Real-Time Prediction of Student Performance in Game-Based Learning

Aviral Chauhan
BTech CSE 4th Year
2021029

Siddhant Rana
BTech CSE 4th Year
2021101

Mahansh Aditya
BTech CSD 4th Year
2021334

Ayaan hasan
BTech CSE 3rd Year
2022121

1. Abstract

The advent of game-based learning has transformed the educational landscape, making learning more engaging and interactive. However, many educational games lack personalized feedback mechanisms, limiting their effectiveness. This research focuses on developing a machine learning model to predict student performance in real-time using extensive game log data. By analyzing student behavior through event-level data—including elapsed time, user interactions, and level progression—this model aims to facilitate adaptive learning experiences tailored to individual needs. The findings of this study will advance knowledge-tracing methods, contributing to more effective and personalized educational tools that enhance student engagement and performance in game-based learning environments. The GitHub link for the code: Ml_project.

2. Introduction

Game-based learning offers an engaging way for students to interact with educational content, but many games lack effective personalized feedback. This research aims to develop a machine learning model that predicts student performance in real time using detailed game log data, including user actions, level progression, and cursor movement. By monitoring how users spend their time—such as tracking cursor movements, focus on in-game elements, and whether the music is on—we can better understand factors influencing their likelihood of clearing the game.

This information can also assist game developers in optimizing gameplay by strategically placing important objects in areas with high user interaction. By analyzing these data points, the model will help create adaptive learning experiences tailored to individual needs. This study seeks to advance knowledge-tracing methods and enhance the effectiveness of game-based learning, making it more personalized and impactful for students.

3. Literature Review

- **Machine Learning Models for Predicting Student Performance**: Several studies focus on using machine learning algorithms to predict student outcomes in game-based learning environments. A study by Patki et al. compared various algorithms for predicting performance in online mathematics games, concluding that Random Forest provided the most accurate results by identifying crucial in-game features such as mathematical transformations performed early in the game [1].

- **Hidden Markov Models and Time-Series Data**: Tadayon and Pottie utilized hidden Markov models to predict student performance through time-series analysis of game logs, highlighting that sequential gameplay data effectively forecasts learning outcomes [2].

- **Feature Importance in Prediction Models**: Research on math learning games has shown that early in-game behaviors, like completing specific tasks or achieving milestones, are strong predictors of posttest scores. Nguyen et al. found that pretest scores combined with students' in-game achievements (e.g., mastering certain buckets) were highly associated with improved performance, supporting the notion that prior knowledge and key actions during gameplay are critical for learning success [3].

- **Random Forest and K-Nearest Neighbors for Student Modeling**: Another study examined several machine learning models, including Random Forest and K-Nearest Neighbors (KNN), for modeling student performance in digital learning environments. This study emphasized the efficiency of KNN for smaller datasets and the accuracy gains provided by Random Forest in more complex settings with larger data inputs [4].

- **Adaptive Learning and Game Design Influence**: Shute et al.'s investigation into "Physics Playground," a digital physics game, found that students' in-game progress, measured by earning trophies, predicted their conceptual understanding of physics. The research suggests that adaptive elements in games can significantly enhance learning outcomes by aligning challenges with student abilities [5].

4. Dataset Description

Each game session is identified by a unique `session_id`. The dataset contains rows that represent game events, with each row corresponding to a specific event and described by several key attributes, as detailed below:

- **session_id**: The unique identifier for the session in which the event occurred.
- **index**: The index of the event for the session, indicating its order.
- **elapsed_time**: The time elapsed (in milliseconds) since the start of the session when the event was recorded.
- **event_name**: The type of event (e.g., click, hover, notebook interaction).
- **name**: The specific name of the event (e.g., indicates whether a notebook click is opening or closing the notebook).
- **level**: The level of the game in which the event occurred (ranges from 0 to 22).
- **page**: The page number of the event, used only for notebook-related events.
- **room_coor_x**, **room_coor_y**: The x and y coordinates of the click within the in-game room, used only for click events.
- **screen_coor_x**, **screen_coor_y**: The x and y coordinates of the click relative to the player's screen, used only for click events.
- **hover_duration**: The duration (in milliseconds) of the hover action, used only for hover events.
- **text**: The text that the player sees during the event.
- **fqid**: The fully qualified ID (FQID) of the event.
- **room_fqid**: The fully qualified ID (FQID) of the room where the event took place.
- **text_fqid**: The fully qualified ID (FQID) of the text shown to the player during the event.
- **fullscreen**: A binary indicator of whether the player was in fullscreen mode (e.g., `1` for fullscreen, `0` for windowed mode).
- **hq**: A binary indicator of whether the game was running in high-quality mode (e.g., `1` for high-quality, `0` for low-quality).
- **music**: A binary indicator of whether the game music was on or off.
- **level_group**: Groups of levels to which the event belongs, categorized as follows:
  - **0-4**: Levels 0 through 4
  - **5-12**: Levels 5 through 12
  - **13-22**: Levels 13 through 22

Player Movement
Player progression through the game is tracked using the following columns:

- **room_fqid**: The fully qualified ID of the room the player is in during the event.
- **level**: The game level the player is in at the time of the event (from 0 to 22).

Event-Specific Columns
The remaining columns capture details about specific types of events:
Click Events: For click events, the following columns record the coordinates of the click:

- **room_coor_x**, **room_coor_y**: The x and y coordinates of the click within the in-game room.
- **screen_coor_x**, **screen_coor_y**: The x and y coordinates of the click on the player's screen.

Notebook-Related Events: For events related to notebook interactions, the following column is used:

- **page**: The page number in the notebook that the player interacts with.

Hover Events: For hover events, the following column captures the duration of the hover:

- **hover_duration**: The duration of the hover action, measured in milliseconds.

4.1. Feature Engineering

Feature engineering was performed to enhance the model's predictive power through the following steps:

- **Vocabulary Creation:** Unique tokens were extracted from the categorical columns (`room_fqid` and `text_fqid`) by splitting values on the delimiter ('.'). This process ensured uniqueness by storing tokens in a set.
- **Encoding:** Each unique token was mapped to a unique integer to transform categorical variables into a numerical format. This involved sorting the vocabulary and assigning integer values. Subsequently, the original categorical values were converted into their corresponding integer representations, with leading zeros added to maintain consistency.
- **Aggregation:** The dataset was aggregated based on `session_id` and `level_group`. Unique counts of categorical features were computed, and statistical measures such as mean and standard deviation were calculated for numerical features. Missing values were filled with a placeholder (-1) to enhance robustness.

4.2. Data Preprocessing

The preprocessed dataset was split into features and labels, and categorical features were encoded appropriately. Irrelevant columns (e.g., `level_group`) were excluded from the final feature set.

4.3. Train-Test Split

To prevent data leakage and ensure that the model was evaluated on unseen data, the dataset was divided into training and test sets using an 80-20 split ratio prior to any model training or feature preprocessing.

5. Methodology

5.1. Model Selection

Given the nature of the problem, we selected multiple machine learning models for comparison and evaluation. The models were chosen based on their ability to handle complex patterns in the data, as well as their suitability for the specific task at hand. The models implemented include:

- Logistic Regression
- Decision Trees
- Random Forest
- Gradient Boosting Machines (GBM)
- AdaBoost
- Bagging
- Support Vector Machines (SVM)
- XGBoost
- CatBoost

Each model offers unique advantages, such as interpretability, accuracy, and computational efficiency.

5.2. Model Training

The training process is structured around the following steps:

- The data was split into two main subsets: training (80%) and testing (20%).
- **Model Training:** Each model was trained using the training set, and hyperparameters were tuned using Grid Search or Random Search (depending on computational efficiency) to find the optimal set of parameters.

For Random Forest, Grid Search was performed to fine-tune hyperparameters such as the number of estimators, maximum depth of trees, and minimum samples for splitting a node.
Additionally, ensemble methods like Bagging, AdaBoost, and Gradient Boosting were trained with default parameters initially, followed by hyperparameter tuning.

5.3. Evaluation and Metrics

Once the models were trained, they were evaluated using appropriate metrics. The evaluation process included the following:

- **Accuracy, Precision, Recall, and F1-Score** were calculated for classification problems.
- **Confusion Matrix** was used to visualize the performance and errors made by the model.

The models were compared based on these metrics to identify the best-performing approach.

5.4. Hyperparameter Tuning

Hyperparameter tuning is crucial for optimizing model performance. We used techniques such as Grid Search and Random Search to tune the parameters of each model:

- **Logistic Regression:** Tuning the regularization strength (C).
- **Decision Trees:** Tuning the max depth, min samples split, and min samples leaf.

- **Random Forest:** Tuning the number of estimators, max depth, and min samples split.

- **Gradient Boosting:**Learning rate, number of estimators, and maximum depth of trees.

- **AdaBoost**: Tuned the number of estimators and learning rate.

We also evaluated models using learning curves to visualize how the model performs with increasing amounts of training data. Hyperparameter tuning was done to prevent overfitting and ensure the model generalizes well to new, unseen data.

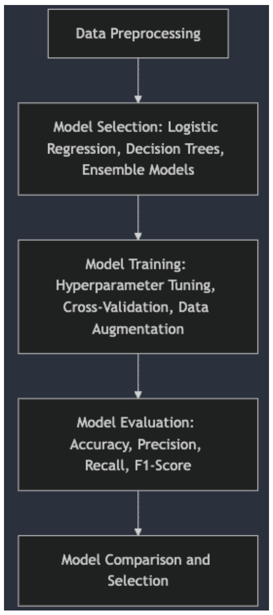
Data Preprocessing
Model Selection: Logistic Regression, Decision Trees, Ensemble Models
Model Training: Hyperparameter Tuning, Cross-Validation, Data Augmentation
Model Evaluation: Accuracy, Precision, Recall, F1-Score
Model Comparison and Selection

Figure 1. Workflow for Model Development

5.5. Model Comparison and Selection

Once all models were trained and evaluated, we compared their performance based on accuracy, precision, recall, and F1-score. The best-performing model, in terms of these metrics, was selected. We also considered factors such as training time, inference speed, and model interpretability during the selection process.

6. Results and analysis

The performance of the models was evaluated based on several key metrics: accuracy, precision, recall, F1-score, and confusion matrix for classification tasks. After training and hyperparameter tuning, we compared the results to identify the best-performing model.

6.1. Model Performance

- **AdaBoost**, **Gradient Boosting Machines (GBM)**, and **CatBoost** outperformed other models in terms of accuracy, with **AdaBoost** slightly ahead in accuracy (0.748746), followed by **GradientBoosting** (0.748682) and **CatBoost** (0.748305).

- **SVM**, followed by **RandomForest** and **GradientBoosting**, outperformed in precision, with **SVM** leading in precision (0.747588), followed by **RandomForest** (0.742794) and **GradientBoosting** (0.743647).

- **SVM**, followed by **GradientBoosting** and **AdaBoost**, outperformed in recall, with **SVM** achieving the highest recall (0.886799), followed by **GradientBoosting** (0.878227) and **AdaBoost** (0.876340).

- The highest **F1-Score** was from **CatBoost** (0.788872), followed by **AdaBoost** (0.787987) and **GradientBoosting** (0.787628).

- **XGBoost** also showed competitive performance, with a slight edge over other models in precision (0.733569) and recall (0.835053), indicating its ability to minimize false positives and false negatives.

- **Logistic Regression** and **Decision Trees** had lower performance, especially in terms of precision and recall, likely due to their simpler nature and lower capacity to capture complex patterns in the data. Logistic Regression had an accuracy of 0.740397, while Decision Trees had the lowest accuracy of 0.656769.

6.2. Hyperparameter Tuning Impact

Hyperparameter tuning, especially for **Random Forest** and **AdaBoost**, significantly improved model performance. Tuning the number of estimators(200), tree depth(10), min samples split(10), min samples leaf(4) and learning rate helped prevent overfitting and enhanced generalization. It increased the accuracy of Random Forest from 0.747056 to 0.75139. Grid Search and Random Search were effective in finding the optimal parameters for each model, leading to better model fit and predictive performance

6.3. Implications of Results

The results indicate that ensemble models, particularly **Random Forest**, **AdaBoost**, and **Gradient Boosting Machines (GBM)**, are well-suited for this problem, providing robust and reliable predictions. **AdaBoost** emerged as the top-performing model overall, while **GBM** and **CatBoost** also delivered competitive results. **XGBoost** proved to be a strong candidate for precision-driven tasks, making it suitable for scenarios where minimizing false positives or false negatives is crucial.

While simpler models like **Logistic Regression** and **Decision Trees** performed well for baseline comparisons, they were outperformed by the more complex models in capturing intricate patterns in the data.

Additionally, we employed data augmentation using the **SMOTE** library to address class imbalance and trained models such as **SVM**, **KNN**, and **CNN** on the augmented data. However, this approach led to a decrease in accuracy for these models, likely due to the synthetic nature of the augmented data not fully capturing the complexities of the original dataset.

6.4. Limitations and Considerations

Although the best models performed well, they are computationally intensive, especially during training. Therefore, further considerations should be given to the balance between model complexity and runtime. Additionally, feature engineering and data preprocessing played a significant role in model performance and could be further refined to boost accuracy.

In conclusion, **Random Forest, GBM, and AdaBoost** stand out as the most effective models for this task, but careful consideration of computational constraints and model interpretability should guide the final model selection for deployment.

7. Conclusion

Throughout this project, we explored various machine learning models to address a classification problem, gaining valuable insights into how data preprocessing, feature engineering, and model selection contribute to predictive performance.

7.1. Key Findings

- **Data preprocessing and feature engineering** were essential in improving model accuracy. By encoding categorical variables and aggregating session-based data, we were able to better capture the patterns in user interactions, significantly enhancing the performance of models such as **Logistic Regression** and **Decision Trees**.

- **Ensemble models** like **Random Forest**, **AdaBoost** and **GBM** outperformed simpler models, particularly when handling complex patterns in the data. **XGBoost** also showed strong precision and recall, offering robust performance in cases where minimizing false positives or negatives is crucial.

7.2. Learnings

- We learned that **feature engineering**, including the transformation of categorical features and aggregation of session data, directly impacts the model's ability to understand user behavior. This was particularly evident in the context of analyzing user interactions in a gaming environment.

- **Hyperparameter tuning**, through methods like **Grid Search**, played a key role in optimizing performance and preventing overfitting, allowing the models to generalize better to unseen data.

7.3. Challenges Faced

- **Computational resources** were a bottleneck, especially when training resource-intensive models like **Random Forest** and **AdaBoost**. These models required significant time and memory, which can limit scalability.

- **Overfitting** presented challenges, particularly during hyperparameter tuning. While methods like cross-validation helped address this, it remained a key area to monitor.

- **Model interpretability** was difficult with more complex algorithms such as **GBM** and **XGBoost**, making it harder to explain model decisions clearly, which is a crucial consideration for deployment in sensitive applications.

In conclusion, we gained a deeper understanding of how user interaction data can be leveraged for insights into engagement patterns. This understanding is valuable for applications like game design, where predicting user behavior and enhancing the gaming experience are critical.

8. Individual Task

All team members will collaboratively contribute to all tasks outlined in the timeline. The distribution of work will be based on individual expertise and availability, ensuring a balanced division of responsibilities throughout the project.

References

[1] Sanika Nitin Patki Ashish Gurung Reilly Norum Erin Ottmar Ji-Eun Lee, Amisha Jindal. A comparison of machine learning algorithms for predicting student performance in an online mathematics game.

[2] Gregory J. Pottie Manie Tadayon. Predicting student performance in an educational game using a hidden markov model.

[3] et al. (2020) Nguyen. Analyzing student performance in math learning games using behavioral metrics.

[4] others. (2023) Pouriyeh, S. Modeling student performance in game-based learning environments.

[5] others. (2015) Shute, V. Exploring adaptive learning and performance prediction in physics playground.