Predict Student Performance from Game Play

Machine Learning Final Project

Jaisal Patel

April 25, 2023

ABSTRACT

Despite the vast number of educational games available, research datasets have been limited in the field. To combat this, the competition tasks competitors with using time series forecasting to predict whether a student will get a question right or not in a given game session on the Jo Wilder platform. If successful, the models will be used to help enhance educational games for children. The dataset contains 13,174,211 rows of data and 20 columns (features). Among these features are the session id of an event—an action like a click—(session_id), the index of an event for a game session (index), the time elapsed in milliseconds between the start of an event and when the event was record (elapsed_time), the event name (event_name), the result of an action—like opening or closing—(name), and the level of the game from 0 to 22 (level). Other features include page number of a storybook event (page), the room coordinates and screen coordinates of in-game rooms and players' screens (room_coor_x, room_coor_y, screen_coor_x, and screen_coor_y), the hover duration for hover events (hover_duration), the text a player sees during an event (text), various fully qualified IDs—fqid, room_fqid, and text_fqid—(which are for the event, room, and text), whether the player was in fullscreen mode (fullscreen), whether the game is in high-quality (hq), whether the game music is on (music), and the group of levels and questions of the row (0-4, 5-12, 13-22; level_group). For pre-processing, I reduced the size of the dataset, removed columns which I noticed to have all nulls (fullscreen, hq, and music), imputed nulls in the numerical columns with the mean and in nulls in the categorical columns with the mode, checked for label imbalance in the level_group column, and converted categorical columns into numerical ones using get_dummies() and LabelEncoder(). During pre-processing, using boxplots and a heatmap, I noticed that the dataset was variable, uniform, contained few outliers, and had a good spread, meaning it could properly be used for machine learning. However, I will have to make permutations and generate more features as I move along making a machine learning model in order to increase its time series forecasting performance. The submission file with the class labels contains a session_id, the class labels we must predict , it must contain the session_id, followed by the class labels/ # correct column (which contains a 0 or 1 depending on whether the machine learning model predicts the player to get the question right or not), and a session_level.

Figure 1: Class Labels & Submission Sample File

	session_id	correct	session_level
0	20090109393214576_q1	0	20090109393214576_0-4
1	20090312143683264_q1	0	20090312143683264_0-4
2	20090312331414616_q1	0	20090312331414616_0-4
3	20090109393214576_q2	0	20090109393214576_0-4
4	20090312143683264_q2	0	20090312143683264_0-4
5	20090312331414616_q2	0	20090312331414616_0-4
6	20090109393214576_q3	0	20090109393214576_0-4
7	20090312143683264_q3	0	20090312143683264_0-4
8	20090312331414616_q3	0	20090312331414616_0-4
9	20090109393214576_q4	0	20090109393214576_5-12

CCS Concepts

● Computing méthodologies → Machine Learning → Learning paradigms; *Supervised learning*; Supervised Learning by classification

● Computing méthodologies → Machine Learning → Machine Learning approaches**; *Neural networks***

● Computing méthodologies → Machine Learning → Machine Learning algorithms**; *Ensemble methods*; Boosting**

Keywords

datasets, neural networks, ensemble modeling, boosting, classification

BENCHMARKING OF OTHER SOLUTIONS

Figure 2: Model Approaches

Notebook Name	Feature Approach	Model Approach	Train/Test Performance
Random Forest Baseline - [0.664]	Calculates number of unique values for each categorical column, and the mean and standard deviation for each numerical column for each unique combination of session_id and level_group, and concatenates each new feature to a dataframe.	Trains Random Forest-based GroupKFold with 5 folds, doing this for each of the 18 different questions.	Train Performance Metric: 0.659 Test Performance Metric: 0.664
PSP-NN model	Splits data into two dataframes based on level_group. Groups data by session using text_fqid. Converts elapsed_time column to natural log of elapsed_time in	Trains neural network model that contains dropout layers, linear layers,	Train Performance Metric: 0.803

	minutes. Creates pivot table where mean of natural log for each session is used for rows and text_fqid for columns. Pivot table is adjusted by subtracting the overall mean natural logarithm of elapsed time and clipping the values between -log_elapsed_mean and 5. Merges label data with correct dataframe.	and batch normalized layers.	Test Performance Metric: 0.617
LGBM + Early Stopping [LB 0.670]	This model uses the exact same feature approach as the Random Forest model despite being authored by separate people.	Trains LightGBM GroupKFold while using early stopping if the model doesn't improve validation score by a set figure with 5 folds, doing this for each of the 18 different questions.	Train Performance Metric: 0.665 Test Performance Metric: 0.670

Random Forest Baseline - [0.664]

This model adds a large number of permutations of features by generating statistical values, such as mean and standard deviation for each column of numerical data. The beauty of this approach is made even more elegant by the fact a unique feature is added for every combination of session_id and level_group. This increases the number of features that the model has by a significant value. While adding too many features can lead to overfitting, the nature of this Kaggle competition relies on it for accurate time series forecasting. While the permutation generation approach by this Jupyter Notebook means features are less catered to the problem than notebooks that had over 200 hand-made features, it beats methods that use SimpleImputer to simply impute numerical data by the mean and categorical data by the mode. Training a Random Forest-based GroupKFold is a good approach as the most accurate version of the model would be used. This is a significant advantage in comparison to models that don't use KFold testing. However, this approach fails in that it uses a Random Forest approach for a time series forecasting problem. According to Aman Arora from the Medium, Random Forest models don't "fit very well for increasing or decreasing trends which are usually encountered when dealing with time-series analysis, such as seasonality." [1] This likely led to lower accuracy within the model. While the notebook's approach is relatively better than many others', it lacks in not using hand-generated features as well as using a Random Forest approach for a time series problem.

PSP-NN model

This notebook's approach is one of the most unique approaches within the entire competition. For feature engineering, this approach uses the mean of natural logs of elapsed_time. This approach, then adjusts a pivot table by clipping the values between -log_elapsed_mean and 5. This approach takes advantage of mathematics to generate permutations not achievable by hand. In this way, this approach innovates and sets itself apart from the vast amount of other approaches that generate permutations by hand or permutations through using means, standard deviations, and modes. This unique mathematical technique for generating permutations likely leads to unique data which other models are not trained on, enabling the model to surpass many others. This allows for more accuracy than other approaches on the training data, which is evident by its 0.803 performance evaluation and a high F1 score—the harmonic mean of precision and recall—when training. However, this approach ultimately falls short in that there is clear overfit of the training data because the testing performance metric was only 0.617. This approach should have checked for issues like label imbalance after making permutations using an approach none other does. Label imbalance could have been fixed by using an approach like Random Over Sampling. While this model's feature approach is unique, its model is not much different from other neural network solutions that use dropout layers to reduce overfitting, batch normalization, and linear layers. The approach could have been made much better, however, through taking more time to perfect the feature approach.

LGBM + Early Stopping [LB 0.670]

This approach uses the same permutation approach for features that the random forest model uses. This means, while the model is strong in terms of the fact it generates a large range of permutations, it falls prey to not using unique mathematical techniques to generate permutations or hand-crafted permutations—a part of the downfall of the first approach. It does compete well against many approaches however. The LGBM—Light Gradient Boosting Machine—approach, which is based on decision trees similar to XGBoost (eXtreme Gradient Boosting), to the model coupled with using GroupKFold to obtain the best performance possible with the given model in its state is a powerful approach to a model as it takes advantage of LGBM's capabilities in training on large datasets, capabilities not present in models that use XGBoost. [2] The model approach for this Jupyter Notebook is powerful, but the approach ultimately fails in beating other models it should beat due to using the same approach for feature engineering and permutation generation as the Random Forest model, thus falling victim to feature engineering flaws that, while ultimately don't make the model approach bad, lead to the model being less performant than it theoretically should be.

DATA DESCRIPTION AND INITIAL PROCESSING

Data Description

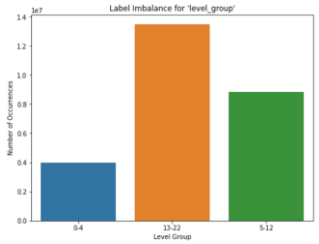
The dataset used in the Predict Student Performance from Game Play Kaggle Competition is a research dataset of game logs for educational game platform games, Jo Wilder. The dataset is meant to be used as a time series forecasting dataset to predict student performance during game-based learning in real-time, enabling game developers to fine tune educational games. It contains 13,174,211 rows of data. The label rows are identified with <session_id>_<question #> and each session having 18 rows for 18 different columns. The questions are ordered in levels—0 to 4, 5 to 12, and 13 to 22. The columns in the training and testing data consist of 20 categories, both numerical and categorical in nature.

The first category is session ID—the session ID of an event. The second category is index—the index of the event. The third category is elapsed_time, the amount of time in milliseconds that has passed between the start of the session and the event recording. The fourth category is event_name, the name of an action someone performs (such as a click). The fifth category is name, name of the event in terms of what specific thing an action does (such as a click closing a tab). The sixth category is level, the level of the game in which the action occurred—which is between 0 and 22. The seventh category is page, the page number of the event if it's a notebook-related event. The next two categories are room_coor_x and room_coor_y, the respective coordinates of a click in reference to in-game rooms for click events. The next two categories are screen_coor_x and screen_cor_y, the coordinates of a click in reference to the player's screen. The twelfth category is hover_duration, how long in milliseconds a hover happened for if a hover event occurred. The thirteenth category is text, which is the text players see during an event. The next three categories are fqid, room_fqid, and text_fqid—the respective fully qualified IDs of the event, room, and text. The seventeenth category is fullscreen, whether the player was in fullscreen mode. The eighteenth category is hq—whether the game is in high-quality. The nineteenth category is music, whether the game music is on or off. The last category is level_group. The group of levels and questions a row belongs to—0 to 4, 5 to 12, or 13 to 22. As for the submission file, it must contain the session_id, followed by the # correct column (which contains a 0 or 1 depending on whether the machine learning model predicts the player to get the question right or not), and a session_level column.

Initial Processing

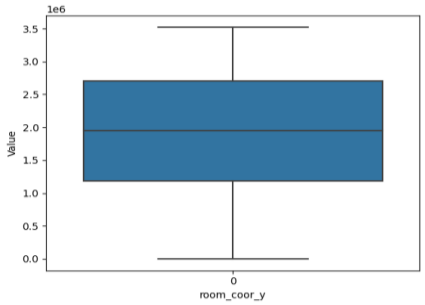
The data set contains 13,174,211 rows of data. This large quantity of data caused my computer and virtual machines difficulties in handling the data due to hardware limitations. As a result, I locally reduced the data to 6,500,000 rows of data using a command line argument. The full dataset was used when I ran my notebook on Kaggle. I then checked the shape of the data to make sure it contained 6.5 million rows of data and 20 columns (features). To ensure that I didn't have null values I checked for the sum of null values with isnull(). However, both categorical data and numerical data produced null values. For numerical columns (the aforementioned times, level, page, the room coordinates, the screen coordinates, and the hover duration), I initially used fillna() to impute the nulls with the means of the columns. However, this significantly dropped performance, so I removed it. For categorical columns (text, event_name, name, and level_group), I used a similar approach, opting to impute the columns with the mode of the columns. I then checked that none of the columns had nulls by using isnull() again. Once satisfied with this check, I proceeded to use seaborn to create custom barplots of the level_group feature to ensure there was no label balance. However, plotting the data showed level imbalance (from highest to smallest: 13 to 22 followed by 5 to 12 followed by 0 to 4).

Figure 3: level_group Label Imbalance Barplots



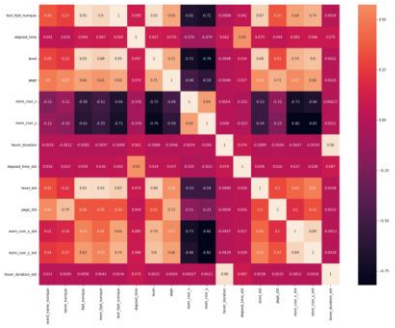
Label imbalances could lead to skewed data and hinder model performance. I then use Random Oversampling on my dataset to ensure the data is approximately even and not imbalanced. I then recheck for imbalance. With initial pre-processing done, I wanted to make sure my data statistically made sense to be used for machine learning. To do this, I created a custom function to create boxplots for every category. My categories' statistically made sense due to approximately uniform distribution, fairly little outliers, and a non-alarming spread. For instance, my boxplot for room_coor_y displayed this behavior.

Figure 4: room_coor_y Boxplot



After making sure my categories statistically made sense to use machine learning methods on, I moved onto feature engineering. For my feature engineering approach, I used groupby() for "session_id" and "level_group" to aggregate into the dataframe features for both existing categorical columns and existing numerical columns. For categorical columns, I aggregated features using the unique count of each category (nunique). For numerical columns, I aggregated features using the standard deviation for each "session_id" and "level_group." I also aggregated features using the mean for each "session_id" and "level_group" for numerical columns. However, I did not use new feature names for aggregated features that used the mean. After feature engineering, I used seaborn to generate a heatmap to make sure that high existence of variation doesn't happen within a singular column, so that way its variation doesn't overshadow other columns' and the data makes sense statistically to use for machine learning.

Figure 5: Training Data Heatmap



Once all my data was pre-processed, I was ready to train my machine learning model.

Modeling

General Training Approach

For training, I use a Group K-fold approach to find the best model to use on each of the 18 questions. This Group K-fold approach ensures that "the same group will not appear in two different folds" [3], thus making the K-fold approach more effective. I found that using 5 folds resulted in a decent approach and was standard across notebooks that used Group K-fold approaches. I first create an out-of-fold dataframe. This was a common technique notebooks that used K-folds had utilized. Out-of-fold predictions are "predictions made by models during the k-fold cross-validation procedure" [4]. These predictions help "estimate the performance of the model on unseen data," [5] thus allowing me to better understand my models' performances. I then iterated over each split of the dataset using Python's enumerate, which allows me to "keep a count of the iterations" [5], thus enabling me to better understand the process of training. I then, in a nested loop, iterated over each of the 19 questions and split them into groups. Group "0-4" contained questions 1 to 3, group "5-12" contained questions, 4 to 13, and group "13-22" contained questions 14 to 18. I then manually created training data and testing data splits based on approaches many of the public notebooks used. I then created the models, fitted them, and saved the models into a set of models. I added to out-of-fold dataframe with validation users at the current questions. For this step, the out-of-fold dataframe was populated with the class probabilities using predict_proba() for the XGBClassifier model I used. This is similar to how other notebooks did it for simple models. However, predict_proba() didn't work for the neural network model and ensemble model I made, so I used predict() with them and altered how I added to the submission file.

Feature Importances

After training my data, I found the feature importances of the models. Feature importances indicate how "useful [features are to] predicting a target variable" [6]. They were all roughly the same across models.

Figure 6: Feature Importances

	Feature	Importance
0	room_fqid_nunique	0.120788
1	text_fqid_nunique	0.095968
2	hover_duration	0.057128
3	level	0.056084
4	hover_duration_std	0.054545
5	elapsed_time_std	0.054170
6	name_nunique	0.053822
7	fqid_nunique	0.053314
8	page_std	0.053266
9	elapsed_time	0.052762
10	page	0.051540
11	level_std	0.051262
12	room_coor_x_std	0.050609
13	room_coor_x	0.049344
14	room_coor_y_std	0.049220
15	room_coor_y	0.049155
16	event_name_nunique	0.047024

The top 5 features in my feature importances table in order were "room_fqid_nunique," "text_fqid_nunique," "hover_duration," "level", and "hover_duration_std." The feature importance table surprised me. I did not expect the unique count of ids for rooms and text to be the most important features to classification of whether a student got a question correct. If the unique count of ids were to be the top feature towards classification, I would have expected the unique count of ids of the events (fqid) to be more prominent as different events could drastically impact whether a student got a question right. However, my assumption was wrong. The other three top features made sense, however. Hover duration can be correlated with the time students are thinking for. If the hover duration is shorter, students are more likely to have guessed on a question. If the however duration is longer, students are more likely to have thought out a question, thus a longer time before interacting with the computer. The standard deviation-aggregated hover duration feature is a top feature for a similar reason. Level being a top feature also made sense since levels could vary drastically, thus impacting student performance. The feature importances generally made sense, however, aside from the top two features.

Maximizing F1-score and Predicting on Testing Data

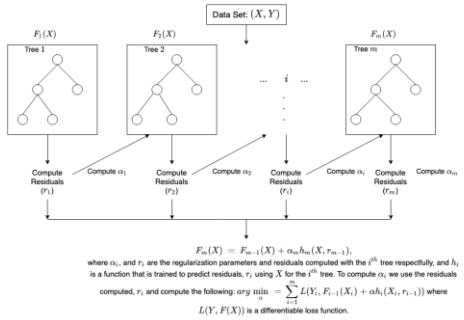
After figuring out feature importances, I optimized F1-score. To do this, I made a "true" dataframe that was a copy of the out_of_fold dataframe used to help estimate model performance on unseen data. I then used a similar approach to many notebooks that feature engineered in finding the best threshold and F1-score. The approach iterates over thresholds from 0.4 to 0.8, uses a technique in which the predictions are equivalent to whether or not the class prediction probabilities were greater than the threshold, and updates the best F1-score and threshold if they were. This maximizes the F1-score and threshold. After that, I created a Threshold vs. F1-score graph for the current model and a precision vs. recall graph for the current model on each of the 18 questions. I then predicted on test data to create the submission.csv file by using iter_test and jo_wilder testing

environment that is standard across nearly all notebooks. I iterated over the sample submissions testing dataset with iter_tests, feature engineered the testing dataset, figured out what group the sample submission was in, iterated through the 18 questions, and used predict_proba() and predict() depending on the model to make the predictions and create the submission.csv file.

Model #1: XGBClassifier Model

My first model was an XGBClassifier. I had used an XGBClassifiers previously in a side project and thought it might be cool to further explore its use in a new problem. The XGBClassifier was effective. XGBoost had the second highest F1-score performance out of my three models at 0.661 on the competition's test. This is likely due to the unique boosting and residuals for classification approach it uses to make weak models strong.

Figure 7: XGBClassifier Model [7]

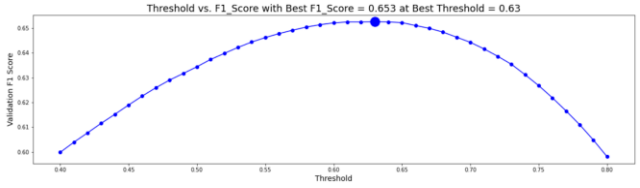


XGB stands for Extreme Gradient Boosting. XGBClassifiers use a random forest approach in combination with boosting, whereby various weak models are used in series to then make a strong classifier [8]. Interestingly, XGBoost models use residuals (differences between predicted and actual values) even for classification tasks. The boosting approach to XGBoost models makes it a strong approach.

After making the XGBoostClassifier, I tampered with different hyperparameters. However, much like my initial preprocessing approach, excessive modification led to the model performing worse when tuning hyperparameters. Therefore, I reverted the XGBoost model to default parameters and "logloss" for the evaluation metric.

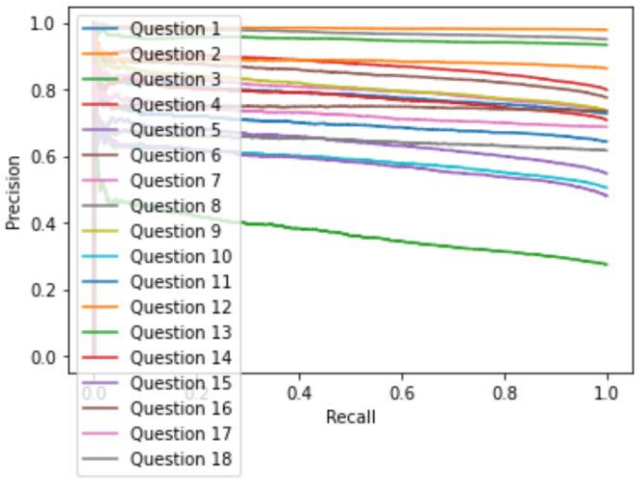
My F1-score vs. Threshold graph showed my training F1-score was at a peak of 0.653 and dropped off after, reaching a best threshold of 0.63. This means that my model struggled as I moved away from the optimal threshold of 0.63.

Figure 8: XGBClassifier Model F1-score vs. Threshold Graph



I also graphed the precision-recall curve for my XGB model across the 18 questions since I technically use a new XGB model for each question.

Figure 9: XGBClassifier Model Precision vs. Recall Curves

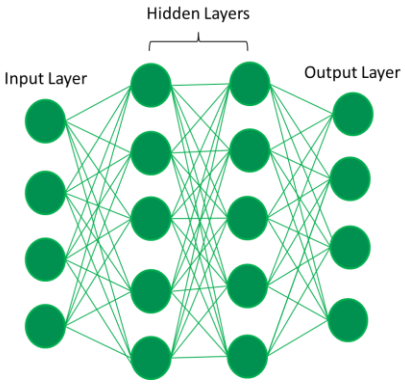


The precision-recall curves for the various question are generally flat indicating consistent performance across thresholds. Questions 2, 8, and 3 are higher in terms of precision, indicating that my model is making fewer false positive predictions compared to true positive predictions. This is a good sign because it indicates that when the model is classifying a student as having gotten a question correct they actually did. Unfortunately, there is a lot of variation amongst where the questions rank in terms of precision. The lower on the y-axis, and hence the lower the precision, the more false positive predictions compared to true positive predictions the model is making. This is a bad sign since many of the questions are lower in precision than others. This is not ideal for research purposes as the model may not truly predict, in some circumstances, if a child got a question right, so the developers of educational game platforms can't, on some questions, accurately improve the educational game to help students learn better since they don't know if a technique is truly leading to the child improving in their academic ability. Question 13 was an outlier and has a diminishing precision as recall increases. This means the model doesn't perform well for question 13, which is not ideal for research purposes. A potential cause could be due to a lack of training data for question 13.

Model #2: Neural Network Model

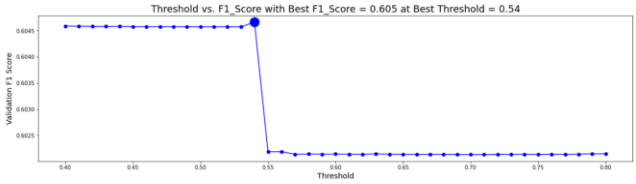
My second model was a neural network. Neural networks imitate the human brain. Input data is passed through a series of hidden layers where each layer conducts math to change the data, according to a set of weights and biases. Neural networks use backpropagation, which means they compute the gradient of mathematical loss functions with respect to weights and biases [9]. My neural network model had the worst performance out of the three models, having an F1-score of 0.612 in the competition test.

Figure 10: Neural Network Model



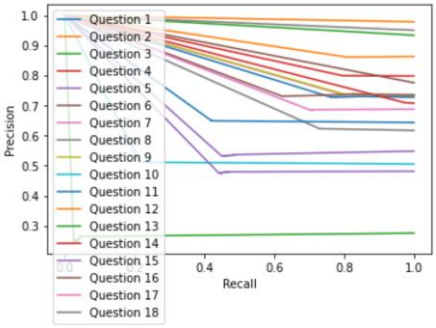
My F1-score vs. Threshold graph showed my training F1-score was at a peak of 0.605 and dropped off after slightly, reaching a best threshold of 0.54. However, before the drop off and after, the graph was fairly horizontal, suggesting that, aside from the initial drop off, my model performed consistently across thresholds.

Figure 11: Neural Network Model F1-score vs. Threshold Graph



The precision-recall curves for the various questions were not generally flat, instead being vertically downward before being flat, indicating that precision vastly fell off before remaining stable as recall increased. This means my model was consistently making more and more false positive predictions as recall increased, regardless of the fact that students actually got the question wrong. This indicates the neural network model I used was insufficient for this problem. Given the widespread distribution of the precision-recall curves for the questions and the fact precision is decreasing as recall increases, the model would not be sufficient for researchers to create a better educational platform for children. It is also interesting to note that question 13 is once again at a low precision and is an outlier just like for the XGBClassifier model, suggesting there is insufficient data for question 13.

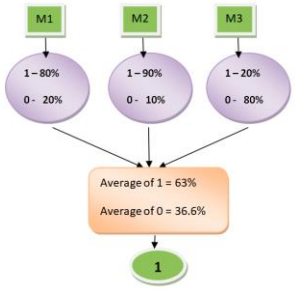
Figure 12: Neural Network Model Precision vs. Recall Curves



Model #3: KNN/Random Forest Ensemble Model

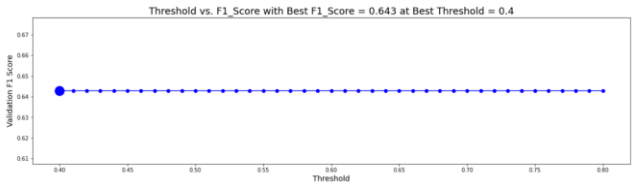
My third model was a K-Nearest Neighbors and Random Forest Ensemble model. Ensemble models combine various similar but different models and aggregate their results into a single score. Ensemble models are powerful because they can take advantage of different approaches models use towards classification and combine their strengths. To make the ensemble model, I used a Voting Classifier. A Voting Classifier is a type of ensemble model that "predicts an output (class) based on the highest probability score of the chosen class (output) as the output [11]. I used hard voting for the Voting Classifier, which means that the predicted output is "the class (output) which has the highest probability of being predicted by each of the classifiers" [11]. For my Random Forest, I used 100 trees (n_estimators). For my K-Nearest Neighbors, I used 5 neighbors (n_neighbors). The KNN/Random Forest Ensemble Model had the highest F1-score on the competition test, 0.663. This F1-score is extremely close to the XGBClassifier's performance and thus can be considered an insignificant difference.

Figure 13: Voting Classifier (Hard Voting)



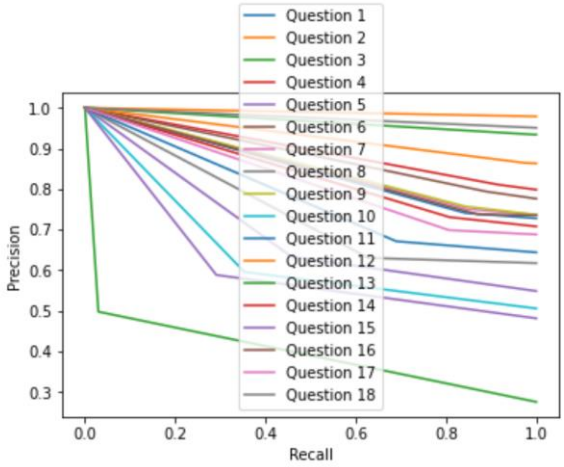
My F1-score vs. Threshold graph showed my training F1-score was at a peak of 0.643 and never dropped off, retaining a best threshold of 0.54. This indicates my model retained a high level of accuracy cross thresholds, performing well across the board. However, the hyperparameters may need to be tuned as a consistent performance across the board is a seemingly unusual result.

Figure 14: KNN/Random Forest Ensemble Model F1-score vs. Threshold Graph



The precision-recall curves for the various questions were not generally flat except for the first few which had a high precision (fewer false positive predictions compared to true positive predictions), instead being vertically downward before being flat, indicating that precision fell off for many of the questions before remaining consistent. This suggests that, during the drop off periods, before graphs became flat, more false positive predictions were being made by the model compared to true positive predictions. However, while there is some variation between where precision lies for the questions, it is not as widespread as the neural network model, suggesting the KNN/Random Forest Ensemble model is more suitable for the task at hand. Interestly, for the third time in a row, question 13 was an outlier that had precision drop off significantly. There is clearly a lack of insufficient data for question 13 because all three models showed similar issues with question 13's precision-recall curve.

Figure 15: KNN/Random Forest Ensemble Model Precision vs. Recall Curves



Which model should educational game developers choose?

All three of my models would require a great deal of additional fine tuning to accurately predict whether a student gets a question right, and thus enable educational game developers to increase the capacity at which their games help students learn. However, from the three models, educational game developers should choose the XGBClassifier model. The XGBClassifier model had very similar performance to the KNN/Random Forest Ensemble model. In addition, the F1-score vs. Threshold curve showed an rise, peak, and drop off as threshold increased. The KNN/Random Forest Ensemble model demonstrated a consistent F1-score across thresholds, which could or could not indicate issues with the model. The neural network model showed the worst performance across all metrics and, thus, it should be avoided. Therefore, the XGBClassifier model is the superior model out of the three models and is the ideal model to predict student performance to develop educational games that further students' academic capabilities.

APPENDIX

[1] Why Random Forests can't predict trends and how to overcome this problem? https://medium.datadriveninvestor.com/why-wont-time-series-data-and-random-forests-work-very-well-together-3c9f7b271631

[2] What is LightGBM? Advantages & Disadvantages? LightGBM vs XGBoost? https://www.kaggle.com/general/264327

[3] Stratified Group k-Fold Cross-Validation

https://www.kaggle.com/code/jakubwasikowski/stratified-group-k-fold-cross-validation

[4] How to Use Out-of-Fold Predictions in Machine Learning

https://machinelearningmastery.com/out-of-fold-predictions-in-machine-learning/#:%7E:text=An%20out%2Dof%2Dfold%20prediction,example%20in%20the%20training%20dataset.

[5] Enumerate() in Python

https://www.geeksforgeeks.org/enumerate-in-python/

[6] How to Calculate Feature Importance With Python

https://machinelearningmastery.com/calculate-feature-importance-with-python/

[7] XGBClassifier Model Picture

https://docs.aws.amazon.com/images/sagemaker/latest/dg/images/xgboost_illustration.png

[8] XGBoost

https://www.geeksforgeeks.org/xgboost/

[9] Artificial Neural Network

https://developer.nvidia.com/discover/artificial-neural-network

[10] In Keras, what is a "dense" and a "dropout" layer?

https://stackoverflow.com/questions/58830573/in-keras-what-is-a-dense-and-a-dropout-layer#:~:text=In%20short%2C%20a%20dropout%20layer,co%2Dadaptation%20of%20feature%20detectors

[11] ML | Voting Classifier using Sklearn

https://www.geeksforgeeks.org/ml-voting-classifier-using-sklearn/

[12] Voting Classifier Diagram

https://vitalflux.com/wp-content/uploads/2020/09/Screenshot-2020-09-07-at-4.40.58-PM-1.png