall

We incremented the instance weights from 0.5 to 1 using Bagging-Classifier as the model. The reason behind this increment was to make sure that the newly curated dataset diverges from the old dataset. If the "Weak Yes" are assigned with less and less weights, then this would mean focusing mostly on the instances which were also part of the older dataset. Keeping this in mind, we changed the model weights and made the required modifications to the model and recorded the results:

Weight for the soft yes instances taken : 0.55

Labels	**Total TP TN FP FN**	**Precision**	**Recall**
Arithmetic	21.0 10 44 1 11	0.90	0.47
Loops	39.0 21 21 6 18	0.77	0.53
TerminalIO	12.0 4 48 6 8	0.4	0.33
Arrays	19.0 12 36 11 7	0.52	0.63
Pointers	5.0 2 58 3 3	0.4	0.4
Conditionals	22.0 11 38 6 11	0.5	0.64
Structures	5.0 2 61 0 3	1.0	0.4
Char-String	7.0 4 57 2 3	0.66	0.57
Functions	31.0 18 35 0 13	1.0	0.58
Algorithms	11.0 2 54 1 9	0.66	0.18

Table 3.2: Label wise results for dynamic weights on taking the average for weight = 0.55

Hamming own 0.81
Hamming np 0.18
Ones acc 0.5
TP 86 TN 452 FP 36 FN 86
Total Postives 172.0
Overall Prec 0.70
Overall Recall 0.5

Weight for the soft yes instances taken : 0.6

Labels	Total TP TN FP FN	Precision	Recall
Arithmetic	21.0 9 42 3 12	0.75	0.42
Loops	39.0 22 21 6 17	0.78	0.56
TerminalIO	12.0 4 49 5 8	0.44	0.33
Arrays	19.0 11 41 6 8	0.64	0.57
Pointers	5.0 2 57 4 3	0.33	0.4
Conditionals	22.0 9 40 4 13	0.69	0.40
Structures	5.0 2 60 1 3	0.66	0.4
Char-String	7.0 2 58 1 5	0.66	0.28
Functions	31.0 17 33 2 14	0.89	0.54
Algorithms	11.0 1 55 0 10	1.0	0.09

Table 3.3: Label wise results for dynamic weights on taking the average for weight = 0.6

Hamming own 0.81

Hamming np 0.18

Ones acc 0.45

TP 79 TN 456 FP 32 FN 93

Total Positives 172.0

Overall Prec 0.71

Overall Recall 0.45

Weight for the soft yes instances taken : 0.65

Labels	Total TP TN FP FN	Precision	Recall
Arithmetic	21.0 8 42 3 13	0.72	0.38
Loops	39.0 21 20 7 18	0.75	0.53
TerminalIO	12.0 5 49 5 7	0.5	0.41
Arrays	19.0 9 38 9 10	0.5	0.47
Pointers	5.0 1 57 4 4	0.2	0.2
Conditionals	22.0 7 38 6 15	0.53	0.31
Structures	5.0 2 61 0 3	1.0	0.4
Char-String	7.0 4 59 0 3	1.0	0.57
Functions	31.0 17 35 0 14	1.0	0.54
Algorithms	11.0 1 55 0 10	1.0	0.09

Table 3.4: Label wise results for dynamic weights on taking the average for weight = 0.65

Hamming own 0.80

Hamming np 0.19

Ones acc 0.436046511627907

TP 75 TN 454 FP 34 FN 97

Total Positives 172.0

Overall Prec 0.68

Overall Recall 0.43

Weight for the soft yes instances taken : 0.7

Labels	Total TP TN FP FN	Precision	Recall
Arithmetic	21.0 7 44 1 14	0.87	0.33
Loops	39.0 22 21 6 17	0.78	0.56
TerminalIO	12.0 6 52 2 6	0.75	0.5
Arrays	19.0 11 40 7 8	0.61	0.57
Pointers	5.0 3 55 6 2	0.33	0.6
Conditionals	22.0 8 39 5 14	0.61	0.36
Structures	5.0 2 61 0 3	1.0	0.4
Char-String	7.0 3 58 1 4	0.75	0.42
Functions	31.0 16 33 2 15	0.88	0.51
Algorithms	11.0 2 55 0 9	1.0	0.18

Table 3.5: Label wise results for dynamic weights on taking the average for weight = 0.7

Hamming own 0.81

Hamming np 0.18

Ones acc 0.46

TP 80 TN 458 FP 30 FN 92

Total Positives 172.0

Overall Prec 0.7272727272727273

Overall Recall 0.46511627906976744

Weight for the soft yes instances taken : 0.8

Labels	Total TP TN FP FN	Precision	Recall
Arithmetic	21.0 7 43 2 14	0.77	0.33
Loops	39.0 21 21 6 18	0.77	0.53
TerminalIO	12.0 7 49 5 5	0.58	0.58
Arrays	19.0 9 39 8 10	0.52	0.47
Pointers	5.0 2 57 4 3	0.33	0.4
Conditionals	22.0 6 36 8 16	0.42	0.27
Structures	5.0 2 60 1 3	0.66	0.4
Char-String	7.0 4 56 3 3	0.57	0.57
Functions	31.0 16 34 1 15	0.94	0.51
Algorithms	11.0 2 55 0 9	1.0	0.1818

Table 3.6: Label wise results for dynamic weights on taking the average for weight = 0.8

Hamming own 0.79

Hamming np 0.20

Ones acc 0.44

TP 76 TN 450 FP 38 FN 96

Total Positives 172.0

Overall Prec 0.66

Overall Recall 0.44

Weight for the soft yes instances taken : 0.9

Labels	Total TP TN FP FN	Precision	Recall
Arithmetic	21.0 11 43 2 10	0.84	0.52
Loops	39.0 23 20 7 16	0.76	0.58
TerminalIO	12.0 4 53 1 8	0.8	0.33
Arrays	19.0 9 38 9 10	0.5	0.47
Pointers	5.0 4 57 4 1	0.5	0.8
Conditionals	22.0 8 37 7 14	0.53	0.36
Structures	5.0 3 61 0 2	1.0	0.6
Char-String	7.0 3 56 3 4	0.5	0.42
Functions	31.0 16 33 2 15	0.88	0.51
Algorithms	11.0 3 53 2 8	0.6	0.27

Table 3.7: Label wise results for dynamic weights on taking the average for weight = 0.9

Hamming own 0.81

Hamming np 0.18

Ones acc 0.48

TP 84 TN 451 FP 37 FN 88

Total Positives 172.0

Overall Prec 0.69

Overall Recall 0.48

Weight for the soft yes instances taken : 1

Labels	Total TP TN FP FN	Precision	Recall
Arithmetic	21.0 7 44 1 14	0.87	0.33
Loops	39.0 21 22 5 18	0.80	0.53
TerminalIO	12.0 8 51 3 4	0.72	0.66
Arrays	19.0 11 36 11 8	0.5	0.57
Pointers	5.0 3 57 4 2	0.42	0.6
Conditionals	22.0 6 39 5 16	0.54	0.27
Structures	5.0 3 61 0 2	1.0	0.6
Char-String	7.0 3 57 2 4	0.6	0.42
Functions	31.0 18 33 2 13	0.9	0.58
Algorithms	11.0 3 53 2 8	0.6	0.27

Table 3.8: Label wise results for dynamic weights on taking the average for weight = 1

Hamming own 0.81

Hamming np 0.18

Ones acc 0.48

TP 83 TN 453 FP 35 FN 89
Total Positives 172.0
Overall Prec 0.70
Overall Recall 0.48

As evident from the results above, there is not much change in recall with an increase in weight, but there is a significant change in precision with the change in weight. The same model at different weights can be thought of as different models. Even though a few models did not produce good enough overall precision-recall, some labels did get good scores for these models, so saving them for future reference can be helpful.

We also plotted the label-wise precision-recall with the varying weights as well as the overall precision-recall with the weights, which can be seen later on in this section.

At first glance, the "Weak Yes" doesn't seem trustworthy because of the high precision and recall at the lower weights. To verify this, we calculated and plotted the precision-recall for the lower weights starting from 0 to 0.5 . The collected results can be seen below:

Weight for the soft yes instances taken : 0

Labels	Total TP TN FP FN	Precision	Recall
Arithmetic	21.0 7 45 0 14	1.0	0.33
Loops	39.0 27 18 9 12	0.75	0.69
TerminalIO	12.0 4 47 7 8	0.36	0.33
Arrays	19.0 8 38 9 11	0.47	0.42
Pointers	5.0 2 58 3 3	0.4	0.4
Conditionals	22.0 11 33 11 11	0.5	0.5
Structures	5.0 1 61 0 4	1.0	0.2
Char-String	7.0 4 58 1 3	0.86	0.61
Functions	31.0 19 32 3 12	0.9	0.58
Algorithms	11.0 3 54 1 8	0.75	0.27

Table 3.9: Label wise results for dynamic weights on taking the average for weight = 0

Hamming own 0.80
Hamming np 0.19
Ones acc 0.5
TP 86 TN 444 FP 44 FN 86
Total Positives 172.0
Overall Prec 0.66
Overall Recall 0.5

Weight for the soft yes instances taken : 0.1

Hamming own 0.8

Labels	Total TP TN FP FN	Precision	Recall
Arithmetic	21.0 10 43 2 11	0.83	0.47
Loops	39.0 23 19 8 16	0.74	0.58
TerminalIO	12.0 6 49 5 6	0.54	0.5
Arrays	19.0 9 39 8 10	0.52	0.47
Pointers	5.0 2 58 3 3	0.4	0.4
Conditionals	22.0 7 35 9 15	0.43	0.31
Structures	5.0 2 61 0 3	1.0	0.4
Char-String	7.0 3 56 3 4	0.5	0.42
Functions	31.0 14 35 0 17	1.0	0.45
Algorithms	11.0 2 55 0 9	1.0	0.18

Table 3.10: Label wise results for dynamic weights on taking the average for weight = 0.1

Hamming np 0.2

Ones acc 0.45

TP 78 TN 450 FP 38 FN 94

Total Positives 172.0

Overall Prec 0.67

Overall Recall 0.45

Weight for the soft yes instances taken : 0.2

Labels	Total TP TN FP FN	Precision	Recall
Arithmetic	21.0 8 44 1 13	0.88	0.38
Loops	39.0 21 19 8 18	0.72	0.53
TerminalIO	12.0 6 49 5 6	0.54	0.5
Arrays	19.0 9 39 8 10	0.58	0.52
Pointers	5.0 4 57 4 1	0.5	0.8
Conditionals	22.0 6 38 6 16	0.5	0.27
Structures	5.0 2 61 0 3	1.0	0.4
Char-String	7.0 2 56 3 5	0.4	0.28
Functions	31.0 16 35 0 15	1.0	0.51
Algorithms	11.0 1 54 1 10	0.5	0.09

Table 3.11: Label wise results for dynamic weights on taking the average for weight = 0.2

Hamming own 0.80

Hamming np 0.19

Ones acc 0.44

TP 76 TN 453 FP 35 FN 96

Total Positives 172.0

Overall Prec 0.68

Overall Recall 0.44

Weight for the soft yes instances taken : 0.3

Labels	Total TP TN FP FN	Precision	Recall
Arithmetic	21.0 10 42 3 11	0.76	0.47
Loops	39.0 26 22 5 13	0.83	0.66
TerminalIO	12.0 6 52 2 6	0.75	0.5
Arrays	19.0 7 38 9 12	0.43	0.36
Pointers	5.0 3 58 3 2	0.5	0.6
Conditionals	22.0 7 39 5 15	0.58	0.31
Structures	5.0 1 61 0 4	1.0	0.2
Char-String	7.0 4 58 1 3	0.8	0.57
Functions	31.0 16 34 1 15	0.94	0.51
Algorithms	11.0 1 54 1 10	0.5	0.09

Table 3.12: Label wise results for dynamic weights on taking the average for weight = 0.4

Hamming own 0.81

Hamming np 0.18

Ones acc 0.47

TP 81 TN 458 FP 30 FN 91

Total Positives 172.0

Overall Prec 0.72

Overall Recall 0.47

Weight for the soft yes instances taken : 0.4

Labels	Total TP TN FP FN	Precision	Recall
Arithmetic	21.0 8 44 1 13	0.88	0.38
Loops	39.0 23 23 4 16	0.85	0.58
TerminalIO	12.0 5 51 3 7	0.625	0.41
Arrays	19.0 12 42 5 7	0.70	0.63
Pointers	5.0 2 56 5 3	0.28	0.4
Conditionals	22.0 4 39 5 18	0.44	0.18
Structures	5.0 2 61 0 3	1.0	0.4
Char-String	7.0 3 59 0 4	1.0	0.42
Functions	31.0 18 35 0 13	1.0	0.58
Algorithms	11.0 2 55 0 9	1.0	0.18

Table 3.13: Label wise results for dynamic weights on taking the average for weight = 0.3

Hamming own 0.82

Hamming np 0.17

Ones acc 0.45

TP 79 TN 465 FP 23 FN 93

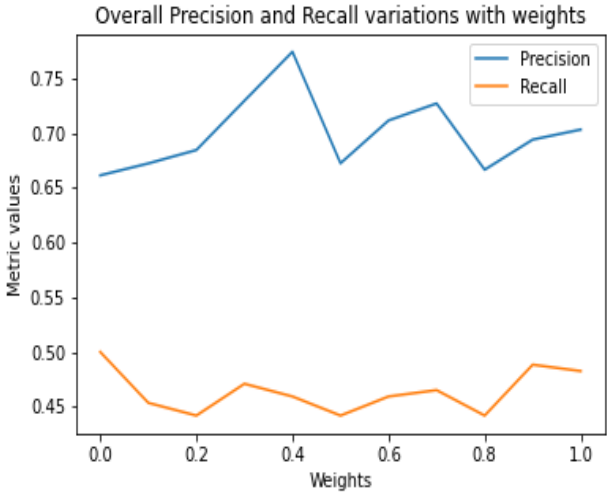


Figure 3.14: Overall precision-recall vs weight variation

total Positives 172.0
Overall Prec 0.77
Overall Recall 0.45

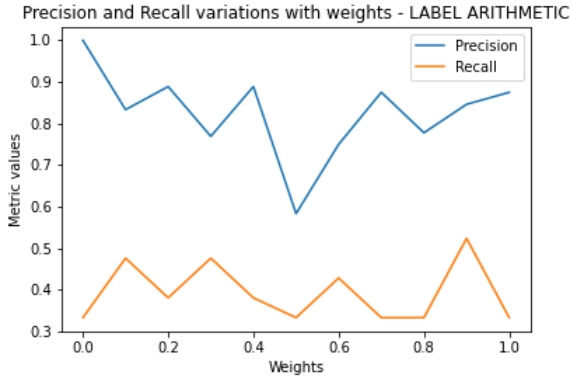
Weight for the soft yes instances taken : 0.45

Labels	Total TP TN FP FN	Precision	Recall
Arithmetic	21.0 10 43 2 11	0.83	0.47
Loops	39.0 22 22 5 17	0.81	0.56
TerminalIO	12.0 5 50 4 7	0.55	0.41
Arrays	19.0 13 38 9 6	0.59	0.68
Pointers	5.0 2 57 4 3	0.33	0.4
Conditionals	22.0 7 41 3 15	0.70	0.31
Structures	5.0 3 61 0 2	1.0	0.6
Char-String	7.0 4 59 0 3	1.0	0.57
Functions	31.0 17 33 2 14	0.89	0.54
Algorithms	11.0 3 54 1 8	0.75	0.27

Table 3.14: Label wise results for dynamic weights on taking the average for weight = 0.45

Hamming own 0.82
Hamming np 0.17
Ones acc 0.5
TP 86 TN 458 FP 30 FN 86
Total Positives 172.0
Overall Prec 0.74
Overall Recall 0.5

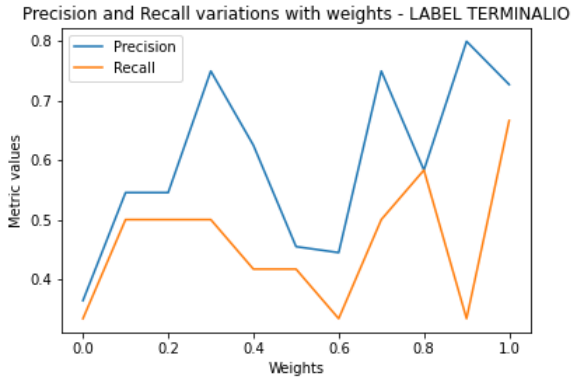
From the graphs we can observe that the modifications we did for the construction of the new



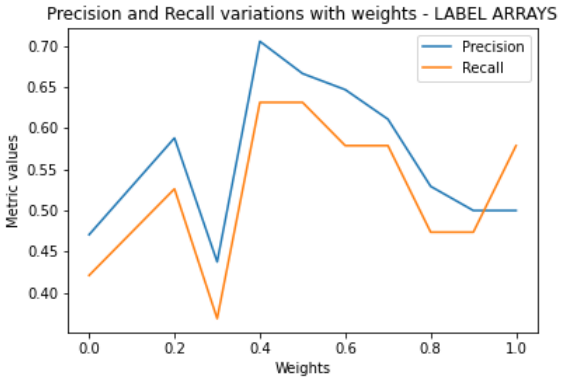
(a) Label Arithmetic

(b) Label Loops

Figure 3.15: Precision-recall vs weight plot for Labels: Arithmetic and Loops



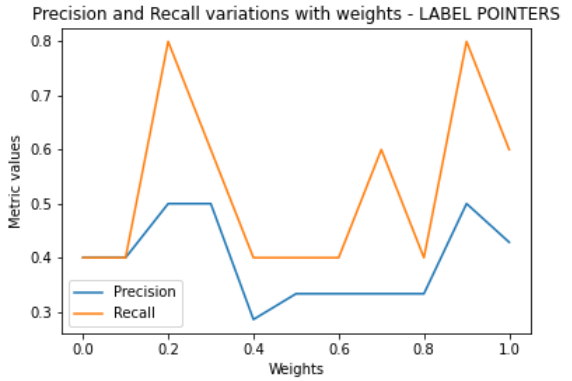
(a) Label Terminal IO



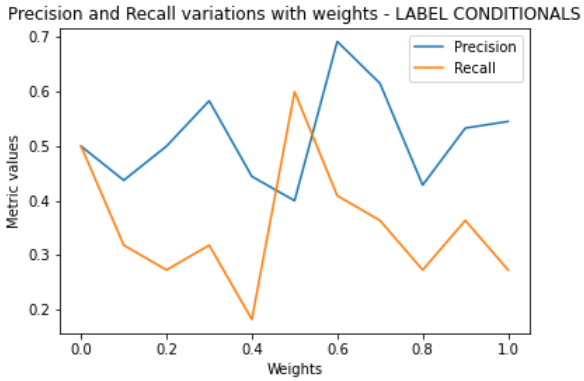
(b) Label Arrays

Figure 3.16: Precision-recall vs weight plot for Labels: Terminal IO and Arrays

dataset, proves to be beneficial to some of the labels. "Terminal IO" and "Char-string" are the labels which sees a drastic improvement in precision and recall at some of the weights. The Label

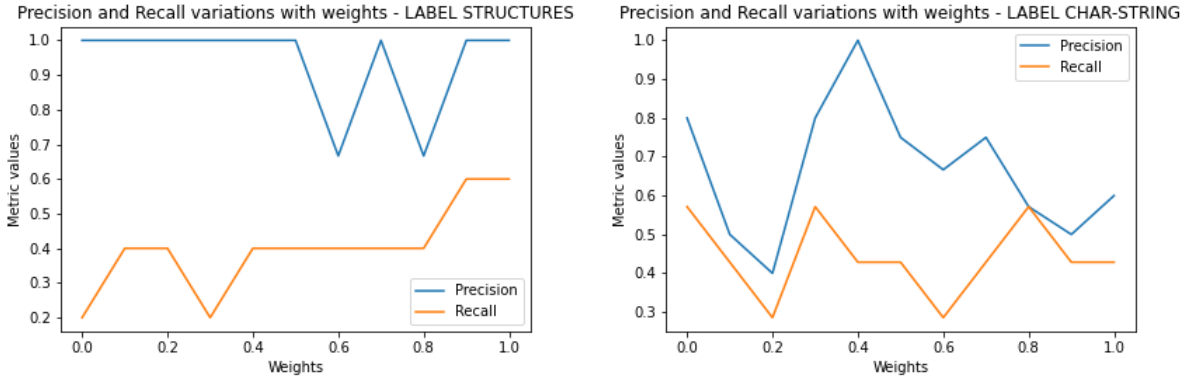


(a) Label Pointers



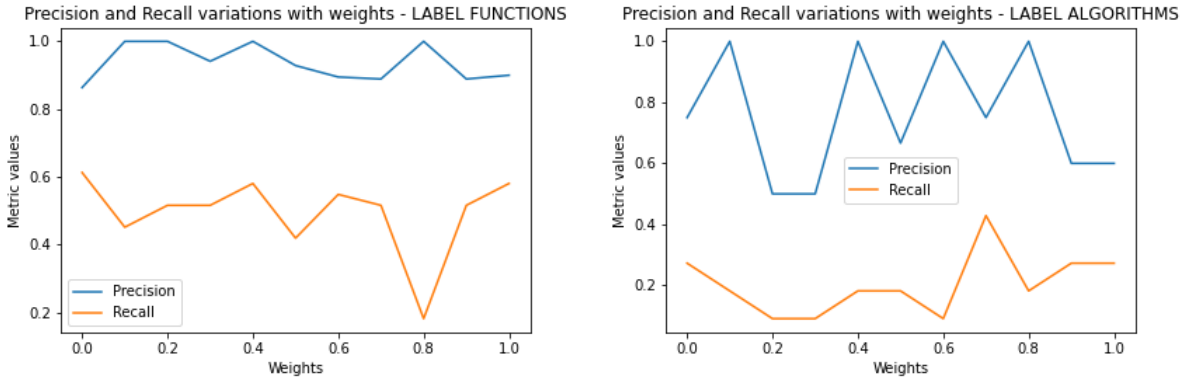
(b) Label Conditionals

Figure 3.17: Precision-recall vs weight plot for Labels: Pointers and Conditionals



(a) Label Structures (b) Label Char-string

Figure 3.18: Precision-recall vs weight plot for Labels: Structures and Char-string



(a) Label Functions (b) Label Algorithms

Figure 3.19: Precision-recall vs weight plot for Labels: Functions and Algorithms

"Arrays" observes the same pattern with an increase in both precision and recall from weights 0.3 to 0.4 .

The label "Conditionals" doesn't gain the same benefits as the previously mentioned labels. The range in which precision increases, we see a decrease in recall and vice-versa is true as well.

The label "Functions" is not much affected by the modifications. The label "Structures" is a rare label because of the low number of problems/instances it has been assigned/labelled to with.

3.6 Dynamic weights

Previously, we assigned instances classified as "Hard Yes" with a weight of 1 and experimented with different weights for the "Weak Yes" instances. Those weights were static in nature. We decided to try dynamic weights for the "Weak Yes" instances for more flexibility and label suitability.

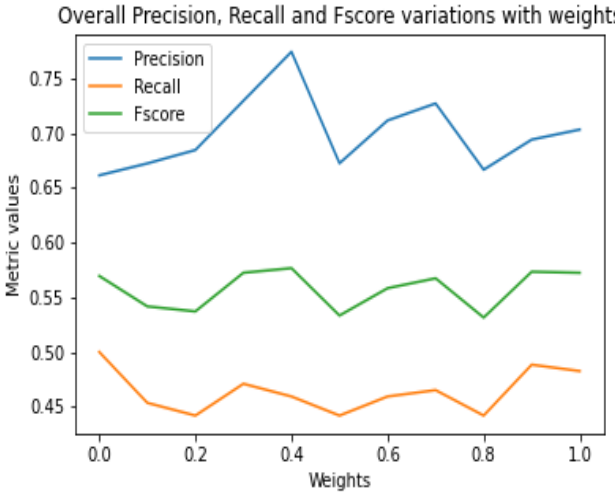


Figure 3.20: Overall Precision, Recall and Fscore variations with weights

For every label, we found out the weight that produced the highest f-measure or the f-score. Even though precision and recall are metrics that are more intuitive, comparing algorithms as well as results with each other on the basis of these two metrics is not easy to interpret. F-score helps by combining these two metrics into one.

Using the previously calculated and stored data, we found the weights giving the highest f-measure for each of the labels. The values can be seen below:

Labels	**Maximum F-score**	**Weight**
Arithmetic	0.64	0.9
Loops	0.74	0.3
TerminalIO	0.69	1.0
Arrays	0.66	0.4
Pointers	0.61	0.2
Conditionals	0.51	0.6
Structures	0.75	0.9
Char-String	0.66	0.0
Functions	0.73	0.4
Algorithms	0.54	0.7

Table 3.15: Label wise maximum f-score corresponding to the weight

After finding out the weights for each of the labels, we dynamically calculated the instance weights for each of the “Weak Yes” instances. As we know, a problem can be categorized into multiple labels. For each of the labels, we have the weight associated with them. We calculate the average of the weights for each of the “Weak Yes” classified instances and assign that as the instance weight. This is more dynamic than the previous approach of assigning the whole instance with one fixed weight. This approach takes into account the label wise weight-age which we found via the f-measure and rather than each of the instance getting assigned the same weight, now instead, each of the instance weight depends upon the labels present.

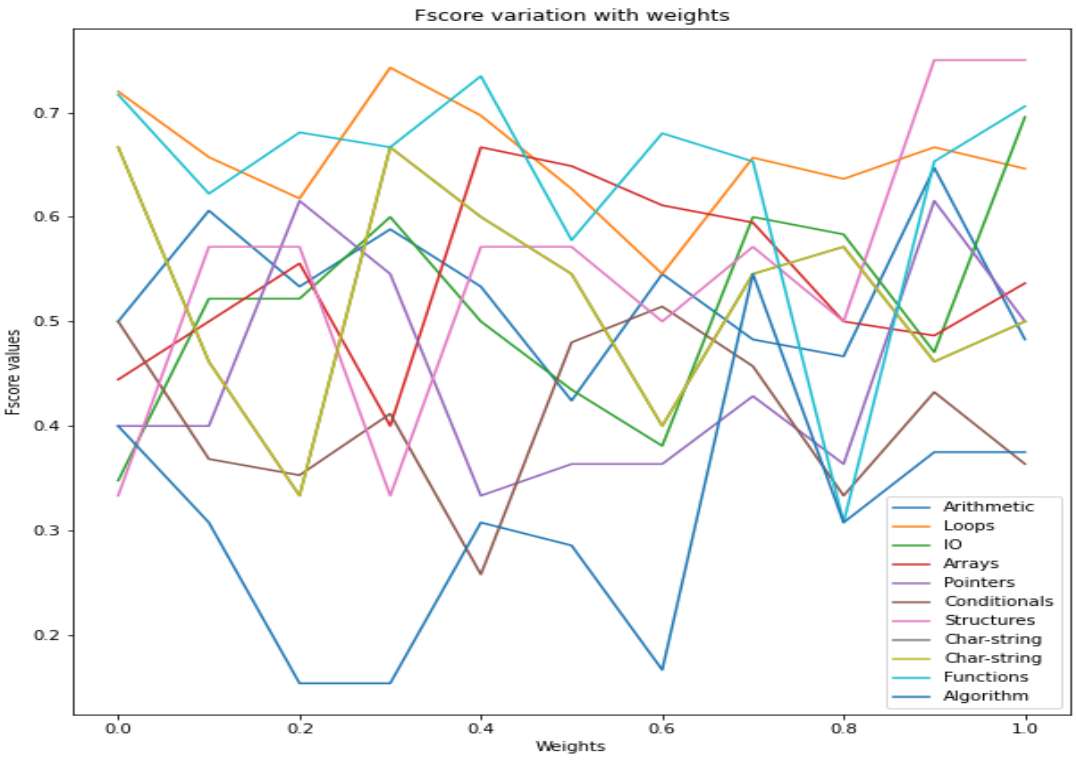


Figure 3.21: Label wise F-score variation with weights

The results obtained have been provided below:

Labels	Total TP TN FP FN	Precision	Recall
Arithmetic	21.0 10 42 3 11	0.76	0.47
Loops	39.0 23 19 8 16	0.74	0.58
TerminalIO	12.0 7 48 6 5	0.53	0.58
Arrays	19.0 9 39 8 10	0.52	0.47
Pointers	5.0 4 55 6 1	0.4	0.8
Conditionals	22.0 6 39 5 16	0.54	0.27
Structures	5.0 1 61 0 4	1.0	0.2
Char-String	7.0 3 59 0 4	1.0	0.42
Functions	31.0 13 34 1 18	0.92	0.41
Algorithms	11.0 3 54 1 8	0.75	0.27

Table 3.16: Label wise results for dynamic weights on taking the average

Hamming own 0.80

Hamming np 0.19

Ones acc 0.45

TP 79 TN 450 FP 38 FN 93

Total Positives 172.0
Overall Precision 0.67
Overall Recall 0.45

A precision of 1 for some of the labels (structures and char-string) provided good results for the experiment.

Upon close inspection of the results, we observed that an average weight of 0.45 provides the best possible results in terms of overall precision and recall. Based on this, we hypothesized that by taking the average, the overall weight is getting pulled down, which leads to not obtaining the best possible overall results. To verify this hypothesis, we tried other approaches besides taking the average.

As stated above, for each of the "Weak Yes" classified instances, we found the weight producing the highest f-measure and then took the average for all of the labels and assigned that to be the instance weight for the Bagging-Classifier model. For hypothesis testing, we tried to take the minimum and the maximum of the label weights instead of the average.

Taking the maximum would result in an increased weight or a higher weight than the average while taking the minimum would result in a decreased weight or a lower weight than the average. Thus, taking the minimum and the maximum would allow us to inspect results for the two opposite ends of the weight spectrum.

The results for both the minimum and maximum cases are provided below:

3.6.1 Results For dynamic weights (Maximum) -

Labels	Total TP TN FP FN	Precision	Recall
Arithmetic	21.0 7 44 1 14	0.875	0.33
Loops	39.0 20 24 3 19	0.86	0.51
TerminalIO	12.0 8 48 6 4	0.57	0.66
Arrays	19.0 11 40 7 8	0.61	0.57
Pointers	5.0 3 56 5 2	0.375	0.6
Conditionals	22.0 8 39 5 14	0.61	0.36
Structures	5.0 2 61 0 3	1.0	0.4
Char-String	7.0 4 57 2 3	0.66	0.57
Functions	31.0 14 32 3 17	0.82	0.45
Algorithms	11.0 4 54 1 7	0.8	0.36

Table 3.17: Label wise results for dynamic weights on taking the maximum

Hamming own 0.81
Hamming np 0.18
Ones acc 0.47

TP 81 TN 455 FP 33 FN 91
Total Postives 172.0
Overall Prec 0.71
Overall Recall 0.47

3.6.2 Results For dynamic weights (Minimum) -

Labels	Total TP TN FP FN	Precision	Recall
Arithmetic	21.0 9 43 2 12	0.81	0.42
Loops	39.0 24 22 5 15	0.82	0.61
TerminalIO	12.0 5 49 5 7	0.5	0.41
Arrays	19.0 12 41 6 7	0.63	0.66
Pointers	5.0 2 58 3 3	0.4	0.4
Conditionals	22.0 7 39 5 15	0.58	0.31
Structures	5.0 2 61 0 3	1.0	0.4
Char-String	7.0 4 58 1 3	0.8	0.57
Functions	31.0 17 34 1 14	0.94	0.54
Algorithms	11.0 3 54 1 8	0.75	0.27

Table 3.18: Label wise results for dynamic weights on taking the minimum

Hamming own 0.82
Hamming np 0.17
Ones acc 0.49
TP 85 TN 459 FP 29 FN 87
Total Postives 172.0
Overall Prec 0.74
Overall Recall 0.49

Our earlier assumptions were that we were getting better results at higher weights and that the averaging of the weights was pulling the weights down. However, upon taking the minimum and the maximum instead of averaging them out, we found that we were getting a better output corresponding to the minimum.

Upon tracking the minimum weights that were found for each of the “Weak Yes”, we found that the weights with the highest occurrences are 0.3 and 0.4. After inspecting the results for the dynamic weights (minimum), we observed that the same weights, i.e., weights of 0.3 and 0.4, produced the best precision-recall amongst the possible weights.

The high occurrences of the weights: 0.3 and 0.4 are the reasons for the relatively lower weight average, and the higher weights were, in fact, pulling the weights up. This proved our earlier hypothesis wrong. However, after accommodating feedback from the user, we obtained improved results than in the previous iteration of the work done in PRIORITY.

CHAPTER 4

Conclusion

In this thesis, we presented various optmisiations to the working of PRIORITY. We introduced the use of confidence scores and threshold in the label propagation method. Earlier, the label propagation had no threshold to stop for, and thus this resulted in labelling all of the unlabelled data. By introducing a confidence or threshold, we placed a cut-off on the procedure, only allowing the labelling of those instances about which the model is highly confident. The graphs provided for comparison show a minor improvement in the results. This resulted in speedup as well due to an early stoppage in the label propagation.

Feedback is a strong tool, which was missing in the previous work. We also curated a new dataset to incorporate the feedback that is stored by Priority. We classified the feedback as "Hard Yes" and "Weak Yes", and then experimented with the different weights to be assigned to the "Weak Yes" instances ranging from 0 to 1 with an ensemble learning algorithm as the base classifier.

Furthermore, we moved away from the idea of static weights and tested with dynamic weights, which resulted in an improvement in both precision and recall collectively, as can be seen from the tables included.

Bibliography

[1] Fitting a gamma distribution with (python) scipy.

[2] Debanjan Chaterjee. *Intelligent Program Analysis and Program Indexing - I*. M.tech. thesis, Indian Institute of Technology Kanpur, 2022. published thesis.

[3] Darshak Chhatbar, Umair Z. Ahmed, and Purushottam Kar. MACER: A modular framework for accelerated compilation error repair. *CoRR*, abs/2005.14015, 2020.

[4] Rajdeep Das, Umair Z. Ahmed, Amey Karkare, and Sumit Gulwani. Prutor: A system for tutoring CS1 and collecting student programs for analysis. *CoRR*, abs/1608.03828, 2016.

[5] Paul Denny, Andrew Luxton-Reilly, and Ewan Tempero. All syntax errors are not equal. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, 07 2012.

[6] Sharath HP. *Real World Deployments of AI-assisted compilation error repair and program retrieval*. M.tech. thesis, Indian Institute of Technology Kanpur, 2021. published thesis.

[7] The imbalanced-learn developers. Imblearn documentation balancedbaggingclassifier.

[8] Scipy Documentation API Reference. scipy.stats.gamma.

[9] sci-kit learn. Ensemble - baggingclassifier.

[10] Fahad Shaikh. *Advancements in AI-assisted compilation error repair and program retrieval*. M.tech. thesis, Indian Institute of Technology Kanpur, 2021. published thesis.

[11] Preeti Singh. *Intelligent Program Analysis and Program Indexing - II*. M.tech. thesis, Indian Institute of Technology Kanpur, 2022. published thesis.

CHAPTER 5

Appendix

5.1 Labels used in PRIORITY:

As per [10] the labels used in Priority can be seen below :

1. Difficulty [1,2,3,4,5]
    - 1. Very easy
    - 2 : Easy
    - 3 : Medium
    - 4 : Difficult
    - 5 : Very difficult
2. Terminal IO [Basic, Advanced]
    - Basic : simple IO with various data-types, use of escape sequences
    - Advanced : pretty patterns/word art/non-trivial formatting, format specifiers e.g. %5.4f or %0.2e, heavily formatted input e.g. (%d-%d-%d) to input (02-12-89)
3. Arithmetic [Basic, Advanced, Bit]
    - Basic : simple arithmetic operations (+,-,*,/,%,++,–), expressions, bracketing
    - Advanced : mixed type operations (e.g. long+int), explicit typecasting, math.h
    - Bit : use of bit-wise operators, left/right shift, bit masks
4. Conditionals [Basic, Switch, Advanced, Flag]
    - Basic : simple if/if-else statements, relational and logical operators
    - Switch : use of switch statements
    - Advanced : use of nested conditionals, ternary statements
    - Flag : use of flags e.g. isSorted, isFirstIteration
5. Loops [Basic, Advanced, In-variants]
    - Basic : simple use of for, while, do-while loops
    - Advanced : nested loops, use of break/continue, use of infinite while loops e.g. while(1)... and loops with empty headers e.g. for(;;)...
    - In-variants : Use of partial sums, running counts, running products and others
6. Arrays [Basic, Advanced, Memory]
    - Basic : 1D numeric arrays, creation, traversal, modification
    - Advanced : 2D or nD arrays
    - Memory : memory management using sizeof, malloc, calloc, realloc, free, stdlib.h
7. Pointers [Basic, Advanced]

- Basic : referencing, dereferencing, pointer arithmetic
- Advanced : arrays of pointers, pointers to pointers

8. Char-String [Basic, Advanced]
   - Basic : character IO, character arithmetic, string IO, NULL, EOF
   - Advanced : sub-string manipulation, strings and pointers, string.h
9. Functions [Basic, Advanced]
   - Basic : one or more scalar arguments and scalar return
   - Advanced : pointer/reference/array arguments, pointer/reference/array return
10. Structures [Basic, Advanced, DS]
    - Basic : storing user input in structures, arrays of structures
    - Advanced : pointers to structures, nested structures
    - DS : use/implementation of data structures e.g. linked list, stacks, (circular) queues, trees, graphs possibly using struct, or even using arrays
11. Algorithms [DC, Recursion, Greedy, DP]
    - DC : divide and conquer, bisection search etc
    - Recursion : self/mutual recursion
    - Greedy : greedy algorithms
    - DP : dynamic programming