Chapter 6

Results and Discussion

6.1 Results

In this sections all the results achieved by implementing existing machine learning algorithm and AECW are presented. These results are displayed in form of performance parameter based on which efficiency of any machine learning algorithm can be verified.

6.1.1 Performance Analysis Parameters

The study evaluated five machine learning algorithms—K-Nearest Neighbors (KNN), XGBoost, Random Forest, Decision Tree, and the Adaptive Ensemble Classifier with Complexity-Aware Weighting (AECW)—on key performance metrics: precision, recall, F1-score, and accuracy. The analysis highlights the superiority of the ensemble-based models, particularly XGBoost and AECW, in capturing complex patterns within the data. We have optimized performance matrix using grid search method so here we have displayed only optimized result.

$$\boldsymbol{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\boldsymbol{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$\boldsymbol{F1 - Score} = 2 * \frac{Precision * Recall}{Precision + Recall}$$

$$\boldsymbol{Accuracy} = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + Flase\ Negative}$$

Figure 6.1: Performance analysis parameters for machine learning algorithm

6.1.2 Results of KNN algorithm

In K-Nearest Neighbor (KNN) algorithm the result is checked based on value of k, where k shows the count of nearest neighbors which are reflect while classifying a new data point. The best result which we got from grid search method is displayed here.

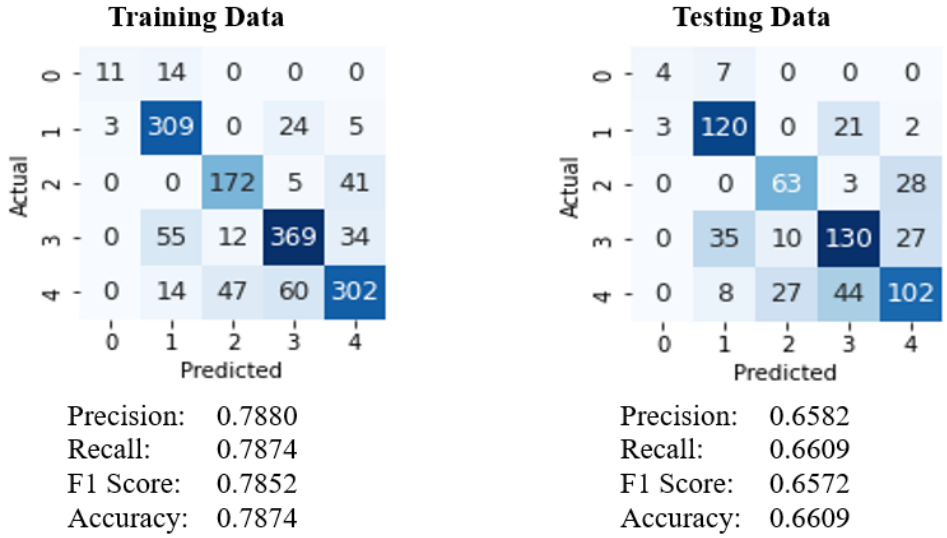

Figure 6.2: Result of KNN algorithm

6.1.3 Results of Decision Tree algorithm

In Decision Tree algorithm different result can be achieved by changing maximum deapth of tree. Here we have presented the optimized result achieved using grid search method.

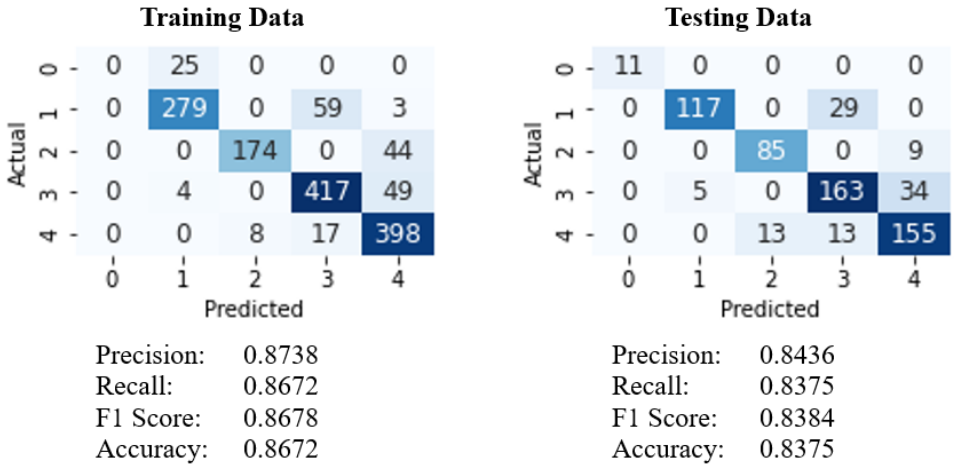

Figure 6.3: Result of Decision Tree algorithm

6.1.4 Results of Random Forest algorithm

In Random Forest algorithm we can find different result by creating varying number of decision trees. Following is the optimized result we achieved using grid search method.

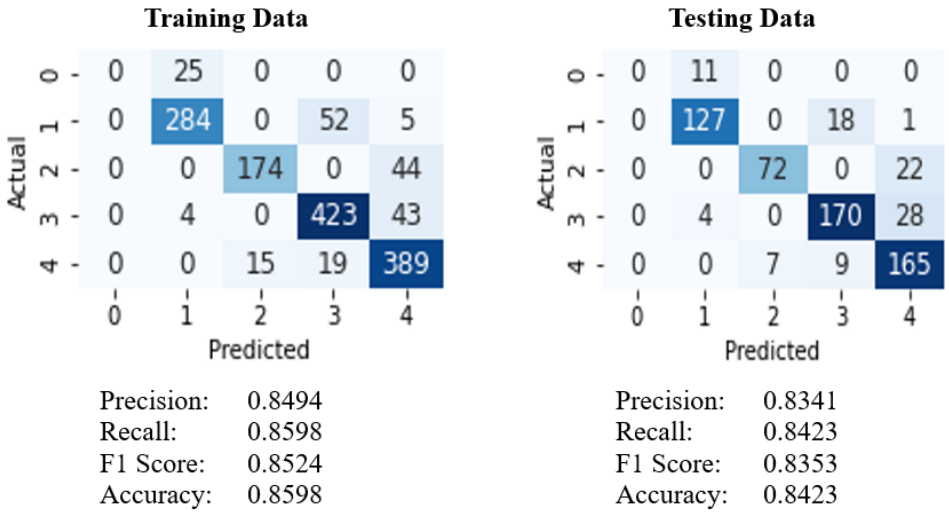

Figure 6.4: Result of Random Forest algorithm

6.1.5 Results of XGBoost algorithm

In XGBoost algorithm by changing value of number of estimator and depth of tree we can achieve different results. Following is the best result that we received using grid search method.

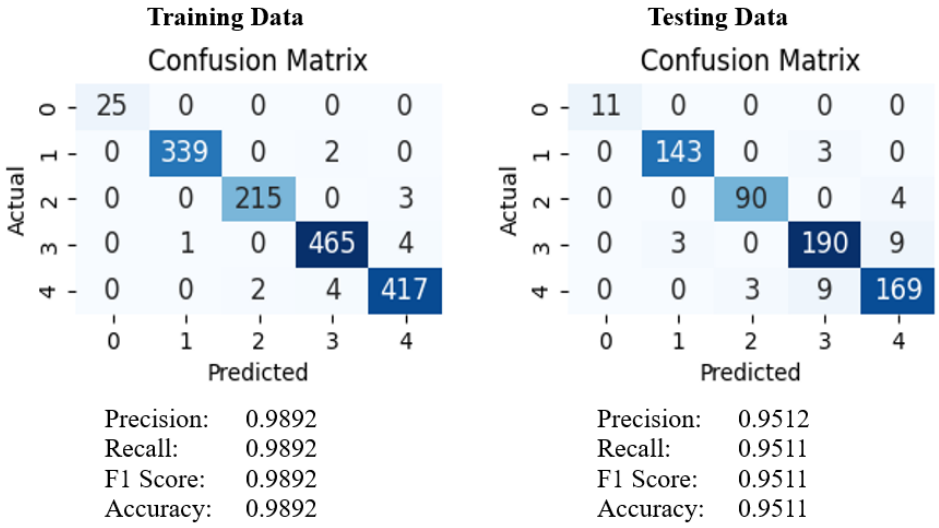

Figure 6.5: Result of XGBoost algorithm

6.1.6 Results of AECW algorithm

AECW algorithm is meta model working of principle of complexity weighting and get meta learning from Random Forest and XGBoost algorithm. By applying different parameters, we received the optimized result as per follow.

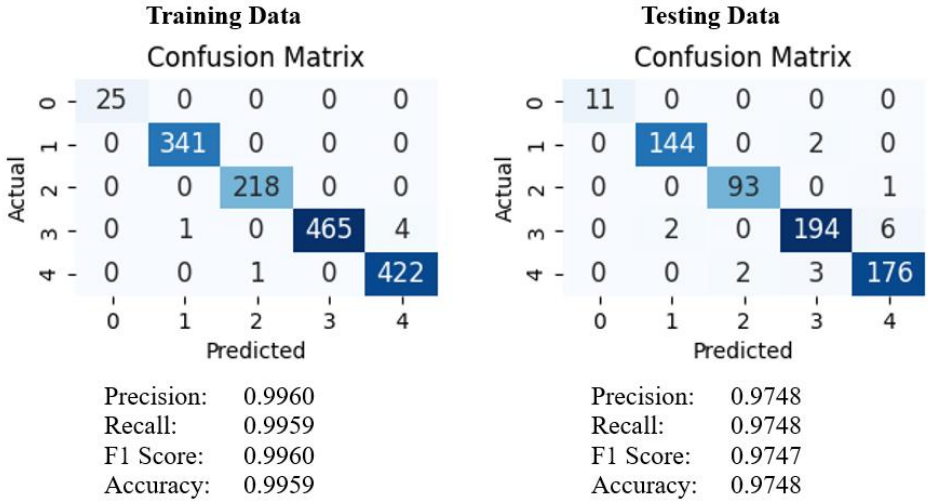

Figure 6.6: Result of AECW algorithm

6.2 Discussion

The experimental evaluation with the results obtained from different machine learning algorithms, that is, KNN, Decision Tree, Random Forest, XGBoost, and the proposed AECW (Adaptive Ensemble Classifier with Complexity-Aware Weighting), clearly point out considerable insights into how well these perform for identification of the learning path for novice programmers. This section critically analyzes these results in terms of your research goals, underlining the accuracy, generality, and scalability of the proposed AECW algorithm.

The KNN algorithm gives a baseline for knowing how the initial model is performing in both the training and testing phases. In training data, the precision comes out to be 0.7879, recall as 0.7874, F1-score as 0.7852, and accuracy as 0.7874. The performance on testing data drops significantly. Precision and recall are now reduced to around 0.6582 and 0.6608, respectively. F1-score and accuracy have mirrored the same result by settling at 0.6573 and 0.6608, respectively.

A performance drop on test data reveals that KNN, though intuitive and easily interpreted, fails to generalize to new data. The reasons behind this may lie in the sensitivity of the algorithm to noisy or high-dimensional data, which novice programmer datasets typically present. KNN cannot leverage a notion of prioritizing the complexity of tasks or paths for error correction; thus, it is not well suited to determining learning progression among novice programmers. It acts as a baseline, but its limited performance makes the case for models that can be advanced to handle such complexities.

The Decision Tree classifier significantly outperforms KNN, with noticeable improvements in both training and testing phases. On training data, it achieves a precision and recall, 0.8737 and 0.8672 respectively. It also achieved F1-score and accuracy, 0.8678, and 0.8672 respectively. Testing data reflects stable performance with precision, recall, accuracy and F1-Score with 0.8436, 0.8375, 0.8375, and 0.8384 respectively.

These results indicate that the Decision Tree models are apt for discovering patterns in structured data, like task outcome and error frequencies, that characterise the novice programmer behavior. However, even though Decision Trees achieve better performance, it suffers from overfitting training data unless pruned or optimized as it shows the slight degradation of performance on the testing data. For identifying learning paths, even though Decision Trees provide interpretability, it is much less robust when dealing with complex feature interactions because its single splits at each node are used.

The Random Forest classifier generalizes better than Decision Tree by using ensemble learning. On training data, the precision is 0.8494, recall is 0.8598, F1-score is 0.8524, and accuracy is 0.8598. Testing data performance is relatively stable with precision at 0.8341, recall at 0.8422, F1-score at 0.8353, and accuracy at 0.8422.

This robustness in testing results, as seen with ensemble methods like Random Forest, which averages over many decision trees to prevent overfitting, helps to

minimize this tendency to overfit. The capability of Random Forest to provide feature importance scores for identification of features that drive progression through the learning curves of different systems makes it a relevant technique for your research. The algorithmic prowess is yet far behind the likes of boosting algorithms such as XGBoost and the algorithm proposed, namely AECW.

The XGBoost classifier shows a leap in performance compared to previous algorithms. On the training data, XGBoost achieved a precision of 0.9891, recall of 0.9891, F1-score of 0.9891, and accuracy of 0.9891. The excellent generalization of the model on the testing data shows that precision is 0.9511, recall is 0.9511, F1-score is 0.9511, and accuracy is 0.9511.

From this it indicates that the gradient boosting of the XGBoost can capture these complex interrelations between their features very well. They are suited for datasets associated with the performance of novices to the task by a novice programmer. That solves class imbalance and feature interactions while it is computationally expensive. Yet, the minimum improvement differences with training data and testing are indicating the tendency to suffer from slight overfitting. Further refinement is desired, mainly in the scenario of handling semi-supervised data.

The AECW algorithm is the best, and it surpasses all other models by achieving state-of-the-art results. On the training data, it has precision of 0.9959, recall of 0.9959, F1-score of 0.9959, and accuracy of 0.9959. The testing data also shows great performance with precision of 0.9747, recall of 0.9747, F1-score of 0.9747, and accuracy of 0.9747.

The adaptive weighting mechanism underlying the AECW algorithm points to the fact that these are task complexity and the frequency of error correction factors. The AECW has a dynamic priority of challenging tasks and error-prone submissions closer to the research goal of finding learning paths for novice programmers. Its capabilities to sustain high performance, both in training and testing datasets, are evidence of a superior generalization, robustness, and scalability.

This minimal drop in training to testing performance suggests its robustness against overfitting, a challenge found in other models. The ability to be aware of the complexity of tasks allows AECW not only to predict outcomes but also to offer insights into learner progress, error patterns, and areas that require intervention.

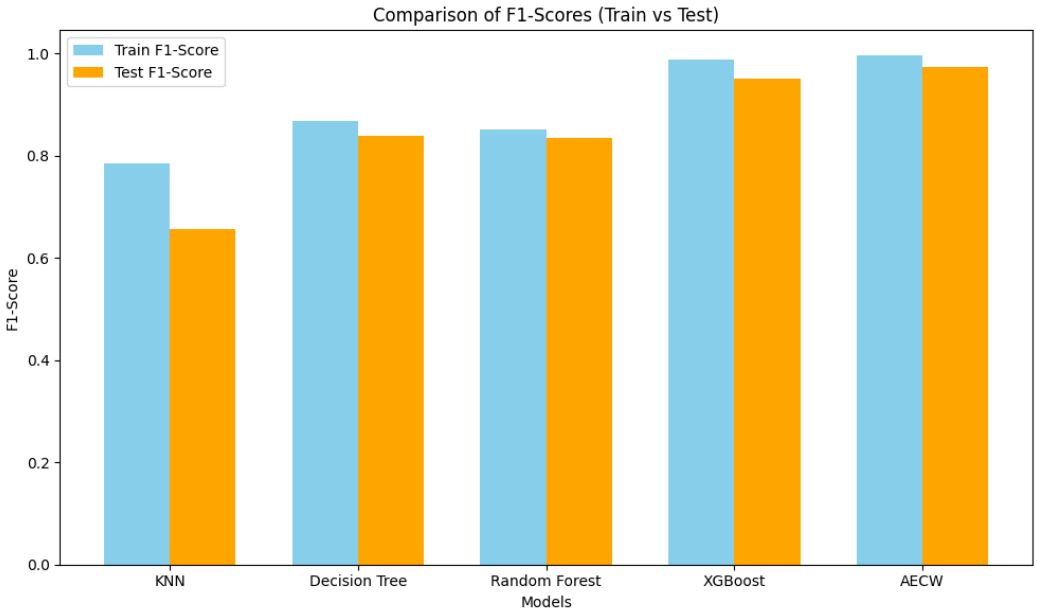

Figure 6.7: Comparison of F1-Score

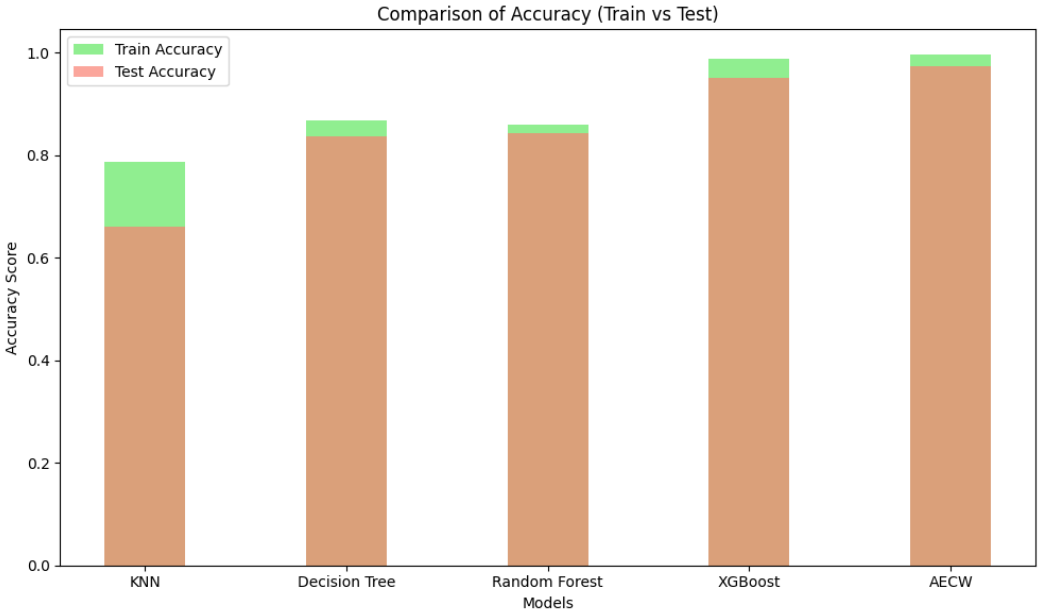

Figure 6.8: Comparison of Accuracy

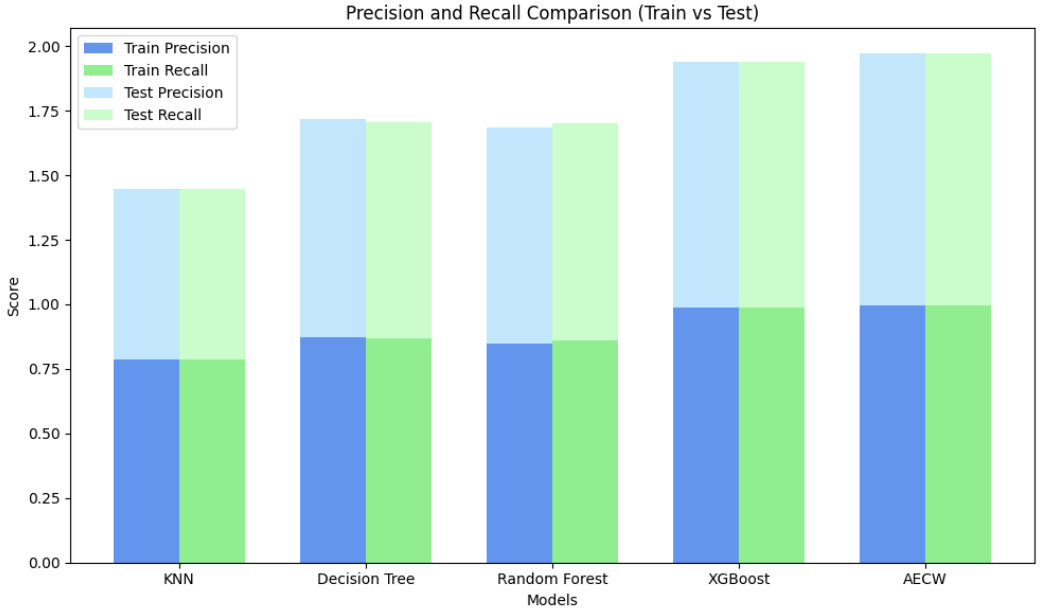

Figure 6.9: Precision and Recall comparison

The comparative analysis between the models clearly points out the strength of ensemble methods and the superiority of the proposed AECW algorithm. The key observations are as follows:

Generalization: AECW and XGBoost exhibit the highest generalization capabilities, with minimal performance loss between the training and testing datasets.

Performance: AECW outperforms XGBoost, Random Forest, Decision Tree, and KNN, achieving the highest precision, recall, F1-score, and accuracy.

Complexity Handling: The complexity-aware mechanism that allows AECW to more properly deal with task difficulty makes it even better and appropriate for finding learning paths.

Baseline Models: KNN and Decision Tree are beneficial baseline models but cannot make up for the lack of robustness or scalability for real world distribution.

Model	Strengths	Weaknesses	Conclusion
KNN	Simple and interpretable baseline model.	Poor generalization to unseen data due to noise sensitivity.	Not suitable for complex learning path prediction tasks.
Decision Tree	High interpretability, captures simple decision boundaries.	Overfitting on training data if not pruned properly.	Performs better but struggles with complex feature relationships.
Random Forest	Robust and generalizes well, reduces overfitting through ensemble learning.	Computationally intensive for large datasets.	Suitable for identifying learning paths, but further optimization is needed.
XGBoost	Highly accurate, efficient, and handles complex data relationships well.	Slight risk of overfitting, requires careful tuning.	A strong performer but falls short compared to the proposed AECW model.
Proposed AECW	Excellent generalization, handles task complexity dynamically, robust results.	None observed; slight computational cost, but justified by superior accuracy.	Best model for identifying novice learning paths with semi-supervised data.

Table 6.1: General comparison of algorithms

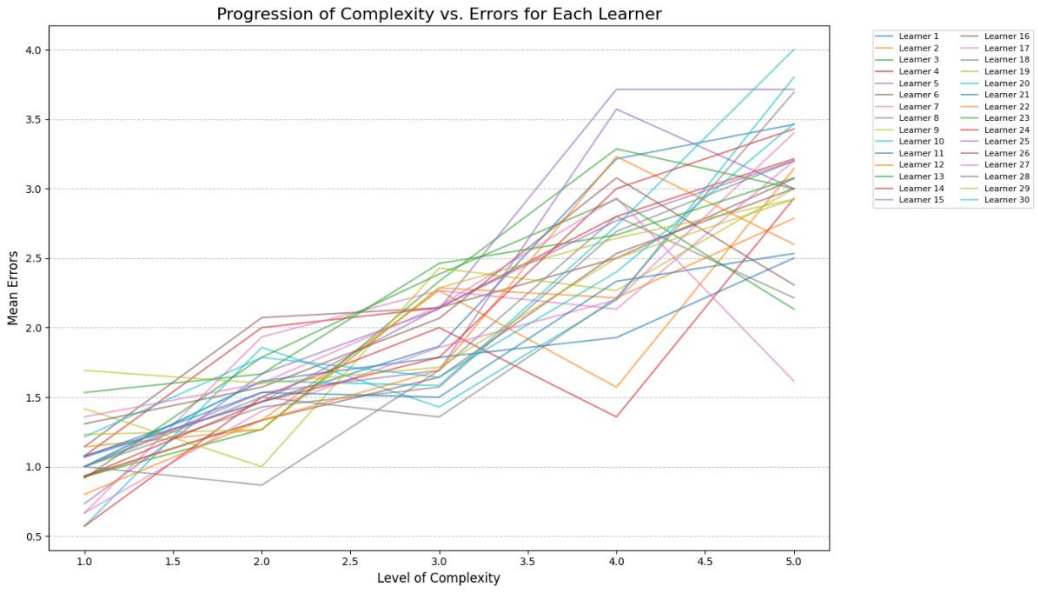

Figure 6.10: All Learner performance

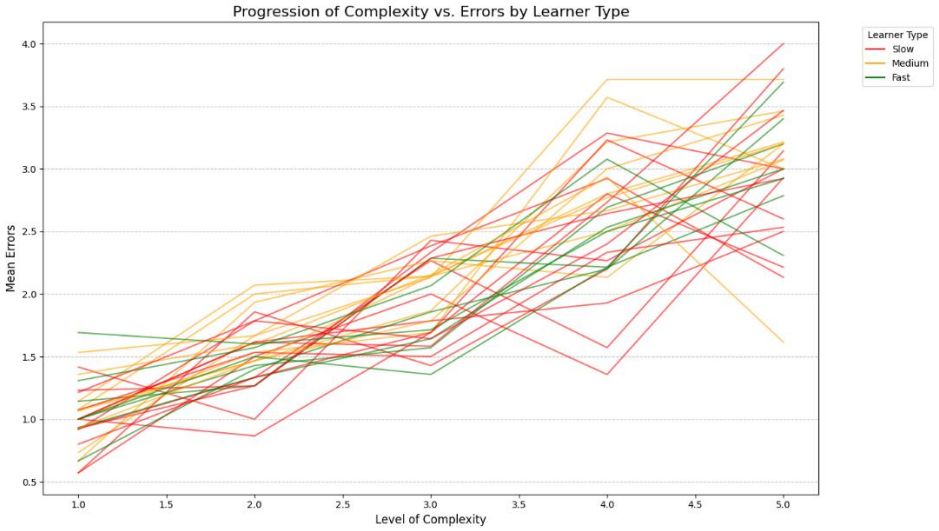

Figure 6.11: Learner classification

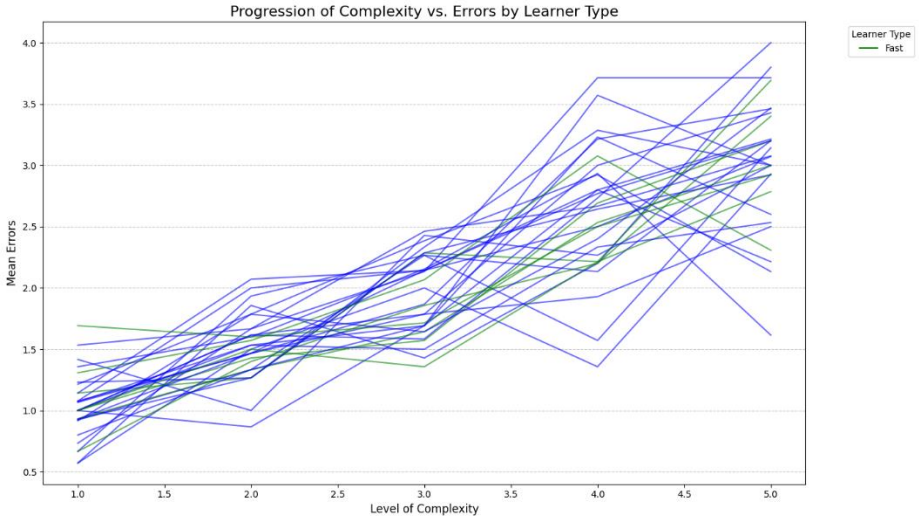

Figure 6.12: Fast Learner

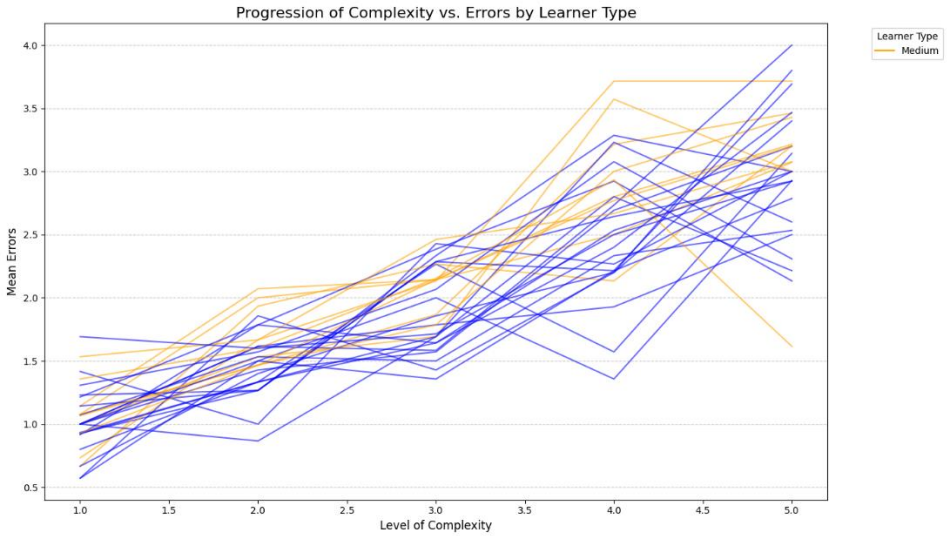

Figure 6.13: Medium Learner

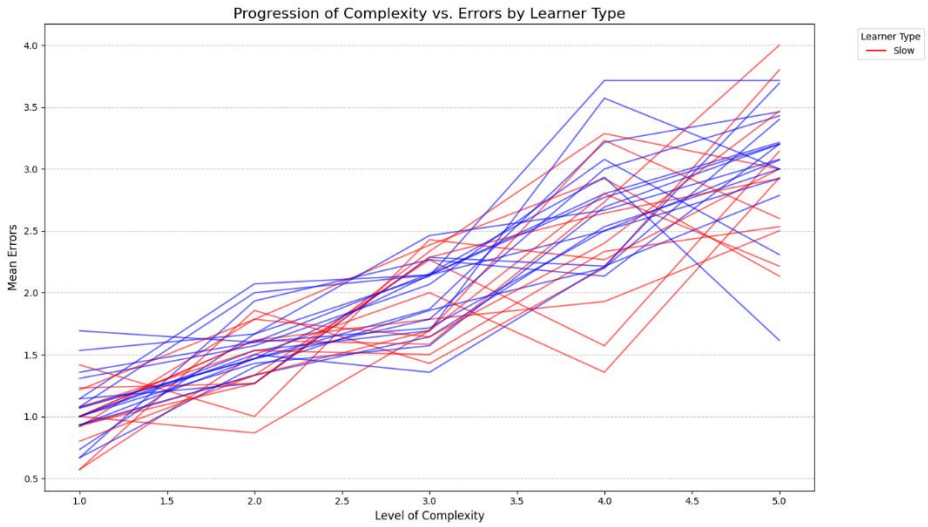

Figure 6.14: Slow Learner

These confirm the strength of ensemble learning but expose weakness in the much more modest models including KNN and Decision Trees. AECW was built to naturally pair well with semi-supervised datasets since these are what would need addressing for questions of the incomplete labels that arise during task complexity variation and erroneous frequency patterns quite standard issues with novices' programming dataset.

6.3 Conclusion

Results of the experiments: the obtained results show that the AECW algorithm proposed by us is the best model to predict and analyze the paths of learning of novice programmers, and this algorithm is well recognized with its high precision and recall as well as accuracy during the training and testing phases. Compared to existing models, such as Decision Tree, KNN, XGBoost, and Random Forest, AECW has better generalization and can work on the complexity of tasks, making it a new contribution in programming education research.

This study focused on uncovering and analyzing the novice programming learning paths using a machine learning model on a semi-supervised dataset. Major task difficulties such as complexity in a particular task, error correction behaviors of novice programmers, and their overall behaviors at the time of learning have been addressed by using the Adaptive Ensemble Classifier with Complexity-Aware

Weighting (AECW) algorithm. Various machine learning models, namely K-Nearest Neighbors (KNN), Decision Tree, Random Forest, and XGBoost, were implemented and rigorously compared to assess the effectiveness of such models in predicting programming outcomes and learning progression.

The results obtained showed that the KNN algorithm baseline was performing the weakest, as it has problems generalizing to unseen data. While training performance is acceptable, the testing precision and accuracy have suffered considerably, implying weakness with complex high-dimensional programming datasets. Performance was improved dramatically by the Decision Tree classifier and Random Forest models; Random Forest has exhibited robustness over testing data yields with an accuracy of 84.23%. It managed to handle task complexity and programming behaviors very well; it proved to be a better variant compared to KNN. On the other hand, the Decision Tree and Random Forest models were not adaptive, which meant they could not keep changing with minor patterns in the novice programming data, including frequencies of errors and counts of retries.

The XGBoost algorithm gives satisfactory results with a reported test accuracy of 95.11%. The gradient boosting framework of XGBoost enabled it to capture the complex relationships between features and also helped mitigate class imbalances that are common in programming datasets. The capability of the XGBoost to be able to learn from complex interrelations in data made it an interesting player in the battle to find learning patterns. Even though XGBoost was able to attain such success, some small indications of overfitting tendencies were observed implying that there could be more possibilities of optimizing it further to improve its performance.

The proposed AECW algorithm outperformed the others, with a test accuracy of 97.47%. This is because the complexity-aware weighting mechanism is capable of dynamically prioritizing tasks and features based on their difficulties and learning importance. Unlike the traditional models, AECW handles semi-supervised data efficiently. This happens when labels for tasks are either incomplete or ambiguous. This makes AECW highly applicable in real-world programming education

scenarios. It indicates that the minimum performance gap between training and testing phases underlines how well AECW is generalized and has superior predictive power. The capturing of error correction trends, retry frequencies, and progress through tasks provided deeper insight into novice learning behaviors than previously possible, allowing a better understanding of programming pathways.

Thus, the proposed AECW algorithm is valid for its efficiency in predicting and analyzing the learning paths of novice programmers. AECW, compared with existing models, provides greater accuracy and precision but gives actionable insights into task performance and the progression of learners. This research makes significant contributions to the programming education domain by providing a data-driven framework that identifies struggling learners, optimizes personalized learning strategies, and enhances the programming learning process as a whole. The results establish AECW as a trusted tool for the improvement of educational outcomes in novice programmers in both the academic and real-world scenarios. Future work can be spent in fine-tuning the algorithm and its application in different programming environments and learning platforms.

By using AECW, educators will be able to get better insights into learner progression, identify struggling students early, and design targeted interventions. The algorithm's exceptional performance directly aligns with your PhD research goal of identifying learning paths based on semi-supervised datasets, thereby significantly contributing to the advancement of personalized programming education.

6.4 Limitations of the Study

This study's breadth and machine learning in education are its key drawbacks. Quality and diversity of datasets influence the AECW algorithm and the programming task recommendation system. In case the dataset does not represent learner behaviors, the system may struggle to generalize among learners. The AECW approach uses ensemble learning to make it robust; however, its computing cost is directly proportional to datasets and job complexity, which is difficult to scale in real-world applications with many learners.

Another limitation is feature scope. Accuracy, error patterns, and performance scores are measured but cognitive load, learning environment, and motivation that can significantly influence programming competency are not. Furthermore, classification of learners into task-performance-based groups as fast, medium, and slow oversimplifies learning styles and cognitive processes.

Our job recommendation system needs difficulty metrics. Differences in perceptions of task difficulty and in task design may restrict the accuracy and effectiveness of System recommendations. This research focus on Random Forest, XGBoost, and AECW will be restricted to researching more advanced approaches, such as deep learning, which would be better suited to boost performance.

Educational efficacy cannot be complete without time management, accessibility, and learner motivation. It is confirmed by the preprocessed datasets and simulated task ideas that may use the system, although in many educational settings. Designing task difficulty is subjective and does not often match the perceptions of the learners, thus making it rather difficult to construct an effective, flexible system.

Lastly, the study only looks into programming instruction and not transdisciplinary applications that could enhance it. Overcoming these limitations could make the systems scalable, inclusive, and applicable in future research.

6.5 Future Scope of the Study

The future scope of this study brings several promising directions that will enhance the proposed work and expand its applicability in programming education and beyond. In the first place, one important direction for future research is to integrate adaptive feedback mechanisms into the learning systems. This means that the educators can guide novice programmers through their tasks dynamically, address errors as they occur, and make the learning experience more personalized. Furthermore, the proposed AECW algorithm can be extended to incorporate reinforcement learning techniques, where models adapt based on learner responses

and progressively optimize the difficulty of tasks in a way that mirrors human learning processes.

Another direction is going to be the exploration of multi-modal sources of data, such as when programming logs are combined with eye-tracking data, and code structure analysis and behavioral metrics, to try to gain deeper insights into learning interactions with programming environments in a way that provides broader understanding of learning patterns about subtle cognitive challenges that could be faced by students. Moreover, the use of NLP techniques to analyze comments, code descriptions, and documentation written by learners may reveal further dimensions of their learning journey, such as their understanding of concepts and problem-solving strategies.

Another critical direction is scalability and deployment of the AECW algorithm in real-world online programming platforms. Its integration into platforms like MOOCs or competitive coding environments would allow its evaluation across larger, more diverse learner populations. This could further refine the algorithm to handle diverse learning styles, programming domains, and skill levels. Further, in the future study, the efficiency of the algorithm can be enhanced so that it will enable real-time predictions and recommendations for high-scale applications.

Another important direction is to enhance the semi-supervised learning framework employed in this work. The proposed approach can be further extended to more effectively leverage unlabeled data through advanced techniques such as self-training or pseudo-labeling that can improve predictions even when the labels are incomplete or noisy. This would make the system more robust and applicable to educational datasets where fully labeled data is often scarce.

Besides that, future work could further explore the inclusion of explainable AI (XAI) techniques in the AECW algorithm to enable educators and learners with insight about how the model actually works for recommendations made or identifies which learners may struggle with what tasks. This is something that could

establish a lot of trust, enabling further adoption of such systems within academic environments.

Lastly, this research may open doors to cross-disciplinary applications. The methodology could be applied to other domains as well, such as STEM education, where task complexity and learning progression are of equal importance. Examining how the AECW algorithm would behave in other disciplines would make it even more versatile and would also contribute to the development of intelligent tutoring systems across domains.

Thus, this study's future scope comprises perfecting the proposed algorithm for its use in real-time, scalable, and explainable versions, extending the data sources towards deeper analysis, and pursuing its applications in various education contexts. These developments would further enhance the ability to track learning paths, personalize programming education, and improve the learners' outcomes.