
PROGRAM REPAIR AND RETRIEVAL ON THE PRUTOR PLATFORM - II

A Thesis Submitted

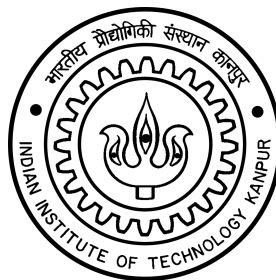
In Partial Fulfillment of the Requirements

For the Degree of M.Tech

by

Jeet Sarangi

21111032



to the

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur

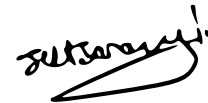
May, 2024

Page intentionally left blank

Declaration

This is to certify that at the thesis titled “**PROGRAM REPAIR AND RETRIEVAL ON THE PRUTOR PLATFORM - II**” has been authored by me. It presents the research conducted by me under the supervision of **PROF. PURUSHOTTAM KAR** and **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: Jeet Sarangi (21111032)

Program: M.Tech

Department of Computer Science and Engineering
Indian Institute of Technology Kanpur, Kanpur, 208016.

May, 2024

Page intentionally left blank

Declaration (To be submitted at DOAA Office)

I hereby declare that

1. The research work presented in the thesis titled “**PROGRAM REPAIR AND RETRIEVAL ON THE PRUTOR PLATFORM - II**” has been conducted by me under the guidance by my supervisor(s) **PROF. PURUSHOTTAM KAR** and **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: Jeet Sarangi

Program: M.Tech

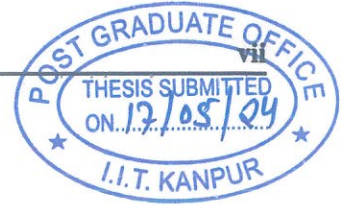
Department of Computer Science and Engineering

Roll No.: 21111032

Indian Institute of Technology Kanpur, Kanpur, 208016.

May, 2024

Page intentionally left blank



Certificate

It is certified that the work contained in the thesis titled “PROGRAM REPAIR AND RETRIEVAL ON THE PRUTOR PLATFORM - II” by JEET SARANGI has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

A handwritten signature in black ink that reads "Amey Karkare". To the left of the signature is a small rectangular stamp containing the letters "KAR" and some illegible scribbles below it.

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

Kanpur, 208016.

May, 2024

Page intentionally left blank

Abstract

Name of student: **Jeet Sarangi** Roll no: **21111032**

Degree for which submitted: **M.Tech**

Department: **Department of Computer Science and Engineering**

Thesis title: **PROGRAM REPAIR AND RETRIEVAL ON THE PRUTOR PLATFORM - II**

Name of thesis supervisor: **Prof. Purushottam Kar, Prof. Amey Karkare**

Month and year of thesis submission: **May, 2024**

In the area of education, the use of machine learning and artificial intelligence has grown a lot in the past few years. This surge has resulted in the development of various AI-assisted solutions aimed at enhancing educational practices. Among these tools is PRIORITY, which was specifically created to support problem setters of ESC101, an introductory programming course offered at IIT Kanpur.

PRIORITY provides a web-based platform that allows users to search for programming queries from previous versions of ESC101 course. It employs machine learning algorithms to label each programming question automatically, making them searchable. Each programming query includes a problem statement and a designated correct answer, making the process of extracting relevant features from this type of data complicated. In addition, PRIORITY entails estimating the level of difficulty for each programming problem which is the other component of PRIORITY system.

In our work on PRIORITY, we have focused on strengthening the label prediction component's robustness. In the previous version, for label prediction only the solution to the programming problem is considered. Nonetheless, we have made some progress by incorporating the problem statement. This modification has improved the system's performance in predicting the correct labels overall.

In our research, we experimented with two distinct methods for representing problem statements: FasText and Bag of Words. We trained the model with N one-vs-rest classifiers (one for each label). Interestingly, we observed that the Bag of Words-based model occasionally yielded better results compared to the FasText-based model, and vice versa. To improve the predictive capabilities of our system, we combined our statement-based model with a previously developed Abstract Syntax Tree (AST) model. This integration aimed to achieve better Label Predictions Overall and for each problem statement. We explored two ways for fusing the results of two models: Late Fusion and Early Fusion. Further we adopted two Ranking-style strategies one which is focused on coverage of important labels and other to avoid inclusion of unsure labels. In sum, our results help move forward priority system's label prediction task that is more reliable.

Acknowledgments

First of all, I want to thank my thesis supervisors, Prof. Purushottam Kar and Prof. Amey Karkare, for all the help and advice they have given me. Without their guidance and support, I would not have been able to finish this Thesis.

Second, I'd like to thank my teammates Utkarsh Shrivastava and Ayush Sahni, with whom I've had many productive conversations and ideas exchanges. In addition, I'd like to express my gratitude to my seniors, Debanjan Chatterjee, Preeti Singh, and Mayank Bansal, for their guidance and support.

In addition, I'd like to thank my CSE department instructors for their guidance and instruction throughout my time at IIT Kanpur. The knowledge I gained here has helped me grow both professionally and personally.

In the end, I want to express my gratitude to my mom for her unwavering faith in me and encouragement throughout my entire 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.

Jeet Sarangi

May, 2024

Contents

Acknowledgments	xi
Contents	xiii
List of Figures	xv
List of Tables	1
1 Introduction	1
1.1 Our Contributions	2
2 Label prediction using text features	5
2.1 Dataset	6
2.2 Data Pre-processing	7
2.3 Text Representation	11
2.4 Removing Irrelevant Windows	14
2.5 ML MODEL	17
2.6 Experiments and Results	24
3 Hybrid Models	27
3.1 Bridging the Gap: Integrating AST and Statement Models	28
3.2 Early Fusion	28
3.3 Deep Models	35
4 Future Work	45
5 Conclusion	47
Bibliography	49

List of Figures

1.1 Priority Search Page Image credits:[10]	2
2.1 Label Frequencies Image Credits[15]	7
2.2 Bag of Words Pipeline	17
2.3 BOW Pipeline with filtered windows	19
2.4 FastText Pipeline with filtered windows	20
3.1 Early Fusion Accuracy Variation Overall Comparison for different T-value in Ranking style prediction	34
3.2 Early Fusion F-score Variation Overall Comparison for different T-values in Ranking style prediction	34
3.3 Wide and Deep Generic Architecture [4]	38

List of Tables

2.1	Bag of words Statement Label Prediction	13
2.2	Bag of words Statement Label Prediction with relevant windows	18
2.3	Relevant Windows Prediction	18
2.4	FastText Statement Label Prediction with Relevant Windows	19
2.5	Fasttext Label Prediction using Multilabel Ranking	21
2.6	FastText Label Prediction using Ranking Style Prediction	23
2.7	Relevant Windows Prediction	25
3.1	Early Fusion with Logistic Regression, Bag of words and same top feature selection	30
3.2	Early Fusion with Logistic Regression, Fasttext embeddings and same top feature selection	31
3.3	Early Fusion with Logistic Regression, Bag of words and different top feature selection	32
3.4	Early Fusion with Logistic Regression, FastText embeddings and different top feature selection	33
3.5	write new	36
3.6	Wide and Deep Predictions	36
3.7	Deep and Deep Predictions	39
3.8	Wide and Deep with custom loss Prediction	40
3.9	Deep and Deep with custom loss predictions	42

Introduction

Contents

1.1 Our Contributions	2
--	----------

Machine Learning and Artificial intelligence at start were in headlines for the claims to surpass humans in games, which was great but not very revolutionary. However, with time these technologies have substantially increased their capabilities in several areas, such as text comprehension, which was for a long time an ability which distinguished humans from machines. As a result, these machine learning algorithms are now getting used in every areas to solve more problems which were earlier only can be handled by humans. One such area is education various new tools and AI-solutions are getting developed for improving the teaching process as shown in [16].

At IIT Kanpur, a popular course called Fundamentals of Computing (ESC101) is conducted annually, attracting a large number of students. As part of this course, weekly lab sessions are held, during which students are given a set of programming questions related to the topics covered that week. Designing a fresh set of problems each year poses a significant challenge for the tutors responsible for this task. To simplify their work, PRIORITY as described in [15] and [10] offers a web-based portal that grants tutors access to an extensive library of programming questions from previous iterations of the course. This resource serves to assist tutors in creating new problem sets effectively.

PRIORITY is an online platform that indexes a database of programming-related queries. It permits users to search the database based on particular programming concepts, required skills, and difficulty levels. As depicted in [Figure 1.1](#) the search page of PRIORITY enables users to conduct such targeted queries.

The PRIORITY database contains roughly 2000 programming queries. In order to create index for the database, each problem must be labelled in accordance with the programming principles outlined on the website and assigned a difficulty score. Nonetheless, manually labelling such a vast collection is a laborious task. As a solution, PRIORITY uses Machine Learning techniques to annotate the database automatically [15].

At first, a small group of past tutors who used to set problems for the ESC101 course were asked to label a small part of the PRIORITY data manually. As a result, we now have a small number of programming questions labelled, while most of them are still unlabelled [10].

The Thesis "Real world deployments of AI-assisted tools for compilation error repair and program retrieval" [10] proposes to create these automated annotations using two separate machine learning tasks. The first task is to make ML models that can predict the programming concepts or skills (Label Prediction) that go with a problem. The second job is to make ML models that can predict how hard a problem is (Difficulty score prediction).

One of the main challenges in the Label Prediction task is to effectively perform feature engineer-

The screenshot shows the Priority Search Page on the PRUTOR platform. The page has a purple header with a search bar, a 'Welcome' message, and a 'Log Out' button. The main content area is divided into two columns of filter boxes. The left column includes filters for Difficulty (Moderate), Arithmetic (Basic, Advanced, Bit operations), Loops (Basic, Advanced, Invariants), Char/String Usage (Basic, Advanced), Functions (Basic, Advanced), and Algorithms (divide & conquer, dp, recursion, greedy). The right column includes filters for Terminal IO (Basic, Advanced), Conditionals (Basic, Advanced, Switch, use of flag variable), Arrays (Basic, Advanced, Memory), Pointers (Basic, Advanced), and Structures (Basic, Advanced, Data Structure). Below the filters are 'Search' and 'Clear' buttons. The search results table below shows 6 results:

No.	Title
1	SEM2:15:16-Substring
2	Who is older?
3	SEM2:17:18-03_Loops_Say the Words
4	LIS1P2-The Imitation Game
5	Partial Palindrome
6	Partial Palindrome

A message at the bottom of the table states '6 Matching results Found below' with an 'OK' button.

Figure 1.1: Priority Search Page

Image credits: [10]

ing. Several approaches have been proposed in theses [3] and [16] to extract significant features from the Abstract Syntax Tree (AST) of the solutions for each programming question. These approaches outline different methods for capturing important information from the AST.

This thesis introduces modifications to the back-end machine learning components of PRIORITY to enhance its performance in the Label Prediction task. The results obtained after implementing these changes demonstrate a substantial improvement. The details of these enhancements are summarized in Section 1.1. Also, now PRIORITY is integrated to PRUTOR platform (an online tool to give programming assignments) the details regarding this are discussed in thesis [17]

This thesis is part of a group of three companion theses, along with [14] and [17]. To gain a comprehensive insight into the research conducted in this specific domain, it is advisable to refer to all three theses.

1.1 Our Contributions

- In the previous version of PRIORITY, feature extraction was based on the Abstract Syntax Tree (AST) of the solution C program for programming questions. This thesis suggests a different approach by utilizing the problem statement itself to extract features for Label Prediction. Various text representation techniques, including Bag of Words (BOW) and deep learning-based techniques like FastText, were explored to represent the question statement. The statement label prediction task was approached in two steps. First, supervised learning

was employed to identify the significant regions in the statement that are relevant to the question. Then, a one-vs-rest SVM classifier was used to predict labels for these important portions. The details of Statement based Label Prediction is discussed briefly in [chapter 2](#).

- In the prior method, Label Prediction involved training a One-vs-All Logistic Regression model on AST features using filter-based supervised feature selection [\[3\]](#). In this thesis we looked into an Early fusion-based Multi-modal method, in which features from both the AST and Statements are combined to train a one-vs-rest classifier. We also looked into how Deep Learning techniques could be used for the Label Prediction job. In particular, we tried out Wide and Deep networks architecture [\[4\]](#) and we also tried to modify the network a bit. [chapter 3](#) talks in brief about the results of these experiments.

Label prediction using text features

Contents

2.1 Dataset	6
2.1.1 Class Imbalance Problem	6
2.1.2 Ignored Labelling	6
2.2 Data Pre-processing	7
2.2.1 Text Pre-processing	8
2.2.1.1 Removal of Html tags and Special characters	8
2.2.1.2 Tokenization	8
2.2.1.3 Lowercasing	9
2.2.1.4 Stopword Removal	9
2.2.1.5 Stemming	9
2.2.1.6 Lemmatization	10
2.2.2 Splitting into windows	10
2.2.2.1 Simple windowing	10
2.2.2.2 Sliding Window	10
2.3 Text Representation	11
2.3.1 Bag of words	11
2.3.1.1 Restricted Vocabulary	12
2.3.1.2 Why Restricted Vocabulary ?	12
2.3.2 FastText	13
2.3.2.1 FastText over Bag of Words	14
2.4 Removing Irrelevant Windows	14
2.4.1 Marking Hot Regions	15
2.4.2 Vocabulary Generation	15
2.4.3 Creating Relevance Dataset	16
2.5 ML MODEL	17
2.5.1 Training Relevance Classifier	17
2.5.2 Hyperparameter Tuning:	20
2.5.3 Predicting Relevant Windows	20
2.5.4 Training Label Predictor	20
2.5.5 Label Prediction	22
2.5.5.1 Multilabel Ranking	22
2.5.5.2 Ranking-Style Prediction	22
2.6 Experiments and Results	24

2.6.1 Micro metric	24
2.6.2 Relevance Classification	25
2.6.3 Standard Label Prediction Conclusion.	25

In order to make the problems database searchable for tutors, PRIORITY [10] introduces two parameters: problem concepts tags and difficulty score. These parameters allow tutors to search and retrieve relevant problems from the database. Machine learning techniques are employed to tag the problems with appropriate labels as defined in [15], which is a crucial component of the tagging process.

The PRIORITY dataset includes problem statements, code solutions, and associated labels for each problem. While previous work in [3] and [16] focused on using code solutions to train a supervised model for label prediction, this chapter explores the utilization of problem statements for the same purpose. We will begin by addressing two important challenges related to the dataset, and then proceed to discuss the step-by-step process of predicting labels using text features.

2.1 Dataset

As per [3] and [15] the PRIORITY dataset comprises 28 labels that represent various programming concepts or skills associated with solving specific problems. Each label corresponds to a specific programming concept or skill. Detailed descriptions of these labels can be found in thesis1. Additionally, the dataset includes two main problems that serve as the focal points of analysis and exploration. These problems provide a concrete context for understanding and applying the label prediction techniques discussed in the thesis .

2.1.1 Class Imbalance Problem

As described in [3], [16] and [15]. The labeled dataset used for training the machine learning models exhibits a significant scarcity of labeled data points. This scarcity is visually depicted in Figure 3.1, which illustrates the frequency of occurrence of the programming labels. The plot clearly demonstrates a heavy class imbalance problem, wherein certain labels have a much higher frequency of data points compared to others. This imbalance is further exacerbated by a long tail distribution, indicating that a few popular labels have a disproportionately larger number of associated data points, while several labels have very few data points linked to them. Specifically, the eight least frequent labels have fewer than 20 programming questions associated with them. The presence of this severe class imbalance in the labeled dataset is anticipated to have a detrimental impact on the performance of machine learning models, particularly for the labels with limited data points.

2.1.2 Ignored Labelling

As noticed by [15] and [3] during the labeling process of the dataset, tutors assigned labels to the programming questions. However, an issue arose when certain labels had multiple levels or

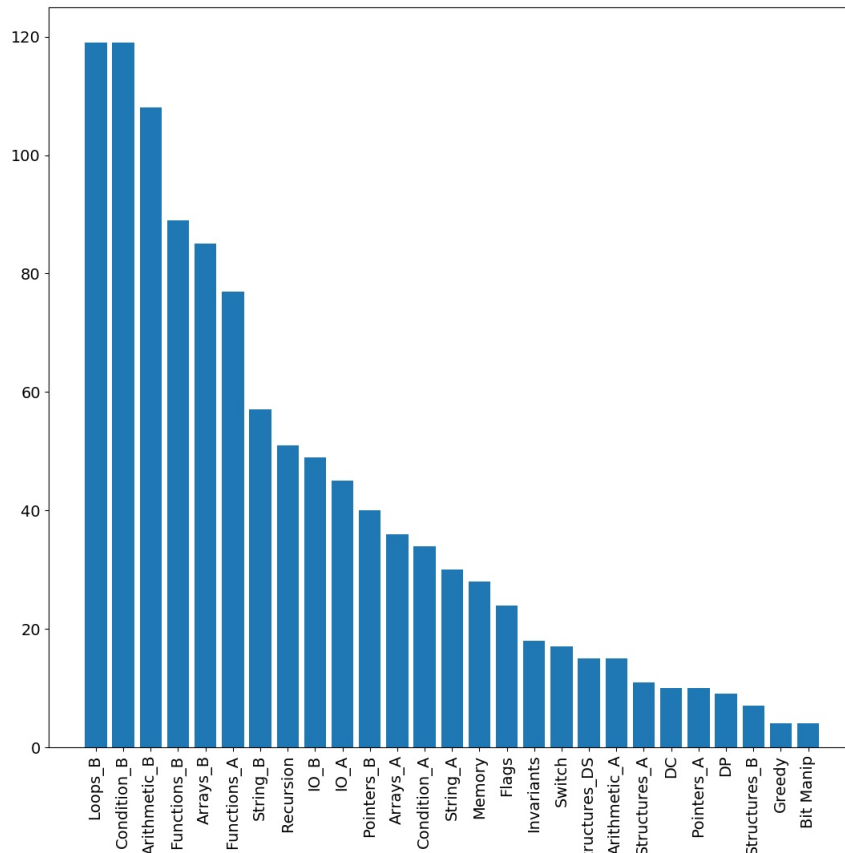


Figure 2.1: Label Frequencies
Image Credits [\[15\]](#)

subcategories. For instance, labels like "arithmetic basic" and "arithmetic advanced" may exist, and a question could possess qualities that correspond to both labels. However, in some cases, only one of the levels, such as "arithmetic advanced," was marked while the other level was left unmarked. This situation created a challenge during the training process and compromised the accuracy of insights obtained from the data. The ambiguity caused by the missing labels and the presence of overlapping qualities underscores the need for careful consideration and standardized guidelines when labeling data with hierarchical or multi-level labels.

2.2 Data Pre-processing

- We began with a csv file including information from the PRIORITY dataset, such as problem statements, solution code, template code, and a list of labels for each coding question. We cleaned the text data in many ways, including removing HTML tags, tokenization, lowercasing, stop word removal, stemming, and lemmatization.
- Following that, we separated the question statements into windows or chunks, because not all windows contribute to a particular label determination.

- We experimented with various approaches for representing these text windows as numeric vectors, including FastText and Bag-of-Words (BoW).
- Finally after all these steps, We acquired a set of vectors for each problem statement, representing each window within the statement.

2.2.1 Text Pre-processing

This section discuss about all steps followed to clean the problem statements before representing it as a numeric vector:

2.2.1.1 Removal of Html tags and Special characters

- HTML tags structure and format websites. They focus on text display rather than content. Focus on meaningful information and remove formatting and unnecessary aspects when processing text data is a crucial step.
- These tags can introduce noise and inconsistencies in the data. These may includes special symbols, foramtting related information which are not really relevant to the actual text of the problem statement.
- Tokenization can be impacted by HTML tags. The failure to remove tags before analysing or representing the text increases the risk that the tags will be included in the tokenization process.
- We have used Python Html Parser [8] to parse the and filter out the Html tags.
- Similar to this, extraneous noise can be introduced into the text by special characters such punctuation marks, symbols, and other non-alphanumeric characters. Eliminating these special characters makes it easier to ignore the background noise and focus on the text's important content.
- For removing special characters we have used python regular expression matching [12].

2.2.1.2 Tokenization

- In both the cleaning and the preparation of text, tokenization is an essential step. Tokens are often words or even smaller linguistic units like punctuation marks or numerals, and this process requires breaking a text down into those component parts. These tokens are then referred to as individual units. For this set up we have used words as the units.
- The process of constructing a vocabulary or word list for a given text can be aided by tokenization. The fact that each token is unique and represents a different word in the text enables us to perform statistical analysis on the text corpus, as well as analyse the frequency and distribution of words, determine which words are significant based on the prediction label.
- Tokenization also serves as a pre-requisite step for stemming and lemmatization.

- For our given setup we have used whitespace tokenizer from NLTK [1] .
- For the statement "consider an array of integers" after tokenization result for it will be `"['consider', 'an', 'array', 'of', 'integers']"`

2.2.1.3 Lowercasing

- Lowercasing is a common step in text cleaning and preprocessing tasks. It involves converting all the text to lowercase letters
- Lowercasing helps maintain consistency in the text. It ensures that the same word appears in the same case throughout the text, avoiding variations due to capitalization errors or different writing styles.

2.2.1.4 Stopword Removal

- Stopword removal is a crucial step in text cleaning and preprocessing tasks. It involves removing common words that do not carry significant meaning or contribute to the overall understanding of the text.
- Stopwords include articles (e.g., "the", "a", "an"), prepositions (e.g., "in", "on", "at"), and conjunctions (e.g., "and", "or", "but"). These words frequently appear in text but have little semantic value. By eliminating stopwords, we can reduce the amount of background noise in the text and concentrate on the more meaningful and informative words.
- The elimination of stopwords can increase the efficiency of topic modelling and text classification tasks. By omitting prevalent words that lack a distinct meaning, we can concentrate on more significant and discriminatory terms. This can result in a more accurate identification of program labels containing problematic text.

2.2.1.5 Stemming

- Stemming is a text preprocessing technique that reduces words to their root or fundamental form, also known as stems. It involves removing suffixes and prefixes from words in order to normalise them and combine words with similar meanings together.
- Stemming reduces vocabulary by simplifying words. It simplifies words to a stem, which aids information retrieval and text analysis. Stemming lets us combine numerous word forms into one. Stemming improves search and retrieval word matching.
- For our given setup we have used PorterStemmer from NLTK [1].
- For example words like "recursion" and "recursively" both when applied stemming produces "recurs".

2.2.1.6 Lemmatization

- The method of lemmatization is used as a text preparation tool, and it boils down words to their dictionary forms. Lemmatization, in contrast to stemming, takes into account the context and meaning of words, producing lemmas that are more linguistically accurate and meaningful.
- Lemmatization is useful since it reduces words to their simplest form without altering their meaning. It simplifies language to its most basic elements, making it easier to comprehend the meaning of text in context. Lemmatization helps in analysis and interpretation because it keeps the essential meaning of words.
- For our given setup we have used WordNetLemmatizer from NLTK [1].
- For example words like "arrays" and "array" both when applied Lemmatization produces "array".

2.2.2 Splitting into windows

2.2.2.1 Simple windowing

- This step is performed after cleaning the text. This procedure includes slicing the statement of the question into a number of smaller windows, each of which comprises a set number of consecutive words.
- By dividing the question statement into windows, we make smaller pieces of text that can be looked at and handled separately. This windowing method lets us figure out the local context and how words relate to each other in a small area. It helps to get the specific information and patterns for each window, which can help with the label prediction job.
- For instance, if we have a question statement that consists of 30 words and choose a window size of 5, we will need to generate six windows, with each window comprising five words in a row in order to complete the analysis. It's possible to label these windows as "Window 1: words 1-5," "Window 2: words 6-10," "Window 3: words 11-15," and so on.
- By breaking down the question statement into smaller windows, we aim to capture the nuanced details and context within the text, improving the model's ability to understand and predict the labels associated with each window.

2.2.2.2 Sliding Window

- We used the window stride method in addition to slicing the query phrase up into windows of a predetermined size each time. When we capture each window, how we slide the window over the text is determined by the window stride value that we use.
- For instance, if we have a window size of 10 and a stride of 5, we start by selecting the first 10 words of the question statement as our initial window. After processing and capturing the information from this window, instead of moving directly to the next 10 words, we slide the window by 5 words.

- If we have a question statement that consists of 30 words and choose a window size m of 10, with a stride value of 5, we will generate five windows each of size 10 having overlapping words with "Window 1: words 1-10," "Window 2: words 6-15," "Window 3: words 11-20," and so on.
- Using the window stride method helps ensure that we do not miss any important information while processing the question statement. It enables us to capture a wider range of context by considering overlapping windows, which can be particularly beneficial in scenarios where the target information is spread across multiple parts of the text.

2.3 Text Representation

Once we have obtained the cleaned windows for the problem statements, the next step is to convert them into numeric vector representations. In our study, we experimented with two approaches that are Bag of Words (BoW) representation and FastText representation.

By experimenting with both BoW and FastText representations, we aimed to compare their performance in capturing the important features of the problem statements. The BoW representation is straightforward and computationally efficient, while the FastText representation offers more nuanced semantic information. The choice between the two depends on the specific requirements of the label prediction task.

2.3.1 Bag of words

- The Bag of Words (BoW) representation is a popular way to show text data, like the windows in our study that were made from the problem statements. It is a simple yet effective approach that captures the presence of individual words in a document.
- In the BoW representation, each window is turned into a vector, where each dimension represents a different word in the language. The parts of the vector show how often or whether certain words are present or absent in the window. This representation doesn't care about the order of the words; it only looks at the binary presence of the words in the vocabulary.
- Let's refer to the vocabulary size as V , the window size as m and let for a problem statement we have p windows. We construct a V -dimensional vector for every window. Each dimension's value reflects the presence or absence of the relevant word in the window. A word's matching dimension value is one if it is visible in the window, while it is 0 if it is not. Finally, we'll get a boolean matrix of size $p * V$.
- To show this, let's look at an example with vocabulary "array","matrix","addition" If we have a window that says "Perform a matrix addition" the BoW representation would be [0, 1, 1]. This means that the words "matrix" and "addition" are in the window, but the word "array" is not.
- The Bag of Words method ignores grammar, syntax, and word order, which can be good and bad. It loses text sequence and context but is computationally efficient and straightforward

to implement. The BoW format is good for prediction of labels, where word presence is more important than word order e.g arrays, Arithmetic bit etc.

- In our approach, we have taken a more focused approach to constructing the vocabulary for each label in the label prediction task. Instead of considering the complete vocabulary of the problem statements, we have restricted the vocabulary to specific words that are relevant to each target label. This is discussed briefly in [Section 2.2.1.1](#)

2.3.1.1 Restricted Vocabulary

Let m denote the total number of documents, w represent the number of unique words in the vocabulary, and k be the number of top labels to consider (in this case, $k = 30$).

For each document i , we construct a boolean label vector, denoted as \mathbf{y}_i , of size $|\mathcal{L}|$. Each entry y_i^j is either 1 or 0, indicating the presence or absence of label j in document i , respectively.

Similarly, we construct a word matrix \mathbf{X} of size $w \times m$. Each entry X_i^j is a binary value indicating the presence (1) or absence (0) of word j in document i .

To determine the correlation between each label and word, we calculate the Pearson correlation coefficient, denoted as $\rho(l, w)$, between label l and word w . This correlation coefficient is computed as:

$$\rho(l, w) = \frac{1}{m} \sum_{i=1}^m \left(y_i^l - \text{mean}(\mathbf{y}^l) \right) \left(X_i^w - \text{mean}(\mathbf{X}^w) \right)$$

where the sum ranges over all documents i , $\text{mean}(\mathbf{y}^l)$ represents the mean of the label vector \mathbf{y}^l , and $\text{mean}(\mathbf{X}^w)$ represents the mean of the word vector \mathbf{X}^w .

Next, for each label l , we select the top k words with the highest absolute correlation coefficients. These k words form the restricted vocabulary \mathcal{V}_l for label l .

Therefore, for each label l in the top k labels, the restricted vocabulary \mathcal{V}_l is given by:

$$\mathcal{V}_l = \{w \mid |\rho(l, w)| \text{ is among the } k \text{ highest absolute correlation coefficients}\}$$

By selecting the top k labels and their corresponding words based on the highest absolute correlation coefficients, we obtain a restricted vocabulary that captures the most relevant and informative words for each label.

2.3.1.2 Why Restricted Vocabulary ?

- By selecting the words with the highest correlation coefficients for each label, the restricted vocabulary is made more specific to the semantics associated with those labels. This aids in capturing the most pertinent words that contribute to each label's comprehension.
- By concentrating on the most highly correlated words, the restricted vocabulary eliminates less informative or distracting words that may not significantly contribute to the labelling task. This results in a textual representation that is more concise and meaningful.

- The restricted vocabulary provides a clearer insight into the relationship between words and labels. By selecting words based on their correlation with specific labels, it becomes easier to interpret and understand the factors influencing the labeling process.
- Overall, this help in reducing the unwanted computation in per label model’s training and also increases the predictive performance per label.

Label	Accuracy	Precision	Recall	F1_score
Arithmetic_Bit	0.9692	0.0	0.0	0.0
Algorithms_Greedy	0.9846	0.0	0.0	0.0
Arithmetic_Basic	0.6462	0.4118	0.35	0.3784
Loops_Basic	0.5692	0.375	0.25	0.3
TerminalIO_Advanced	0.8462	0.125	0.25	0.1667
Arrays_Basic	0.7385	0.3333	0.2143	0.2609
Pointers_Advanced	0.9077	0.0	0.0	0.0
Conditionals_Flags	0.8923	0.0	0.0	0.0
Structures_Basic	0.9692	1.0	0.3333	0.5
Char-String_Basic	0.7846	0.1667	0.1	0.125
Conditionals_Basic	0.6923	0.2941	0.3846	0.3333
Functions_Advanced	0.7692	0.5333	0.5	0.5161
Arithmetic_Advanced	0.9692	0.5	0.5	0.5
Conditionals_Switch	0.9077	0.0	0.0	0.0
Algorithms_Recursion	0.8769	0.375	0.5	0.4286
Structures_Advanced	0.9692	0.3333	1.0	0.5
Algorithms_DP	0.9538	1.0	0.25	0.4
Conditionals_Advanced	0.8769	0.0	0.0	0.0
Loops_Advanced	0.6308	0.6667	0.4516	0.5385
Arrays_Advanced	0.9385	0.8333	0.625	0.7143
Char-String_Advanced	0.8308	0.0	0.0	0.0
TerminalIO_Basic	0.8154	0.0	0.0	0.0
Pointers_Basic	0.9385	0.75	0.5	0.6
Structures_DS	0.9846	1.0	0.6667	0.8
Arrays_Memory	0.8154	0.25	0.1	0.1429
Algorithms_DC	0.9692	1.0	0.3333	0.5
Functions_Basic	0.7692	0.6	0.5	0.5455
Loops_Invariants	0.8769	0.0	0.0	0.0
Overall	0.8533	0.3042	0.422	0.3535
Micro	0.8533	0.3057	0.4137	0.3232

Table 2.1: Bag of words Statement Label Prediction

2.3.2 FastText

Fasttext is useful since it uses subword details which capture the context of the sentence very well. Let’s say a window containing some words as $W = (w_1, w_2, \dots, w_n)$, where each w_i represents a word. To represent this window using FastText, we used pre-trained word embeddings from the FastText model trained on a large corpus such as Wikipedia.

For each word in the window, we retrieve its corresponding pre-trained FastText embedding

vector, denoted as $\mathbf{v}_i \in \mathbb{R}^d$, where d is the dimensionality of the word embeddings. These embedding vectors capture the semantic information of the words in a continuous vector space.

To obtain the FastText representation \mathbf{v}_w of the window W , we can average the individual word embeddings:

$$\mathbf{v}_w = \frac{1}{n} \sum_{i=1}^n \mathbf{v}_i,$$

where $\mathbf{v}_w \in \mathbb{R}^d$ represents the FastText representation of the window W , n is the number of words in the window, and \sum denotes the summation over all word embeddings.

The FastText representation \mathbf{v}_w is a dense vector in a high-dimensional space which in our case was 100, where each dimension captures different aspects of the window's semantic information. This representation can then be used as input features for downstream tasks such as label prediction.

By using FastText representation, we aim to capture the rich semantic meaning of the windows and enable the model to leverage the subword information to handle unseen or morphologically variant words. Our approach works in this context since we had very less data in order to fine tune fastText embeddings.

Overall, the FastText representation provides a powerful and efficient way to capture the semantic information of the window-based text, enabling effective downstream tasks such as statement label prediction.

2.3.2.1 FastText over Bag of Words

- FastText is a more robust representation than bag of words, since it offers the advantage of getting the semantic details which is not same in case of Bag of words.
- FastText embeddings are trained on very large data corpus of wikipedia which brings them with a pre-trained semantic rich information, this enable the model to capture the nuances and context of the text windows in our case for relevance classifier as well as label prediction.
- FastText representations are dense vectors which are on calculated for each of the windows of sentences will be able to give us more robust similarity and differences information to other windows than bag of words.
- Overall, using FastText enables the models to capture the semantic details of the windows in prediction tasks and also with dealing with variations of words for like synonyms and antonyms. These benefits will contribute to increase the model performance on the prediction tasks like label prediction and relevance prediction.

2.4 Removing Irrelevant Windows

Prior to performing label prediction, it was recognized that not all windows in the programming question are relevant for all labels. Additionally, some programming questions included a story

component that was completely unrelated to the actual question. These stories were included to provide a real-life context but did not contribute to the label prediction process. To address this issue, a filter layer was introduced to remove the irrelevant windows and focus on the essential parts of the question. The filter layer acted as a gatekeeper, selectively allowing only the pertinent windows to pass through for the final label prediction. By eliminating the irrelevant content, the model's attention was directed towards the relevant portions of the question, enhancing the accuracy and effectiveness of the label prediction task.

To address the challenge of filtering out irrelevant windows from the programming question, two approaches were considered.

Approach 1: Using similarity matching. A collection of text related to programming and programming in C was gathered from Wikipedia articles. From these texts, word windows were created, and vectors were generated for each window. This resulted in a set of vectors that represented programming-related content. When presented with a problem statement, the approach involved finding the average distance from the k nearest neighbors. Based on these distances, the top T windows were considered relevant.

Approach 2: focused on employing supervised learning to train a model capable of classifying whether a given window was relevant or not. This approach relied on training a model using labeled data to learn the patterns and characteristics of relevant windows. The details and implementation of this approach are discussed in depth in sections 2, 3, and 4.

Considering the availability of sufficient C-related articles for our specific case, it was decided to proceed with Approach 2. The subsequent sections provide a comprehensive exploration of the methodology, experimental results, and analysis related to this approach.

2.4.1 Marking Hot Regions

In order to train a model to classify relevant windows, the problem statements from the PRIORITY dataset were utilized. For each labeled problem in the dataset, the relevant portions of the problem statements were identified and marked. This process was carried out manually by our team members individually. By involving multiple team members in the marking process, we aimed to minimize individual biases and ensure a diverse range of perspectives on what constitutes relevant portions. This approach allowed for a comprehensive and well-rounded assessment of the relevance of different parts of the problem statements.

2.4.2 Vocabulary Generation

Now to determine which window carries how much relevance, we first need to assign a score to each word in the vocabulary. This is achieved by traversing each problem statement in the dataset and incrementing the score value for words that fall within the marked important sections. For instance, let's consider the following text:

"We have apples and oranges with their corresponding quantity, given a 2d array of apples and oranges."

Suppose person 1 marks "given a 2d array of apples and oranges" as relevant, person 2 marks "given a 2d array," and person 3 marks "2d array." In this example, our vocabulary would be

represented as:

$$V = \{ \text{"given"} : 2, \text{"2d"} : 3, \text{"array"} : 3, \text{"apples"} : 1, \text{"oranges"} : 1, \text{"and"} : 1 \}$$

Here, each word in the vocabulary is associated with a score that indicates how many people have marked it as relevant and the frequency of its relevance. By accumulating these scores, we obtain a quantitative measure of the relevance of each word within the problem statements.

2.4.3 Creating Relevance Dataset

After creating the vocabulary for the relevance dataset, we assign a score, denoted as $S(w)$, to each window, w , based on the words present within it. The score of a window is determined by taking the average of the scores of the words it contains. Mathematically, the score of a window can be represented as:

$$S(w) = \frac{1}{|w|} \sum_{i=1}^{|w|} S_{\text{word}}(w_i)$$

where $|w|$ represents the number of words in the window and $S_{\text{word}}(w_i)$ represents the score of the word w_i present in the window.

To create a dataset for the relevant or not classifier, we follow the following steps:

1. For each problem in the dataset, we convert it into windows of words using the techniques outlined in Section 1. We then calculate the scores, $S(w)$, for each window based on the vocabulary obtained in the previous steps.
2. Next, we rank the windows based on their corresponding scores. This ranking allows us to prioritize the windows based on their relevance scores, with higher-ranked windows considered more relevant. Mathematically, the ranking process can be represented as:

$$R(w) = \text{rank}(S(w))$$

where $R(w)$ represents the rank of the window w based on its score $S(w)$.

3. Let the total number of windows be denoted as N . We compare N to a predetermined threshold value, denoted as K . If N is less than K , we mark all windows as relevant since the number of windows is below the threshold.
4. However, if the number of windows N is greater than K , we employ a selection process to determine the relevant windows. We mark the top K windows as relevant, ensuring that the most relevant windows are included. Additionally, we select a percentage p of the remaining windows from the left side (after excluding the top K windows) and mark them as relevant as well. This process can be represented mathematically as:

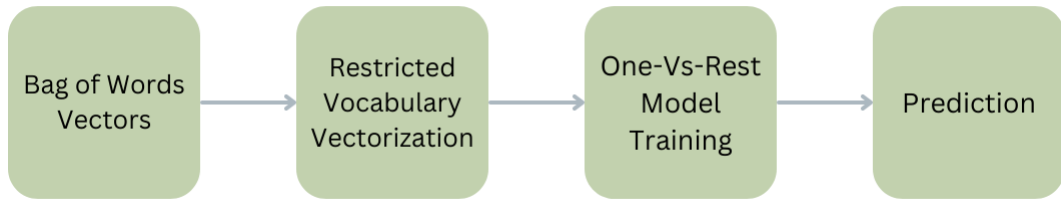


Figure 2.2: Bag of Words Pipeline

$$\text{relevant}(w) = \begin{cases} \text{True} & \text{if } R(w) \leq K \\ \text{True} & \text{if } K < R(w) \leq K + \lceil \frac{N-K}{100} \cdot p \rceil \\ \text{False} & \text{otherwise} \end{cases}$$

By following these steps, we obtain a dataset consisting of windows that are marked as either relevant or irrelevant. This dataset can be used to train a binary classifier to distinguish between relevant and irrelevant windows in future predictions.

2.5 ML MODEL

After obtaining the preprocessed vectors for label prediction, we have two pipelines based on the Bag of Words (BOW) vectors and FastText vectors. The pipelines are as follows:

- For BOW vectors, the pipeline involves using the full vocabulary initially and then reducing the dimensionality of the vectors using the restricted vocabulary. This is depicted in Figure 2.1. Afterward, the one-vs-rest model is trained using these reduced-dimensional vectors for label prediction(as shown in [Figure 2.1](#)). However, when using the relevance classifier, the first step is to remove the irrelevant windows before reducing the dimensionality using the restricted vocabulary. This process is illustrated in (as shown in [Figure 2.3](#)).
- For FastText, the pipeline is similar, but with one key difference. In this we do not have the restricted vocabulary step. So, the FastText vectors are directly fed into the training(as shown in [Figure 2.2](#)) and if relevance classifier is used then after filtering out the irrelevant vectors training step is done(as shown in [Figure 2.4](#)).

The subsequent sections will brief regarding the training and incurred results.

2.5.1 Training Relevance Classifier

In order to train the relevance classifier, we will conduct experiments using both the Bag of Words (BOW) encodings and the FastText encodings. Our objective is to train a single classifier capable of determining the relevance of a given window. To achieve this, we have explored five different classifiers: K nearest neighbors [\[6\]](#), decision trees[\[13\]](#), random forests[\[2\]](#), Support Vector Machines (SVM) [\[5\]](#), and Logistic Regression [\[9\]](#).

Label	Accuracy	Precision	Recall	F1_score
Arithmetic_Bit	0.9538	0.0	0.0	0.0
Algorithms_Greedy	0.9846	0.0	0.0	0.0
Arithmetic_Basic	0.6	0.4118	0.3043	0.35
Loops_Basic	0.5846	0.375	0.2609	0.3077
TerminalIO_Advanced	0.8462	0.0	0.0	0.0
Arrays_Basic	0.7385	0.4444	0.25	0.32
Pointers_Advanced	0.9231	0.0	0.0	0.0
Conditionals_Flags	0.8769	0.0	0.0	0.0
Structures_Basic	0.9692	1.0	0.3333	0.5
Char-String_Basic	0.8154	0.3333	0.2	0.25
Conditionals_Basic	0.6615	0.2941	0.3333	0.3125
Functions_Advanced	0.8	0.6	0.5625	0.5806
Arithmetic_Advanced	0.9692	0.5	0.5	0.5
Conditionals_Switch	0.9077	0.0	0.0	0.0
Algorithms_Recursion	0.8769	0.375	0.5	0.4286
Structures_Advanced	0.9692	0.3333	1.0	0.5
Algorithms_DP	0.9538	1.0	0.25	0.4
Conditionals_Advanced	0.8769	0.0	0.0	0.0
Loops_Advanced	0.6923	0.6667	0.5185	0.5833
Arrays_Advanced	0.9538	0.8333	0.7143	0.7692
Char-String_Advanced	0.8615	0.0	0.0	0.0
TerminalIO_Basic	0.8308	0.0	0.0	0.0
Pointers_Basic	0.9231	0.75	0.4286	0.5455
Structures_DS	0.9846	1.0	0.6667	0.8
Arrays_Memory	0.8462	0.0	0.0	0.0
Algorithms_DC	0.9538	1.0	0.25	0.4
Functions_Basic	0.7231	0.4667	0.4118	0.4375
Loops_Invariants	0.8923	0.0	0.0	0.0
Overall	0.856	0.309	0.4162	0.3547
Micro	0.856	0.3116	0.4085	0.3257

Table 2.2: Bag of words Statement Label Prediction with relevant windows

Model	Accuracy	Precision	Recall	F1_score
KNN	0.9504	0.9546	0.995	0.9744
Decision Tree	0.9266	0.9571	0.9658	0.9614
Random Forest	0.9542	0.957	0.9965	0.9763
Logistic Regression	0.8694	0.9693	0.8903	0.9281
SVM	0.9485	0.9484	1.0	0.9735

Table 2.3: Relevant Windows Prediction

By utilizing these diverse classifiers, we aim to evaluate their performance in accurately classifying windows as relevant or irrelevant. Each classifier brings its own strengths and characteristics, allowing us to analyze their effectiveness in this specific task. Through rigorous experimentation and evaluation, we can determine which classifier exhibits the highest accuracy and precision in capturing the relevance of the windows.

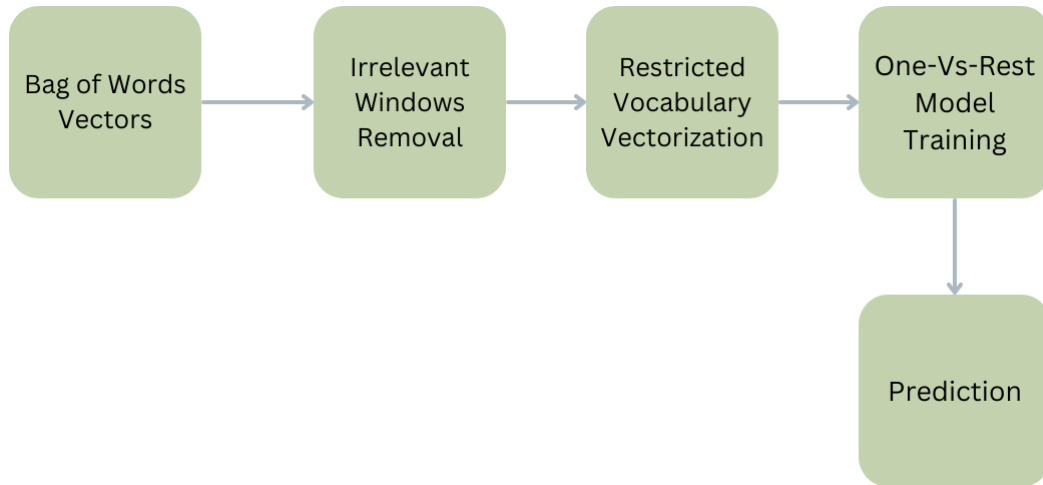


Figure 2.3: BOW Pipeline with filtered windows

Label	Accuracy	Precision	Recall	F1_score
Arithmetic_Bit	0.9846	0.0	0.0	0.0
Algorithms_Greedy	0.9846	0.0	0.0	0.0
Arithmetic_Basic	0.8	0.4118	0.7	0.5185
Loops_Basic	0.6615	0.4375	0.35	0.3889
TerminalIO_Advanced	0.8615	0.125	0.3333	0.1818
Arrays_Basic	0.8615	0.3333	0.5	0.4
Pointers_Advanced	0.9846	0.0	0.0	0.0
Conditionals_Flags	0.9538	0.0	0.0	0.0
Structures_Basic	0.9538	0.0	0.0	0.0
Char-String_Basic	0.8615	0.1667	0.2	0.1818
Conditionals_Basic	0.6154	0.1176	0.1667	0.1379
Functions_Advanced	0.8308	0.3333	0.8333	0.4762
Arithmetic_Advanced	0.9692	0.0	0.0	0.0
Conditionals_Switch	0.9846	0.0	0.0	0.0
Algorithms_Recursion	0.9231	0.625	0.7143	0.6667
Structures_Advanced	0.9538	0.0	0.0	0.0
Algorithms_DP	1.0	1.0	1.0	1.0
Conditionals_Advanced	0.9692	0.0	0.0	0.0
Loops_Advanced	0.7231	0.619	0.5652	0.5909
Arrays_Advanced	0.9077	0.1667	0.5	0.25
Char-String_Advanced	0.9385	0.0	0.0	0.0
TerminalIO_Basic	0.9385	0.0	0.0	0.0
Pointers_Basic	0.9385	0.25	0.5	0.3333
Structures_DS	0.9846	0.5	1.0	0.6667
Arrays_Memory	0.9385	0.25	0.5	0.3333
Algorithms_DC	1.0	1.0	1.0	1.0
Functions_Basic	0.7846	0.1333	0.6667	0.2222
Loops_Invariants	0.9385	0.0	0.0	0.0
Overall	0.9016	0.4727	0.3006	0.3675
Micro	0.9016	0.4344	0.3094	0.3306

Table 2.4: FastText Statement Label Prediction with Relevant Windows

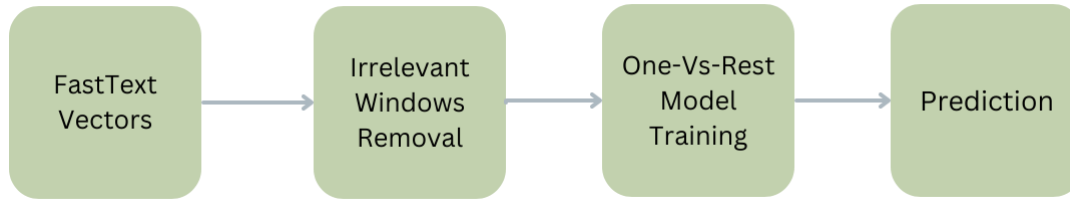


Figure 2.4: FastText Pipeline with filtered windows

2.5.2 Hyperparameter Tuning:

We first extract the set of windows from the training set of all the problem statements. The validation set is then built using these windows using which the classifier hyper-parameter is tuned for relevance classifier. And to the label classifier we get a subset of problems gradually based on the population for each label then we treat this subset as the validation set to tune the parameters.

2.5.3 Predicting Relevant Windows

Given a question consisting of multiple windows, our goal is to assign a relevance score to each window. Let's denote the relevance score of window i as $R(i)$ ($R(i) \in \{0, 1\}$).

To calculate the relevance score, we first obtain the probability score for each window, denoted as $P(i)$. This probability score represents the likelihood of a window being relevant. We can use various techniques, such as machine learning models, to estimate these probabilities based on the characteristics of the window.

Once we have the probability scores for all windows, we rank them in descending order. This ranking enables us to identify the windows with higher probabilities and, hence, higher relevance. To determine the threshold for relevance, we introduce a parameter K , which represents the desired number of relevant windows. If the total number of windows in the question is denoted as N , we apply the following criteria:

- If $N < K$, we consider all windows as relevant ($R(i) = 1$ for all windows).
- If $N \geq K$, we assign a relevance score of 1 to the top K windows based on their probability scores ($R(i) = 1$ for $i = 1$ to K).
- For the remaining $(N - K)$ windows, we introduce another parameter, $p\%$ ($0 \leq p \leq 100$). We select the top $p\%$ of these windows based on their probability scores and assign them a relevance score of 1. The remaining $(1 - p)\%$ windows are assigned a relevance score of 0.

By following this approach, we can effectively determine the relevance of each window within the question, providing a quantitative measure of their importance.

2.5.4 Training Label Predictor

To train for label prediction based on the window vectors, we adopt a one-vs-rest approach, where we train individual classifiers for each of the 28 labels. This approach allows us to handle the

Label	Accuracy	Precision	Recall	F1_score
Arithmetic_Bit	0.9846	0.0	0.0	0.0
Algorithms_Greedy	0.9846	0.0	0.0	0.0
Arithmetic_Basic	0.6615	0.7647	0.4194	0.5417
Loops_Basic	0.4308	0.9375	0.2941	0.4478
TerminalIO_Advanced	0.8769	0.5	0.5	0.5
Arrays_Basic	0.6923	0.6667	0.2609	0.375
Pointers_Advanced	0.9846	0.0	0.0	0.0
Conditionals_Flags	0.9077	0.0	0.0	0.0
Structures_Basic	0.9846	1.0	0.5	0.6667
Char-String_Basic	0.8923	1.0	0.4615	0.6316
Conditionals_Basic	0.4615	0.8235	0.3043	0.4444
Functions_Advanced	0.7692	0.6	0.5	0.5455
Arithmetic_Advanced	0.9692	0.0	0.0	0.0
Conditionals_Switch	0.9846	0.0	0.0	0.0
Algorithms_Recursion	0.9385	0.625	0.8333	0.7143
Structures_Advanced	0.9538	0.0	0.0	0.0
Algorithms_DP	0.9846	1.0	0.5	0.6667
Conditionals_Advanced	0.8769	0.0	0.0	0.0
Loops_Advanced	0.4769	0.9048	0.3725	0.5278
Arrays_Advanced	0.9077	0.3333	0.5	0.4
Char-String_Advanced	0.9077	0.0	0.0	0.0
TerminalIO_Basic	0.7846	0.25	0.0833	0.125
Pointers_Basic	0.9077	0.5	0.3333	0.4
Structures_DS	1.0	1.0	1.0	1.0
Arrays_Memory	0.9077	0.25	0.25	0.25
Algorithms_DC	1.0	1.0	1.0	1.0
Functions_Basic	0.5692	0.5333	0.2759	0.3636
Loops_Invariants	0.9231	0.0	0.0	0.0
Overall	0.8473	0.3385	0.6358	0.4418
Micro	0.8473	0.3385	0.6615	0.4327

Table 2.5: Fasttext Label Prediction using Multilabel Ranking

multi-label nature of the problem effectively. Since each problem is associated with multiple windows, we make an assumption that if a label is present (on) for a particular problem, it is also considered present (on) for all the windows associated with that problem. Similarly, if a label is absent (off) for a problem, it is assumed to be absent (off) for all the windows of that problem. By leveraging this assumption, we train the classifiers using the labeled problem statements and their corresponding windows.

For the label prediction task, we utilize linear support vector machines (SVM) as our classifier of choice. SVMs are well-suited for multi-label classification problems and have shown good performance in various text classification tasks. The results obtained from the trained classifiers will be further discussed and analyzed in the subsequent section, shedding light on the effectiveness and performance of the approach in accurately predicting the labels for the given problem statements.

2.5.5 Label Prediction

In the standard one-vs-rest prediction approach, we utilize the trained one-vs-rest classifiers to predict the labels for each individual problem. For a given problem, we apply the each one-vs-rest classifier to predict the labels for its associated windows. If any of the windows is predicted to have that label "on", then the problem is considered to have that label "on" in the final output. This approach ensures that if a label is present in any of the windows of a problem, it is considered as an active label for that problem.

To obtain the final label predictions for all 28 labels, we sequentially apply all the trained one-vs-rest classifiers to the problems. Each classifier predicts the presence or absence of its specific label for the associated windows of each problem. By iterating through all the classifiers, we accumulate the label predictions and generate the final output, indicating the presence or absence of each label for the given set of problem statements. This one-vs-rest prediction strategy allows us to capture the label associations at the window level, providing a comprehensive representation of the label presence across the different windows associated with each problem.

2.5.5.1 Multilabel Ranking

In the context of multi-label ranking, obtaining probability scores for each label is crucial. To achieve this, we employed one-vs-rest classifiers to predict the probability scores for each label independently. This approach allows us to estimate the likelihood of each label being active or inactive for a given problem.

Since we have multiple windows associated with each question, we explored two different approaches to aggregate the scores from the window level to the problem level. The first approach involved averaging the scores across all the windows, considering the normalized scores for fair comparison. This approach assumes that the overall label relevance can be determined by considering the collective contributions from all the windows.

The second approach involved selecting the maximum score among the scores obtained from the windows. This approach assumes that a single window might have a significant impact on determining the relevance of a label. By considering the maximum score, we ensure that if at least one window indicates a high relevance score for a label, it will have a greater influence on the final ranking.

After obtaining the probability scores for each label for every problem, we proceed to rank the labels based on their corresponding scores. For each problem, we iterate through the labels and arrange them in descending order based on their probability scores. We then select the top T labels to be active for that specific problem. This ensures that we always consider the most relevant labels for each problem, even if the model may not accurately rank all the labels. By setting a threshold of T, we aim to capture the most significant labels for each problem and reduce the chances of all labels being inactive.

2.5.5.2 Ranking-Style Prediction

In the context of Ranking-Style prediction, we follow a similar procedure as in multi-label ranking. Firstly, we obtain the probability scores for each label and each problem using the one-vs-rest

Label	Accuracy	Precision	Recall	F1_score
Arithmetic_Bit	0.9846	0.0	0.0	0.0
Algorithms_Greedy	0.9846	0.0	0.0	0.0
Arithmetic_Basic	0.8	0.4118	0.7	0.5185
Loops_Basic	0.6615	0.4375	0.35	0.3889
TerminalIO_Advanced	0.8615	0.125	0.3333	0.1818
Arrays_Basic	0.8615	0.3333	0.5	0.4
Pointers_Advanced	0.9846	0.0	0.0	0.0
Conditionals_Flags	0.9538	0.0	0.0	0.0
Structures_Basic	0.9538	0.0	0.0	0.0
Char-String_Basic	0.8615	0.1667	0.2	0.1818
Conditionals_Basic	0.6154	0.1176	0.1667	0.1379
Functions_Advanced	0.8308	0.3333	0.8333	0.4762
Arithmetic_Advanced	0.9692	0.0	0.0	0.0
Conditionals_Switch	0.9846	0.0	0.0	0.0
Algorithms_Recursion	0.9231	0.625	0.7143	0.6667
Structures_Advanced	0.9538	0.0	0.0	0.0
Algorithms_DP	1.0	1.0	1.0	1.0
Conditionals_Advanced	0.9692	0.0	0.0	0.0
Loops_Advanced	0.7231	0.619	0.5652	0.5909
Arrays_Advanced	0.9077	0.1667	0.5	0.25
Char-String_Advanced	0.9385	0.0	0.0	0.0
TerminalIO_Basic	0.9385	0.0	0.0	0.0
Pointers_Basic	0.9385	0.25	0.5	0.3333
Structures_DS	0.9846	0.5	1.0	0.6667
Arrays_Memory	0.9385	0.25	0.5	0.3333
Algorithms_DC	1.0	1.0	1.0	1.0
Functions_Basic	0.7846	0.1333	0.6667	0.2222
Loops_Invariants	0.9385	0.0	0.0	0.0
Overall	0.9016	0.4727	0.3006	0.3675
Micro	0.9016	0.4344	0.3094	0.3306

Table 2.6: FastText Label Prediction using Ranking Style Prediction

classifiers, as previously described. Additionally, we rank the labels based on their corresponding scores, allowing us to prioritize the most relevant labels.

To make predictions, we utilize the results obtained from the standard predictions of the one-vs-rest classifiers. For each problem, we check if the number of predicted labels is less than a predefined threshold, denoted as T . If the number of predicted labels is already below k , we retain all the predicted labels as active for that particular problem.

However, if the number of predicted labels exceeds T , we apply the ranking of labels based on their scores. We select the top T labels with the highest scores and consider them as active predictions. For the remaining labels, we set their predictions to zero, indicating that they are not relevant for the given problem. This approach ensures that we only predict labels for which we have higher confidence and avoids making excessive predictions.

By incorporating label ranking and the threshold of T , we aim to improve the precision of predictions and focus on the most reliable label assignments. This methodology enables us to make informed decisions regarding label activation, considering both the probability scores and the rank-

ing information.

2.6 Experiments and Results

2.6.1 Micro metric

Apart from conventional evaluation metrics like precision, recall, F-score, and accuracy for the overall labels, we used the micro metric to evaluate the performance of our model. While the overall metric provides an average performance measure for all labels, the micro metric provides a more detailed analysis by taking into account each individual data point in the dataset. For the micro metric, we first use our multiclass classification model to predict the labels for every data point. The accuracy, F-score, precision, and recall are then calculated based on the anticipated accurate labels for that particular data point. Each and every data point in the dataset goes through this process, yielding values for micro precision, micro recall, micro F-score, and micro accuracy by averaging over all of them.

Let \mathbf{y}_i denote the predicted label vector for data point i , where \mathbf{y}_i is a 1×28 -dimensional vector representing the predicted labels for the 28 classes. Similarly, let $\mathbf{y}_{\text{true},i}$ denote the ground truth label vector for data point i , where $\mathbf{y}_{\text{true},i}$ is also a 1×28 -dimensional vector.

To calculate precision (Prec_i), recall (Rec_i), F-score (F_i), and accuracy (Acc_i) for each data point i , we use the following formulas:

$$\begin{aligned}\text{Prec}_i &= \frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i}, \\ \text{Rec}_i &= \frac{\text{TP}_i}{\text{TP}_i + \text{FN}_i}, \\ F_i &= \frac{2 \cdot \text{Prec}_i \cdot \text{Rec}_i}{\text{Prec}_i + \text{Rec}_i}, \\ \text{Acc}_i &= \frac{\text{TP}_i + \text{TN}_i}{\text{TP}_i + \text{TN}_i + \text{FP}_i + \text{FN}_i}\end{aligned}$$

where TP_i is the number of true positives, FP_i is the number of false positives, FN_i is the number of false negatives, and TN_i is the number of true negatives for data point i . These values are calculated by comparing \mathbf{y}_i with $\mathbf{y}_{\text{true},i}$.

Once we have computed Prec_i , Rec_i , F_i , and Acc_i for each data point, we can then calculate the micro-average precision ($\text{Prec}_{\text{micro}}$), micro-average recall ($\text{Rec}_{\text{micro}}$), micro-average F-score (F_{micro}), and micro-average accuracy ($\text{Acc}_{\text{micro}}$) as follows:

$$\begin{aligned}\text{Prec}_{\text{micro}} &= \frac{1}{N} \sum_{i=1}^N \text{Prec}_i, \\ \text{Rec}_{\text{micro}} &= \frac{1}{N} \sum_{i=1}^N \text{Rec}_i, \\ F_{\text{micro}} &= \frac{1}{N} \sum_{i=1}^N F_i,\end{aligned}$$

$$\text{Acc}_{\text{micro}} = \frac{1}{N} \sum_{i=1}^N \text{Acc}_i$$

where N is the total number of data points in the dataset. These micro metrics provide a comprehensive evaluation of the model’s performance across all classes by considering each data point individually.

Using the micro metric will give us a detailed understanding of how the model is performing in data at atomic level and will serve as a effective tool to evaluate the model in this problem setup.

2.6.2 Relevance Classification

Model	Accuracy	Precision	Recall	F1_score
KNN	0.9504	0.9546	0.995	0.9744
Decision Tree	0.9266	0.9571	0.9658	0.9614
Random Forest	0.9542	0.957	0.9965	0.9763
Logistic Regression	0.8694	0.9693	0.8903	0.9281
SVM	0.9485	0.9484	1.0	0.9735

Table 2.7: Relevant Windows Prediction

The presented in table 2.1 provides a comprehensive overview of the outcomes derived from the Bag of Words (BoW) representation technique. In this context, the vectors generated are of dimensionality k , signifying the selection of the top k correlated words associated with each label. Notably, each label undergoes training using an individual model in a one-vs-rest fashion, with Support Vector Machines (SVM) serving as the chosen algorithm for this purpose. The table encapsulates the results of this approach, shedding light on the performance and efficacy of the BoW representation coupled with SVM in the context of label prediction.

2.6.3 Standard Label Prediction Conclusion:

The results highlight a few key observations:

- Comparing 2.1 and 2.2 reveals that the relevance model contributes positively to the predictions, demonstrating improved results when irrelevant windows are excluded.
- Analyzing 2.2 and 2.4 indicates that FastText vectors outperform BOW vectors in label prediction.
- Examining 2.5, 2.4, and 2.6 emphasizes the effectiveness of the multilabel ranking method, showcasing superior performance in terms of accurate ranking. While the overall prediction accuracy might not be optimal, the method excels in assigning correct ranking orders based on probabilities.

Hybrid Models

Contents

3.1 Bridging the Gap: Integrating AST and Statement Models	28
3.2 Early Fusion	28
3.2.0.1 Advantage of using Early fusion	29
3.2.0.2 Combining Strategies for features:	30
3.2.0.3 Hyperparameter tuning:	33
3.2.0.4 Model Prediction	34
3.2.1 Results	35
3.3 Deep Models	35
3.3.1 Wide and Deep	37
3.3.1.1 Implemented Architecture:	37
3.3.2 Deep and Deep	38
3.3.2.1 Deep and Deep Architecture:	39
3.3.2.2 Loss Funtion:	41
3.3.2.3 Custom Loss:	41
3.3.2.4 Optimizer:	42
3.3.3 Results of Deep learning based predictions:	43

As we saw in [Chapter 2](#) the how to use the text features, we discovered the usefulness of problem statements(text features) in the process of predicting program labels. The effectiveness of the AST-based approach was demonstrated over a comprehensive range of programming use cases as described in the theses [\[3\]](#) and [\[16\]](#). However, as per a comparative analysis we can see that there were times when the statement-based technique was able to outperform its AST equivalent. This was the case in a number of different scenarios. This frequently occurred when the AST did not provide a detailed picture of the approach that was being taken to solve the problem.

In this chapter, we will discuss various ways to combine these two methodologies since both the methods have there own strengths and are two powerful models, each of which excels in its own unique field of application. Our objective is to design a label prediction system that draws on the positive aspects of both AST and Statement models. This should result in a tool that is both flexible and powerful, making it suitable for use by both educators and programmers.

3.1 Bridging the Gap: Integrating AST and Statement Models

We have investigated three different methodologies—Late Fusion (This is discussed in Utkarsh’s thesis [17]), Early Fusion, and Deep Learning-based models—to combine two models for prediction tasks. Each of these strategies has its own pros and drawbacks.

However, there have been a number of difficulties we have faced along the way, such as:

- **Result Representation:** One key problem is effectively describing the results provided by each model so that they may be seamlessly combined to achieve the final results.
- **Generalization Across Labels:** Creating a combination approach that can be applied consistently to all labels presents another noteworthy problem. Relying on a static merging strategy is not always acceptable because some labels may benefit more from one model than the other. Therefore, it is necessary to adopt a more dynamic approach to combining that takes into account the distinctive qualities of each label.

In the following parts, a brief discussion will be held on how actually two of these approaches were put into practice and evaluated. The detail regarding the third approach which is late fusion can be found on Utkarsh’s thesis [17].

3.2 Early Fusion

Early fusion, also known as feature-level fusion, involves merging features from multiple sources before feeding them into a unique model. This approach aims to create a joint feature representation that captures complementary information from different models.

Steps involved in achieving the above are as follows:

1. Feature Selection:

Early Fusion involves merging features from different sources before inputting them into a unified model. Let X_s and X_a represent the feature vectors from the Statement-Based Model and AST Model, respectively. The combined feature vector X_{combined} is obtained by merging or concatenating these features:

$$X_{\text{combined}} = [X_s, X_a]$$

Here, $[X_s, X_a]$ denotes the concatenation of feature vectors X_s and X_a .

2. Model Prediction:

Once the features are merged, they are used as input for a machine learning model. Let F be the mapping function that takes X_{combined} as input and produces predicted probabilities P_{combined} :

$$P_{\text{combined}} = F(X_{\text{combined}})$$

The model F can be any machine learning algorithm. During training, the parameters of F are optimized to effectively leverage the combined information from both models. This process enhances the model’s ability to capture intricate patterns and relationships in the data.

3.2.0.1 Advantage of using Early fusion

Using early fusion for hybrid model give us various benefits as follows:

- **Information Integration:**
 - Early fusion integrates features from both the AST and the problem statement at an early stage, allowing the model to consider a holistic representation that captures the intricacies of both sources.
- **Optimal Model Training:**
 - By combining features early, the model is trained on a unified input space, facilitating more effective learning and adaptation to the combined information.
- **Enhanced Generalization:**
 - Shared Feature Space: Early fusion promotes improved generalization by creating a shared feature space that captures common patterns present in both the AST and problem statement.
- **Interpretability in Label Prediction:**
 - Unified Output Interpretation is achieved as the combined features lead to a unified output, making it easier to interpret and understand how both the AST and problem statement contribute to the predicted labels.
- **Efficient Inference:**
 - During inference, a single model processes the combined features, streamlining the prediction process and reducing computational complexity.
- **Robustness to Missing Information:**
 - Partial Information Handling: In situations where either the AST or the problem statement is incomplete or missing, early fusion models can still make predictions based on the available information, maintaining robustness.
- **Adaptability to Label Prediction:**
 - Early fusion offers flexibility in combining features, allowing adaptation to the label prediction task. Different fusion schemes can be explored based on the characteristics of the data.
- **Preservation of Contextual Information:**
 - Early fusion ensures that contextual information from both the AST and the problem statement is retained, which is crucial for accurately predicting labels.
- **Reduced Risk of Information Loss:**

- Maximized information utilization by combining information early, the risk of losing essential details is minimized, ensuring that both modalities contribute effectively to the label prediction.

- **Simplified Model Deployment:**

- The early fusion approach simplifies the deployment of the model, as there is a single model that integrates information from both sources for predicting labels.

3.2.0.2 Combining Strategies for features:

Program Label	Accuracy Score	Precision Score	Recall Score	F1 Score
Arithmetic_Bit	0.9846	0.0	0.0	0.0
Algorithms_Greedy	0.9846	0.0	0.0	0.0
Arithmetic_Basic	0.7692	0.5625	0.5294	0.5455
Loops_Basic	0.6769	0.3529	0.375	0.3636
TerminalIO_Advanced	0.8615	0.4545	0.625	0.5263
Arrays_Basic	0.7538	0.36	1.0	0.5294
Pointers_Advanced	0.9385	0.0	0.0	0.0
Conditionals_Flags	0.8769	0.0	0.0	0.0
Structures_Basic	1.0	1.0	1.0	1.0
Char-String_Basic	0.8769	0.4167	0.8333	0.5556
Conditionals_Basic	0.6615	0.4074	0.6471	0.5
Functions_Advanced	0.7538	0.4839	1.0	0.6522
Arithmetic_Advanced	0.9692	0.0	0.0	0.0
Conditionals_Switch	1.0	1.0	1.0	1.0
Algorithms_Recursion	0.9538	0.7273	1.0	0.8421
Structures_Advanced	0.9538	0.5	0.3333	0.4
Algorithms_DP	0.9385	0.2	1.0	0.3333
Conditionals_Advanced	0.8615	0.1818	1.0	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	0.8923	0.0	0.0	0.0
TerminalIO_Basic	0.8615	0.0	0.0	0.0
Pointers_Basic	0.8923	0.3333	0.75	0.4615
Structures_DS	0.9692	0.5	1.0	0.6667
Arrays_Memory	0.9692	0.6667	1.0	0.8
Algorithms_DC	0.9692	0.0	0.0	0.0
Functions_Basic	0.6769	0.4167	1.0	0.5882
Loops_Invariants	0.9385	0.0	0.0	0.0
Overall Metrics	0.8874	0.4416	0.6994	0.5414
Micro	0.8874	0.4707	0.6974	0.5319

Table 3.1: Early Fusion with Logistic Regression, Bag of words and same top feature selection

To combine the features effectively, two crucial parameters were considered in our approach:

1. **Feature Selection Based on Pearson Score:**

Before feeding the combined feature set into the machine learning model, a careful feature selection process was implemented. For each label x , a specific set of top k features was

Program Label	Accuracy Score	Precision Score	Recall Score	F1 Score
Arithmetic_Bit	0.9846	0.0	0.0	0.0
Algorithms_Greedy	0.9846	0.0	0.0	0.0
Arithmetic_Basic	0.7231	0.4706	0.4706	0.4706
Loops_Basic	0.6769	0.3529	0.375	0.3636
TerminalIO_Advanced	0.8462	0.4286	0.75	0.5455
Arrays_Basic	0.6769	0.3	1.0	0.4615
Pointers_Advanced	0.9538	0.0	0.0	0.0
Conditionals_Flags	0.9077	0.2	0.3333	0.25
Structures_Basic	1.0	1.0	1.0	1.0
Char-String_Basic	0.8615	0.3846	0.8333	0.5263
Conditionals_Basic	0.6615	0.4074	0.6471	0.5
Functions_Advanced	0.7538	0.4839	1.0	0.6522
Arithmetic_Advanced	0.9692	0.0	0.0	0.0
Conditionals_Switch	1.0	1.0	1.0	1.0
Algorithms_Recursion	0.9538	0.7273	1.0	0.8421
Structures_Advanced	0.9538	0.5	0.3333	0.4
Algorithms_DP	0.9692	0.0	0.0	0.0
Conditionals_Advanced	0.8615	0.1818	1.0	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	0.9077	0.1429	1.0	0.25
TerminalIO_Basic	0.8615	0.0	0.0	0.0
Pointers_Basic	0.8769	0.3	0.75	0.4286
Structures_DS	0.9692	0.5	1.0	0.6667
Arrays_Memory	0.9692	0.6667	1.0	0.8
Algorithms_DC	1.0	1.0	1.0	1.0
Functions_Basic	0.6769	0.4167	1.0	0.5882
Loops_Invariants	0.9385	0.0	0.0	0.0
Overall Metrics	0.8857	0.4377	0.711	0.5419
Micro	0.8857	0.4621	0.703	0.5261

Table 3.2: Early Fusion with Logistic Regression, Fasttext embeddings and same top feature selection

determined from the combined features set. This selection was based on the Pearson correlation score, a measure of the linear correlation between two variables. There were two distinct strategies employed to identify these top features:

(a) **Single Top k Relevant Feature:**

In this approach, a single set of top k relevant features was selected from the combined features for each label x .

(b) **Set-wise Top k Relevant Features:**

Alternatively, a two-step process was followed. Initially, the top k_1 relevant features were identified from the AST feature set, and another set of top k_2 relevant features from the Statement feature set was determined. These sets of features were then used collectively.

2. **Handling Statement-Based Features:**

In the case of statement-based features, a decision had to be made regarding whether to use all the words or employ FastText-based feature embeddings. Two distinct experiments were conducted to explore both approaches and assess their impact on the model’s performance.

This process emphasizes the importance of selecting relevant features tailored to each label, ensuring that the most informative features contribute to the model’s learning. The Pearson correlation score serves as a valuable metric in identifying the strength and direction of the linear relationship between features, guiding the selection of top features for optimal predictive performance. The experimentation with different feature selection strategies and the consideration of various options for statement-based features aim to refine and enhance the model’s ability to capture the nuances present in both the AST and statement-based representations.

Program Label	Accuracy Score	Precision Score	Recall Score	F1 Score
Arithmetic_Bit	0.9846	0.0	0.0	0.0
Algorithms_Greedy	0.9846	0.0	0.0	0.0
Arithmetic_Basic	0.8	0.625	0.5882	0.6061
Loops_Basic	0.6462	0.3478	0.5	0.4103
TerminalIO_Advanced	0.9077	0.5714	1.0	0.7273
Arrays_Basic	0.7692	0.375	1.0	0.5455
Pointers_Advanced	0.9846	0.5	1.0	0.6667
Conditionals_Flags	0.9077	0.2	0.3333	0.25
Structures_Basic	1.0	1.0	1.0	1.0
Char-String_Basic	0.9077	0.5	0.8333	0.625
Conditionals_Basic	0.6769	0.4375	0.8235	0.5714
Functions_Advanced	0.7692	0.5	0.9333	0.6512
Arithmetic_Advanced	0.9692	0.5	0.5	0.5
Conditionals_Switch	1.0	1.0	1.0	1.0
Algorithms_Recursion	0.9538	0.7273	1.0	0.8421
Structures_Advanced	0.9846	0.75	1.0	0.8571
Algorithms_DP	1.0	1.0	1.0	1.0
Conditionals_Advanced	0.8769	0.2	1.0	0.3333
Loops_Advanced	0.9231	0.9	0.8571	0.878
Arrays_Advanced	0.9692	0.75	1.0	0.8571
Char-String_Advanced	0.9231	0.1667	1.0	0.2857
TerminalIO_Basic	0.8308	0.0	0.0	0.0
Pointers_Basic	0.9077	0.4	1.0	0.5714
Structures_DS	1.0	1.0	1.0	1.0
Arrays_Memory	0.9692	0.75	0.75	0.75
Algorithms_DC	1.0	1.0	1.0	1.0
Functions_Basic	0.7538	0.4815	0.8667	0.619
Loops_Invariants	0.8769	0.1429	0.3333	0.2
Overall Metrics	0.9027	0.4928	0.7861	0.6058
Micro	0.9027	0.5214	0.7786	0.5933

Table 3.3: Early Fusion with Logistic Regression, Bag of words and different top feature selection

Program Label	Accuracy Score	Precision Score	Recall Score	F1 Score
Arithmetic_Bit	0.9846	0.0	0.0	0.0
Algorithms_Greedy	0.9846	0.0	0.0	0.0
Arithmetic_Basic	0.8	0.5909	0.7647	0.6667
Loops_Basic	0.6769	0.3529	0.375	0.3636
TerminalIO_Advanced	0.8769	0.5	0.875	0.6364
Arrays_Basic	0.7692	0.375	1.0	0.5455
Pointers_Advanced	0.9846	0.5	1.0	0.6667
Conditionals_Flags	0.9077	0.2	0.3333	0.25
Structures_Basic	1.0	1.0	1.0	1.0
Char-String_Basic	0.9538	0.6667	1.0	0.8
Conditionals_Basic	0.7385	0.5	0.7647	0.6047
Functions_Advanced	0.8462	0.619	0.8667	0.7222
Arithmetic_Advanced	0.9538	0.0	0.0	0.0
Conditionals_Switch	1.0	1.0	1.0	1.0
Algorithms_Recursion	0.9538	0.7273	1.0	0.8421
Structures_Advanced	0.9538	0.5	0.6667	0.5714
Algorithms_DP	1.0	1.0	1.0	1.0
Conditionals_Advanced	0.8308	0.1538	1.0	0.2667
Loops_Advanced	0.9385	0.8696	0.9524	0.9091
Arrays_Advanced	0.9538	0.7143	0.8333	0.7692
Char-String_Advanced	0.9231	0.1667	1.0	0.2857
TerminalIO_Basic	0.8308	0.0	0.0	0.0
Pointers_Basic	0.8923	0.3333	0.75	0.4615
Structures_DS	1.0	1.0	1.0	1.0
Arrays_Memory	0.9692	0.6667	1.0	0.8
Algorithms_DC	0.9231	0.1667	1.0	0.2857
Functions_Basic	0.7385	0.4688	1.0	0.6383
Loops_Invariants	0.9385	0.0	0.0	0.0
Overall	0.9044	0.4982	0.7803	0.6081
Micro	0.9044	0.549	0.7752	0.6088

Table 3.4: Early Fusion with Logistic Regression, FastText embeddings and different top feature selection

3.2.0.3 Hyperparameter tuning:

We have two hyperparameter values to tune, the two values are K_1 (number of top features to be selected for Statement features), K_2 (number of top features to be selected for AST features). Given the sparse labeling in our training dataset and our use of a Ranking-style prediction method for early fusion, choosing the best hyperparameters (i.e., the number of top features from both statement-based and AST-based features) proved to be difficult. We used an iterative process to gradually raise these hyperparameter values until we saw appreciable improvements in the overall F-score. However, reaching a saturation limit beyond which additional increments were ineffective in producing appreciable gains in the training set's F-score. This saturation point is where we set the hyperparameters.

3.2.0.4 Model Prediction

For model prediction, we leverage two distinct methodologies utilizing the predicted label probability scores outputted by the model:

1. Prediction up to Top T Labels:

In this method, the labels are initially predicted using the trained model. The resulting probability scores for each label, denoted as P_i , are obtained. Subsequently, only the top T labels with the highest scores are retained. Mathematically, the prediction is expressed as:

$$\text{Predicted Labels} = \{i \mid P_i \text{ are in the top } T \text{ highest probabilities}\}$$

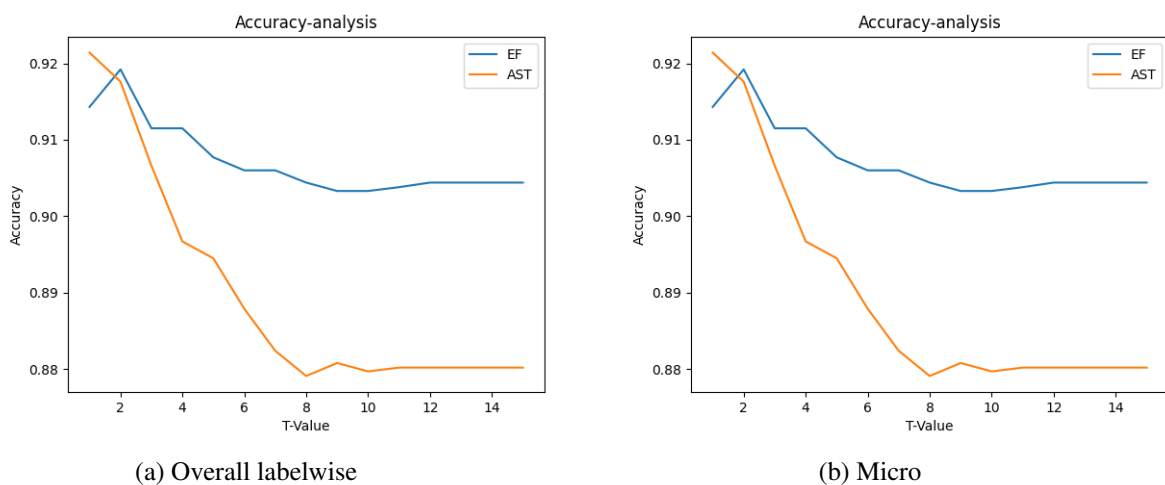


Figure 3.1: Early Fusion Accuracy Variation Overall Comparison for different T-value in Ranking style prediction

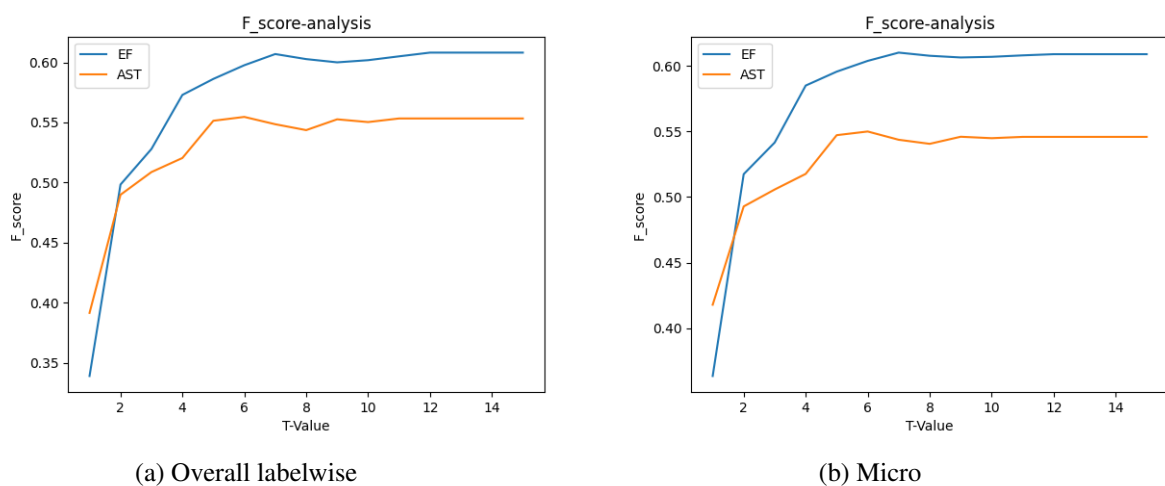


Figure 3.2: Early Fusion F-score Variation Overall Comparison for different T-values in Ranking style prediction

These prediction strategies enable the model to provide nuanced outputs based on the confidence levels of each label. The mathematical formulations encapsulate the process of selecting the

top T labels either directly from the sorted probability scores or from the initial set of predicted labels. These methodologies are essential for tailoring the model's output to meet specific criteria, such as choosing only the most probable labels for further consideration or limiting the output to a predefined number of top labels.

3.2.1 Results

- The results from table 3.1 and table 3.2 shows that the model's performance in ranking style prediction with logistic regression. This uses same k features selected from pool of all the features combined. Now it can be seen that the Fasttext based vectors outperforms the Bag of word based vectors in overall average over all the labels. However, Bag of words is also performing good in terms of the Micro metrics.
- By optimizing feature selection to focus on the most informative features from both the AST and statement feature sets, the model can better capture the nuanced characteristics of the input data. This targeted approach ensures that the model leverages the strengths of each feature type, thereby enhancing its ability to discern meaningful patterns and relationships in the data which can be seen in table 3.3 and table 3.4.
- Now, as can be seen in table 3.3 and table 3.4, fasttext based feature vectors outperforms the Bag of words features in both Micro and overall metric of accuracy and fscore.
- This model which used the ranking style prediction with different set of top features also shows a substantial increase in the accuracy and fscore of the model and beats the AST based model of these [3] and [16].
- As this is ranking-style prediction we are using in early fusion, we can see in the figures 3.1 and 3.2 the Early fusion model is able to beat the AST based model for different T values.

3.3 Deep Models

In this section, we delve into the application of Deep Neural Networks (DNNs) to address the challenges posed by the aforementioned problem. We aim to explore the versatility and expressive power of DNNs in capturing intricate patterns and relationships within the given Priority dataset. This section provide insights into the experimentation process and present the results obtained through the application of deep learning techniques.

Utilizing deep neural networks for this problem presents several advantages:

1. Capturing Complex Relationships:

Deep neural networks excel at capturing intricate and non-linear relationships within data. Given the complex nature of programming problems and their corresponding labels, deep networks can potentially unveil intricate dependencies that may be challenging for traditional models to discern.

Table 3.5: write new

Label	Accuracy	Precision	Recall	F1_score
Arithmetic_Bit	0.9846	0.0	0.0	0.0
Algorithms_Greedy	0.9846	0.0	0.0	0.0
Arithmetic_Basic	0.8	0.5294	0.6429	0.5806
Loops_Basic	0.7846	0.625	0.5556	0.5882
TerminalIO_Advanced	0.9231	0.375	1.0	0.5455
Arrays_Basic	0.8769	0.7778	0.5385	0.6364
Pointers_Advanced	0.9846	0.0	0.0	0.0
Conditionals_Flags	0.9385	0.0	0.0	0.0
Structures_Basic	0.9846	0.0	0.0	0.0
Char-String_Basic	0.9231	0.5	0.6	0.5455
Conditionals_Basic	0.7385	0.3529	0.5	0.4138
Functions_Advanced	0.8462	0.6667	0.6667	0.6667
Arithmetic_Advanced	0.9692	0.0	0.0	0.0
Conditionals_Switch	1.0	1.0	1.0	1.0
Algorithms_Recursion	0.8769	0.5	0.5	0.5
Structures_Advanced	0.9538	0.3333	0.5	0.4
Algorithms_DP	0.9846	0.0	0.0	0.0
Conditionals_Advanced	0.9538	0.0	0.0	0.0
Loops_Advanced	0.8615	0.8095	0.7727	0.7907
Arrays_Advanced	0.9538	0.6667	0.8	0.7273
Char-String_Advanced	0.9692	1.0	0.3333	0.5
TerminalIO_Basic	0.9231	0.75	0.4286	0.5455
Pointers_Basic	0.9538	0.5	0.6667	0.5714
Structures_DS	0.9692	0.0	0.0	0.0
Arrays_Memory	0.9538	0.5	0.6667	0.5714
Algorithms_DC	0.9846	0.0	0.0	0.0
Functions_Basic	0.7385	0.3333	0.4167	0.3704
Loops_Invariants	0.9538	0.0	0.0	0.0
Overall	0.9203	0.5087	0.5946	0.5483
Micro	0.9203	0.5645	0.6128	0.5634

Table 3.6: Wide and Deep Predictions

2. Automatic Feature Learning:

Deep networks possess the ability to automatically learn hierarchical representations of features from raw data. This can be particularly advantageous for tasks where the relevant features are not explicitly defined, allowing the model to autonomously discover relevant patterns and representations.

3. End-to-End Learning:

Deep networks facilitate end-to-end learning, enabling the model to directly map input data to output predictions. This can streamline the training process and potentially enhance the model's ability to discern intricate relationships between problem statements and corresponding labels.

4. Transfer Learning:

Deep neural networks, especially those pre-trained on large datasets, can leverage transfer

learning. This allows the model to transfer knowledge gained from a different but related task, potentially boosting performance, especially in scenarios like Priority dataset with limited labeled data.

5. **Adaptability to Various Data Types:**

Deep networks are adaptable to different data types, including text and numerical data. Given the diverse nature of the features involved in programming problems, the flexibility of deep learning architectures makes them well-suited for handling a variety of input modalities.

6. **Potential for Ensemble Learning:**

Deep networks can be seamlessly integrated into ensemble learning frameworks. Combining multiple deep models can lead to improved predictive performance, harnessing the diversity of individual models to enhance overall accuracy and robustness.

In summary, leveraging deep neural networks for this problem offers the potential to uncover intricate relationships, automatically learn relevant features, handle large-scale datasets, and benefit from transfer learning and ensemble strategies. In the following sections we discuss two approaches we took to get the desired predictions:

3.3.1 Wide and Deep

Wide and Deep Neural Networks represent a hybrid architecture that combines the strengths of both deep learning and traditional machine learning. The "wide" component refers to a linear model with a broad feature space, capturing simple and direct relationships. In contrast, the "deep" component involves a neural network with multiple hidden layers, allowing for the extraction of complex hierarchical patterns. The generic architecture for is shown in [3.3](#) [\[4\]](#)

The wide component excels at memorizing feature combinations, while the deep component focuses on generalization and learning intricate relationships. Together, they offer a powerful framework for handling a variety of data types and capturing both shallow and deep insights from the input data.

By combining memorization and generalization capabilities, wide and deep neural networks provide a flexible and effective solution for tasks that demand a balance between capturing simple and complex patterns in the data.

3.3.1.1 Implemented Architecture:

In our implementation of the wide and deep neural network, we propose the following architecture.

- Wide Component (AST-based features)

The wide component is designed to capture immediate correlations and interactions between features. For AST-based features, which represent the structural properties of code, this component can be expressed as:

$$Y_{\text{wide}} = \text{Dense}(28, \text{activation} = \text{'relu'})(X_{\text{wide}})$$

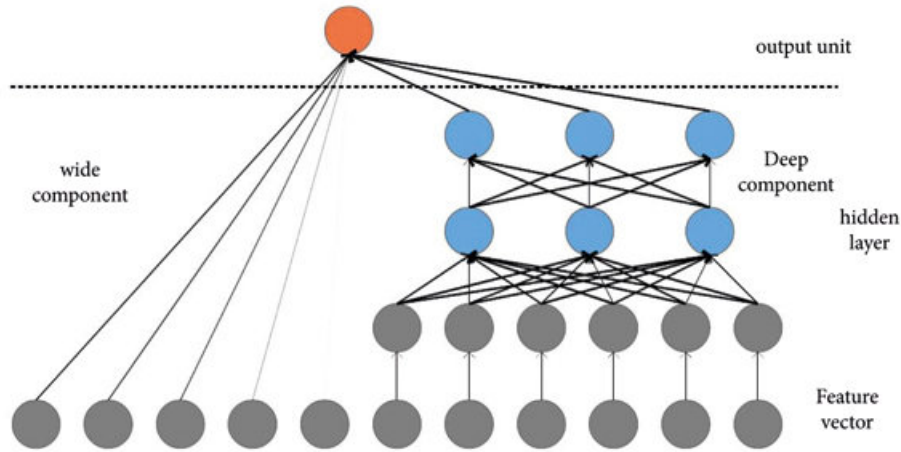


Figure 3.3: Wide and Deep Generic Architecture [4]

where X_{wide} is the input vector of AST-based features.

- Deep Component (FastText embeddings)

The deep component excels at learning complex and hierarchical representations. For FastText embeddings, capturing semantic information from code snippets, the deep component can be expressed as:

$$Y_{\text{deep}} = \text{Dense}(30, \text{activation} = \text{'relu'}) (\text{Dropout}(0.2) (\text{Dense}(64, \text{activation} = \text{'relu'}) (\text{Dropout}(0.2) (X_{\text{deep}}))))$$

Here, X_{deep} represents the input vector of FastText embeddings. The deep component consists of two dense layers with ReLU activation and dropout layers in between for regularization.

- Combining Wide and Deep Components

The outputs from the wide and deep components are combined to obtain the final predictions:

$$Y_{\text{final}} = \text{Dense}(28, \text{activation} = \text{'sigmoid'}) (\text{Concatenate}([Y_{\text{deep}}, Y_{\text{wide}}]))$$

Here, $[Y_{\text{deep}}, Y_{\text{wide}}]$ denotes the concatenation of predictions from the deep and wide components. The final combination includes a dense layer with a sigmoid activation to ensure the output is in the $[0, 1]$ range for binary classification.

3.3.2 Deep and Deep

For experimentation, this approach was explored, termed "Deep and Deep," wherein both AST and Statement features were separately fed into distinct neural networks. Each network independently processed the features, capturing intricate patterns and representations. Subsequently, the outputs of these networks were combined into a unified representation. This approach aimed to harness the unique strengths of each feature set through dedicated processing, enabling the networks to

Label	Accuracy	Precision	Recall	F1_score
Arithmetic_Bit	0.9846	0.0	0.0	0.0
Algorithms_Greedy	0.9846	0.0	0.0	0.0
Arithmetic_Basic	0.8154	0.5294	0.6923	0.6
Loops_Basic	0.7077	0.5625	0.4286	0.4865
TerminalIO_Advanced	0.9385	0.625	0.8333	0.7143
Arrays_Basic	0.8308	0.6667	0.4286	0.5217
Pointers_Advanced	0.9846	0.0	0.0	0.0
Conditionals_Flags	0.9077	0.0	0.0	0.0
Structures_Basic	0.9846	0.0	0.0	0.0
Char-String_Basic	0.9231	0.6667	0.5714	0.6154
Conditionals_Basic	0.7077	0.2353	0.4	0.2963
Functions_Advanced	0.8462	0.6667	0.6667	0.6667
Arithmetic_Advanced	0.9692	0.0	0.0	0.0
Conditionals_Switch	1.0	1.0	1.0	1.0
Algorithms_Recursion	0.8769	0.375	0.5	0.4286
Structures_Advanced	0.9538	0.0	0.0	0.0
Algorithms_DP	0.9692	0.0	0.0	0.0
Conditionals_Advanced	0.8923	0.0	0.0	0.0
Loops_Advanced	0.8615	0.7143	0.8333	0.7692
Arrays_Advanced	0.9538	0.6667	0.8	0.7273
Char-String_Advanced	0.9846	0.0	0.0	0.0
TerminalIO_Basic	0.9385	0.5	0.5	0.5
Pointers_Basic	0.9385	0.5	0.5	0.5
Structures_DS	0.9846	0.5	1.0	0.6667
Arrays_Memory	0.9538	0.75	0.6	0.6667
Algorithms_DC	0.9846	0.0	0.0	0.0
Functions_Basic	0.7538	0.2667	0.4444	0.3333
Loops_Invariants	0.9538	0.0	0.0	0.0
Overall	0.9137	0.474	0.5541	0.5109
Micro	0.9137	0.5346	0.6423	0.5505

Table 3.7: Deep and Deep Predictions

specialize in their respective domains. The intention was to leverage the complementary nature of AST and Statement features, allowing each network to focus on its inherent characteristics. This experimentation sought to evaluate whether the collaboration of two individually trained networks could yield enhanced predictive performance compared to other fusion strategies.

3.3.2.1 Deep and Deep Architecture:

The model comprises two main components: one for processing AST inputs and another for processing text inputs.

- **Text Part**

It takes the FastText Embeddings as input and then use for further processing.

$\text{Text_Out} = \text{Dense}(28, \text{ReLU})(\text{Text_Inputs})$

Here, Text_Inputs is the Fasttext Embedding.

- **AST Part**

It takes the AST features vector as input and transforms it to a 30 sized concentrated vectors.

$AST = \text{Dense}(30, \text{ReLU})(AST_Inputs)$ $AST_Out = \text{Dense}(30, \text{ReLU})(AST)$

- **Combining Parts**

Before, Final output combining both the new sets.

$\text{Combined} = \text{Concatenate}([AST_Out, \text{Text_Out}])$

- **Output Layer**

Outputs a set of 28 vectors where each corresponds to a label.

$\text{Output} = \text{Dense}(28, \text{Sigmoid})(\text{Combined})$

This architecture effectively integrates information from both AST and text representations, providing a comprehensive approach for label prediction.

Label	Accuracy	Precision	Recall	F1_score
Arithmetic_Bit	0.9846	0.0	0.0	0.0
Algorithms_Greedy	0.9846	0.0	0.0	0.0
Arithmetic_Basic	0.8	0.4706	0.6667	0.5517
Loops_Basic	0.7692	0.5625	0.5294	0.5455
TerminalIO_Advanced	0.9231	0.375	1.0	0.5455
Arrays_Basic	0.8462	0.6667	0.4615	0.5455
Pointers_Advanced	0.9846	0.0	0.0	0.0
Conditionals_Flags	0.9538	0.0	0.0	0.0
Structures_Basic	0.9538	0.0	0.0	0.0
Char-String_Basic	0.9077	0.5	0.5	0.5
Conditionals_Basic	0.7538	0.2353	0.5714	0.3333
Functions_Advanced	0.8154	0.4	0.6667	0.5
Arithmetic_Advanced	0.9692	0.0	0.0	0.0
Conditionals_Switch	1.0	1.0	1.0	1.0
Algorithms_Recursion	0.8769	0.375	0.5	0.4286
Structures_Advanced	0.9385	0.0	0.0	0.0
Algorithms_DP	0.9846	0.0	0.0	0.0
Conditionals_Advanced	0.9385	0.0	0.0	0.0
Loops_Advanced	0.8615	0.7143	0.8333	0.7692
Arrays_Advanced	0.9692	0.8333	0.8333	0.8333
Char-String_Advanced	0.9385	0.0	0.0	0.0
TerminalIO_Basic	0.9692	0.75	0.75	0.75
Pointers_Basic	0.9385	0.5	0.5	0.5
Structures_DS	0.9692	0.0	0.0	0.0
Arrays_Memory	0.9538	0.5	0.6667	0.5714
Algorithms_DC	1.0	1.0	1.0	1.0
Functions_Basic	0.7385	0.2667	0.4	0.32
Loops_Invariants	0.9538	0.0	0.0	0.0
Overall	0.917	0.4335	0.5859	0.4983
Micro	0.917	0.4897	0.6154	0.5211

Table 3.8: Wide and Deep with custom loss Prediction

3.3.2.2 Loss Funtion:

We have used Binary Cross Entropy Loss. Binary Cross-Entropy is a loss function commonly used for binary classification problems. In the context of this experimentation, when dealing with a multi-label classification task (each label is treated independently as a binary classification), binary cross-entropy is suitable. Mathematically, for a single data point with n labels, the binary cross-entropy loss is calculated as follows:

$$L(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^n [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

where y_i is the true label, \hat{y}_i is the predicted probability of the label being 'on', and the sum is taken over all labels losses.

3.3.2.3 Custom Loss:

Now, since the DNN based models were noticed to be biased for the frequently occurring labels and is doing really good in them but many minority labels were getting ignored by the model. Hence, in order to filter this phenomena we modified the Binary Cross-entropy loss to uplift the minority labels. The method adopted was some sort of weighted Binary CrossEntropy loss. Following are the steps to calculate it:

- Calculate the Binary CrossEntropy loss per class (L_i).
- Calculate the scale value for each class as below:
 1. Count the number of samples in each class in your training data. Let's denote the counts as *classcounts*, where *classcounts*[i] represents the number of samples in class i.
 2. Compute the inverse of the class frequencies by dividing the total number of samples by the number of samples in each class:

$$\text{classfrequencies} = \text{sum}(\text{classcounts}) / \text{classcounts}$$

3. Normalize the class frequencies to sum up to the number of classes. This step ensures that the scale factors form a valid probability distribution:

$$\text{normalizedfrequencies} = \text{classfrequencies} / \text{sum}(\text{classfrequencies})$$

4. adjust the scale factors by raising them to a power (k). This can be useful to further emphasize the importance of smaller classes:

$$\text{adjustedfrequencies} = \text{normalizedfrequencies}^k$$

5. The *adjustedfrequencies* array now contains the scale factors for each class. You can use these scale factors as class weights during the training of your deep model.

- Now, the Cross-entropy loss can be taken as:

$$L_{new} = L_{old} * adjustedfrequencies$$

Label	Accuracy	Precision	Recall	F1_score
Arithmetic_Bit	0.9846	0.0	0.0	0.0
Algorithms_Greedy	0.9538	0.0	0.0	0.0
Arithmetic_Basic	0.8308	0.5294	0.75	0.6207
Loops_Basic	0.8308	0.5625	0.6923	0.6207
TerminalIO_Advanced	0.8769	0.375	0.5	0.4286
Arrays_Basic	0.8462	0.5556	0.4545	0.5
Pointers_Advanced	0.9692	0.0	0.0	0.0
Conditionals_Flags	0.9231	0.0	0.0	0.0
Structures_Basic	0.9846	0.0	0.0	0.0
Char-String_Basic	0.8923	0.3333	0.4	0.3636
Conditionals_Basic	0.7385	0.2353	0.5	0.32
Functions_Advanced	0.8615	0.4	1.0	0.5714
Arithmetic_Advanced	0.9692	0.0	0.0	0.0
Conditionals_Switch	1.0	1.0	1.0	1.0
Algorithms_Recursion	0.8615	0.125	0.3333	0.1818
Structures_Advanced	0.9385	0.0	0.0	0.0
Algorithms_DP	1.0	1.0	1.0	1.0
Conditionals_Advanced	0.9077	0.0	0.0	0.0
Loops_Advanced	0.8308	0.6667	0.7778	0.7179
Arrays_Advanced	0.9385	0.6667	0.6667	0.6667
Char-String_Advanced	0.9692	0.0	0.0	0.0
TerminalIO_Basic	0.9385	0.5	0.5	0.5
Pointers_Basic	0.9231	0.25	0.3333	0.2857
Structures_DS	1.0	1.0	1.0	1.0
Arrays_Memory	0.9538	0.5	0.6667	0.5714
Algorithms_DC	0.9846	0.0	0.0	0.0
Functions_Basic	0.7692	0.4	0.5	0.4444
Loops_Invariants	0.9231	0.0	0.0	0.0
Overall	0.9143	0.4162	0.5669	0.48
Micro	0.9143	0.4709	0.5951	0.4934

Table 3.9: Deep and Deep with custom loss predictions

3.3.2.4 Optimizer:

Adam (Adaptive Moment Estimation) [11] is used for our problem combined with the Binary cross entropy loss. It is a widely used algorithm for multi-label classification. The most important feature of Adam optimizer is adaptive learning rates for particular parameters.

Mathematically, the update rule for the parameters θ in Adam is given by:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

where η is the learning rate, \hat{m}_t is the exponentially decaying average of past gradients, \hat{v}_t is the

exponentially decaying average of past squared gradients, and ϵ is a small constant to prevent division by zero. Using binary cross-entropy loss with Adam optimizer is a common for multi-label classification tasks. Overall, the training process involves iteratively updating the model parameters using the Adam optimizer and binary cross-entropy loss, while leveraging the complementary information from both statement-based and AST-based features to improve the accuracy and effectiveness of the multiclass classification model.

3.3.3 Results of Deep learning based predictions:

- The tables [3.6](#) and [3.7](#) results shows how the wide and deep architecture performs better than the deep and deep architecture in terms of both the overall and micro metrics including the accuracy and fscore.
- The wide and deep network architecture combines the strengths of both wide and deep learning approaches, leveraging a hybrid model that incorporates both shallow and deep layers. This design allows the model to capture both low-level features and high-level representations, enabling it to learn complex patterns and relationships in the data more effectively.
- Another thing to notice here is that despite the deep learning model's considerable capabilities it fails to surpass the performance achieved by the early fusion approach. This is mostly because the deep learning models is getting optimised over the overall metric due to which it is more biased towards the dominant classes.
- To overcome this, we tried adding a custom loss which takes a scaled loss which will try to create a more balanced loss for the overall classes. The results from tables [3.9](#) and [3.8](#) shows the output after adding custom loss.
- Still the deep learning based methods is failing to address the class imbalance issue and continues to learn the dominant classes.

Future Work

In this chapter, we will discuss some future avenues for research and improvements to enhance the performance and capabilities of the existing models. The following are the directions which can be further explored for making further advancements in label prediction and problem tagging:

- SaimeseXML [7] could be used to make the label prediction task more robust by using extreme multiclass classifiers.
- Some more data collection for the data set will also increase the robustness of the models. So, a direction to explore it is data collection to apply the techniques to a furthermore diverse set of data points for training.
- Better ensemble techniques can be explored in direction to combine the statement features and AST based features.
- Graph Neural Networks (GNNs) to improve the learning of AST-based features. By representing the abstract syntax tree (AST) as a graph structure, GNNs can effectively capture the hierarchical relationships between different nodes in the tree. This approach has the potential to better exploit the inherent structure and semantics of the code, leading to more accurate label predictions as discussed in thesis [3].

Conclusion

In this thesis, we have made significant revisions to the Prutor platform's backend machine learning algorithm, PRIORITY, before integration. By combining statement features and AST-based feature vectors in a clever way and adding careful feature selection and model tweaking, we have seen notable improvements in F-score and overall accuracy. Notably, combining these factors has not only made forecasts more accurate overall but has also made it possible to identify and anticipate classes with lower enrollment. These improvements highlight how well our improved algorithm performs in raising the bar for the machine learning features of the Prutor platform.

Bibliography

- [1] Steven Bird, Edward Loper, and Ewan Klein. *Natural Language Processing with Python*, 20XX. Online documentation.
- [2] Leo Breiman. Random forest. In *Machine Learning*, volume 45, pages 5–32, 2001.
- [3] Debanjan Chatterjee. Intelligent program analysis and program indexing - i, 2022.
- [4] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems, 2016.
- [5] Corinna Cortes and Vladimir Vapnik. Support vector machines. *Machine Learning*, 20(3):273–297, 1995.
- [6] Thomas M. Cover and Peter E. Hart. k-nearest neighbors. In *IEEE Transactions on Information Theory*, volume 13, pages 21–27, 1967.
- [7] Kunal Dahiya, Ananye Agarwal, Deepak Saini, Gururaj K, Jian Jiao, Amit Singh, Sumeet Agarwal, Purushottam Kar, and Manik Varma. Siamesexml: Siamese networks meet extreme classifiers with 100m labels. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2330–2340. PMLR, 18–24 Jul 2021.
- [8] Python Software Foundation. html.parser - simple html and xhtml parser, 2023.
- [9] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Logistic regression. *The Elements of Statistical Learning*, 2009.
- [10] Sharath Hp. Real world deployments of ai-assisted tools for compilation error repair and program retrieval, 2021.
- [11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- [12] Python Software Foundation. re — regular expression operations, 2021.
- [13] J. Ross Quinlan. Decision trees. *Machine Learning*, 1(1):81–106, 1986.
- [14] Ayush Sahni. Program repair and retrieval on the prutor platform - i(unpublished), 2024.
- [15] Fahad Shaikh. Advancements in ai-assisted compilation error repair and program retrieval, 2021.
- [16] Preeti Singh. Intelligent program analysis and program indexing - ii, 2022.
- [17] Utkarsh Srivastava. Program repair and retrieval on the prutor platform - iii(unpublished), 2024.