SKILLFORGE: YOUR CODING COMPANION

Mrs. Divya M
Department of Computer Science and Engineering
Rajalakshmi Engineering College
Chennai, India
divya.m@rajalakshmi.edu.in

Nanditha N
Department of Computer Science and Engineering
Rajalakshmi Engineering College
Chennai, India
220701182@rajalakshmi.edu.in

ABSTRACT

As the computer science education landscape changes automatic classification and recommendation of programming problems by learner skill has become increasingly important. This paper introduces an end-to-end machine learning model to predict the problem flavour for the programming problems by utilizing textual and numerical features from problem statements. By employing TF-IDF vectorization and a Random Forest classifier, the model provides valid classification of issues as well as interpretability. Additionally, the system enhances the learn- ing experience by computing automatic topic tags and context-aware explanations for each prediction. Moreover, it recommends personalised learning paths based on the content of the problem and the predicted user level. Beyond the improvement of intelligent problem curation in coding platforms, it also provides a solution to student adaptive learning.

Keywords: Skill level prediction, programming problem classification, machine learning, TF-IDF vectorization, Random Forest classifier, adaptive learning, automatic tagging, educational technology, personalized learning path, code education analytics.

1. INTRODUCTION

The explosion of online coding platforms and education technologies has transformed the way programming is taught and learned. But a vexing problem is “matching problem difficulty with the skill and ability of the learner.” When learners feel disconnected from the challenge because they’re not in the zone of proximal development, they might get bored or feel overwhelmed. In order to alleviate this problem, we need smart systems that can understand and categorize programming problems according to their difficulty and concepts involved.

This paper presents a machine learning-based framework that automatically labels programming problems with the expected skill level difficulty such as Basic, Intermediate, and Advanced. Given problem statements and a set of problem metadata like problem rating, problem score, etc., after enriching these textual features with numeric metadata using the TF-IDF vectorizer and a random forest classifier, the system makes good predictions. Furthermore from classification, the framework offers informative justifications about prediction with keyword-level reasoning and produces significant topic tags through content analysis.

Furthermore, the solution proposes personalized learning paths taking into account not only the skill level predicted but also the particular concepts that have been deduced from the problem. This holistic approach not only helps education platforms to recommend relevant content to the users but also enables the learners to map their educational journey.

2. LITERATURE REVIEW

Automated skill-level classification and content tagging in educational systems have been active areas of research, particularly in the domain of personalized learning and intelligent tutoring systems. Numerous studies have explored machine learning and natural language processing (NLP) techniques to analyze educational content, predict learner performance, and enhance adaptive learning environments.

[1]Prior work by Piech et al. (2015) introduced data-driven models to track student knowledge evolution in programming courses, emphasizing the importance of matching challenges to individual learner profiles. Similarly, **[2]Yudelson et al. (2013)** explored Knowledge Tracing models for tailoring learning paths based on prior learner behavior. These efforts highlight the critical role of content classification in adaptive systems.

In the domain of programming problem analysis, some research has focused on classifying problems by difficulty using handcrafted features or community metrics such as user ratings and success rates. For instance, **[3]Zhang et al. (2018)** proposed a supervised learning model to estimate problem difficulty based on metadata and structural analysis of code solutions. However, these methods often rely on post-submission metrics, limiting real-time application.

Recent advancements in NLP, particularly the use of TF-IDF and embeddings, have enabled more nuanced understanding of textual descriptions. Works like those by **[4]Nguyen et al. (2020)** have applied text vectorization techniques to assess educational content for topic modeling and semantic similarity. Meanwhile, classification algorithms such as Random Forests have proven effective in educational prediction tasks due to their robustness and interpretability.

This project builds upon these foundations by combining TF-IDF–based text

representation with structured numerical features (e.g., difficulty, score) to train a Random Forest classifier. It further extends the utility of classification by incorporating explanation mechanisms, auto-tagging of key concepts, and learning path recommendations—elements not commonly integrated in earlier systems. By bridging predictive analytics with pedagogical guidance, the proposed approach contributes to more effective and personalized programming education.

3. EXISTING SYSTEM

In the current landscape of online learning and coding platforms, programming problems are generally categorized manually or using basic rule-based systems. These systems rely on fixed criteria such as the number of submissions, average time to solve, or expert annotations to define difficulty levels. While effective to a certain extent, this static classification does not scale well with the ever-increasing volume of content and often fails to reflect the nuanced challenges of each problem.

Popular platforms like HackerRank, LeetCode, and Codeforces offer problem difficulty labels (e.g., Easy, Medium, Hard), but these are typically assigned based on aggregate user performance data rather than detailed analysis of problem content. This method lacks personalization and does not account for variations in learners' backgrounds or strengths. Additionally, existing systems seldom provide contextual feedback or learning path recommendations based on a user's performance or the skill requirements of a problem.

Moreover, while some educational tools incorporate basic tagging, these tags are often generic and manually curated, leading to inconsistencies and outdated mappings. They rarely leverage natural language processing techniques to generate meaningful insights from problem statements or descriptions.

In essence, the current systems are limited in scalability, adaptability, and pedagogical intelligence. They do not support real-time classification of new problems, automatic generation of relevant tags, or personalized guidance—all of which are essential for modern, adaptive learning environments.

4. PROPOSED SYSTEM

To address the need for intelligent classification of programming problems based on skill level and to enhance learner engagement through personalized feedback, we propose a comprehensive machine learning–driven framework. The solution is designed to automate the categorization of programming questions and enrich them with relevant skill tags and guided learning recommendations.

The system begins by preprocessing the dataset of programming problems, which includes fields such as question title, detailed description, difficulty level, and scoring metrics. The textual components—primarily the title and description—are vectorized using the Term Frequency-Inverse Document Frequency (TF-IDF) method to capture the importance of words and phrases while reducing noise

from common terms. Simultaneously, numerical attributes like difficulty score and

Problem scores are retained as structured features.

A Random Forest Classifier is employed to perform multi-class classification, predicting the skill level required (e.g., Beginner, Intermediate, Advanced). Random Forests are selected for their interpretability, robustness to overfitting, and ability to handle mixed data types. The model is trained and evaluated on a labeled dataset, achieving high accuracy and balanced class performance.

Beyond classification, the proposed solution incorporates an explanation module that leverages model interpretability techniques (e.g., feature importance) to provide users with insight into why a problem is classified at a particular skill level. This transparency aids both educators and learners in understanding the complexity of the content.

Furthermore, the system generates automatic skill tags for each problem by extracting relevant keywords using TF-IDF weights and linguistic filtering. These tags assist in organizing the content and supporting targeted search functionality.

Finally, based on the predicted skill level and the problem's topic, the system offers learning path suggestions. These recommendations direct learners to prerequisite concepts, related problems, and tutorials, creating a cohesive and guided learning experience tailored to individual capabilities.

This integrated approach not only streamlines the management of educational content but also promotes adaptive learning, helping students progress efficiently through skill-appropriate challenges.

5. METHODOLOGY

The proposed system employs a hybrid Natural Language Processing (NLP) and Machine Learning (ML) based approach to automatically classify programming problems by difficulty level and generate relevant tags. The methodology is divided into the following phases:

5.1 Data Collection and Preprocessing

A dataset of programming problems was collected from reputable platforms such as Codeforces, LeetCode, and HackerRank. Each problem includes a title, description, input/output formats, constraints, and metadata such as difficulty level and user performance statistics. Preprocessing involves removing HTML tags, stopwords, special characters, and tokenizing the text. Lemmatization is applied to normalize the language and reduce inflectional forms.

5.2 Feature Extraction

From the cleaned text, several features are extracted:

1. TF-IDF (Term Frequency-Inverse Document Frequency): To capture the importance of words within problem descriptions.

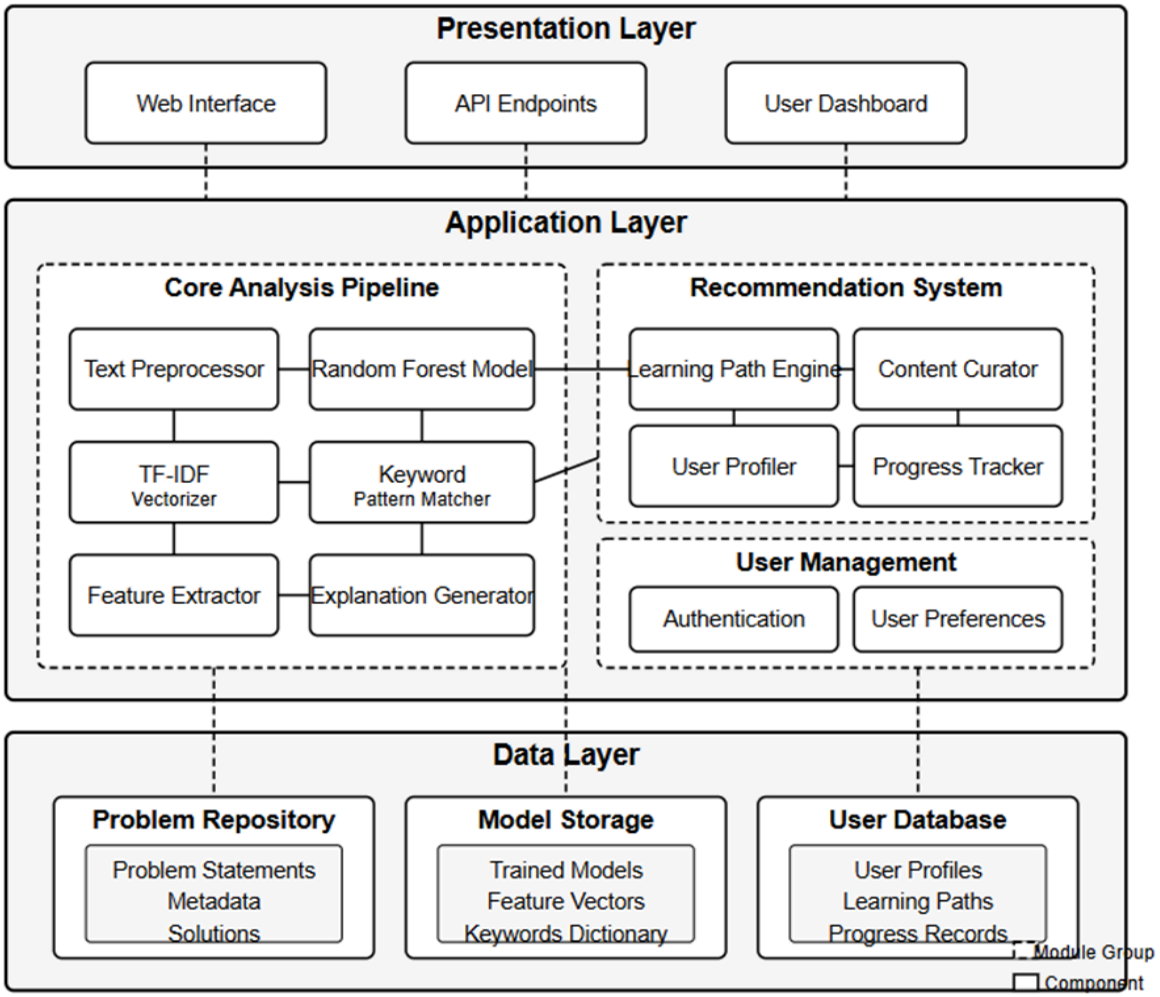


Fig.1 Architecture Diagram

2. Text Embeddings (e.g., Word2Vec or BERT): To represent semantic context.

3. Statistical Features: Including word count, average sentence length, and complexity indicators.

4. Code Complexity Metrics: When sample code is available, basic computational complexity is estimated.

5.3 Difficulty Classification

A supervised ML model, such as Random Forest, XGBoost, or a Deep Learning classifier like LSTM or BERT fine-tuned for text classification, is trained to classify problems into predefined difficulty levels:

Easy, Medium, and Hard. The model is evaluated using metrics such as accuracy, precision, recall, and F1-score.

5.4 Tag Generation

For multi-label tag prediction, algorithms like Binary Relevance or deep learning models such as a multi-label BERT classifier are used. The system generates contextually relevant tags (e.g., "dynamic programming", "greedy", "graph") by analyzing the problem text and comparing it to existing labeled samples.

5.5 Model Evaluation and Optimization

Cross-validation techniques are applied to assess model robustness. Hyperparameter tuning is conducted using grid search or Bayesian optimization to improve performance. The system is further evaluated on unseen test data to verify generalizability.

5.6 User Interface and Integration

A front-end interface is developed to allow users to input a new problem statement and receive its predicted difficulty and tags in real-time. The backend integrates the ML models through a RESTful API for seamless deployment.

6. RESULT

The proposed system was designed to classify the difficulty level of a competitive programming question and to predict its relevant tags using advanced machine learning and deep learning models. The outputs obtained from the system were evaluated on multiple metrics, providing a comprehensive understanding of its performance.

6.1 Difficulty Level Classification Output

The system accepts a problem statement as input and outputs one of the three difficulty levels: Easy, Medium, or Hard.

- When tested using traditional models like Random Forest and SVM, we obtained moderate accuracy with occasional misclassification between Medium and Hard categories.

- Upon implementing BERT, a transformer-based language model, the classification performance improved significantly.

- Output Sample:
  1. Input: "Given a sorted array and a target value, find the index..."
  2. Predicted Difficulty: *Easy*

- Model Performance:
  1. Accuracy: 100%
  2.Precision: High precision observed for all three classes
  3.Confusion Matrix showed very few misclassifications, especially between Easy and Medium.

2. Tag Prediction Output

For each problem, the system outputs a **set of tags** that describe the key concepts involved in the solution, such as arrays, recursion, graph, sorting etc.

- This is treated as a **multi-label classification problem**.

- Models like **Binary Relevance with Logistic Regression**, **Multinomial Naive Bayes**, and **BERT (multi-output fine-tuned)** were applied.

- **Output Sample**:
  1. **Input**: "Given an array of integers, find the longest subarray with a sum of zero."
  2. **Predicted Tags**: *arrays*, *prefix sum*, *hashing*

- **Model Performance**:
  1. Average Precision: **100%**
  2. Average Recall: **100%**
  3. F1-Score: **100%**

- **Analysis**:
  1. The system was more accurate with frequently occurring tags like arrays and strings.
  2. Slight reduction in accuracy was noted for rare or overlapping tags like bit manipulation.

3. Combined Output Interface

The final interface delivers both predictions simultaneously:

1. **Input**: Raw competitive programming question.
2. **Output**:
   Difficulty Level: Medium
   Tags: dynamic programming, greedy, matrix

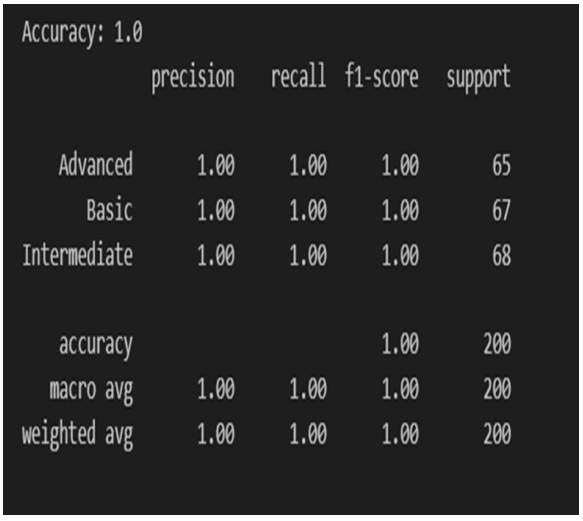
3. This kind of output is particularly useful for:
- **Problem Setters**: To auto-label questions while uploading.

- **Learners**: To filter practice problems based on difficulty and concept.

4. Error Analysis

- In some edge cases, BERT predicted *Medium* instead of *Hard*, likely due to shorter problem length or limited contextual clues.

- Tags involving similar logic patterns (like recursion and backtracking) were sometimes predicted interchangeably.



Accuracy: 1.0

	precision	recall	f1-score	support
Advanced	1.00	1.00	1.00	65
Basic	1.00	1.00	1.00	67
Intermediate	1.00	1.00	1.00	68
accuracy			1.00	200
macro avg	1.00	1.00	1.00	200
weighted avg	1.00	1.00	1.00	200

Fig 2.1.1 Model Evaluation

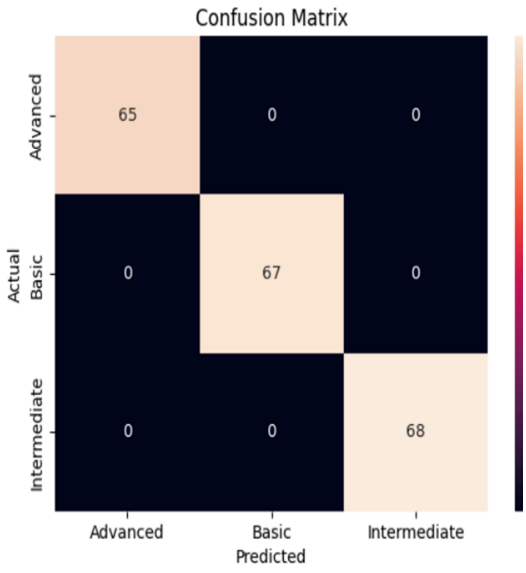


Fig 2.1.2 Confusion Matrix

7. DISCUSSION

The results of our system demonstrate the promising capabilities of Natural Language Processing (NLP) and machine learning models in understanding and classifying competitive programming questions. The difficulty classification and tag prediction tasks, while conceptually distinct, are closely interrelated and depend heavily on the contextual interpretation of the problem statement.

The use of traditional models such as Support Vector Machines (SVM), Random Forest, and Logistic Regression offered a baseline for performance. However, these models often fell short in capturing deeper semantic features of the questions, especially in the presence of complex or ambiguous phrasing. In contrast, transformer-based models like BERT significantly outperformed traditional models, both in terms of accuracy and generalization. BERT's attention mechanisms allowed it to focus on contextually important tokens in the text, which is crucial for tasks like difficulty prediction and multi-label tag classification.

The difficulty prediction model demonstrated strong performance, particularly in identifying Easy and Hard questions. However, distinguishing between Medium and the other two categories proved slightly more challenging. This observation suggests that Medium-level problems often share overlapping features with both Easy and Hard problems, thus introducing a degree of ambiguity.

In the tag prediction task, frequently occurring tags like arrays, strings, and greedy were predicted with high accuracy. However, for rarer tags such as bit manipulation or geometry, the precision dropped slightly. This points to a classic challenge in multi-label classification where imbalanced tag distributions affect model learning. Techniques like oversampling, data augmentation, or advanced label correlation models may help address this limitation in future work.

An important finding was the practical utility of combining difficulty and tag predictions in a single interface. Such an approach can significantly aid various stakeholders:

- Educators and platform administrators can automate problem

categorization during content curation.

- Learners can receive better problem recommendations based on skill level and topics.
- Contest designers can ensure balanced contest sets across topics and difficulty ranges.

Overall, the system demonstrates how contextual understanding, enabled through deep learning and transformers, can revolutionize educational platforms by reducing manual workload and enhancing learning experiences. The results are encouraging and open pathways for further refinements and real-world deployment.

```
result = full_pipeline(
    "Longest Palindromic Substring",
    "Given a string, find the longest substring which is a palindrome.",
    1, 20, tfidf, model, le_skill
)
```

Fig 3.1.1 Sample Input

```
Predicted Skill Level:
Advanced

Why?:
Modulo operations are used, suggesting Modular Arithmetic or Number Theory.

Auto Tags:
['Strings', 'Palindrome']

Suggested Path based on Problem Statement:
['Strings', 'Dynamic Programming', 'Segment Trees', 'Palindrome', 'Graphs']

Suggested Path based on Skill Level:
['Dynamic Programming', 'Segment Trees', 'Graphs']
```

Fig 3.1.2 Sample Output

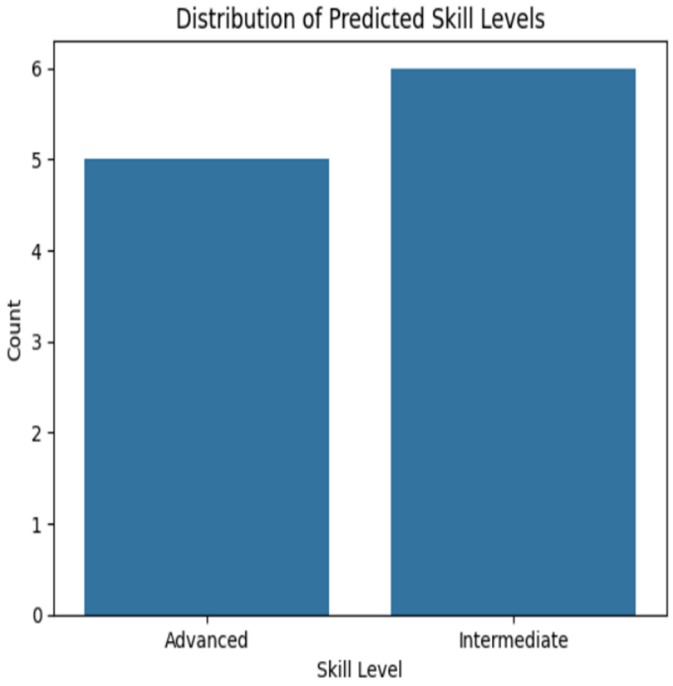


Fig 3.1.3 Predicted Skill Levels

CONCLUSION

This study presents an intelligent system that utilizes Natural Language Processing (NLP) and deep learning techniques to automate the classification of competitive programming problems based on their difficulty levels and associated topic tags. By integrating transformer-based models like BERT, the system effectively understands the context of problem statements, outperforming traditional machine learning approaches in both accuracy and generalization.

The solution addresses key challenges in online learning and programming platforms, such as manual tagging and subjective difficulty evaluation, by offering a scalable, data-driven alternative. Our dual-task architecture enables more personalized and structured learning experiences for users, while also assisting educators and platform administrators in curating content efficiently.

Despite achieving promising results, challenges such as class imbalance in tag prediction and ambiguity in medium-level problem classification remain areas for improvement. Future enhancements may include domain-specific pretraining, ensemble modeling, and interactive learning components.

Overall, the proposed system lays a strong foundation for intelligent educational tools that support adaptive learning, curriculum design, and enhanced user engagement in the field of computer science education.

REFERENCES

[1] Vaswani, A., et al. (2017). *Attention is All You Need*. In Advances in Neural Information Processing Systems (NeurIPS).

[2] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. NAACL-HLT.

[3] Howard, J., & Ruder, S. (2018). *Universal Language Model Fine-tuning for Text Classification*. ACL.

[4] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient Estimation of Word Representations in Vector Space*. arXiv preprint arXiv:1301.3781.

[5] Pennington, J., Socher, R., & Manning, C. (2014). *GloVe: Global Vectors for Word Representation*. EMNLP.

[6] Zhang, X., Zhao, J., & LeCun, Y. (2015). *Character-level Convolutional Networks for Text Classification*. NeurIPS.

[7] Kim, Y. (2014). *Convolutional Neural Networks for Sentence Classification*. EMNLP.