Intelligent Program Analysis and Program Indexing - II

A Thesis Submitted

In Partial Fulfillment of the Requirements

For the Degree of M.Tech.

by

Preeti Singh

20111044

to the

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur

Aug, 2022

Declaration

This is to certify that at the thesis titled **"INTELLIGENT PROGRAM ANALYSIS AND PROGRAM INDEXING - II"** 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.

Preeti

Name: Preeti Singh (20111044)
Program: M.Tech.
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur, Kanpur,
208016.
Aug, 2022

iv

Declaration (To be submitted at DOAA Office)

I hereby declare that

1. The research work presented in the thesis titled **"INTELLIGENT PROGRAM ANALYSIS AND PROGRAM INDEXING - II"** has been conducted by me under the guidance by my supervisor(s) **PROF. PURUSHOTTAM KAR, PROF. AMEY KARKARE**.
2. The thesis has been formatted as per Institute guidelines.
3. The content of the thesis (text, illustration, data, plots, pictures etc.) is original and is the outcome of my research work. Any relevant material taken from the open literature has been referred and cited, as per established ethical norms and practices.
4. All collaborations and critiques that have contributed to giving the thesis its final shape have been duly acknowledged and credited.
5. Care has been taken to give due credit to the state-of-the-art in the thesis research area.
6. I fully understand that in case the thesis is found to be unoriginal or plagiarized, the Institute reserves the right to withdraw the thesis from its archive and also revoke the associated degree conferred. Additionally, the Institute also reserves the right to apprise all concerned sections of society of the matter, for their information and necessary action (if any).

Preeti

Name: Preeti Singh
Program: M.Tech.
Department of Computer Science and Engineering
Roll No.: 20111044
Indian Institute of Technology Kanpur, Kanpur, 208016.
Aug, 2022

Roll No - 20111044

POST GRADUATE OFFICE
THESIS SUBMITTED
ON 14-11-2022
I. I. T., KANPUR - 16

Certificate

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

[illegible] Amey Karkare

Prof. Purushottam Kar, Prof. Amey Karkare
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
Kanpur, 208016.
Aug, 2022

Abstract

Name of student: **Preeti Singh** Roll no: **20111044**
Degree for which submitted: **M.Tech.**
Department: **Department of Computer Science and Engineering**
Thesis title: **Intelligent Program Analysis and Program Indexing - II**

Name of thesis supervisor: **Prof. Purushottam Kar, Prof. Amey Karkare**
Month and year of thesis submission: **Aug, 2022**

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. One such tool has been created for the problem setters of the course ESC101, which is taught to novice programmers each semester at the Indian Institute of Technology, Kanpur.

PRIORITY offers a web site where users can look up programming queries from previous ESC101 course offerings.It uses machine learning methods to automatically label each programming question so that it may be searched and could help the tutors to prepare the programming questions. Problem statement and golden solution has been provided for each particular programming questions. The task of feature engineering is therefore not straightforward when trying to extract features from such data. Predicting the programming labels and estimating the level of difficulty of the challenge are the other two elements.

This thesis suggests a number of modifications to the PRIORITY back-end machine learning architecture.The label prediction problem is primarily the subject of this thesis. By traversing the abstract syntax tree of C programmes, we were able to extract the features, which contributed to the more reliable creation of features. Features have been extracted using graph methods and depth first traversal.

We have experimented with various correlation strategies for feature selection. The effective feature selection carried out by the Pearson correlation algorithm resulted in a significant rise in the f1-score for the test data points. The model is trained via supervised feature selection followed by one-vs-rest logistic regression. Programming labels exist that require various quantities of features to be identified because they rely heavily on programming language tokens. As a result, we tried different number of features for each programming label and observed a substantial rise in the f1-score.We suggested additional optimization by using the ensemble technique in the feature selection stage. This strategy combines Pearson correlation and normalised mutual information for various values of K, and it significantly improves accuracy and f1-score.

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 Debanjan and Mayank who have been my partner during this thesis and have been a constant source of support. I also extend thanks to my batch mates, friends and family members. A special thanks to Fahad and Sharath, our seniors in the M.Tech program from the graduating batch of 2021 at the Indian Institute of Technology Kanpur.

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.

Preeti Singh
Aug, 2022

Contents

List of Figures

List of Tables

PART I : PRIORITY

CHAPTER 1

Introduction

Contents

In today's Era, Artificial Intelligence and Machine Learning are now being utilized to solve a wide range of real-world problems in a wide range of industries. Several intelligent tools are continually being created in the field of education to improve the teaching process.
Every semester, the Indian Institute of Technology, Kanpur, teaches ESC101, which is an introductory programming course to a considerable number of students. As it is an introductory programming course, hence Lab sessions are conducted on a regular basis. In each lab session, programming questions are provided to students which need to be solved within the allotted time. Hence every year, tutors and faculty of ESC101 need to prepare programming questions from scratch.The re-usability of programming questions is one of the key challenges faced by such a huge course comprising hundreds of students. PRIORITY is an online portal that was created to give course problem setters access to a library of a large number of programming questions from previous offerings.
PRIORITY creates a search index for all of the programming challenges in the database. Users can look for programming questions based on the programming principles or skills needed to solve the problem, as well as the difficulty level of the challenge. Figure 1.1 depicts the PRIORITY web portal's problem/programming query search page. PRIORITY has roughly two thousand programming questions in total. Now, in order to index this set of programming problems, each problem must be labeled using the web portal's search parameters. As a result, each programming question must have a label for programming skills as well as a mark for difficulty. Given the difficulty of annotating such a vast corpus, PRIORITY relies on machine learning algorithms to label the whole corpus of problems. The collection of programming queries is referred to as the PRIORITY data.
The first step is to create machine learning models that forecast the programming concepts/skills (also known as programming labels) that are connected with a problem. The second objective is to create machine learning models to forecast the difficulty of the problem. One important problem in using machine learning algorithms on the PRIORITY data is conducting suitable feature engineering. Each programming query must be converted into a carefully chosen set of quality attributes that will aid in the learning of the aforementioned prediction tasks.
We propose various improvements to PRIORITY's back-end Machine Learning architecture in this thesis, affecting all three primary components: feature engineering, programming label prediction, and difficulty score prediction. The results reveal that the recommended improvements improved the performance of the two prediction tasks significantly.

1.1 Our Contributions

The Key Contribution has been enumerated below:

1. Learning the confidence score : Previously In Priority, label propagation algorithm had been run for around twenty times to label the unlabelled data points and every time top fifty data points had been selected to include them in training data points for the next round of the algorithm. Hence, we thought that instead of running the algorithm twenty times, we could learn the confidence score and run the algorithm as many times as it is making appropriate changes in unlabelled data points. Details of this could be found in Mayank's thesis[3].

2. Featurization : For syntactic analysis of the solution of the programming question, we have used an abstract syntactic tree and extracted the programming terms needed for predicting the labels. For each data point, we have around 4992 feature vectors.

3. Experimented with correlation Techniques(Feature Selection) : For feature selection, we have experimented with some correlation techniques such as mutual information, Pearson, Spearman, precision, recall and f1-score.

4. With variable K value : As we have some labels for which only one feature is enough to predict the labels such as Conditional_Switch and some are for which more than one features are required to predict the labels such as Function_advanced. Hence we have also experimented with different values of K for each label during feature selection. We got an appropriate rise in f1-score with variable K values.

5. Ensemble Techniques(with correlation) : We have further improved the feature selection by performing ensemble technique. We have performed an ensemble of Pearson correlation techniques with K=12 and mutual information correlation techniques with K=20 and found some improvement in f1-score, precision, recall and accuracy of the model.

6. Feedback : We have also tried to use feedback provided by the users, who use the web application to make the new questions. We are using text based feedback to improve the model by changing the model weights. Details of this could be found in Mayank's thesis[3].

7. Difficulty Score: We have followed a mixture of Experts techniques to improve the prediction of difficulty score. We found that the Random forest classifier model is performing better for (1-2) difficulty score whereas Gradient boosting regressor model is performing better for (3-5) difficulty score. Hence we used a combination of both algorithms. Details of this could be found in Debanjan's thesis[5].

CHAPTER 2

Related Works

Technology and information utilisation in education could give students quick access to information and speed up their learning. The teachers, tutors, and students are the most significant participants in any course. One such course is ESC101, which is provided to new students at the Indian Institute of Technology Kanpur each semester . As a result, since this course is an introduction to programming, it has been necessary to prepare or set up programming problem statements in accordance with the subjects covered in earlier weeks. Since the students in this course are absolute beginners, instructors must create the topic-based problem statement for the lab session from scratch each semester.
Tutors could not use the previous offers' problem statements because they could not be searched. In order to make the issue statements searchable based on programming concepts like strings, arrays, loops, etc., Fahad[13] and Sharath[10] developed PRIORITY. We have outlined some of the background information in brief below.

- **Data Preparation:** There is a large and unlabeled corpus of problem data accessible from ESC101's previous offers. There is a SQL dump available with roughly 2K data points obtained from PRUTOR[7]. These issues cover problem statements, ideal solutions, and templates for addressing the problem statements. Despite the fact that the entire corpus is unlabeled, a few senior tutors have been asked to classify certain data points from the corpus in accordance with the level of programming expertise required to resolve the given issue. We have approximately 461 tagged data points from the 2K corpus after conducting this experiment.

- **Front-end Design:** A web application has been developed to make the problem statements searchable. It could be helpful for the tutors and faculty to search for a particular problem that covers certain programming topics. Both non-functional and functional requirements are considered while building the web application. The front-end part was built using AngularJS. Simplicity, intuitiveness, and user-friendliness are certain terms that are considered while building a web application. Login-based authenticity, login and logout functionality, feedback submission by registered users, display of search results based on query entered by user are the functional requirement implemented by Fahad[13] and Sharath[10].

- **Back-end Design:**They have created a backend using Django, with MYSQL running on the database server, in addition to the front-end component. The set of API has been created and put into use to ensure that the web application functions properly. Following is a brief description of some of the API:

  - **Create User:**username and password are the two parameters taken by this API and saved it in auth_user table. Password are saved in encrypted form.

  - **Authenticate User:**When registered user tries to login this API has been used to authenticate the credentials of the user.
  - **Text Feedback:** user, query,number of results, version and feedback are the five argument taken by this API and saved it in text_feedback table.
  - **Feedback Rating:** user,problem feedback,query, time, position of recommendation and rating are the six parameter taken by this API and returns the same data entered by user with auto generated id.
  - **Feedback:** user_id, problem_id and new label annotation has been asked from the user as feedback. users can also provide difficulty score. In this, if the user think that particular problem should be under some additional labels then they can provide it as feedback.
  - **Passive Feedback:** user, problem feedback, query, start time, end time, position of recommendation and category are the arguments passed to this API and it will save in the table.
  - **Problem Details:** This API returns the JSON object which contains details about the problem like golden solution, problem statements and template etc specified by the problem id parameter.
  - **Problem Search:** labels has been provided to this API as parameter and it will return the set of problems which matches the labels provided in the query section and also according to the difficulty score.

- **ML Model:** As we have 461 labeled data and around 1539 data points are unlabelled. Hence Fahad and Sharath have tried to label the unlabeled data using the labeled data points. They have used semi-supervised algorithm names such as Label Propagation to mark the unlabeled data points. Machine learning will predict the labels for unlabelled data points with a confidence score and then will choose the top fifty data points from that and then include those data points in the training portion in the next iteration.This process will repeat itself for about twenty iterations.

- **Web Application Deployment:** There has been two separate docker container on which front end and backend is hosted. It is hosted on cloud computer server instance of Indian Institute of Technology of Computer Science and Engineering Department.

CHAPTER 3

PRIORITY

Contents

PRIORITY is the web portal designed to make the corpus of the previous offering programming question searchable so that tutors don't need to prepare programming problems from scratch. We are aware that we have a more number of unlabeled data points than identified ones.The machine learning model must be trained in order to forecast the labels and difficulty score for each unlabeled data item.
Prediction of Label and difficulty scores are the two most crucial objectives of the thesis.Each question includes a problem statement, a golden solution, and a template taken from the PRUTOR SQL dump.Users of the portal could look for programming questions based on the level of complexity and the amount of programming knowledge needed to solve them.As we have about 2K data points, each problem must be indexed by the programming knowledge needed to solve it as well as on the level of complexity if we want it to be found based on the passed query.

3.1 Key challenges:

- **Label De-duplication:** In the annotation stage, some problems have been given to many tutors so that we could get some sort of consistency in the labeled data points.Although, redundant data points are the major challenge after this stage. Therefore, those redundant

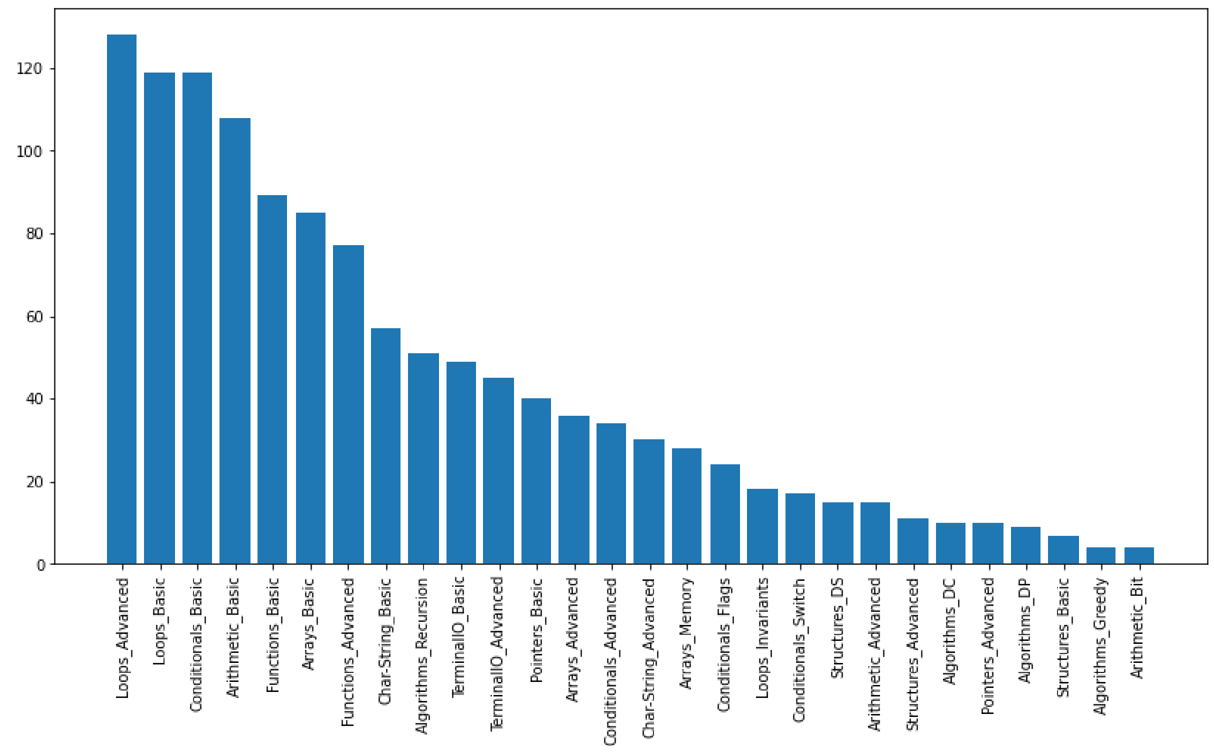

Figure 3.1: Distribution of Labeled Data Points. Image Credit [13]

data points need to be removed. The redundant data points are being merged using union technique in the current PRIORITY scenario.

- **Missing Label Problem:** Numerous labels that are used frequently in programming questions include TerminalIO, Arithmetic, Conditionals Statements, and others. Since every tutor has a unique combination of skills and perspective, some tutors may choose to overlook labels that are frequently used, such as understanding of terminal input/output, and instead annotate the issue with the most crucial label. For instance, if a specific programming challenge calls for an understanding of conditional statements, arrays, and recursion. But the tutors just annotate this issue with the recursion label because they believe that if a student knows about recursion, he or she should also be familiar with conditional statements and arrays, hence decided that other label are irrelevant to annotate. As a result, we had a problem with missing labels. To fix it, we combined the labels and the difficulty score averages for the data points annotated by two or more instructors.

- **Problem of Class Imbalance:** In our dataset, we have fewer labeled data points.Figure 3.1 shows the distribution of problems among twenty eight labels in sorted form. We can observe from the plot that there is a significant imbalance in the labels.Some labels that are more popular have many problems labelled, while some labels that are less popular have fewer problems identified.Therefore, a machine learning model that uses labels with fewer problems that have been marked may perform poorly.

3.2 AST Analysis of Each Label:

Previously in Priority, our seniors have used bag of words techniques for performing feature selection which didn't capture the syntactic and semantic features of the source code. Abstract Syntax Tree is the tree representation of the source code of any programming language. In this, each node will depict the programming terms or constructs occurring in the source code which will help to capture syntactic behavior of the source code . We have used the pycparser[4] compiler for getting the abstract syntax tree of the source code. As it is a kind of tree so traversal will be easier. In pycparser[4], there are a total of 49 node types such as For,PtrDecl,ArrDecl, etc.
We have 461 labeled data points which were labeled by tutors and faculties who conduct the course ESC101 in IITK every semester and prepare lectures and questions for the lab sessions. We have analyzed AST of problems tagged under each labels and tried to find which all nodes are necessary for predicting the particular labels. We have ten major labels and two or three minor labels under each major label.

1. **TerminalIO:**
   Basic and Advanced are the two minor labels which come under the TerminalIO major label. The presence of console input such as scanf,getline,gets and console output such as puts, printf states about the terminalIO label.

   - **Basic:** It includes simple console input and output statements with simple format specifiers.

```
printf("%d\n", cur);
```

printf syntax with simple format specifier

```
FuncCall:  (at temp.c:20:17)
  ID: printf (at temp.c:20:17)
  ExprList:  (at temp.c:20:24)
    Constant: string, "%d\n"
    ID: cur (at temp.c:20:32)
```

AST representation of basic printf

Table 3.1: Example of TerminalIO Basic

   - **Advanced:**It includes console input and output with complex format specifier eg.$\%0.4lf$.

```
printf("%0.4lf", area);
```

printf syntax with complex format specifier

```
FuncCall:  (at temp.c:18:5)
  ID: printf (at temp.c:18:5)
  ExprList:  (at temp.c:18:12)
    Constant: string, "%0.4lf"
    ID: area (at temp.c:18:22)
```

AST representation of advanced printf

Table 3.2: Example of TerminalIO Advanced

   After Analysis we have come to conclusion that for this label, we need to check for FuncCall, ID and Constant node type.

2. **Arithmetic:**
   This major label includes three minor labels such as basic,advanced and bit. This label includes arithmetic operations present in C.

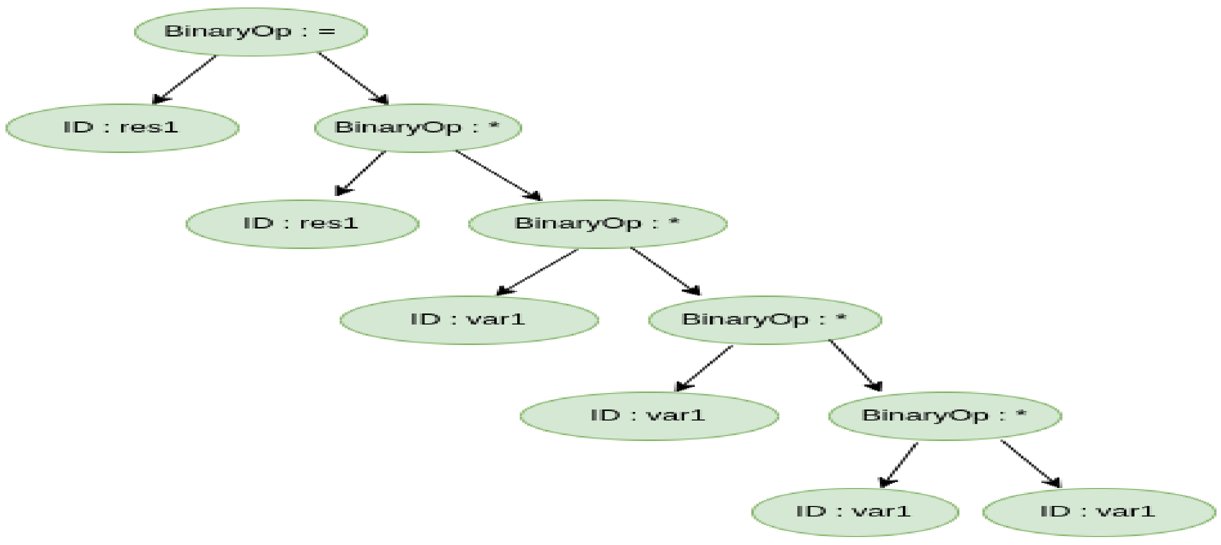

Figure 3.2: AST of sample code of arithmetic advanced

- **Basic:** This label includes binary operation in simple format. There should not be any nested form of binary operation. All binary operators are included except the bit operators.
- **Advanced:** All binary operators except the bit operators are present in nested form.

```
int main()
{
int res=1,var1=4;
res1=res1*var1*var1*var1*var1;
}
```

Table 3.3: Example of Arithmetic Advanced

Figure 3.2 shows the level of BinaryOp present in above sample code.Hence to detect this level we need feature which should count the number of level of same node types.

- **Arithmetic Bit:** All bit operators such as bitwise AND, bitwise OR, bitwise NOT are included in this label.

```
#include<stdio.h>
int main()
{    int a,b;
    scanf("%d %d",&a,&b);
    printf("%d",a^b); //xor is
     false if both the bits
    are 1
    return 0;
}
```

```
FuncCall:  (at temp.c:8:5)
  ID: printf (at temp.c:8:5)
  ExprList:  (at temp.c:8:12)
    Constant: string, "%d"
    BinaryOp: ^ (at temp.c
   :8:17)
      ID: a (at temp.c:8:17)
      ID: b (at temp.c:8:19)
```

Table 3.4: Sample Code and AST of example of Arithmetic Bit

Here, we can see that we only need to check the operator of the BinaryOp node type. Hence we need features for all unary,binary, ternary and bit operators which keeps the cunt of each operators.

3. **Conditionals:** This major label includes four minor labels such as basic, advanced, switch and flags.

   - **Conditionals Basic:** This minor labels includes problems which has simple if-else and if conditionals statements.Hence, here we should define feature which could find whether any conditional node type is present or not.
   - **Conditional Advanced:** This label includes problems which have nested if-else statements.Hence, here bigram feature and count of number of same node type feature is required as we need to check whether presence of bigram of (IF-IF).
   - **Conditional Switch:** This minor label includes problems which have switch statements. The count of switch statements and nested switch statements defines the difficulty score. Hence here, we can say that unigram feature of switch node type is enough to detect this label.
   - **Conditionals Flags:** : This label includes the problems where some boolean variable define the flow of the code.

```c
#include<stdio.h>
int main(){
        int n,mat[n][n];
        scanf("%d",&n);
        for(int i=0;i<n;i++){
          for(int j=0;j<n;j++){
             scanf("%d",&mat[i][j]);}}
        int skew=1;
        for(int i=0;i<n&&skew;i++){
            for(int j=0;j<n;j++){
               if(mat[i][j]!=-mat[j][i]){
                   skew=0;
                   break;
            }}}
        if(skew)        printf("SKEW MATRIX\n");
        else            printf("NOT SKEW\n");
    return 0;
}
```

Table 3.5: Example of Conditional Flags

Table 3.5shows the example of flag(skew=0) used. Presence of flag could be analysed logically. Hence we could not analyse it with the help of AST.

4. **Loops:**

   - **Basic:** In this minor label, we have simple for,while and do-while loop. In this the program should not contain any form of nested loop. Hence just the presence of for,

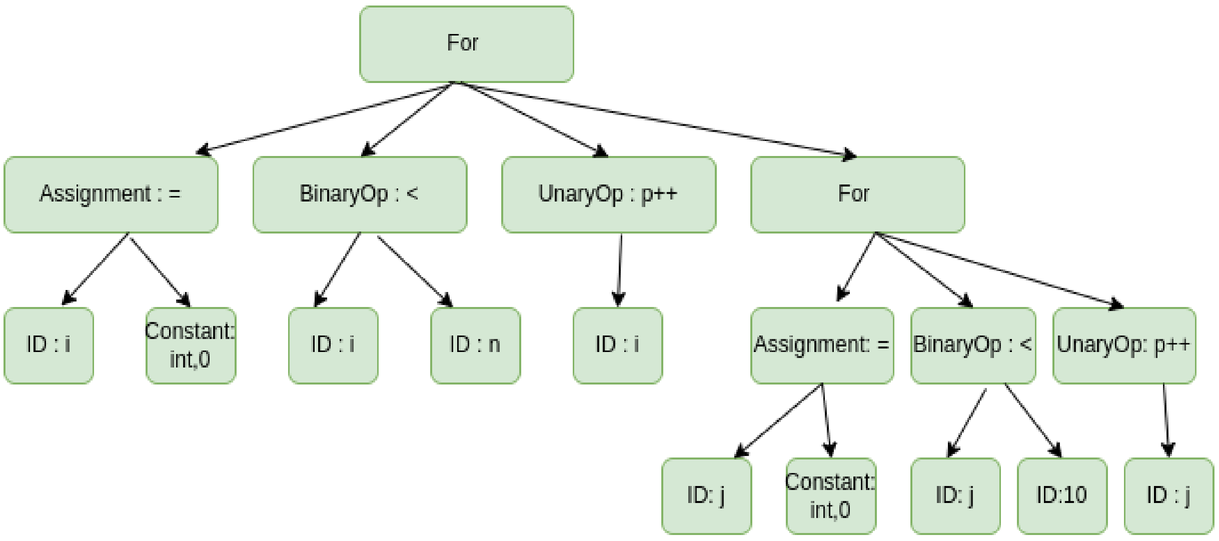

Figure 3.3: AST of nested for loop (Advanced Loop)

while node type is enough to detect this label. Unigram of for or while loop is required to detect this loop and count of these loop could contribute to get the difficulty score of the problem.

- **Advanced :** In this minor label, we have nested form of the loop.Hence just unigram is not enough to detect this label. Combination of unigram and bigram is required to detect this label. Below example briefly described the fact.

```
for(i=0;i<n;i++)
   for(j=0.j<10;j++)
      printf("%c", a[i][j]);
     }
```

Table 3.6: Sample code of Nested For loop

Figure 3.3 clearly shows that we need to check for bigram feature (For-For) from the AST and also level of same node type to detect this label.

- **Loop-Invariant:** As loop invariant is the condition which is true at every iteration. Evaluating this label comes under dynamic analyses. Dynamic analysis could not be performed by abstract syntax tree as it is used to extract syntactic details.

5. **Arrays:**

- **Basic:** This minor label includes one dimensional array with all the operation such as creation,manipulation and iteration. Hence, here we need to check for the presence of ArrDecl node type and compute the dimension of the array.
- **Advanced:** This label includes 2D array or nD arrays. As dimension is the only factor which create the difference between basic and advance label hence dimension computation is required. We are employing the same procedure we used in the conditional

statement and loop to check for nested form here. Bigram feature of parent−child relationship could be used to detect this label.

- **Memory:** This minor label includes some of the memory management function such as sizeof,malloc,calloc,realloc,free. As all these are function call hence we need to check ID of FuncCall node type from the abstract syntax tree to detect this label.

6. **Pointers:**

- **Basic:** This minor label includes the declaration of pointers and simple operation with pointers such as referencing,pointer arithmetic and dereferencing. The abstract syntax tree of the problem solution should contain PtrDecl node type. There could be any number of this node, however it should be present in nested form.
- **Advanced:** This minor label includes complex declaration of pointers such as pointer to pointer and array of pointers. Below examples show the representation of abstract syntax tree of pointer to pointer declaration.

```
int *C;
```

```
int **D;
```

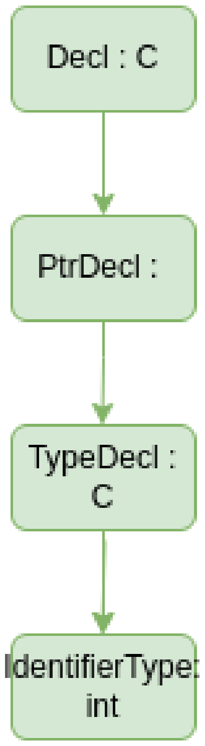

Figure 3.4: Example of basic Pointers

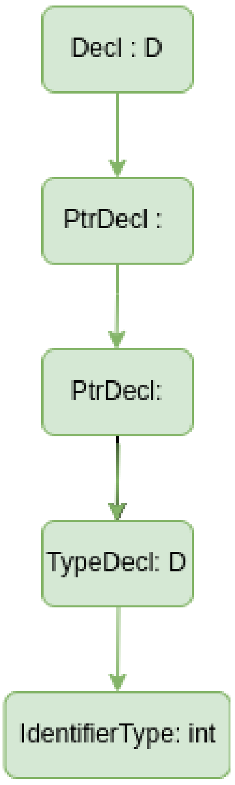

Figure 3.5: Example of Advanced Pointers

Figure 3.4Figure 3.5shows the AST of basic and advanced pointers respectively. It clearly states that unigram and bigram feature of parent−child relationship could be used to detect basic and advanced pointers respectively.

7. **Char-String:**

   - **Basic:** This minor label includes character declaration, character arithmetic, EOF and NULL value.
   - **Advanced:** This minor label includes declaration of character array, string function and string manipulation. Currently we are ignoring the headers of the solution , however if we use, we could extract string.h from the header to detect this label.

8. **Functions:** For this label, we need to check for the parameter declared and passed to function and also the return value from the function.

   - **Basic:** This minor label includes the declaration of function with simple parameter and should return simple values.
   - **Advanced:** This minor label includes the declaration of function with complex parameter such as arrays, structure or pointers. It could also return the array , structure or pointer.

```c
#include <stdio.h>
float exponential(int n, float x) {
  float sum = 1.0f;
  int i;
  for (i = n - 1; i > 0; --i )
  sum = 1 + x * sum / i;
  return sum;
}
```

```c
#include <stdio.h>
void product(int p[], int n, int* pro){
int i;
for(i=0; i<n; i++){
    *pro = *pro * p[i];
    }
return;
}
```

Table 3.7: Example of Basic & Advanced Function

9. **Structures:**

   - **Basic:** Members of the structure in this label should be basic. arrays could be a member, but pointers shouldn't be present.Hence, We need to check Decl node type of each member.
   - **Advanced:** Members of the structure in this label may be complicated types, such as pointers to structures and pointers to pointers. This category also includes nested structures. Hence, we need to check unigram and bigram features of each member.
   - **DS(Data Structures):** This minor label implements a number of significant data structures, including linked lists, stacks, queues, trees, and graphs. These implementations are logical, therefore employing an abstract syntax tree was unable to capture them. We have determined that there is a strong likelihood that this label will be present if we have an advanced structure and recursion.

10. **Algorithms:**

   - **Recursion:** We used a call graph to detect recursion. The easiest technique to spot recursion is to create a directed call graph where each function is a node and all the other functions it calls are connected by edges. Cycles in the graph lead to recursion.Many static-analysis programs can produce call graphs from C source. However, function pointers make it difficult for analyzers to find recursion since they would have to look at every conceivable value that the program may assign the pointer to. If the code contains function pointers, this could result in false negative results when employing static analysis. Details of the implementation of call graph could be found in Debanjan's thesis[5].

- **DC(Divide and Conquer):** This label includes the problem which divide the problem into sub problem and solve each sub problem and combine the results of sub problem to solve the problem.
- **Greedy:** This label includes the problem which could be solved by adopting best possible approach.
- **Dynamic Programming:** This label includes the problem which could be solved by algorithms based on dynamic programming.
  Three labels—Algorithm DC, Algorithm Greedy, and Algorithm DP—need a dynamic analytic approach for detection.Since the programming problem covered by these labels may be resolved by identifying its logical behaviour, utilising an abstract syntax tree is not an option.

3.3 Featurization :

According to our dataset, there are more unlabeled data points in PRIORITY; therefore, if we use a machine learning model to predict labels for these data points, these data points will serve as the train and test datasets. Each programming question must be modelled as a feature vector for this. Since the problem setter typically presents the problem statements in the form of a situation or a tale, we have totally dismissed the programming problem statements for feature extraction.
Prior to this, our seniors in PRIORITY have extracted the features using bag of words methodologies. We need strong features that can help predict the label for unlabeled data points and enhance the performance of the machine learning model because we have less labelled data points.For each solution code, we have employed an abstract syntax tree for feature extraction. In above section 3.2, we have briefly described the analysis of each label using abstract syntax tree. Following the analysis stage, we have carefully selected the features that would offer useful data and aid in forecasting the labels and difficulty score for the specific challenge.We have taken into account categorical and ordinal features to offer a more robust feature set.. Below is a brief description of the features.

3.3.1 Categorical Features:

- **Unigram Features:** We have used Pycparser[4] to generate the abstract syntax tree. For C language, Pycparser provide forty nine node types in abstract syntax tree. Hence there will forty nine unigrams which will provide usful information and help in predicting some labels such as Conditional Switch.Details of this features could be found in Debanjan's thesis[5]

- **Parent−Child Bigram Features:**To identify the nested form of programming phrases like nested loop, nested if-else, etc., we employed the parent−child bigram. We must navigate the C code's abstract syntax tree to accomplish this. Since Pycparser offers 49 types of nodes, hence the parent-child bigram will have a total of 2401 features.Details of this features could be found in Debanjan's thesis[5]

- **Parent−GrandChild Bigram Features:** When two for loops or if statements are separated by curly brackets, we can use the parent−grandchild bigram to detect the nested form of the programming terminology such as nested loop, nested if−else, etc. We must navigate the C code's abstract syntax tree to accomplish this. Since Pycparser offers 49 node kinds, the parent−grandchild bigram will have 2401 features in total.Details of this features could be found in Debanjan's thesis[5]

- **Recursion check Features:** Call graph technology has been used to implement this feature. It is a binary feature, with a value of 1 if the cycle is present in the call graph and a value of 0 otherwise. As a result, there will be one recursion check feature overall. Cycle in the call graph has been found using depth first search.Details of this features could be found in Debanjan's thesis[5]

- **Boolean Function Call Features:** There is an easy way to tell if a function call is present in C code. To accomplish this, we must navigate the abstract syntax tree and look for nodes of the FuncCall node type. It is a binary feature that, if the FuncCall node type is present, will be set to 1; otherwise, it will be set to 0.As a result, there will be one boolean function Call feature overall.Details of this features could be found in Debanjan's thesis[5].

3.3.2 Ordinal Features:

- **Maximum Count of Same Node Type Sub−Tree Features:** In place of basic unigram and bigram features, we can extract more effective features. We could capture how frequently that node type appears in the subtree. We simply take into account the maximum frequency out of all the frequencies. We shall have features of dimension 49 because there are 49 different sorts of nodes provided by Pycparser[4].Details of this features could be found in Debanjan's thesis[5].

- **Maximum Cycle length in Call Graph Features:** We should calculate the call graph's cycle duration to provide some effective features. This will increase the accuracy of detecting direct recursion and also aid in capturing indirect recursion.Maximum length should be taken into consideration. As a result, there will be one maximum cycle length feature overall. Details of this features could be found in Debanjan's thesis[5].

- **Maximum Depth of Same Node Type Features:** In the AST, we have the frequency of occurrence for each type of node. However, looking at the overall frequencies won't reveal as much about the program's syntactic structure. As a result, we suggest that we count the instances of each node type from the subtree's root to its leaves for each subtree that is rooted at that node type. We only use the highest possible count for each type of node. We shall have features of dimension 49 because there are 49 different sorts of nodes provided by Pycparser[4].Details of this features could be found in Debanjan's thesis[5].

- **Count of Operators Features:** As we have discussed in section3.2, We need to feature some of the values acquired from node types instead than just taking into account node types. One feature determines whether the ID of the FuncCall is console input/output, and if it is, another feature for the format specifier is available. We require features for each BinaryOp, UnaryOp, and TernaryOp because we need to determine whether the BinaryOp is a bit operator for the Arithmetic bit label. These features will keep track of frequency of each operator appears in the programming problem's solution. Debanjan's feature set now includes a total of 40 additional functionalities which has been by considering some of the labels which requires value of the node type.

3.3.3 Feature Extraction:

Our seniors have employed bag−of−words for feature extraction, as we covered in chapter 2. In order to create a robust feature set given the limited number of labelled data points, we employed abstract syntax trees to extract the features from the answers to the programming problems. We have discussed feature extraction for label prediction in this thesis. We have a 4992-dimensional feature vector for each programming questions.

3.3.4 Feature Selection:

For each data point, we obtained a 4992 dimension feature vector after evaluating the abstract syntax tree. We won't use all the features to identify a specific label for the label prediction job. Reducing the number of features for a learning task frequently improves how well machine learning models perform.A subset of the features is sufficient to detect the specific label because the features are resilient in nature.
As a result, we explored supervised feature selection strategies that filtered the features depending on how input and output were related.

3.3.4.1 Correlation Approach:

The concept of correlation describes the relationship between two or more input variables that can be utilised to forecast a target variable. We have experimented with a variety of correlation techniques, such as selecting features based on precision, recall, or f1-score, and in doing so, we have run into difficulties determining the threshold value that will convert non-binary features to binary features. The pearson, spearman, and mutual information correlation approaches have also been tested. All other methodologies are outperformed by the pearson correlation method by tuning the hyperparameter K.

3.3.4.2 Variable K value for each labels:

By altering the value of K, we have conducted a number of experiments to obtain the right number of features for each label that could be useful for prediction. As there are some programming labels for which we required only one or two features that might help in detecting the particular label for test cases, while there are other programming labels for which we need more than 10 features. Therefore, we considered testing out different K values for each programming label.We conducted the experiment by varying the value of K from 1 to 40 and then selected the value of K that produced the highest f1-score for each individual label.

3.3.4.3 Ensemble Techniques:

When we explored with correlation strategies, we found that the mutual information approach and the Pearson correlation approach both produced good results for certain labels. Therefore, we considered using an ensemble approach to integrate the outcomes of both approaches before selecting the features. We have compiled the results of each strategy's score for the Ensemble approach using the formula below.

```
    Final scores = T * P +(1- T) * M;
    where T is hyperparameter, P is scores by Pearson approach,
    M is scores by Mutual information approach.
```

In order to conduct this experiment, we first chose the same value of K for both approaches before varying K values for each approach. Different values outperformed, as we discovered.

3.4 Machine Learning Model:

Our next objective is to select a suitable machine learning model that could employ these feature vectors and aid in predicting labels for test data points after we have completed feature sets for each programming label.
In chapter 2, we reviewed how Fahad and Sharad had previously trained the Balanced Random Forest Classifier with bag-of-words features.In order to label unlabelled labels, they employed a semi-supervised approach called label propagation. They have chosen the top fifty data points from the unlabeled portions after each label propagation step and added those data points to the training set for the subsequent iteration.Labels and difficulty scores are the two search criteria being offered to search for a particular type of programming question. Therefore, label prediction and difficulty score are the two main goals discussed below.

3.4.1 Label Prediction:

Predicting all the pertinent programming concepts linked to a given programming topic is what is meant by the term "label prediction task." There are 28 different programming labels used in all. An effective model for this task is a multi-label classification problem. After carefully examining each programming label and its abstract syntax tree to obtain a feature vector of dimension 4992 for each data point, we will now examine the various machine learning approaches that can be used to accomplish this task.

3.4.1.1 Training the Model:

There are roughly twenty eight programming labels, and we have chosen features for each of them. We will individually train 28 distinct binary classifiers after feature selection. In this instance, we applied the one-vs-all (OVA) method, which is intended to address the multi-label classification problems during training. After the training phase, we will have 28 binary classifiers that can predict a specific label. Gradient boosting Regressors and Logistic Regression cross validation are two classifier algorithms that we have explored during training.

3.4.1.2 Prediction by the Model:

We have examined the One-vs-Rest technique for model prediction. For one-vs-Rest classification, we must create N-binary classifier models for the dataset of N-class examples. The number of created binary classifiers and the number of class labels contained in the dataset must be equal. We have 28 binary trained classifiers that will determine whether or not the label is present. This is one of the common methods for resolving multi-label categorization problems.

3.4.2 Difficulty score Prediction:

The instructors might use an important search criteria to discover an appropriate problem, which is the level of difficulty of a programming task. It is necessary to create a machine learning model to estimate the difficulty score of a programming question because a significant portion of the PRIORITY data points are left unlabeled. As discussed in chapter 2, Previously this task was considered as multi class classification problem and the Random Forest Classifier method was used to perform a straightforward classification with five classes. However, this method ignores the ordinal relationship between the difficulty scores that exists.
This task is a good illustration of ordinal regression. We have investigated numerous regression strategies to tackle this problem. For lower difficulty levels (1 and 2) across many metrics, we discovered that the Random Forest classifier model performed better than the Gradient Boosting Regressor model, whereas for higher difficulty scores(3, 4, 5), the Gradient Boosting Regressor model performed better than the Random Forest Classifier. In order to benefit from both methodologies, we have used an ensemble approach.Details of this could be found in Debanjan's Thesis[5].

CHAPTER 4

Experiments

Contents

We will examine the experiments that we have conducted for the label prediction task in this chapter and their accompanying findings.

4.1 Feature Selection:

4.1.1 Selection of Correlation Technique:

The concept of correlation describes the relationship between two or more input variables that can be utilised to forecast a target variable. We have experimented with a variety of correlation strategies to choose features, such as employing precision, recall, or f1-score as correlation techniques. Both categorical and ordinal characteristics have been defined in our feature set, as it was explained in the section above. For categorical features, we can choose the essential traits for each programming label by using precision, recall, or the f1-score as a correlation strategy. We had difficulty with ordinal features, though, because we had to turn them into categorical characteristics in order to use the correlation strategy. The threshold value is important information for this conversion. Assuming a threshold value is challenging since some labels just require one or two features to be identified while other labels require multiple features. Even though we tried with a few threshold values, our findings weren't very promising.
Following that, we tried with a few more correlation techniques, including mutual information and Pearson's correlation. Pearson Correlation is use to test whether there is a statistically significant linear relationship among the features and also help to determine the strength and direction of the association. After experimenting with Pearson's approach, we found some promising results hence we proceed with Pearson for future experiments. For different values of k, we compared various metrics from the Logistic Regression Cross Validation model, such as accuracy, precision, recall, and f1-score.
For feature selection , we are considering top-k features which has highest correlation score for each label. Hence k is the hyperparameter and we have experimented with many values of K.

Program Label	Accuracy @5	Precision @5	Recall @5	F1-Score @5
Arithmetic_Bit	0.9846	0	0	0
Algorithms_Greedy	0.9846	0	0	0
Arithmetic_Basic	0.7538	0.5263	0.5882	0.0.5556
Loops_Basic	0.6769	0.3529	0.375	0.3636
TerminalIO_Advanced	0.8	0.381	1	0.5517
Arrays_Basic	0.6308	0.2727	1	0.4286
Pointers_Advanced	0.9538	0	0	0
Conditionals_Flags	0.9077	0.2	0.3333	0.25
Structures_Basic	0.9846	0.5	1	0.6667
Char-String_Basic	0.6923	0.2083	0.8333	0.3333
Conditionals_Basic	0.5538	0.3571	0.8824	0.5085
Functions_Advanced	0.6769	0.4167	1	0.5882
Arithmetic_Advanced	0.9692	0	0	0
Conditionals_Switch	1	1	1	1
Algorithms_Recursion	0.9538	0.7273	1	0.8421
Structures_Advanced	1	1	1	1
Algorithms_DP	0.9231	0.1667	1	0.2857
Conditionals_Advanced	0.8615	0.1818	1	0.3077
Loops_Advanced	0.8923	0.7917	0.9048	0.8444
Arrays_Advanced	0.9538	0.7143	0.8333	0.7692
Char-String_Advanced	0.9231	0.1667	1	0.2857
TerminalIO_Basic	0.7077	0	0	0
Pointers_Basic	0.9077	0.3333	0.5	0.4
Structures_DS	9846	0.6667	1	0.8
Arrays_Memory	0.9692	0.75	0.75	0.75
Algorithms_DC	0.9692	0.3333	1	0.5
Functions_Basic	0.6769	0.4167	1	0.5882
Loops_Invariants	0.9385	0	0	0
Overall	**0.8654**	**0.3935**	**0.7688**	**0.5205**

Table 4.1: Logistic Regression model Metrics using Pearson's Correlation Approach

Table 4.1 shows the metrics of the Logistic Regression Cross validation model by using Pearson's Correlation approach.

4.1.2 Ensemble Approach Used in Correlation Technique:

To choose a better feature that could aid in forecasting the programming labels, we have tried with both Pearson correlation and mutual information correlation strategies. We have looked at many values of k, which range from five to twenty. As was mentioned in the subsection 4.1.1, the Pearson Correlation approaches produced encouraging findings.
However, after more analysis, we discovered that for some programming labels, Pearson correlation is performing better and for some programming labels, mutual information is performing better.As a result, we considered combining the findings from the two correlation methodologies in some way so that we might benefit from both. There are two methods for doing this: the first is to simply combine the results using simple statistical equations, and the second is to learn the combination using a machine learning model.

The first strategy, which involves combining the results from the two correlation approaches using a statistical formula, has been used in this thesis. The formula is presented in the chapter 3. This combination of results is known as ensemble technique. Figure 4.1 shows the comparison of the metrics of Mutual information and Person Correlation techniques at k=5. From the figure, we have analyzed the f1-score obtained from the Pearson Correlation approach for some programming labels such as Char-String Advanced, Arrays_Memory, TerminalIO_Advanced, Char-String_Basic, and Conditionals_Basic is more than obtained from the mutual information approach. However, there are some labels such as Arithmetic_Basic, Loops_Basic, Arrays_Basic, Pointers_Advanced, and TerminalIO_Basic for which the f1-score of mutual information is more than Pearson. Rest labels have the same values for both approaches.Table 4.2 shows the metrics obtained by the ensemble approach. This experiment has been performed by considering the same value of k for both the correlation approaches.

Program Label	Accuracy@5	Precision@5	Recall@5	F1-Score@5
Arithmetic_Bit	0.9846	0	0	0
Algorithms_Greedy	0.9846	0	0	0
Arithmetic_Basic	0.7692	0.5714	0.4706	0.5161
Loops_Basic	0.6769	0.3529	0.375	0.3636
TerminalIO_Advanced	0.9077	0.5714	1	0.7273
Arrays_Basic	0.7538	0.36	1	0.5294
Pointers_Advanced	0.9692	0	0	0
Conditionals_Flags	0.9077	0.2	0.3333	0.25
Structures_Basic	0.9846	0.5	1	0.6667
Char-String_Basic	0.8	0.2667	0.6667	0.381
Conditionals_Basic	0.6923	0.2083	0.8333	0.3333
Functions_Advanced	0.6769	0.4167	1	0.5882
Arithmetic_Advanced	0.9692	0	0	0
Conditionals_Switch	1	1	1	1
Algorithms_Recursion	0.9538	0.7273	1	0.8421
Structures_Advanced	1	1	1	1
Algorithms_DP	0.9231	0.1667	1	0.2857
Conditionals_Advanced	0.8615	0.1818	1	0.3077
Loops_Advanced	0.8923	0.7917	0.9048	0.8444
Arrays_Advanced	0.9538	0.7143	0.8333	0.7692
Char-String_Advanced	0.9538	0	0	0
TerminalIO_Basic	0.8154	0	0	0
Pointers_Basic	0.9077	0.3333	0.5	0.4
Structures_DS	0.9846	0.6667	1	0.8
Arrays_Memory	0.9692	0.75	0.75	0.75
Algorithms_DC	0.9692	0.3333	1	0.5
Functions_Basic	0.6769	0.4167	1	0.5882
Loops_Invariants	0.9385	0	0	0
Overall	**0.8846**	**0.4332**	**0.6936**	**0.5333**

Table 4.2: Logistic Regression Model Metrics using Ensemble approach

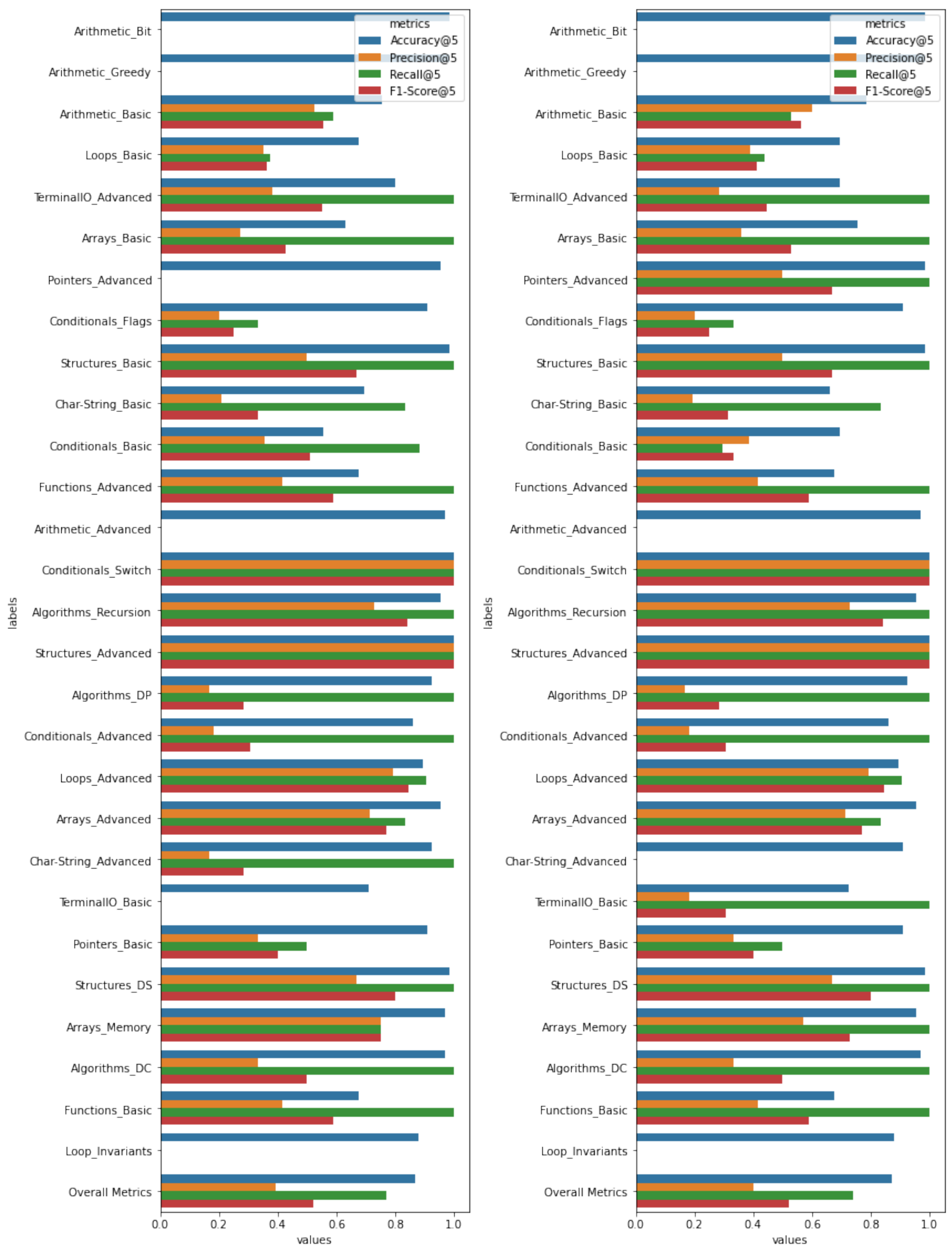

Figure 4.1: Comparison of Metrics of Mutual information and Pearson Correlation Approach @5

4.1.3 Ensemble Approach with variable K value for each Labels:

We experimented with a combination of mutual information and Pearson's correlation approach at different values of k after studying the outcomes of a mixture of two correlation techniques for the same value of k. According to Debanjan's thesis[5], mutual information performs better at k=20 and from subsection 4.1.1 we have found that Pearson performs better at k=5, respectively. When we compare both the Table 4.2,Table 4.3, we can see that there is significant increment in f1-score for the ensemble approach with different k value for each correlation technique.

Program Label	Accuracy	Precision	Recall	F1-Score
Arithmetic_Bit	0.9846	0	0	0
Algorithms_Greedy	0.9846	0	0	0
Arithmetic_Basic	0.8	0.6667	0.4706	0.5517
Loops_Basic	0.7385	0.4615	0.375	0.4138
TerminalIO_Advanced	0.9077	0.5714	1	0.7273
Arrays_Basic	0.7538	0.36	1	0.5294
Pointers_Advanced	0.9846	0	0	0
Conditionals_Flags	0.9231	0	0	0
Structures_Basic	1	1	1	1
Char-String_Basic	0.7692	0.2353	0.6667	0.3478
Conditionals_Basic	0.6923	0.4348	0.5882	0.5
Functions_Advanced	0.7538	0.4828	0.9333	0.6364
Arithmetic_Advanced	0.9692	0	0	0
Conditionals_Switch	1	1	1	1
Algorithms_Recursion	0.9538	0.7273	1	0.8421
Structures_Advanced	1	1	1	1
Algorithms_DP	1	1	1	1
Conditionals_Advanced	0.8769	0.2	1	0.3333
Loops_Advanced	0.8923	0.85	0.8095	0.8293
Arrays_Advanced	0.9538	0.7143	0.8333	0.7692
Char-String_Advanced	0.9538	0.25	1	0.4
TerminalIO_Basic	0.8615	0	0	0
Pointers_Basic	0.9231	0.4	0.5	0.4444
Structures_DS	1	1	1	1
Arrays_Memory	0.9692	0.75	0.75	0.75
Algorithms_DC	0.9846	0.5	1	0.6667
Functions_Basic	0.6769	0.4167	1	0.5882
Loops_Invariants	0.9385	0	0	0
Overall	**0.9016**	**0.4879**	**0.6994**	**0.5748**

Table 4.3: Logistic Regression Model Metrics using Ensemble approach with different k

4.1.4 Using variable value of K for each Labels:

By adjusting the amount of K, we have run a number of tests to obtain the right number of features for each label that could be useful for prediction. For some programming labels, we just require one or two qualities that could aid in identifying the specific label for test cases, however for other programming labels, we need more than ten features that could aid in identifying the specific label for test instances.

Therefore, we considered testing out numerous K values for each programming label. We conducted the experiment by varying the value of K from one to forty, and then we chose the value of K that produced the highest f1-score for each individual label. Table 4.4shows the metrics of the Logistic Regression model using variable value of k for each programming labels. After comparing the scores of Table 4.1,Table 4.2, Table 4.4, we can see that there is significant increase in the f1-score in Table 4.4 for the newly proposed approach of using different value of k for each label.

Program Label	Accuracy@5	Precision@5	Recall@5	F1-Score@5
Arithmetic_Bit	0.9846	0	0	0
Algorithms_Greedy	0.9385	0.2	1	0.3333
Arithmetic_Basic	0.7692	0.55	0.6471	0.5946
Loops_Basic	0.6923	0.3889	0.4375	0.4118
TerminalIO_Advanced	0.8308	0.4211	1	0.5926
Arrays_Basic	0.7692	0.375	1	0.5455
Pointers_Advanced	0.9846	0.5	1	0.6667
Conditionals_Flags	0.9077	0.2	0.3333	0.25
Structures_Basic	1	1	1	1
Char-String_Basic	0.8154	0.2857	0.6667	0.4
Conditionals_Basic	0.6615	0.4194	0.7647	0.5417
Functions_Advanced	0.7538	0.4839	1	0.6522
Arithmetic_Advanced	0.9231	0.2	0.5	0.2857
Conditionals_Switch	1	1	1	1
Algorithms_Recursion	0.9538	0.7273	1	0.8421
Structures_Advanced	1	1	1	1
Algorithms_DP	1	1	1	1
Conditionals_Advanced	0.8615	0.1818	1	0.3077
Loops_Advanced	0.9077	0.8571	0.8571	0.8571
Arrays_Advanced	0.9538	0.7143	0.8333	0.7692
Char-String_Advanced	0.9231	0.1667	1	0.2857
TerminalIO_Basic	0.9385	0	0	0
Pointers_Basic	0.8923	0.3333	0.75	0.4615
Structures_DS	1	1	1	1
Arrays_Memory	0.9692	0.6667	1	0.8
Algorithms_DC	0.9846	0.5	1	0.6667
Functions_Basic	0.6769	0.4167	1	0.5882
Loops_Invariants	0.9385	0	0	0
Overall	**0.894**	**0.4658**	**0.7861**	**0.5849**

Table 4.4: Logistic Regression Model Metrics using Variable value of K for each labels

4.2 Comparison of Results of Ensemble Approaches:

Metrics	Ensemble@same_K	Ensemble@different_K	Ensemble@variable_K
Accuracy Score	0.8846	0.9016	0.894
Precision Score	0.4332	0.4879	0.4658
Recall Score	0.6936	0.6994	0.7861
F1-Score	0.5333	0.5748	0.5849

Table 4.5: Comparison of the metrics of different Ensemble approaches

From Table 4.5,we can say that Ensemble approaches with variable value of k shows promising results in f1-score. However one-vs-rest algorithm suffer from low precision score. As PRIORITY suffer from missing label which results in high recall score.

CHAPTER 5

Conclusion and Future Work

Contents

5.1 Conclusion:

This thesis makes several suggestions for enhancements to the machine learning models used in the PRIORITY portal's back-end.As discussed in chapter 3, we mainly concentrated on the feature-stretching methods employed in PRIORITY. It is obvious that the suggested improvements have greatly enhanced label prediction performance.
The improvements are the result of a careful examination of the abstract syntax tree and testing a number of correlation strategies in order to identify the most effective correlation strategy for feature selection. We need to extract better features because there is a shortage of labelled data.In order to combine two or more correlation techniques at once and choose more reliable characteristics, we have experimented with ensemble strategies. For the model's training and prediction, we employed a conventional logistic regression model. The explanation for the higher scores is that the machine learning models were trained using more robust features.

5.2 Future Work:

- We are not using the problem statement for prediction labels, as we have described in this thesis. In order to use the problem statement for label prediction in future work, some methodologies could be investigated.
- At the training and prediction stages, we have only investigated the One-Vs-Rest algorithm. As a result, several algorithms for training and prediction might be investigated.

CHAPTER 6
Appendix

Contents

6.1 PycParser Node Types for C language:

- ArrayDecl
- ArrayRef
- Assignment
- Alignas
- BinaryOp
- Break
- Case
- Cast
- Compound
- CompoundLiteral
- Constant
- Continue
- Decl
- DeclList
- Default
- DoWhile
- EllipsisParam
- EmptyStatement
- Enum
- Enumerator
- EnumeratorList
- ExprList
- FileAST
- For
- FuncCall
- FuncDecl
- FuncDef
- ID
- Goto
- IdentifierType
- If
- InitList
- Label
- NamedInitializer
- ParamList
- PtrDecl
- Return
- StaticAssert
- Struct
- StructRef
- Switch
- TernaryOp
- TypeDecl
- Typedef
- Typename
- UnaryOp
- Union
- While
- Pragma

PART II : MACER

CHAPTER 7

Introduction

Contents

Research is currently being done on the use of AI-assisted tools for educational and software engineering applications. Compilation mistakes represent a significant learning obstacle for beginning programmers. Additionally, the compiler's comments generally addresses to more experienced programmers and may not make sense to novices. Programmers can benefit substantially from automated compilation error repair, which generates fixes for flawed programmes using AI-based techniques. For instance, if we use the condition "1=a" in a loop (where an is an integer variable), the compiler will generate the error "expression is not assignable," which is challenging for a new programmer to comprehend. Although the error statement mentions expression, the problem in this case is with assignment.

Therefore, it could be frustrating for a novice programmer to comprehend these error messages produced by the compiler and implement the fix. Although an experienced coder should have no trouble understanding it because he has probably encountered this issue before. However, in this case, if the inexperienced programmers receive some assistance from some knowledgeable instructors, that might aid them to learn quickly. It is impossible to provide human mentors to every student, despite the fact that the number of students is growing every year.

A lot of attention has grown in recent years around employing a machine learning method to correct flawed programmes. We have seen some encouraging outcomes in this area. One of the first studies, DeepFix[8], used a sequence-to-sequence machine learning model to produce right target code from flawed source code.The next strategy in this field was TRACER[1], which relies on source and target pairs provided from PRUTOR[7] and helps to identify student errors so that programmes can be corrected based on this data rather than artificially manufactured data.

A simple machine learning model that predicts the types of repairs that could be made as well as the locations where they should be made has been utilised by MACER[6] to establish six major modules for predicting repairs for the incorrect programme.As a result, it has demonstrated improvement over TRACER[1] in terms of accuracy as well as training and testing time.MACER[6] just uses the line number supplied by the compiler generated message and does not fully utilise the compiler diagnostic error message.

DrRepair[14] is one method for fixing a source programme with a compilation error. To generate the correct target code from the incorrect source code, they employed a sequence to sequence model with a graph attention mechanism, which completed the operation in a single pass.

The approach that SampleFix[9] recommended focused mostly on the variety and uncertainty of source code. We know that several types of code may have the same outcome, thus they propose a one-to-many mapping between the source and target programmes to repair the flawed programme.In order to increase accuracy, especially when there are fewer labels, and scale up the XML demands with many labels, DECAF[12] created initialization procedures for categorization.

Every module was optimised by MACER++[13] using the MACER[6] principle. They made use of the MACER[6] repair class generation idea, and TRACER was employed to provide synthetic data. PLT[11] (Probabilistic Label Tree), DiSMEC (one-vs-rest classifier for multi-label classification), and Tree Based XML approaches have been utilised to enhance the MACER[6] pipeline.

The MACER++[13] pipeline was initially developed for the C programming language, but as Python gained popularity and adoption, a tool was created that provides interactive feedback to help programmers fix compilation errors in Python scripts. There is now an application called PyMACER[10] that uses the MACER++[13] pipeline to correct Python code problems.

Below is the example which shows the erroneous source code , corrected target code and compiler generated error message.

```
1  #include<stdio.h>
2  int check_prime(int num)
3  {
4  int i;
5  for(i=2;i<=num-1;i++)
6  {
7     if(num%i=!0)
8     {
9     return 0;
10    }
11 }
12 return 1;
13 }
```

```
1  #include<stdio.h>
2  int check_prime(int num)
3  {
4  int i;
5  for(i=2;i<=num-1;i++)
6  {
7     if(num%i != 0)
8     {
9     return 0;
10    }
11 }
12 return 1;
13 }
```

Compiler generated error message: test.c:6:12: error: expected expression
Repair predicted by MACER: Replace =! with != on line 7.

Table 7.1: Example of repair prediction

7.1 Our Contribution:

- We have analyzed the repair classes provided by MACER. We have found that some of the repair classes are giving promising results.
- We have proposed changes in Tokenization step by correcting the first token of the erroneous source line.

CHAPTER 8

Related Works

Contents

In recent years, there has been a lot of interest in using artificial intelligence and deep learning to fix flawed programming codes . For the automatic repair of programs, this area has produced many encouraging outcomes.
The compiler creates messages that display program errors and include the line number where the error occurred. However, those messages produced by the compiler are challenging for new programmers to grasp. These compiler-generated messages are more obscure in nature, making them difficult for beginning programmers to understand yet simple for expert programmers. Beginner programmers need to spend a lot of time understanding the messages produced by the compiler as a result. Due to this, many researchers have developed a keen interest in this field. We've covered some of the relevant research that has been done in this area below.

8.1 TRACER (Targeted RepAir of Compilation ERrors):

- The repair process was originally broken down into several stages by TRACER[1], including Abstraction of the Code, Concretization, Localization of the Error, and Abstract Repair of Code.
- Instead of training on the entire incorrect source code, TRACER[1] was trained on the pair of incorrect source line and rectified target line.
- TRACER[1] abstracts identifiers and literals into certain generic tokens like TYPEKIND_INT, TYPEKIND_Literals, etc. instead of working directly on the source code.
- TRACER[1] took three lines from the source code; the first is the error line produced by the compiler, and the other two are the lines immediately above and below it. The deep learning model was fed these three lines.

8.2 TEGCER:

- Based on the solutions or modifications submitted by their peers, TEGCER[2] is a system that suggests corrections to the student's incorrect code.
- With the aid of the messages the compiler produces, TEGCER[2] initially creates the error-repair classes and offers advice on how to fix erroneous lines.

- The compiler reported error lines and the bigrams and unigrams of the incorrect line are used to generate an encoded feature vector.
- A dense feedforward neural network is trained using this feature vector to forecast the error-repair classes.

8.3 MACER (Modular Accelerated Compilation Error Repair):

In addition to locating the lines that require repair, MACER[6] also identifies the type of repair each line requires as well as the location of the repair.TEGCER[2]-generated repair classes have been used in MACER[6]. They have investigated multi-class and multi-label learning tasks, such as hierarchical classification and reranking algorithms, for the challenge of code repair.The six modules that make up MACER[6] are outlined below.

- **Repair Lines :** This module aids in locating lines of the source code that require repairing.
- **Feature Encoding :**Code abstraction has been carried out in this module for the specified lines to produce a feature vector with a 2239-dimension.
- **Repair Class Prediction :**Out of 1016 repair classes, the type of repair to be employed in the source code was selected using the feature vector.
- **Repair Profile prediction :**The position of the repair inside the source code has been predicted in this module using the feature vector.
- **Repair Application :** Predicted repairs have been used in this module at predicted sites.
- **Repair Concretization :** In this module, undo the code's abstraction before compiling it to verify that the repair was successful or not.

8.4 Dr Repair:

- The proposed Graph Attention-based network and compiler diagnostic feedback are used to correct compiler errors. It encodes diagnostic compiler feedback and flawed source code as inputs, then decodes the error line and fixed code using LSTM and graph attention models.
- All of the source code identifiers and diagnostic parameter symbols were taken into consideration as nodes, and it was suggested that instances with the same symbols that characterise sematic behaviour be connected.Then, they created a neural network model using a graph attention mechanism that could simulate the process of tracking symbols.
- By employing a feedforward network and a pointer-generator decoder, they were able to model the likelihood that a line would be incorrect. Training is more time-consuming and requires a hugely powerful equipment.The hardest part is fine-tuning the hyperparameter so that it can concentrate on particular kinds of faults.
- They have suggested a method to produce more fictitious training data. Based on three scenarios—beginner programmer, professional programmer, and predicted code—they have examined compiler faults.They have investigated several datasets based on these cases and divided them into four main groups, including syntax, ID-type, keyword, and ID-typo. Following that, they looked into codeforces websites to gather more syntactically sound programmes and looked into probability distribution to produce additional source-target pairs.

8.5 SampleFix:

- The method that SampleFix[9] suggested was primarily concerned with the diversity and ambiguity of source code. They suggest a one-to-many mapping between the source and target programmes in order to fix the flawed programme because we know that different types of code may generate the same result.
- A conditional variational autoencoder is trained to offer efficient fixes to flawed programmes, and after interacting with the compiler, the repair's correctness is assessed. The regularizer which penalises the repairs has increased the effectiveness of the fixes.

8.6 DECAF:

- For challenges involving multi-label categorization, DECAF[12] introduced an XML technique. According to this technique, each data point is given a specific subset of labels from the overall huge collection of labels. When we have less training data points, collaborative learning is employed and this label metadata is utilised.
- According to the one-vs-rest methodology, they have learned a unique linear classifier for each label in the DECAF[12] paper. These classifiers use the label's metadata and offer helpful data.
- DECAF[12] developed initialization strategies for categorization that let them improve accuracy, particularly when there are less labels, and scale up the XML workloads with large numbers of labels.
- They introduce the following three topics: A lightweight text embedding block generated distinct one-vs-rest classifiers for each label and a shortlisted that offers high recall for data points for short-text applications.

8.7 MACER++:

- The MACER[6] approach, which was previously discussed, has been improved with MACER++. On MACER++, Fahad[13] and Sharath[10] have collaborated. This offers improvements to a few MACER[6] modules while essentially maintaining the same pipeline overall. Since the pipeline is essentially unchanged, it provides all of MACER's[6] advantages while also being more efficient and helpful.
- Basically, it presents a brand-new method for producing synthetic data. Using existing data (source programmes), the synthetic data creation technique creates fresh samples with realistic errors that match those committed by inexperienced programmers. Considering how little of the Python dataset we have to work with there is, this is also really helpful.
- MACER++ has been developed for C programming languages and dataset was obtained from TRACER[1].

8.8 PyMACER:

- MACER++ was performing better and becoming more well-known. The development of an interactive and practical application that informs students of Python programme mistakes was carried out.

- As mentioned above, the MACER++ pipeline was developed for the C programming language. PYMACER is a Python programming language implementation that uses a machine learning model as its backend and follows the MACER++ pipeline.

CHAPTER 9

Proposed Approach

Contents

A light machine learning model that identifies the types of repairs that could be made as well as the locations where they should be made has been utilised by MACER[6] to describe the methods of predicting repair for the incorrect programme in six different modules. In terms of precision of repair, MACER's[6] performance is more encouraging than that of earlier approaches, and it also requires fewer resources.

Full compiler diagnostic error messages are not used by MACER[6]. They merely use the line number that the compiler provides. Machine learning model performance may be improved by using the knowledge included in compiler diagnostic error signals, as demonstrated by DrRepair[14]. The problem with MACER[6] is that zero shot cases cannot be repaired using it. Zero shot cases are those in which the machine learning model cannot make a repair because they are not included in the training dataset. Additionally, MACER[6] does not make use of potentially useful information offered by the repair classes. We have made an effort to investigate some of the concerns raised above and also to enhance the repair class prediction.

9.1 Analysis of Repair Class of MACER:

In order to remove compilation errors from the abstracted source code, repair classes combine the insertion and deletion of tokens. It is one of MACER's[6] modules. We examined a few of the repair classes discovered by MACER[6], which are briefly stated below:

- **Case 1: Incorrect position of the ; in a line:** From the Table 9.1, we can see that there is same type of error in all the three examples such as the semicolon is present before the bracket. After analysis, it seems that the repair class depends on the type of bracket. As for common and square bracket is insertion and deletion of bracket whereas for curly braces, it is insertion and deletion of semicolon. It seems that these types of errors are handled by using multiple repair classes.

Source Line	**Target Line**	**Repair class**
printf("%c",a1[i];)	printf("%c",a1[i]);	Delete:) Insert:)
else { return b};	else { return b;}	Delete: ; Insert: ;
str[j]=str[j+(count/2)+1;]	str[j]=str[j+(count/2)+1];	Delete:] Insert:]

Table 9.1: Example of incorrect position of semicolon(;)

- **Case 2: Invalid Token Identification:** As we can see in Table 9.2, the repair could be deletion of ! and insertion of a backslash symbol (\ ) and there is no need of deletion of this token "); and insertion of again) and ; as individual token. Hence in this case while tokenization, "); is identified as single token which is invalid if we consider the rules of C programming language.

Source Line	**Target Line**	**Repair class**
printf (" \\\" " ! ");	printf (" \\\" \" ") ;	- ! - "); +) + ; + \"

Table 9.2: Example of Invalid token identification

- **Case 3: Unbalanced Parentheses:** Another frequent repair class was formed due to unbalanced parentheses which leads the parser to consider valid function calls as invalid tokens. In Table 9.3, we can see both text and abstracted form of the source and target line.If we keep each token in stack then we can clearly see that ceilf function is valid function and printf is invalid if one bracket is missing. However, it is identifying valid function as invalid and vice versa.

Source Text/Abstracted	Target Text/Abstracted	Repair class
printf("%f",ceilf(rad); printf ("%f " , TypeKind.INVALID TypeKind.FLOAT) ;	printf("%f",ceilf(rad)); printf (" %f " , TypeKind.FUNCTIONCALL (TypeKind.FLOAT)) ;	- TypeKind.INVALID + TypeKind.FUNCTIONCALL +)

Table 9.3: Example of repair class of unbalanced parantheses

- **Case 4: Same Insertion and deletion:** Another frequent repair class was formed due to deletion and insertion of same type of token. The example has been shown for this type of case in Table 9.4. As we know that MACER[6] has introduced three type of operation such as insert,delete and replace which was performed for repairing the source code , hence if we could introduce one more operation named as shift. This shift operation should be performed when we have same deletion and insertion and it could provide repair by shifting that token either by left or right then there is no need to perform deletion and insertion of same token.

Source Line	**Target Line**	**Repair class**
if else (TypeKind.INT != TokenKind.LITERAL_INT)	else if (TypeKind.INT != TokenKind.LITERAL_INT)	- else + else
TypeKind.POINTER = (* int) malloc ((TypeKind.INT + TypeKind.INT) * sizeof (int)) ;	TypeKind.POINTER = (int *) malloc ((TypeKind.INT + TypeKind.INT) * sizeof (int)) ;	- int + int

Table 9.4: Example of repair class of same insertion and deletion

- **Case 5: Repair of First token:** There are approximately 289 data points in our datasets where the error is in only the first token, according to our analysis of repair classes. These token are the reserved words of the C programming language. In Table 9.5, we can see

Source Text	Source Abstracted Form
whileb (i = 10)	"TypeKind.FUNCTIONNOPROTO (TypeKind.INT = TokenKind.LITERAL_INT)

Table 9.5: Example of repair class having error in first token

that whileb is identified as TypeKind.FUNTIONNOPROTO which means it is considering whileb as function name. However if we could extract first token and correct it then this source could compile successfully.

9.2 Changes made in Abstraction:

The source code in MACER[6] has been abstracted, and error lines provided by the compiler have been retrieved from the abstracted code. The source lines include a number of user-defined literals and identifiers but are not helpful for error correction. It is typical in the literature to lower the input vocabulary size in order to prevent saturating the machine learning approach with these meaningless tokens.This is accomplished by MACER[6] by maintaining keywords and symbols while substituting literals and identifiers by their associated abstract LLVM token types.

By taking into account its type information, MACER[6] is substituting literals and identifiers. An exclusive token named INVALID is used to replace unrecognised identifiers. For example, the expression double a=4.5; will have the abstracted form as double VARIABLE DOUBLE = LITERAL DOUBLE;.

After analysing the repair classes, we have conducted experiment to reduce the vocabulory size to some extent. This attempt is performed by removing the type consideration used by MACER[6] at the time of substituting literals and identifiers. We have replaced the all literals by special token called LITERALS and replaced all identifiers by special tokens called NON_NUMERIC and NUMERIC. For example whether the identifier is int or double , it will be replaced by NUMERIC token. We have trained the model and tested on TRACER[1] dataset, however there was no improvement in the accuracy. The accuracy was similar to MACER[6] accuracy.

9.3 Changes made in Tokenization:

We discovered problems in the tokenization and abstraction steps after analysing the repair class in section 9.1. In the tokenization stage, we attempted to give some modifications. Lightweight tokenization has been employed. For the C programming language, we have created lexer rules. ANTLR was used to write these lexer rules. ANTLR is a robust parser generation tool that processes, executes, and translates binary or structured text files.

Rules for doing Tokenization:

- All reserved word,format specifiers,escape sequence and operators from C programming language should be identified as single token.
- All other terms, excluding those stated in first point , are tokenized as characterwise.
- With the exception of situations when the second operator is a unary operator, there should never be any whitespace between two operators.

Lexer helped us remove whitespaces by identifying them as unknown due to the inconsistent spacing in the source line token. For example, the statement *if(num != = 0)* will be tokenized using lexer rules as ['if','n','u','m','!==','0',')'] .Lexer rules were used to generate tokens, which we then used to create bigrams and apply the sliding window technique on.

To locate the token that has to be added or removed to make the source code accurate, we used the sliding window technique.Let's examine the sliding window method for windows with a size of two. Let's look at an illustration where the source = 'abcd' and target = 'abd'. The example makes it abundantly evident that the 'c' token needs to be deleted. As the window size is two for sliding window hence token for source will be ['ab','bc','cd'] and for target will be ['ab','bd']. After comparing the source and target, we have found that ['bc','cd'] token from source is not present in target token list and last term of 'bc' is same as first term of 'cd',hence if we could delete the common term and combine it with similar window size of two then we will get similar token list as target. If we have this common term in target token list then, insertion need to be performed.However, for window sizes greater than two, the same concept for discovering insertion and deletion did not function effectively. As a result, we have abandoned the sliding window method and instead used the Sequence Matcher algorithm of difflib library of Python to match the tokens.

Our vocabulary size has expanded after training the model with the adjustment in the tokenization stage, which includes roughly 4k models trained as bigrams which was previously around 2k. We didn't acquire any encouraging outcomes from this experiment. As we were getting many spurious bigrams which reduces the accuracy of repair.Let's use the word "num" as an example to understand about spurious bigrams. Since it is not a reserved word so it will follow characterwise tokenization, the unigrams and bigrams for this word will be ['n','u','m','nu','um']. Three unigrams and two bigrams were produced from this, however they are useless since they will confuse the machine learning model and act as noise.The spurious bigrams is the reason to remove characterwise tokenization. However, characterwise tokenization has removed some of the issue from repair class discussed in section 9.1.

After this attempt, we have set new rules for tokenization. We will be removing whitespaces between two operators, with the exception of situations when the second operator is a unary operator. The first token, which is mostly a reserved token from the C programming language, will be corrected from the source code. We discovered about two hundred data points from the datasets with only the first token being incorrect.

We have taken the first token from the source text line and used the Python NLTK package to run a similarity check against the list of reserved words for the C programming language in order to rectify the first token of the incorrect line. The correct reserved term has been substituted in the source text line if the similarity score of the token is more than 0.8, which strongly indicates that it is a reserved word. Following the first token's correction, we invoked the abstraction function, which transformed the source text line into an abstracted form. Finally, using the exception rules, we eliminated the whitespaces between the operators, completing the tokenization process. Following that, we used the MACER[6] pipeline to train and test the model using the TRACER[1] dataset.

CHAPTER 10

Experiments and Results

Contents

We will examine the experiments that we have conducted for the optimizing the modules of MACER[6] in this chapter and their accompanying findings.

10.1 Tune the Hyperparameter:

This experiments has been conducted on MACER[6].After analysing the repair classes, we discovered that several of them have lengths of more than one hundred.Hence we have performed experiment by tuning the length of the repair classes and run the model on TRACER[1]dataset at different pred@K value. Pred@K is the proportion of a model's programmes that successfully matched the target code. K indicates that the number of repairs anticipated by the model were taken into consideration.

Length of repair class	No.of data points	Pred@5	Pred@8	Pred@10
4	3791	0.616	0.629	0.633
10	713	0.662	0.67	0.673
20	150	0.65	0.66	0.664
30	54	0.636	0.65	0.652

Table 10.1: Results of tuning the length of repair classes

In Table 10.1,we have shown the results of tuning the length of repair classes of MACER[6].Here length of repair class means we have restricted the length of repair class and we have removed those repair classes from the diffset who has more length then the specified one. Number of data points defines the data points from the dataset which has more length then the specified one. Though we didn't get any promising results with this experiment as the accuracy at Pred@5 mentioned in MACER[6] paper is 0.693.

10.2 Changes made in Tokenization

After examining the repair class, we found issues in the tokenization and abstraction phases.We have made some rules for tokenization.With the exception of circumstances when the second operator is a unary operator, whitespace will no longer be allowed between two operators. We will correct the first token from the source code, which is mostly a reserved token from the C programming language. About two data points were extracted from the databases, and only the first token was inaccurate.

To repair the first token of the wrong line, we have taken the first token from the source text line and performed a similarity check against the list of reserved words for the C programming language. If the similarity score of the token is more than the threshold value, which strongly suggests that it is a reserved word, the correct reserved term has been substituted in the source

Dataset	Pred@5		Rep@5	
	Old	New	Old	New
TRACER	0.69	0.65	0.79	0.75

Table 10.2: Comparison of Results after making changes in tokenization stage

text line. After adjusting the first token, we used the abstraction function to turn the original text line into an abstracted version. The tokenization procedure was finally finished by removing the whitespace in between the operators using the exception rules. Following that, we trained and tested the model using the TRACER[1] dataset using the MACER[6] process. We have shown the comparison of results from the original MACER[6] after making changes in tokenization stages. However, we have not received any promising results.

Bibliography

[1] Umair Z. Ahmed, Pawan Kumar, Amey Karkare, Purushottam Kar, and Sumit Gulwani. Compilation error repair: for the student programs, from the student programs. *International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, 2018.

[2] Umair Z. Ahmed, Renuka Sindhgatta, Nisheeth Srivastava, and Amey Karkare. Targeted example generation for compilation errors. *In 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 327–338, 2019.

[3] Mayank Bansal. Intelligent program analysis and program indexing - iii. M.tech. thesis, Indian Institute of Technology Kanpur, 2022. unpublished thesis.

[4] Eli Bendersky et al. Pycparser: https://github.com/eliben/pycparser.

[5] Debanjan Chatterjee. Intelligent program analysis and program indexing - i. M.tech. thesis, Indian Institute of Technology Kanpur, 2022. unpublished thesis.

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

[7] 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.

[8] Rahul Gupta, Soham Pal, Aditya Kanade, and Shirish Shevade. Deepfix: Fixing common c language errors by deep learning. *In 31st AAAI Conference on Artificial Intelligence (AAAI)*, pages 1345–1351, 2017.

[9] Hossein Hajipour, Apratim Bhattacharyya, and Mario Fritz. Samplefix: Learning to correct programs by sampling diverse fixes. *CoRR*, abs/1906.10502, 2019.

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

[11] Kalina Jasinska, Krzysztof Dembczynski, Róbert Busa-Fekete, Karlson Pfannschmidt, Timo Klerx, and Eyke Hullermeier. Extreme f-measure maximization using sparse probability estimates. *In 33rd International Conference on Machine Learning (ICML)*, pages 1435–1444, 2016.

[12] Anshul Mittal, Kunal Dahiya, Sheshansh Agrawal, Deepak Saini, Sumeet Agarwal, Purushottam Kar, and Manik Varma. Decaf: Deep extreme classification with label features.. *In Proceedings of the Fourteenth ACM International Conference on Web Search and Data Mining (WSDM '21), Virtual Event, Israel. ACM, New York, NY, USA*, 2021.

[13] Fahad Shaikh. Advancements in ai-assisted compilation error repair and program retrieval. M.tech. thesis, Indian Institute of Technology Kanpur, 2021. unpublished thesis.

[14] Michihiro Yasunaga and Percy Liang. Graph-based, self-supervised program repair from diagnostic feedback. *Proceedings of the 37 th International Conference on Machine Learning, Vienna, Austria, PMLR 119.*, 2020.