Intelligent Program Analysis and Program Indexing - I

A Thesis Submitted

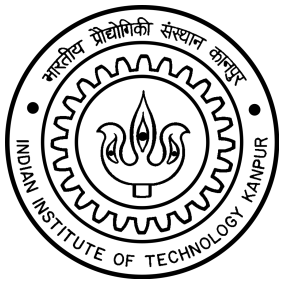
In Partial Fulfillment of the Requirements

For the Degree of M.Tech.

by

Debanjan Chatterjee

20111016



to the

Department of Computer Science and Engineering

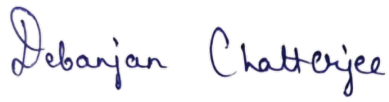
Indian Institute of Technology Kanpur

May, 2022

Declaration

This is to certify that at the thesis titled **"INTELLIGENT PROGRAM ANALYSIS AND PROGRAM INDEXING - I"** 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: Debanjan Chatterjee (20111016)
Program: M.Tech.
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur, Kanpur, 208016.
May, 2022

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 - I"** 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).

Name: Debanjan Chatterjee
Program: M.Tech.
Department of Computer Science and Engineering
Roll No.: 20111016
Indian Institute of Technology Kanpur, Kanpur, 208016.
May, 2022

Page intentionally left blank

20111016

Declaration

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

Debanjan Chatterjee

Name: Debanjan Chatterjee (20111016)
Program: M.Tech.
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur, Kanpur, 208016.
May, 2022

Page intentionally left blank

Abstract

Name of student: **Debanjan Chatterjee** Roll no: **20111016**
Degree for which submitted: **M.Tech.**
Department: **Department of Computer Science and Engineering**
Thesis title: **Intelligent Program Analysis and Program Indexing - I**

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

In recent years, the application of Machine Learning and Artificial Intelligence in the field of education has observed immense growth. Numerous AI-assisted tools are constantly being developed to be used in pedagogical settings. PRIORITY is one such tool indented to assist the problem setters for ESC101, an introductory programming course offered at IIT Kanpur.

PRIORITY provides a web portal where users can search for programming questions from past offerings of the ESC101 course. It relies on machine learning algorithms to provide an automatic solution to labeling each programming question, such that they can be made searchable.

There are three main components to the back-end machine learning architecture that enable PRIORITY to label programming problems automatically. The programming questions are comprised of the problem statement and the solution C program. Hence, extracting features from such data is a non-trivial feature engineering problem. The other two components are predicting the programming labels and the difficulty score of the problem.

This thesis proposes several changes to the back-end machine learning architecture of PRIORITY across all the three major components. We propose to use a more robust set of features. These features have been extracted from the Abstract Syntax Trees and the Function Call Graphs of the solution C programs. Various tree traversal and graph algorithms have been used to perform feature extraction.

For the label prediction task, we use a filter-based supervised feature selection algorithm using normalized mutual information score to train a one-vs- all logistic regression model. We adopt a ranking-style approach during prediction to ensure that our model predicts only the relevant labels.

We utilize ensemble learning techniques to solve the problem difficulty score prediction task. We have used a Mixture of Experts (MOE) model to combine the predictions of the best performing regressor and classifier models. We propose a further optimization where we suggest using the intermediate predictions of the MOE model to train a neural network model.

Acknowledgments

First and foremost, I want to express my sincerest gratitude to Prof. Purushottam Kar and Prof. Amey Karkare, my thesis supervisors, for their constant guidance and support. Without their mentorship and encouragement, it would have been impossible for me to complete this thesis.

Secondly, I want to express my gratitude to Mayank Bansal and Preeti Singh, with whom I've had numerous discussions and brainstorming sessions. I would also like to thank my seniors, Fahad Shaikh and Sharath HP, for mentoring me throughout the past year. I am tremendously grateful for their patience and constant support.

I would also like to express my gratitude to all of the professors in the CSE Department for providing me with invaluable technical knowledge throughout my time at IIT Kanpur. The lessons that I learned here have enhanced my technical expertise and helped me improve as a human being.

I would also like to thank all my friends and batchmates who have been an integral part of my academic journey during M.Tech.

Lastly, I want to thank my mother for always believing in me and supporting me in all my endeavors throughout my life.

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.

Debanjan Chatterjee
May, 2022

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Contents

Nowadays, Artificial Intelligence and Machine Learning are being used to solve a variety of real-world challenges across a variety of fields. In the field of education, several Intelligent tools are constantly being developed to enhance the process of teaching.

ESC101 is the introductory course in programming offered by the Indian Institute of Technology, Kanpur, every semester to a large number of students. One major challenge faced by such a large course, which is being taken by several hundreds of students, is the re-usability of programming questions or tasks. PRIORITY [10] [6]is a web portal that was developed to provide problem setters of the course access to a collection of a large number of programming questions from previous offerings of the course.

PRIORITY provides a search index over the entire collection of programming problems. Users can search for programming questions based on the programming concepts or skills required to solve the question and the problem's difficulty level. Figure 1.1 shows the problem/programming question search page in the PRIORITY web portal.

PRIORITY has a total of around two thousand programming questions. Now to index this set of programming questions, each problem must be labeled based on the search parameters that are allowed in the web portal. Thus, each programming question must be annotated with the programming skills label as well as its difficulty score label. Annotating such a large corpus is a cumbersome task; thus, PRIORITY relies on machine learning algorithms to label the entire corpus of problems. We refer to the collection of programming questions as the PRIORITY data.

Initially, some past tutors (problem setters for the ESC101 course) were requested to manually annotate a small portion of the PRIORITY data. Therefore, we have a small portion of labeled data points (programming questions) and a large portion of unlabeled data points.

[10] proposes two separate machine learning tasks to label the unlabeled data. The first task is to develop ML models to predict the programming concepts/skills (referred to as the programming labels) associated with a problem. The second task is to develop ML models to predict the problem's difficulty score.

In order to use machine learning algorithms on the PRIORITY data, one major challenge is performing appropriate feature engineering. Each programming question needs to be transformed into a careful selection of quality features that can help in learning the above-mentioned predictive tasks.

In this thesis, we propose several changes to the back-end Machine learning architecture of PRIORITY across all the three major components: i.e. feature engineering, programming label prediction, and difficulty score predictions. The results show that the proposed changes have led to

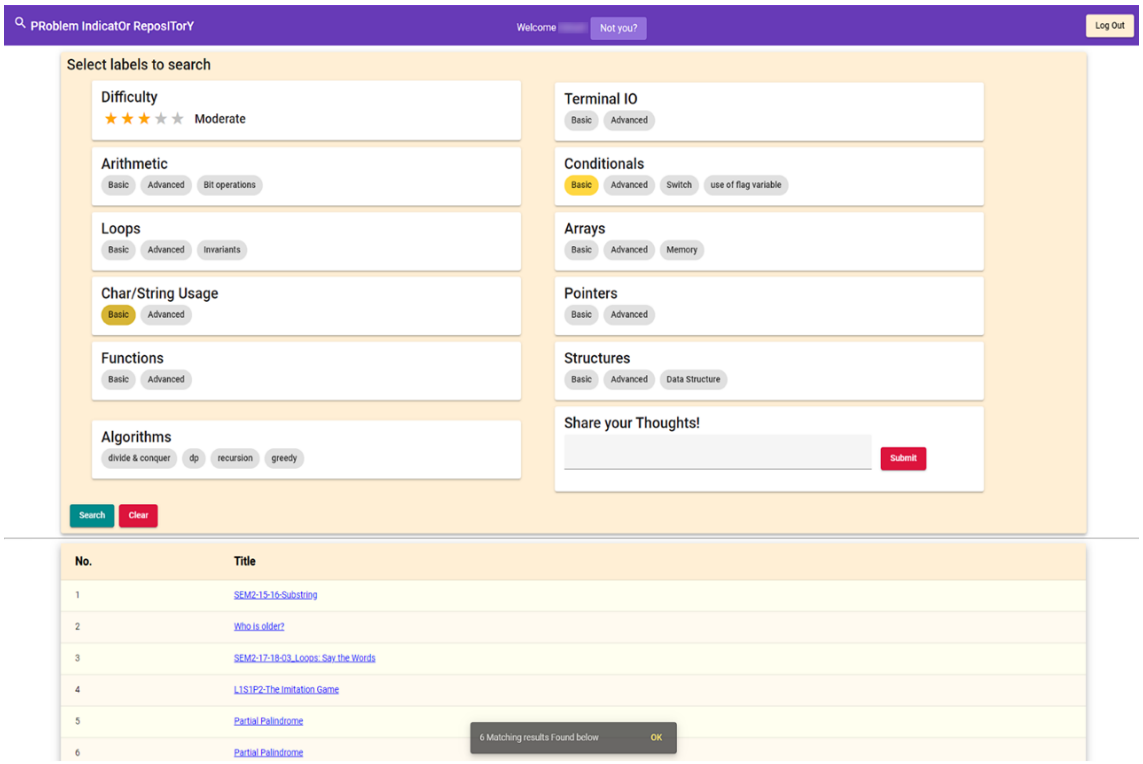


Figure 1.1: Problem/Programming Question search page in the PRIORITY portal. Image Credit [6]

.

a significant boost in the performance of the two prediction tasks. These proposed improvements have been described briefly in section 1.1.

1.1 Our Contributions

- In the previous version of PRIORITY, the Abstract Syntax Tree (AST) of the solution C program of the programming questions was used to perform feature extraction. The extracted features were simple bag-of-words style features. This thesis proposes to use a more robust set of features. We have extracted features using more elaborate tree traversals of the AST. We have also generated the Function Call Graphs of the C programs (solution code) from the ASTs and extracted some interesting features from the call graph. These new features were carefully chosen to help identify the programming labels and the difficulty scores of the problems. The complete description of the newly proposed features is available in chapter 2.

- Previously, the task of Label Prediction was solved using a Balanced Random Forest Classifiers along with a semi-supervised learning technique called Label Propagation [13]. In this thesis, we perform filter-based supervised feature selection on the new features to train a One-vs-All Logistic Regression model. We use a ranking-style approach during prediction. The newly proposed method outperforms the previously used technique, as seen in chapter 3.

- The task of difficulty score prediction was previously modeled as a multi-class classification problem. This thesis explores several regression techniques to predict difficulty score. This thesis proposes utilizing ensemble learning techniques to solve this task. We have used a Mixture of Experts (MOE) model to combine the predictions of the best performing regressor and classifier models. We propose a further optimization to the MOE model, where we suggest using the intermediate predictions of the MOE model to train a neural network model. The newly proposed method for this task has been described in chapter 4 .

CHAPTER 2

PRIORITY: Featurization

Contents

As discussed in chapter 1, PRIORITY is a web portal that provides a search index over a large corpus of programming questions. This large corpus of programming questions makes up the PRIORITY data. There is a problem statement for each programming question and a solution C program.

PRIORITY allows users to search for programming questions using both programming labels as well as difficulty scores. Now a large portion of the dataset is unlabeled. Hence it is necessary to train machine learning models to perform these prediction tasks to assign labels and difficulty scores to the unlabeled data.

If we apply machine learning to predict the labels for the unlabeled portion of the PRIORITY data, the programming questions will act as the train and test data points. Therefore, it is necessary to represent each programming question as a feature vector.

In [10], to extract features from the programming question, the problem statement has completely been ignored since they tend to contain unnecessary information as the problem setters often describe the problem statement indirectly using a story.

[10] have used only the solution C program for feature extraction. They are extracting several bag-of words-like features from the Abstract Syntax Tree of the solution C program. From here onwards, we will refer to this feature representation as old features.

As already discussed in chapter 1, PRIORITY has a very small amount of labeled data. Learning predictive models using such limited training data is difficult even for very powerful machine learning algorithms. One approach to improve the performance of machine learning models is to extract better features.

This thesis proposes a new set of features that can be extracted from the solution C program using its Abstract Syntax Tree (AST). A tool named Pycparser[2] has been used to generate the

AST of the programs. Several features have been carefully chosen to provide the necessary information, which may help identify the programming labels and difficulty scores of the problem. These features have been discussed in detail in the following sections.

2.1 Unigram Features

Every solution code, i.e., C program, can be converted into its Abstract Syntax Tree representation. The AST being a tree, will contain several nodes. These individual nodes in the AST are referred to as unigrams. These nodes are of certain node types. Pycparser supports forty-nine node types in total. The first type of feature that we extract is a one-hot unigram feature encoding of the node types present in the AST.

```
1 int main(){
2     int a=5;
3 }
```

Figure 2.1 demonstrates the AST of the above sample C program (let us call it Sample C Program 1).

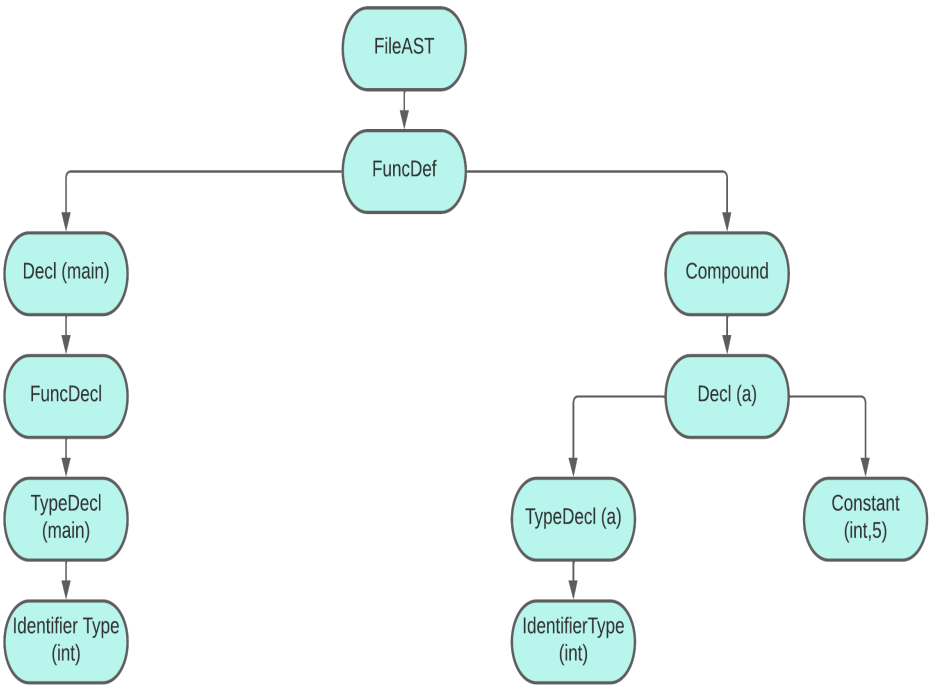


Figure 2.1: AST of of Sample C Prgoram 1.

The unigram node-types of the AST shown in Figure 2.1 are:- 'FuncDecl', 'FileAST', 'FuncDef', 'Compound', 'IdentifierType', 'Constant', 'TypeDecl', 'Decl'. The one-hot encoding of the unigram node-types will give a feature vector of size 49. It will take a value of 1 for the unigram node-types that are present in the AST, and 0 for the absent node-types. To extract these unigrams, we simply need to do a traversal of the AST.

2.2 Parent-Child Bigram Features

Similar to how we extract the unigram features in section 2.1, we can do a tree traversal on the AST of each programming question's solution C code to generate Parent-Child bigrams. In a tree, all the (parent, child) pairs are referred to as Parent-Child bigrams. To illustrate this, let us consider a simple tree where there are only three nodes: A, B, and C. A is the root node, with two children, B and C. Here, (A, B) and (A, C) will be the Parent-Child bigrams of the tree. For the above sample C program and it's corresponding AST in Figure 2.1, we get the following Parent-Child bigrams:

('FuncDecl', 'TypeDecl'), ('FuncDef', 'Compound'), ('FuncDef', 'Decl'), ('Compound', 'Decl'), ('TypeDecl', 'IdentifierType'), ('FileAST', 'FuncDef'), ('Decl', 'TypeDecl'), ('Decl', 'FuncDecl'), ('Decl', 'Constant').

The one-hot encoding of the Parent-Child bigram node-types will give a feature vector of size 2401. It will take a value of 1 for the parent-child bigram node-types that are present in the AST, and 0 for the absent bigram node-types.

2.3 Parent-Grandchild Bigram Features

Like Parent-Child Bigrams, we can do a similar tree traversal on the AST of each programming question's solution C code to generate Parent-Grandchild bigrams. In a tree, all the (parent, grandchild) pairs are referred to as Parent-Grandchild bigrams. To illustrate this, let us consider a simple tree where there are only seven nodes: A, B, C, D, E, F, and G. A is the root node, with two children, B, and C. B has D and E as its children, and C has F and G as its children. Here, (A, D), (A, E), (A, F), and (A, G) will be the Parent-grandchild bigrams of the tree.

For the above sample C program and it's corresponding AST in Figure 2.1, we get the following Parent-Grandchild bigrams:

('Decl', 'IdentifierType'), ('FileAST', 'Compound'), ('FileAST', 'Decl'), ('FuncDef', 'FuncDecl'), ('FuncDecl', 'IdentifierType'), ('Compound', 'TypeDecl'), ('Compound', 'Constant'), ('Decl', 'TypeDecl'), ('FuncDef', 'Decl').

The one-hot encoding of the Parent-Grandchild bigram node-types will give a feature vector of size 2401. It will take a value of 1 for the parent-grandchild bigram node-types that are present in the AST, and 0 for the absent bigram node-types.

2.4 Maximum Same Node-type Depth Features

In the Abstract Syntax Tree of a C program, we can extract richer features than just extracting the unigram node types. We can consider the frequency of occurrence of each node type in the AST. However, taking the overall frequencies will not tell as much about the syntactic structure of the program. Therefore, we propose to extract non-trivial count features in this section and section 2.5.

In an AST, for each subtree rooted at a particular node type, we can count the number of times that node type occurs from the root of the subtree to the leaves. We take only the maximum of such counts for each node type.

```
int main(){
    int a=5+3-2*8;
}
```

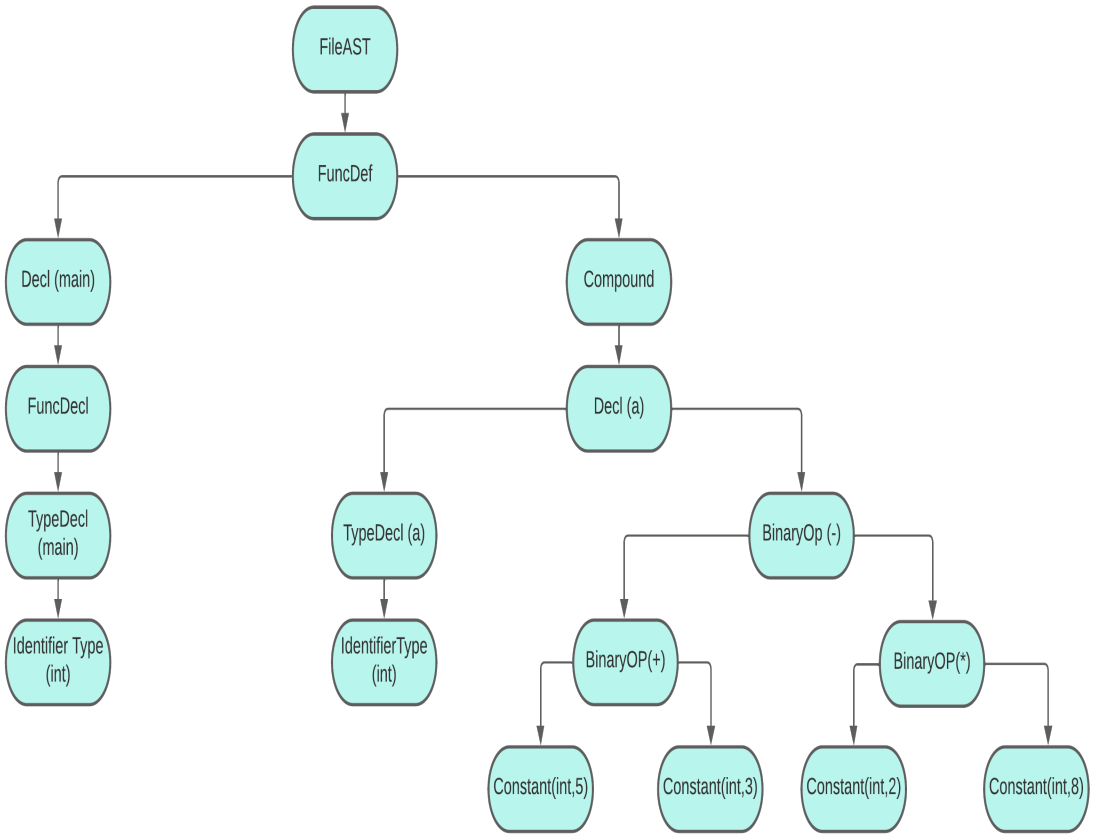


Figure 2.2: AST of of the above Sample C program 2.

For the above code snippet(Sample C Program 2), the corresponding AST is given in Figure 2.2. The maximum same node type depth features for this AST, will be:- ('BinaryOp': 2), ('Compound': 1), ('Constant': 1), ('Decl': 1), ('FileAST': 1), ('FuncDecl': 1), ('FuncDef': 1), ('IdentifierType': 1), ('TypeDecl': 1). These are expressed as (node type name: count) pairs.

Since there are a total of forty-nine node types, these types of features will generate features of dimension 49. For each node type that is present, we consider the count value discussed here, and for the absent node types, we take a value of 0.

2.5 Maximum Same Node-type Sub-tree Count Features

In an AST, for each subtree rooted at a particular node type, we take the frequency of occurrence of that node type in the subtree. We take only the maximum of such counts for each node type.

For the above code snippet(Sample C Program 2), the corresponding AST is given in Figure 2.2. The maximum same node type subtree count features for this AST, will be:- ('BinaryOp': 3), ('Compound': 1), ('Constant': 1), ('Decl': 1), ('FileAST': 1), ('FuncDecl': 1), ('FuncDef': 1), ('IdentifierType': 1), ('TypeDecl': 1). These are expressed as (node type name: count) pairs.

Since there are a total of forty-nine node types, these types of features will generate features of dimension 49. For each node type that is present, we consider the count value discussed here, and for the absent node types, we take a value of 0.

2.6 Boolean Recursion Check Feature

From the Abstract Syntax Tree of a C program, we can generate its function call graph. A function call graph is a directed graph that contains all the functions as nodes, and if an edge exists from one function A to another function B, it means function A is making a call to function B.

Generating the function call graph from the AST is pretty straightforward and can be done by tracking the FuncDef and FuncCall nodes. Once the function call graph is generated, we can check if recursion exists in a C program by checking the presence of a cycle in the directed function call graph. Cycle detection can be done using a depth-first search.

```
1  void a(int n)
2  {
3      if(n==0)
4          return;
5      return a(n-1);
6  }
7  int main()
8  {
9      a(5);
10 }
```

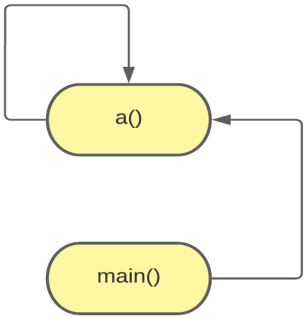


Figure 2.3: Function Call Graph of the above sample C program.

Figure 2.3 shows the function call graph of the above C program (let us call this sample C program 3). The di-graph contains a cycle as it should since the given program is recursive. Therefore, this binary recursion check feature will be 1. If there was no recursion, the binary feature would assume a value of 0.

2.7 Maximum Cycle Length in Function Call Graph

Along with having a binary feature to indicate the presence of recursion, we can also have a feature to count the maximum cycle length for the directed function call graph. This feature will help us

to track the presence of indirect or cross recursion in a C program.

Counting the maximum cycle length in a directed graph can also be solved using depth-first search.

```
void c()
{
    a();
}

void b()
{
    c();
}

void a()
{
    b();
}
int main()
{
    a();
}
```

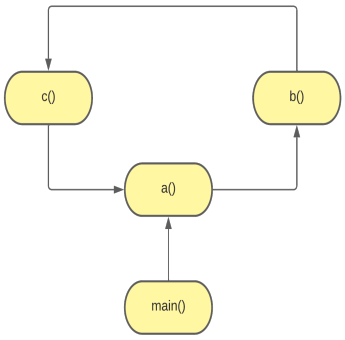


Figure 2.4: Function Call Graph of the above sample C program.

Figure 2.4 shows the function call graph of the above C program (let us call this sample C program 3). The di-graph contains a cycle of maximum length 3. Therefore, this feature will take a value of 3. Make a note, that there can be multiple cycles in the di-graph, this feature will take only the maximum length cycle value. If there is no recursion in the C program, this feature takes a value of 0.

2.8 Boolean Function Call Check Feature

From the function call graph of a program, we can also detect the presence of a function call made simply by checking if there is an edge in the di-graph. This is also a binary feature, as it takes a value of 1 if the program contains at least one function call and 0 otherwise.

2.9 The New Feature Set

To summarize this chapter, we have proposed a new way of extracting features for each programming question. We use the Abstract Syntax Tree and the Function Call Graph of the solution C programs to generate the features.

Our proposed feature set will represent each programming question as a 4952 dimensional feature vector. The first 49 dimensions will comprise the one-hot unigram encoding of the node types. It will be followed by two sets of 2401 dimensional features that make up parent-child and parent-grandchild bigrams. Next, there will be two sets of 49 dimensional features that are described in section 2.4 and section 2.5. Finally, there will be two binary (boolean) features to check recursion and 'check function call, followed by a count feature stating the maximum cycle length in the corresponding function call graph of the program. For the remainder of this thesis, we will refer to this feature representation as the 'new features'.

CHAPTER 3

PRIORITY: Label Prediction

Contents

The primary objective of the PRIORITY tool is to provide a search index over the extensive corpus of programming problems available from previous ESC101 course offerings. The future problem setters for ESC101 can search for specific problems based on the programming skills and concepts required to solve these problems and the programming question's difficulty. The problem setters can get to know the type of questions that were asked previously and use these as benchmarks to set future problems, thus making the task of problem setting very convenient.

In this chapter, we will take a look at how a search index be created over programming questions such that they can be filtered out using the specific programming concepts used to solve the questions.

The label prediction task is defined as predicting all the relevant programming concepts associated with a programming question given that problem. This task can be modeled as a multi-label classification problem. There are a total of 28 programming labels used. Over the course of this chapter, we will take a close look at all the programming labels used and how this task can be solved using different machine learning techniques, and all the challenges faced in solving this task.

3.1 Programming Labels

Past problem setters were asked to annotate a small portion of the entire corpus of programming questions available in the PRIORITY data. We will be referring to this as the labeled data. For each programming question, the annotators gave a set of programming labels that they felt were relevant.

The labels indicate the programming concepts that would be required to solve the programming question. There are a total of ten major labels and twenty-eight minor labels.

The major labels include TerminalIO, Arithmetic, Conditionals, Loops, Arrays, Char-String, Pointers, Functions, Structures, and Algorithms. Each major label was further split into minor labels. For example, Algorithms is divided into Divide and Conquer, Greedy, Recursion, and Dynamic Programming.

These minor labels were used to train the machine learning models. The PRIORITY portal can perform a search based on how many of these minor label tags are switched on. The complete description of all programming labels are given as follows.

- **TerminalIO Basic:** Problems requiring simple I/O (printf/scanf) statements.
- **Terminal Advanced:** Problems requiring advanced format specifiers for printf/scanf statements.
- **Arithmetic Basic:** Problems requiring simple use of arithmetic operators.
- **Arithmetic Advanced:** Problems requiring advanced use of arithmetic operators, complex expressions, explicit type casting, use of math.h, etc.
- **Arithmetic Bit:** Problems requiring use of bit-wise operators.
- **Conditionals Basic:** Problems requiring use of simple if/else conditional statements.
- **Conditionals Switch:** Problems requiring use of switch statements.
- **Conditionals Advanced:** Problems requiring advanced concepts such as use of ternary operators, nested if/else statements, etc.
- **Conditionals Flag:** Problems requiring use of flag variables.
- **Loops Basic:** Problems requiring simple iterative/loop statements.
- **Loops Advanced:** Problems requiring advanced looping concepts such as nesting, using jump statements, etc.
- **Loops In-variants:** Problems requiring use of non-trivial loop in-variants.
- **Arrays Basic:** Problems requiring use of simple one-dimensional arrays.
- **Arrays Advanced:** Problems requiring use of multi-dimensional arrays.
- **Arrays Memory:** Problems requiring concepts of memory management and involves usage of malloc(), calloc(), realloc(), sizeof(), free().

- **Pointers Basic:** Problems requiring basic use of pointers.
- **Pointers Advanced:** Problems requiring complex/non-trivial use of pointers such as arrays of pointers or pointers of pointers,etc.
- **Char-String Basic:** Problems requiring use of character or strings.
- **Char-String Advanced:** Problems involving advanced character/string-based concepts.
- **Functions Basic:** Problems requiring basic use of functions such as functions having one or more primitive parameters and primitive return types.
- **Functions Advanced:** Problems requiring basic use of functions such as functions having arrays, pointers, references as parameters and return types.
- **Structures Basic:** Problems requiring basic use of structures.
- **Structures Advanced:** Problems requiring advanced use of structures such as nesting of structures, pointers to structures, etc.
- **Structures DS:** Problems requiring usage of structures to implement data structures such as linked lists, stacks, queues, graphs, trees, etc.
- **Algorithms DC:** Problems requiring use of algorithms based on divide and conquer approach.
- **Algorithms Recursion:** Problems requiring use of algorithms based on recursion.
- **Algorithms Greedy:** Problems requiring use of algorithms based on greedy approach.
- **Algorithms DP:** Problems requiring use of algorithms based on dynamic programming.

3.1.1 De-duplication of Labels

Some programming questions were intentionally given to multiple problem setters(tutors) to label during the annotation stage. This was done to ensure consistency. Before using this labeled data to train some machine learning model, it is necessary to remove these duplicate data points. In order to remove the duplicates, the current version of PRIORITY [10] takes a union across all the label sets annotated by different tutors for a given problem. For example, if a particular programming question was labeled with (Conditionals Advanced, Algorithms Recursion) by tutor A and the same question was labeled with (Arrays Advanced, Algorithms Recursion) by tutor B. That problem will get ultimately labeled by the union of the two sets, i.e. (Conditionals Advanced, Arrays Advanced, Algorithms Recursion).

3.1.2 Class Imbalance problem

The quantity of labeled data points is significantly less. Figure 3.1 plots the frequency of occurrence of the programming labels. It can be observed from the plot that the labeled data suffers from a heavy class imbalanced problem. Moreover, the label frequencies follow a long tail distribution,

i.e., some popular data points have a significantly larger amount of data points (programming questions) associated with them. However, some labels have very few data points associated with them. The eight least frequent labels have less than 20 programming questions associated with them.

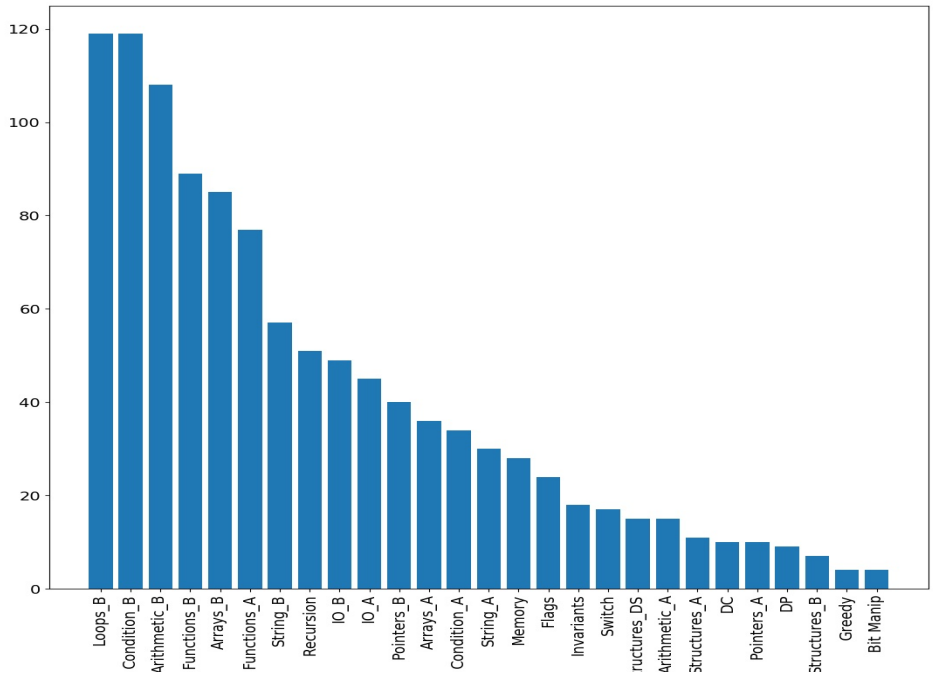


Figure 3.1: Labeled samples of each Programming Label. Image Credit [10]

Because of the severe class imbalance problem existing in the labeled dataset, it is expected to negatively affect the performance of the machine learning models, especially for the scarce labels. Some techniques

3.2 Feature Extraction

As discussed in chapter 2, PRIORITY used bag-of-words-like features previously. For the purpose of this thesis, for the task of label prediction, we propose to use the rich set of new features extracted, as described in chapter 2. Therefore, each programming question is represented by a feature vector of dimension 4952. These features have been carefully extracted from the Abstract Syntax Trees of the corresponding programs and should help identify the programming labels.

3.3 Feature Selection

For the task of label prediction, for each data point(programming question), we have a 4952 dimensional feature vector. Now, not all 4952 features will help identify each programming label. In fact, since the features are considered robust enough, only a small subset of features should be enough to identify the individual programming labels. It is fairly common that reducing the

number of features for a learning task tends to enchance the performance of the machine learning models.

Therefore, we propose a filter-based supervised feature selection technique. Only a subset of features is selected in filter-based feature selection depending on their relationship with the target variable. A correlation-type statistical measure is calculated between input features and target labels, and for each label, only the top-k highest scoring features are selected. Normalized Mutual Information Score [4] is used as the statistical measure.

To illustrate the feature selection algorithm in a more detailed matter, let us consider the label 'Algorithms Recursion'. Let there be n training samples; thus, we will have a vector of size $n \times 1$ corresponding to the 'Algorithms Recursion' label. Since we have 4952 features, we will have a feature matrix of $n \times 4952$. If we consider each feature separately, then we will have 4952 vectors of size $n \times 1$. We calculate the normalized mutual information score between the label vector and each vector corresponding to the 4952 features separately. Finally, we select the top-k features for which the normalized mutual information score was maximum. The same process is repeated for all the twenty-eight labels.

3.4 Machine Learning Models

Now that we have established a set of features for each of the programming labels, our next task is to apply machine learning techniques to develop models capable of predicting the programming labels given a problem.

Previously, PRIORITY used the old features as discussed in chapter 1 and chapter 2, to train a Balanced Random Forest Classifier and used a semi-supervised learning technique called Label Propagation to solve the task of Label Prediction [10].

This thesis proposes an alternative approach to predict the programming labels using supervised learning techniques. Our method is described in two sections. First, we look at how our model will be trained in subsection 3.4.1. After that, we will discuss the different techniques of how the labels could be predicted for a data point (programming question) in subsection 3.4.2. The complete pipeline of the programming label prediction task is given in Figure 3.2.

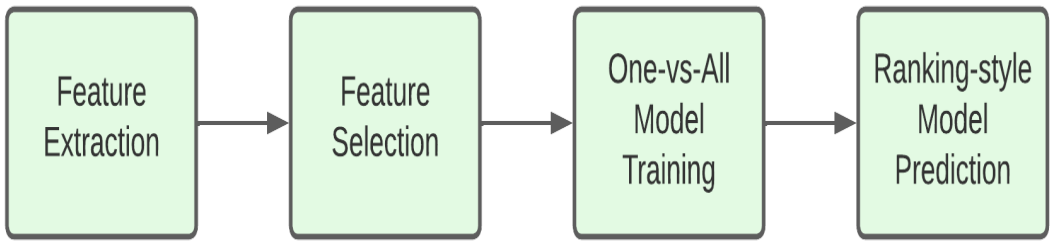


Figure 3.2: Complete pipeline for Label Prediction

3.4.1 Model Training

We will use the features selected for each programming label and train twenty-eight different binary classifiers for each label independently. Thus, we follow a one-vs-all (OVA) approach to solve the multi-label classification task during training. After the training stage, we will have twenty-eight binary classifiers, each capable of predicting a single label. We have tried out two classifiers for our experiments: Logistic Regression [7] and Linear Support Vector Machine. The Logistic Regression One-vs-All model showed superior performance, as explained in detail in subsection 3.5.2.

3.4.2 Model Prediction

Now for model predictions, we have explored two techniques. The first one is to predict the twenty-eight labels in a One-vs-All fashion as described in subsubsection 3.4.2.1. The second one is to perform ranking style prediction, which is explained in subsubsection 3.4.2.2.

3.4.2.1 Standard One-vs-All Predictions

The standard technique would be: given a programming problem, we ask the twenty-eight binary classifiers independently to predict if that label is present or not. This approach is referred to as the One-vs-All prediction style. This is a popular technique used to solve multi-label classification [11] problems. However, there is a significant drawback to using this approach. Since the labels assigned by annotators (tutors) for each programming question are for pedagogical purposes, the annotators would not have labeled each problem with all the satisfied programming labels. Instead, they would have labeled them with only a set of most relevant labels. One-vs-All prediction style is not equipped to tackle this problem since it naively predicts every label that a program would satisfy.

3.4.2.2 Ranking-style Predictions

The standard One-vs-All prediction style would lead to a high Recall score but a low Precision Score, as observed in subsection 3.5.2, as it naively predicts all satisfied labels. To enforce our model, to predict the most relevant set of programming labels, given a problem, we have proposed a Ranking-Style prediction approach. There is an associated problem of multi-label ranking in the literature on multi-label classification. Multi-label ranking [12] refers to when the model not only predicts the labels in a multi-label classification problem but also can rank the predicted labels based on relevance.

In our ranking-style prediction method, we first train the Logistic One-vs-All model as described in. During prediction, we take the probability score that the label is turned on for each of the twenty-eight labels, as predicted by the binary classifiers. We can now rank the predicted labels by sorting them based on the probability scores. It is assumed that this will give the relevant ordering for the labels. Finally, we only predict the top-p labels based on this relevance order. This is the standard ranking-style prediction approach.

We propose a further optimization to this method. A label-specific multiplier score can be used to scale the probability scores obtained for the labels. This will ensure that some labels are

given more significance than other labels. This can be intuitively explained as follows, let us take the labels 'TerminalIO_Basic' and 'Algorithms_Recursion' into consideration. If an annotator feels that a problem is testing the recursion skill, they will not label it with 'TerminalIO_Basic', despite the problem requiring printf()/scanf() statements. Therefore it makes sense to give more importance to 'Algorithms_Recursion' than 'TerminalIO_Basic' .

Hence using these label-specific multiplier scores will allow our model to mimic the labeling style of the annotators. There are two ways to decide on these label-specific multiplier scores. The first one is to learn these multipliers directly from the labeled data. This is an objective technique, and we have used two such ways. The second is a subjective technique where we have to use domain expertise. The techniques adopted by us are given as follows.

- **Objective1:** For each label, we use the count of its occurrence in the labeled data. We use a polynomial function $f(x) = x^{1/5}$,where x is the count value. Here, $f(x)$ serves as a quashing function. These values obtained for each label will become its multiplier during ranking.

- **Objective2:** For each label, we use the ratio of the total labeled samples to the count of its occurrence in the labeled data. We use a polynomial function $f(x) = x^{1/5}$,where x is the count value. Here, $f(x)$ serves as a quashing function. These values obtained for each label will become its multiplier during ranking.

- **Subjective:** Here, we look at the order in which the topics corresponding to the programming labels are taught in the ESC101 course, and assign the first label covered a multiplier score of 1. Each subsequent label is given a score of 0.05 higher than the previous label. Therefore, the second label taught gets a score of 1.05 and so on.

3.5 Experimentation and Results

In this section, we will discuss the various experiments that were carried out for the label prediction task and their corresponding results.

3.5.1 Feature Selection

3.5.1.1 The selected features

We are using a filter-based supervised feature selection algorithm where we are selecting the top-k features with the highest normalized mutual information scores.

Some of the selected features across the different programming labels make a lot of intuitive sense. For example, the top three scoring features for the label 'Algorithms_Recursion' are: the boolean recursion check feature, maximum cycle length in function call graph feature and the boolean function call check feature. All these features are necessary for the label 'Algorithms Recursion' to be turned on.

Similarly, the 'Arrays_Basic' label has a lot of unigram and bigram features have the node type 'ArrayDecl' turned on. The other labels also have some very meaningful features selected

as the highest scoring features. This indicate that the new features does indeed do a good job in being related to the programming labels.

3.5.1.2 Tuning the hyperparameter k

In our feature selection algorithm, we are selecting only the top-k features with the maximum normalized mutual information score for each label. Thus, k acts as a hyper-parameter in our proposed model. Hence it is essential that the appropriate value of k is chosen. We have experimented with various values of k. For determining the correct k value, we have compared various metrics like precision, recall, accuracy, and f1-scores of the logistic regression (OVA) model for different values of k. Table 3.1, Table 3.2, Table 3.3 show the scores of the various metrics considered for a k value of 5, 12, and 20 respectively. From the scores, it has been observed that the performance of the one-vs-all logistic regression is best when k=20. Hence, for all future experiments, it will be assumed that the value of hyper-parameter k is 20.

Program Label	Accuracy Score	Precision Score	Recall Score	F1 score
Arithmetic_Bit	0.9692	0	0	0
Algorithms_Greedy	0.9846	0	0	0
Arithmetic_Basic	0.7077	0.4375	0.4118	0.4242
Loops_Basic	0.6923	0.3889	0.4375	0.4118
TerminalIO_Advanced	0.6923	0.2857	1	0.4444
Arrays_Basic	0.7538	0.36	1	0.5294
Pointers_Advanced	0.9846	0.5	1	0.6667
Conditionals_Flags	0.9077	0.2	0.3333	0.25
Structures_Basic	0.9846	0.5	1	0.6667
Char-String_Basic	0.7692	0.2353	0.6667	0.3478
Conditionals_Basic	0.6923	0.3846	0.2941	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	0.9846	0.75	1	0.8571
Algorithms_DP	0.9231	0.1667	1	0.2857
Conditionals_Advanced	0.8308	0.1538	1	0.2667
Loops_Advanced	0.8923	0.7917	0.9048	0.8444
Arrays_Advanced	0.9538	0.7143	0.8333	0.7692
Char-String_Advanced	0.9077	0.1429	1	0.25
TerminalIO_Basic	0.7231	0.1818	1	0.3077
Pointers_Basic	0.7692	0.2105	1	0.3478
Structures_DS	0.9846	0.6667	1	0.8
Arrays_Memory	0.9538	0.5714	1	0.7273
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 Metrics	0.8659	0.3914	0.7399	0.512

Table 3.1: Metrics of Logistic Regression(OVA) model using top-5 features

Program Label	Accuracy Score	Precision Score	Recall Score	F1 score
Arithmetic_Bit	0.9692	0	0	0
Algorithms_Greedy	0.9846	0	0	0
Arithmetic_Basic	0.7692	0.5455	0.7059	0.6154
Loops_Basic	0.6615	0.4062	0.8125	0.5417
TerminalIO_Advanced	0.6923	0.2857	1	0.4444
Arrays_Basic	0.7538	0.36	1	0.5294
Pointers_Advanced	0.9385	0.2	1	0.3333
Conditionals_Flags	0.6769	0	0	0
Structures_Basic	1	1	1	1
Char-String_Basic	0.8	0.2667	0.6667	0.381
Conditionals_Basic	0.6769	0.4167	0.5882	0.4878
Functions_Advanced	0.7538	0.4828	0.9333	0.6364
Arithmetic_Advanced	0.9538	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	0.8615	0.1818	1	0.3077
Loops_Advanced	0.9077	0.8571	0.8571	0.8571
Arrays_Advanced	0.9231	0.5556	0.8333	0.6667
Char-String_Advanced	0.9077	0.1429	1	0.25
TerminalIO_Basic	0.8154	0.2143	0.75	0.3333
Pointers_Basic	0.8	0.2353	1	0.381
Structures_DS	0.9846	0.6667	1	0.8
Arrays_Memory	0.9692	0.6667	1	0.8
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 Metrics	0.8692	0.4058	0.8092	0.5405

Table 3.2: Metrics of Logistic Regression(OVA) model using top-12 features

3.5.2 ML Model Performance

3.5.2.1 One-vs-All Prediction

As already described in section 3.4, to solve the multi-label classification problem, training is done using a One-vs-All (OVA) approach. Predictions can also be made in the One-vs-All approach. Here we have experimented with two classifiers, namely a Logistic Regression OVA model and a Linear Support Vector Machine OVA model. The results for the Logistic Regression model and Linear SVM model have been given in Table 3.3,Table 3.5, respectively.

Note that here, we are using a simple supervised learning style. In the previous version of PRIORITY, a semi-supervised learning technique named Label Propagation was used with Balanced Random Forest as the base classifier. The results for this older model have been displayed in table Table 3.4.

The results from the three tables have been compared, and it has been observed that both the Logistic Regression one-vs-all model as well as the Linear SVM model have decent improvements in their overall metrics. The f1-score for the Logistic Regression one-vs-all model is the highest.

Program Label	Accuracy Score	Precision Score	Recall Score	F1 score
Arithmetic_Bit	0.9846	0	0	0
Algorithms_Greedy	0.9846	0	0	0
Arithmetic_Basic	0.7538	0.5217	0.7059	0.6
Loops_Basic	0.6615	0.4	0.75	0.5217
TerminalIO_Advanced	0.6923	0.2857	1	0.4444
Arrays_Basic	0.7538	0.36	1	0.5294
Pointers_Advanced	0.9692	0.3333	1	0.5
Conditionals_Flags	0.7846	0.0769	0.3333	0.125
Structures_Basic	1	1	1	1
Char-String_Basic	0.7846	0.25	0.6667	0.3636
Conditionals_Basic	0.6923	0.4348	0.5882	0.5
Functions_Advanced	0.7385	0.4667	0.9333	0.6222
Arithmetic_Advanced	0.9538	0	0	0
Conditionals_Switch	1	1	1	1
Algorithms_Recursion	0.9538	0.7273	1	0.8421
Structures_Advanced	0.9846	0.75	1	0.8571
Algorithms_DP	1	1	1	1
Conditionals_Advanced	0.8308	0.1538	1	0.2667
Loops_Advanced	0.8923	0.85	0.8095	0.8293
Arrays_Advanced	0.9385	0.625	0.8333	0.7143
Char-String_Advanced	0.8769	0.1111	1	0.2
TerminalIO_Basic	0.8154	0.2143	0.75	0.3333
Pointers_Basic	0.8	0.2353	1	0.381
Structures_DS	1	1	1	1
Arrays_Memory	0.9692	0.6667	1	0.8
Algorithms_DC	0.9692	0.3333	1	0.5
Functions_Basic	0.6769	0.4167	1	0.5882
Loops_Invariants	0.9077	0	0	0
Overall Metrics	0.8703	0.4076	0.8035	0.5409

Table 3.3: Metrics of Logistic Regression(OVA) model using top-20 features

Since the Logistic Regression One-vs-All model has the best scores, as well as the fact that it provides a class probability distribution during prediction. Therefore it has been considered the suitable model for training the multi-label classification task in a one-vs-all style, so that during prediction we can rank the labels using the class probability scores.

3.5.2.2 Ranking Style Predictions

We discussed in subsubsection 3.4.2.2 that the PRIORITY labeled data suffers from a problem of missing labels. As we can see in Table 3.3, this holds true that both the overall and the label-wise recall score is very high. Thus, for most of the labels, the One-vs-All model is able to predict the true positives correctly. However, the One-vs-All prediction approach suffers from a low precision score. Even though the One-vs-All model might predict true for a label of a programming question, it might be the case that the annotator had marked the problem with labels that are more relevant than that specific label. This will lead to a low precision score, as the model has learned to blindly predict the labels for a programming question. Instead, it should ideally predict only the most

Program Label	Accuracy Score	Precision Score	Recall Score	F1 score
TerminalIO_Advanced	0.8154	0.3571	0.625	0.4545
TerminalIO_Basic	0.8462	0.2	0.5	0.2857
Arithmetic_Basic	0.8154	0.7273	0.4706	0.5714
Arithmetic_Advanced	0.9231	0.2	0.5	0.2857
Arithmetic_Bit	0.9077	0.1429	1	0.25
Conditionals_Basic	0.6923	0.3636	0.2353	0.2857
Conditionals_Advanced	0.8615	0.1111	0.5	0.1818
Conditionals_Switch	1	1	1	1
Conditionals_Flags	0.8154	0	0	0
Loops_Basic	0.7692	0.52	0.8125	0.6341
Loops_Advanced	0.9231	0.9	0.8571	0.878
Loops_Invariants	0.8462	0	0	0
Arrays_Basic	0.7077	0.3077	0.8889	0.4571
Arrays_Advanced	0.9231	0.6	0.5	0.5455
Arrays_Memory	0.9538	0.6	0.75	0.6667
Pointers_Basic	0.9231	0.4286	0.75	0.5455
Pointers_Advanced	0.9385	0.2	1	0.3333
Char-String_Basic	0.8308	0.3077	0.6667	0.4211
Char-String_Advanced	0.8769	0	0	0
Functions_Basic	0.8154	0.6	0.6	0.6
Functions_Advanced	0.7846	0.5217	0.8	0.6316
Structures_Basic	0.9538	0.25	1	0.4
Structures_Advanced	0.9846	1	0.6667	0.8
Structures_DS	1	1	1	1
Algorithms_DC	0.9231	0.1667	1	0.2857
Algorithms_DP	0.8923	0.125	1	0.2222
Algorithms_Recursion	0.9231	0.6667	0.75	0.7059
Algorithms_Greedy	0.9538	0	0	0
Overall	0.8786	0.4104	0.6358	0.4989

Table 3.4: Metrics of Balanced Random Forest Classifier with Label Propagation

relevant labels given a programming question.

In order to identify the most relevant programming labels given a programming question, we have transitioned to a ranking style prediction. For ranking style predictions, the terminology of the metrics used will slightly be different; instead of Precision, we will now have Precision@p, instead of Recall, we will now have Recall@p, and so on.Table 3.6 shows the Accuracy@p, Precision@p, Recall@p, and F1-score@p of the ranking-style prediction Logistic Regression model. Here, we use the standard style of ranking the labels, i.e., considering only the top-p predicted labels and turning all the other predicted labels off as discussed in subsubsection 3.4.2.2.

Additionally, we have experimented with ranking-style prediction, where the probability scores of each label are scaled with label-specific multipliers. Table 3.7 shows the scores we get from this experiment. Here, the Objective1 label-specific multipliers are used as described in subsubsection 3.4.2.2.

Comparing the scores between Table 3.4, Table 3.3, Table 3.6, Table 3.7 it is observed that there is a significant jump in the f1-score for the newly proposed ranking-style predictions. The ranking-style prediction model using label-specific multipliers(Objective1) has the best overall

Program Label	Accuracy Score	Precision Score	Recall Score	F1 score
Arithmetic_Bit	0.9846	0	0	0
Algorithms_Greedy	0.9231	0	0	0
Arithmetic_Basic	0.7692	0.5556	0.5882	0.5714
Loops_Basic	0.7077	0.4516	0.875	0.5957
TerminalIO_Advanced	0.7231	0.2917	0.875	0.4375
Arrays_Basic	0.7231	0.3333	1	0.5
Pointers_Advanced	0.9538	0.25	1	0.4
Conditionals_Flags	0.7385	0.0625	0.3333	0.1053
Structures_Basic	0.9846	0	0	0
Char-String_Basic	0.7538	0.2222	0.6667	0.3333
Conditionals_Basic	0.5692	0.3721	0.9412	0.5333
Functions_Advanced	0.7538	0.4839	1	0.6522
Arithmetic_Advanced	0.9538	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	0.8462	0.1667	1	0.2857
Loops_Advanced	0.8769	0.8095	0.8095	0.8095
Arrays_Advanced	0.9538	0.7143	0.8333	0.7692
Char-String_Advanced	0.9231	0.1667	1	0.2857
TerminalIO_Basic	0.7385	0.1579	0.75	0.2609
Pointers_Basic	0.9077	0.375	0.75	0.5
Structures_DS	1	1	1	1
Arrays_Memory	0.9692	0.75	0.75	0.75
Algorithms_DC	0.8615	0.1	1	0.1818
Functions_Basic	0.6769	0.4167	1	0.5882
Loops_Invariants	0.9077	0	0	0
Overall Metrics	0.8626	0.3934	0.8208	0.5318

Table 3.5: Metrics of One-vs-All Linear SVM Classifier

F1-score. It should also be noted that even though the ranking-style prediction model has higher f1-scores than the One-vs-All prediction model, it does have lower recall scores. This is expected, as the ranking-style models are trying to predict only the most relevant labels, thus increasing the precision score but reducing the overall recall score.

There were a couple of additional experiments run with different label-specific multipliers. These multipliers have been described in subsubsection 3.4.2.2. Table 3.8 shows the comparison of overall accuracy, precision, recall, and f1-scores of the three models using three types of label-specific multipliers (Objective1, Objective2, Subjective).

As already discussed in subsubsection 3.4.2.2, since we are only considering the top-p predicted labels, the value p becomes another hyperparameter for our model. Therefore, it is necessary to choose the appropriate value for p. Table 3.9 shows the scores for p=4, p=5, and p=6. As observed from the results, when the value of p is 5, we seem to be getting the best scores.

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	0.5217	0.7059	0.6
Loops_Basic	0.6615	0.4	0.75	0.5217
TerminalIO_Advanced	0.7385	0.3043	0.875	0.4516
Arrays_Basic	0.7692	0.375	1	0.5455
Pointers_Advanced	0.9692	0.3333	1	0.5
Conditionals_Flags	0.8462	0	0	0
Structures_Basic	1	1	1	1
Char-String_Basic	0.7846	0.25	0.6667	0.3636
Conditionals_Basic	0.8	0.75	0.3529	0.48
Functions_Advanced	0.7538	0.4815	0.8667	0.619
Arithmetic_Advanced	0.9538	0	0	0
Conditionals_Switch	1	1	1	1
Algorithms_Recursion	0.9538	0.7273	1	0.8421
Structures_Advanced	0.9846	0.75	1	0.8571
Algorithms_DP	1	1	1	1
Conditionals_Advanced	0.8615	0.1818	1	0.3077
Loops_Advanced	0.8923	0.85	0.8095	0.8293
Arrays_Advanced	0.9385	0.625	0.8333	0.7143
Char-String_Advanced	0.9077	0.1429	1	0.25
TerminalIO_Basic	0.8154	0.2143	0.75	0.3333
Pointers_Basic	0.9385	0.5	0.5	0.5
Structures_DS	1	1	1	1
Arrays_Memory	0.9538	0.6	0.75	0.6667
Algorithms_DC	0.9846	0	0	0
Functions_Basic	0.7692	0.5	0.6667	0.5714
Loops_Invariants	0.9385	0	0	0
Overall	0.8907	0.4522	0.711	0.5528

Table 3.6: Metrics of the standard Ranking-Style prediction model with no label-specific multipliers

3.6 Future Work

- A label-wise analysis of the scores could be performed for both the Balanced Random Forest Classifier model (with Label Propagation) and the Logistic Regression Ranking-style model. An ensemble of the two models could be created so as to extract the best possible performance across all the labels. A similar idea has been explored in the Difficulty Prediction task in chapter 4.
- Other statistical measures such as Pearson's correlation, or metrics like Precision, Recall or F1-score, can be used to perform the filter-based feature selection.
- Here we have used only the new features to train the Logistic Regression model. In future, we can also append the old features to create a bigger feature set.
- We have only explored Ranking-style predictions in this thesis, Ranking-style training could also be adopted in the future instead of One-vs-All training.

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	0.5217	0.7059	0.6
Loops_Basic	0.6615	0.4	0.75	0.5217
TerminalIO_Advanced	0.7385	0.3043	0.875	0.4516
Arrays_Basic	0.7538	0.36	1	0.5294
Pointers_Advanced	1	1	1	1
Conditionals_Flags	0.8923	0	0	0
Structures_Basic	1	1	1	1
Char-String_Basic	0.8	0.2667	0.6667	0.381
Conditionals_Basic	0.7538	0.5333	0.4706	0.5
Functions_Advanced	0.7538	0.4815	0.8667	0.619
Arithmetic_Advanced	0.9538	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	0.125	0.5	0.2
Loops_Advanced	0.8923	0.85	0.8095	0.8293
Arrays_Advanced	0.9385	0.625	0.8333	0.7143
Char-String_Advanced	0.9231	0.1667	1	0.2857
TerminalIO_Basic	0.8154	0.2143	0.75	0.3333
Pointers_Basic	0.9385	0.5	0.25	0.3333
Structures_DS	1	1	1	1
Arrays_Memory	0.9692	0.75	0.75	0.75
Algorithms_DC	0.9846	0	0	0
Functions_Basic	0.7692	0.5	0.8667	0.6341
Loops_Invariants	0.9385	0	0	0
Overall	0.894	0.4632	0.7283	0.5663

Table 3.7: Metrics of the Ranking-Style prediction model with Objective1 label-specific multipliers

Metric	Objective1	Objective2	Subjective
Accuracy@5	0.894	0.8841	0.8874
Precision@5	0.4632	0.4301	0.4412
Recall@5	0.7283	0.6763	0.6936
F1-Score@5	0.5663	0.5258	0.5393

Table 3.8: Comparison of metrics of Ranking-Style prediction models with Objective1, Objective2 and Subjective label-specific multipliers

Metric	$p = 4$	$p = 5$	$p = 6$
Accuracy@5	0.9	0.894	0.8857
Precision@5	0.481	0.4632	0.4419
Recall@5	0.659	0.7283	0.7688
F1-Score@5	0.5561	0.5663	0.5612

Table 3.9: Comparison of metrics of Ranking-Style prediction models with Objective1 with p=4,5,and 6

CHAPTER 4

PRIORITY: Difficulty Score Prediction

Contents

As discussed in chapter 1, PRIORITY aims to provide a search index over a significantly large collection of programming problems from past offerings of ESC101. The difficulty of a programming problem is a critical search parameter that the problem setters could use to find a suitable problem. Since a large part of the programming questions in the PRIORITY data is unlabeled, there is a need to develop a machine learning model to predict the difficulty score of a programming question.

The task of difficulty score prediction can be defined as follows, given a programming question, predict the difficulty score associated with that specific problem. The difficulty scores are as follows.

- Difficulty Score '1': Very easy
- Difficulty Score '2': Easy
- Difficulty Score '3': Medium
- Difficulty Score '4': Difficult
- Difficulty Score '5': Very Difficult

4.1 Dataset and Problem Overview:

- There were around two thousand programming questions in the PRIORITY data. Out of which about ten to fifteen percent of the total samples were annotated with the help of tutors.

- The labeled data also suffered a problem of duplicate problems. Some programming questions were labeled by multiple tutors to maintain consistency. Therefore, there was a need to de-duplicate the labeled data. For the de-duplication of difficulty score, the average score given by the tutors was considered. For example, if tutor A marked a programming question with a difficulty score of two and tutor B marked the same question with a difficulty score of four, then in our final labeled dataset, that question was labeled with a difficulty score of three.
- The previously extracted features [10] have been used to train the difficulty score prediction model. The old features has been described in chapter 2. These were mostly bag-of-words style features.
- Using the labeled data, there is a need to train a machine learning model which is able to predict the difficulty score of a new programming question.
- The trained machine learning model would be used to predict the difficulty score for the remaining unlabeled data so that the entire programming corpus can be indexed based on the difficulty score.

4.1.1 Imbalanced dataset:

One major issue in the dataset is that there is no uniform class distribution. There are very problems with difficulty score of 5, whereas problems with a difficulty score of 3 was most frequent followed by problems with difficulty score 2. The histogram depicting the frequencies of problems with various difficulty scores is shown in Figure 4.1.

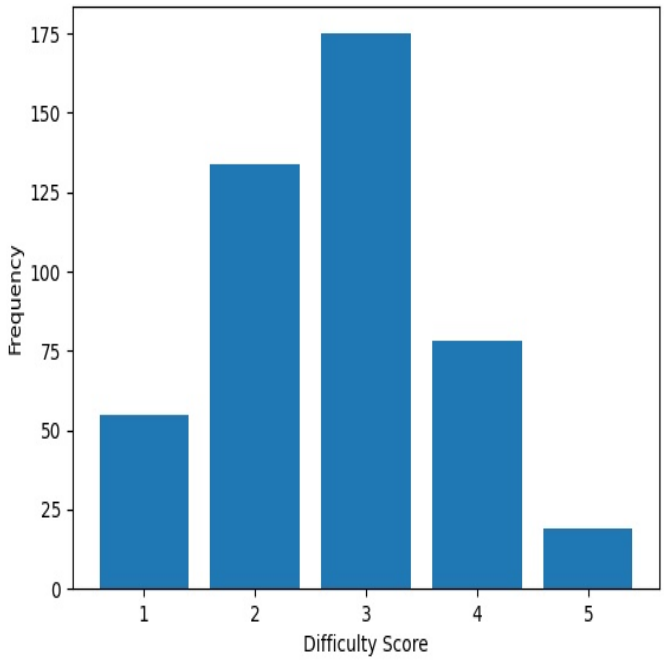


Figure 4.1: Distribution of problems with difficulty scores

Such a skewed distribution of scores generally leads to poor performance in rarer classes as the machine learning models do not get enough instances for that class during training.

4.2 Machine Learning Models:

- The task of problem difficulty score prediction can be modeled as a multi-class classification [1] problem, and that is precisely how this task has been solved till now. Each difficulty score had been considered as a separate class, and then a multi-class classifier was trained on the labeled data. However, using this approach does not capture the ordinal relationship that exists between the difficulty score classes.

- This task is a typical example of ordinal regression [5]. Ordinal regression is a task in machine learning that is used to predict an ordinal variable.

- In this thesis, we have explored several regression techniques to solve this task. Regressor models like Linear Regressor, Least Square Ridge Regressor, Support Vector Regressor, Random Forest Regressor, and Gradient Boosting Regressor [9] were experimented with. Gradient Boosting Regressor turned out to be the best-performing model, outperforming the Random Forest Classifier model, which was the best performing model previously.

- Further careful analysis of the mis-predicted instances by the Gradient Boosting Regressor model and the Random Forest Classifier model was performed. In the study, it was observed that the Random Forest classifier model outperformed the Gradient Boosting Regressor model for lower difficulty scores (1 and 2) across different metrics, and the Gradient Boosting Regressor model outperformed the Random Forest Classifier for higher difficulty scores (3, 4, 5).

- Therefore, we have used an ensemble learning technique called Mixture of Experts to combine the predictions from both the models mentioned above to get the best possible predictions. The layout of the Mixture of Experts model is described in detail in subsection 4.2.1.

- We propose a further optimization of the Mixture of Experts model, where we make use of the intermediate predictions by the gating model and the two expert models and append these predictions along with the initial feature vector and create a new feature vector, which is used to train a Neural Network model.

- So far, we have been using only the old features. We have also tried to predict the difficulty score prediction task by training the Mixture of Expert model using only the new features. The old features and new features are described in chapter 2.

4.2.1 Mixture of Expert model

- The Mixture of Experts [8] is an ensemble learning technique that involves dividing the prediction tasks into sub-tasks. An expert is trained for each sub-task and a gating model that learns which expert's prediction to trust based on the input, therefore combining the predictions from the expert models.

- The Mixture of Experts architecture proposed by us is shown in Figure 4.2. Initially, our difficulty score labels ranged from one to five. From our analysis, it was concluded that the Random Forest Classifier model performed better for lower difficulty scores (1, 2) and the Gradient Boosting Regressor model performed better for higher difficulty scores (3, 4, 5). Hence, the difficulty score labels were transformed into binary values by thresholding the scores at a value of 3. Thus, if a data point (programming question) was initially labeled with a difficulty score of 1 or 2, it got assigned a label of 0, and a data point had a difficulty score of 3, 4, or 5, it got assigned a label of 1. This transformed dataset was used to train a binary classifier (gating model) to solve the task of identifying if a data point belongs to sub-task 1, i.e., has a difficulty score of 1 or 2, or if it belongs to sub-task 2, i.e., has a difficulty score of 3, 4, or 5. A Support Vector Classifier [3] was used as the gating model.

- The expert models, i.e., the Random Forest Classifier (Expert 1) and the Gradient Boosting Regressor models (Expert 2), were trained independently to solve the task of predicting difficulty scores for all five labels and were not restricted to their respective sub-task labels. This decision was taken as the prediction made by the Gating model is not always correct, and in the case when the Gating model made an incorrect prediction, the difficulty score predicted by the mismatched expert would still be quite reliable. Both the Random Forest Classifier as well as the Gradient Boosting Regressor are powerful models, and the predictive capabilities should not be diminished if the number of output classes is increased from three to five.

- Finally, for a new test data point, we first use the Gating model to predict if the data point belongs to sub-task 1 or sub-task-2. Based on the sub-task that the gating model predicts, we will consider the prediction of the expert model associated with that sub-task as the final prediction.

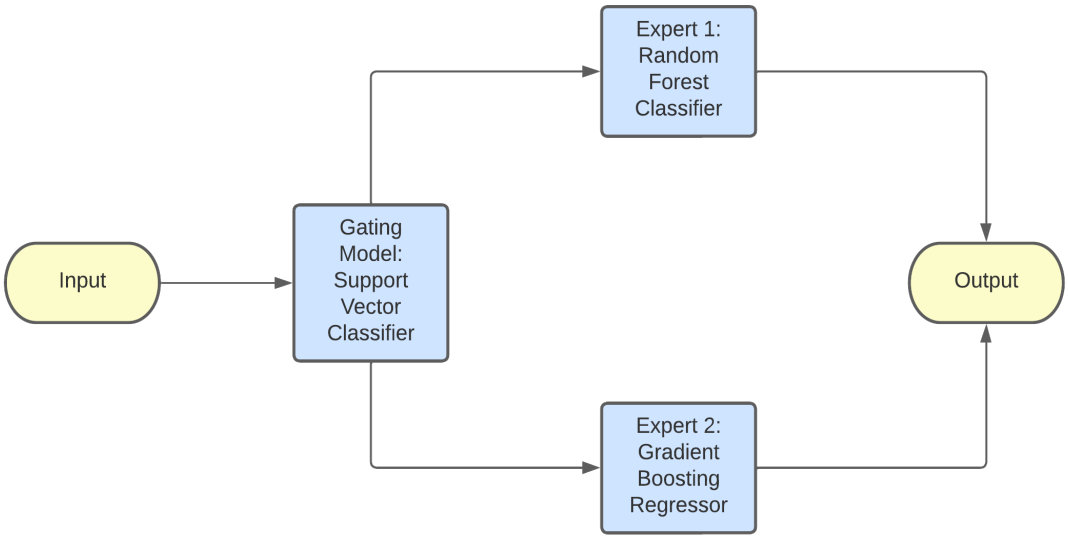


Figure 4.2: Mixture of Experts model

4.2.2 Neural Network Model

- One further optimization to the Mixture of Experts model has been proposed. We will create a new feature vector by appending the class probability scores from the Gating model, class probability scores from the Random Forest Classifier model(Expert 1), and the predicted score from the Gradient Boosting Regressor model to the initial feature vector. These new feature vectors will be used to train a Neural Network model.

- Our Neural Network architecture has two fully connected layers with a ReLU activation function and a softmax layer. Categorical cross-entropy loss is used. Adam has been used as the optimizer.

4.3 Experiments and Results

In this section we will be discussing all the experiments that was carried out for the difficulty score prediction task.

4.3.1 Regression Techniques

We tried out several regression model like Linear Regressor, Least Square Ridge Regressor, Support Vector Regressor, Random Forest Regressor, and Gradient Boosting Regression. Gradient Boosting Regressor gave the best results among them. The scores of Gradient Boosting Regression Model is given in Table 4.2.

4.3.2 MOE model trained with old features

The scores from the best performing model (Random Forest Classifier) is also given in Table 4.2. It is observed that the Gradient Boosting Regressor model has better overall accuracy than the Random Forest Classifier and have comparable scores in rest of the metrics. Here we have considered both Classification and Regression-type metrics. Regression metrics like Mean Absolute Error, Mean Square Error and Mean Absolute Percentage Error are important as they penalize close scores less and faraway scores more.

Even though accuracy score is an important parameter for difficulty score prediction, it is much better to predict a problem with a difficulty score one higher or lower than the true difficulty score, rather than predicting the difficulty score of a problem as 5 when the true difficulty score was 1.

We have plotted the graph of the frequency of test data points vs the difference in its true difficulty score and predicted difficulty score. The above graph has been plotted for both the Random Forest Classifier model and the Gradient Boosting Regressor model in Figure 4.3.

For the Mixture of Experts (MOE) model, the gating model used was a Support Vector Machine classifier model, The accuracy, precision, recall and f1-score of the model is shown in Table 4.1.

Given that the gating model is not always accurate, it made sense to train both the experts on all five difficulty scores, to account for the instances where the gating model makes a incorrect prediction.

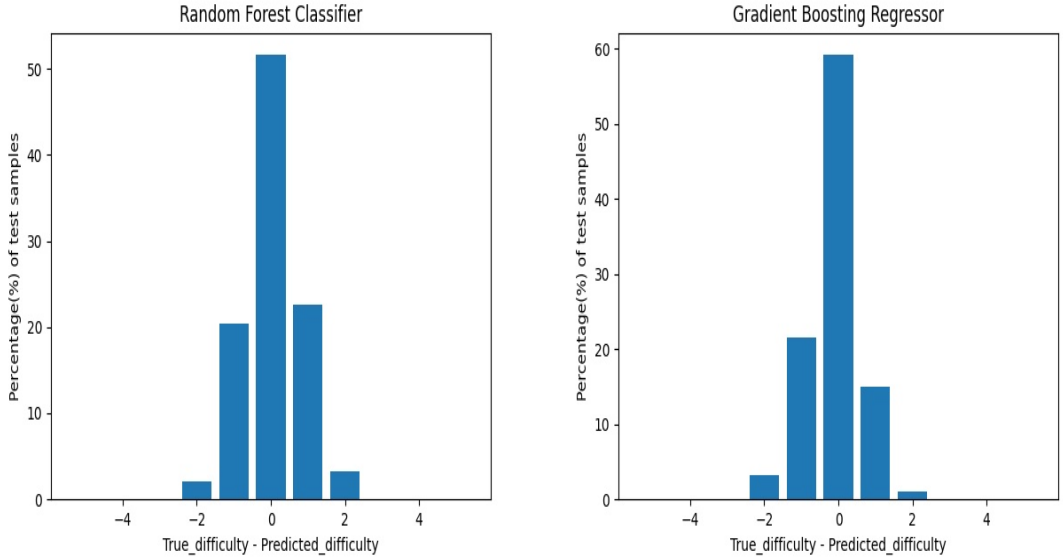


Figure 4.3: Difference in true difficulty score and predicted difficulty scores by RFC and GBR models

Metrics	Score
Accuracy	0.84
Precision	0.82
Recall	0.95
F1-Score	0.88

Table 4.1: Metrics achieved by the Support Vector Classifier gating model

Table 4.2 shows the scores for the Mixture of Experts model. It should be noted that the MOE model is successful in capturing the best scores among the the two expert models.

Table 4.3, Table 4.4, and Table 4.5, provide a more interesting study, Here the Accuracy, Mean Absolute Error, and Mean Absolute Percentage Error across the five difficulty scores have been compared for the three models. From the scores it is observed that the scores for Random Forest Classifier model is superior to the Gradient Boosting Regressor model for difficulty scores of 1 and 2. The Gradient Boosting Regressor model does better for the difficulty scores of 3, 4 and 5. The Mixture of Expert model has manages to capture the best scores for all the five difficulty scores. The scores of the MOE model are comparable to the Random Forest Classifier model for lower difficulty scores, whereas the MOE model scores are comparable to the Gradient Boosting Regressor model for the higher difficulty scores.

4.3.3 MOE model trained with new features

We have also experimented with the Mixture of Experts model by training it with the new features from chapter 2. TTable 4.6 shows the head-to-head comparison of the overall metrics for the MOE model trained with old features vs the MOE model trained with new features. The MOE model

Metrics	Random Forest Classifier	Gradient Boosting Regressor	MOE Model
Accuracy Score	0.52	0.59	0.59
Precision Score	0.56	0.55	0.52
Recall Score	0.46	0.48	0.47
F1-Score	0.49	0.49	0.48
Mean Squared Error	0.65	0.44	0.47
Mean Absolute Error	0.54	0.53	0.5
MAPE	0.22	0.25	0.22

Table 4.2: Random Forest Classifier vs Gradient Boosting Regressor vs MOE Model

Difficulty Score	Random Forest Classifier	Gradient Boosting Regressor	MOE Model
1	0.58	0.58	0.58
2	0.52	0.48	0.44
3	0.67	0.79	0.82
4	0.26	0.53	0.53
5	0.25	0	0

Table 4.3: Random Forest Classifier vs Gradient Boosting Regressor vs MOE Model based on accuracy for difficulty score bins

trained with new features seem to have a significantly higher precision, recall and f1-score.

Table 4.7, Table 4.8, and Table 4.9 provide a comparative study between the MOE model trained on old features and the MOE model trained on new features by reporting the scores across the different difficulty scores.

4.3.4 Neural Network model trained with old features

Table 4.10 reports the scores for the Neural Network model discussed previously. The Neural Network model has been trained using the old features, and this model also shows a significant increase in the overall precision, recall and f1-scores.

Table 4.11, Table 4.12, and Table 4.13 report the scores for the Neural network model compared to the MOE model (trained on the old features) across the different difficulty scores.

4.4 Future-work

- There has been a significant boost in precision, recall and f1-score for the Neural Network model trained on the old features as well as the Mixture of Experts model trained on the new features. Therefore, one obvious future work would be train the proposed Neural Network model using the new features.
- The new features and old features can also be merged, and this merged set of features can be used to train the Mixture of Experts model and the Neural Network model.
- Different architectures for the Mixture of Experts and Neural Networks could be tried out.

Difficulty Score	Random Forest Classifier	Gradient Boosting Regressor	MOE Model
1	0.42	0.67	0.47
2	0.56	0.64	0.63
3	0.33	0.33	0.31
4	0.79	0.55	0.54
5	1.25	0.98	0.98

Table 4.4: Random Forest Classifier vs Gradient Boosting Regressor vs MOE Model based on mean absolute error for difficulty score bins

Difficulty Score	Random Forest Classifier	Gradient Boosting Regressor	MOE Model
1	0.42	0.67	0.47
2	0.28	0.32	0.32
3	0.11	0.11	0.1
4	0.2	0.14	0.14
5	0.25	0.2	0.2

Table 4.5: Random Forest Classifier vs Gradient Boosting Regressor vs MOE Model based on mean absolute percentage error for difficulty score bins

Metrics	MOE model with Old Features	MOE model with New Features
Accuracy Score	0.59	0.58
Precision Score	0.52	0.72
Recall Score	0.47	0.59
F1-Score	0.48	0.63
Mean Squared Error	0.47	0.47
Mean Absolute Error	0.5	0.49
Mean Absolute Percentage Error	0.22	0.22

Table 4.6: MOE model trained on old features vs MOE model trained with new features

Difficulty Score	MOE model with Old Features	MOE model with New Features
1	0.58	0.5
2	0.44	0.48
3	0.82	0.7
4	0.53	0.53
5	0.0	0.75

Table 4.7: MOE model (trained on old features) vs MOE model (trained with new features) based on accuracy for difficulty score bins

Difficulty Score	MOE model with Old Features	MOE model with New Features
1	0.47	0.55
2	0.63	0.51
3	0.31	0.43
4	0.55	0.54
5	0.98	0.62

Table 4.8: MOE model (trained on old features) vs MOE model (trained with new features) based on mean absolute error for difficulty score bins

Difficulty Score	MOE model with Old Features	MOE model with New Features
1	0.47	0.55
2	0.32	0.25
3	0.1	0.14
4	0.14	0.13
5	0.2	0.12

Table 4.9: MOE model (trained on old features) vs MOE model (trained with new features) based on mean absolute percentage error for difficulty score bins

Metrics	MOE model	Neural Network model
Accuracy Score	0.59	0.57
Precision Score	0.52	0.61
Recall Score	0.47	0.56
F1-Score	0.48	0.57
Mean Squared Error	0.47	0.65
Mean Absolute Error	0.5	0.49
Mean Absolute Percentage Error	0.22	0.22

Table 4.10: MOE model vs Neural Network model

Difficulty Score	MOE model	Neural Network model
1	0.58	0.58
2	0.44	0.6
3	0.82	0.58
4	0.53	0.53
5	0.0	0.5

Table 4.11: MOE model vs Neural Network model based on accuracy for difficulty score bins

Difficulty Score	MOE model	Neural Network model
1	0.47	0.5
2	0.63	0.52
3	0.31	0.45
4	0.55	0.47
5	0.98	0.75

Table 4.12: MOE model vs Neural Network model based on mean absolute error for difficulty score bins

Difficulty Score	MOE model	Neural Network
1	0.47	0.5
2	0.32	0.26
3	0.1	0.15
4	0.14	0.12
5	0.2	0.15

Table 4.13: MOE model vs Neural Network model based on mean absolute percentage error for difficulty score bins

CHAPTER 5

Conclusion

This thesis proposes various improvements to the back-end machine learning models and the featurization techniques used in PRIORITY. From the results discussed in chapter 3 and chapter 4, it is apparent that the suggested changes have significantly improved the performance of both the label prediction and difficulty score prediction tasks. The improvements have been a result of incorporating elaborate ML techniques in the two tasks, like using a ranking-style prediction approach for the label prediction task, remodeling the difficulty score prediction task as a regression problem, and applying ensemble learning techniques to it. When the quantity of labeled data is scarce, often it is necessary to extract better features. Therefore, another direct reason for the improved scores is training the machine learning models using more robust features as proposed in chapter 2.

Bibliography

[1] Mohamed Aly. Survey on multiclass classification methods. *Neural Netw*, 19(1-9):2, 2005.

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

[3] Naiyang Deng, Yingjie Tian, and Chunhua Zhang. *Support vector machines: optimization based theory, algorithms, and extensions*. CRC press, 2012.

[4] Pablo A Estévez, Michel Tesmer, Claudio A Perez, and Jacek M Zurada. Normalized mutual information feature selection. *IEEE Transactions on neural networks*, 20(2):189–201, 2009.

[5] Pedro Antonio Gutiérrez, Maria Perez-Ortiz, Javier Sanchez-Monedero, Francisco Fernandez-Navarro, and Cesar Hervas-Martinez. Ordinal regression methods: survey and experimental study. *IEEE Transactions on Knowledge and Data Engineering*, 28(1):127–146, 2015.

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

[7] David G Kleinbaum, K Dietz, M Gail, Mitchel Klein, and Mitchell Klein. *Logistic regression*. Springer, 2002.

[8] Saeed Masoudnia and Reza Ebrahimpour. Mixture of experts: a literature survey. *Artificial Intelligence Review*, 42(2):275–293, 2014.

[9] Peter Prettenhofer and Gilles Louppe. Gradient boosted regression trees in scikit-learn. 2014.

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

[11] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13, 2007.

[12] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Mining multi-label data. In *Data mining and knowledge discovery handbook*, pages 667–685. Springer, 2009.

[13] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. 2002.