COMP9417 Project: Predicting Student Performance for Game Play

GitHub Link: LeechXDD/9417_Pro_Project (github.com)

Group Name: 9417-Pro

Siyuan Wu	**z5412156**
Chao Xu	**z5441640**
Benedicta Chun	**z5260342**
Weizhi Chen	**z5430533**
Liren Ding	**z5369144**

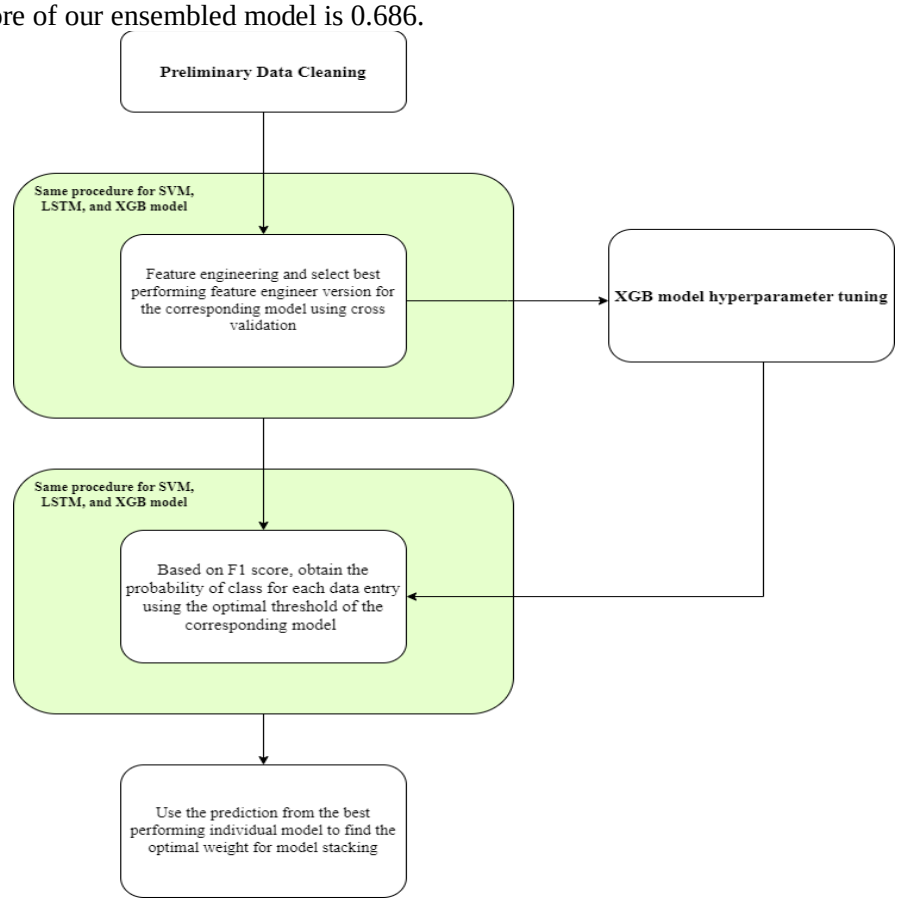
1. Introduction

1.1 Problem Statement

Hosted on Kaggle Predict Student Performance from Game Play, this project employs machine learning for performance prediction, a unique approach that enhances the research into knowledge-tracing methods and bridges the gap in most game-based learning platforms.
Our project employs Kaggle's time series API and data from an online educational game, aiming to determine whether players will answer questions correctly at various checkpoints. By utilizing variables such as session_id, elapsed time, event name, level, coordinates of clicks, hover duration, and seen text, we aim to predict the correctness of each student's answers. The compelling aspect of this project and its chosen focus is the real-world impact it brings along. By harnessing machine learning's power, we can enable game developers to create more effective educational games and provide educators with valuable insights, fostering broader acceptance of game-based learning platforms.

1.2 Description of Project

The diagram below shows the workflow of our project. In this project, we have employed SVM, LSTM and XGB, three different machine learning models to solve the problem of predicting student's performance. The primary goal of this project is to identify the best model that effectively performs the best results. To achieve this, we developed several features engineering functions to preprocess the dataset. Before training our models, we split the dataset into training subset and validation subset. This is useful for avoiding overfitting and underfitting issues. Three models are trained individually at first stage with training subset, then we evaluate on the validation subset based on serval matrix, particularly F1 Score. Our finding is that XGB provided the most accurate predictions due to inner function dealing with overfitting problem and loss function optimization. Finally, we ensembled three models by giving different weights based on their individual F1 Score. This is a common technique that able to improve the performance of predictions. Thus, the result F1 Score of our ensembled model is 0.686.



1.3 Metric

The objective of the competition is to predict the binary label for each session/question number pair. However, the dataset provided in this competition is fairly unbalanced with the majority of the data labeled as correct, which makes the use of accuracy metric inappropriate. Therefore, the suggested evaluation metric for this Kaggle competition is the F1 score. The equation of F1 score is represented below:

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \; where$$

$$precision = \frac{TP}{TP + FP} \; and \; recall = \frac{TP}{TP + FN}$$

Precision and recall values range from 0 to 1. To maximize the F1 score, both precision and recall must be as close to 1 as possible, according to the equation. While precision and recall use false positives and false negatives, the utilization of false positives and false negatives will penalize a model that consistently labels a subject with the same label as the majority of training subjects. Each user will have the answer for 18 questions in the validation dataset, and the model will obtain the F1 for each question. The sum of F1 from each question is then subjected to a macro-averaging procedure. Below is the equation for Marco averaged F1:

$$Marco\ F1 = \frac{\sum_{i=1}^{N} F_{1i}}{N} \; where\ N\ is\ the\ number\ of\ question$$

2. Data Processing

2.2 Feature Engineering

2.2.1 Exploratory Data Analysis

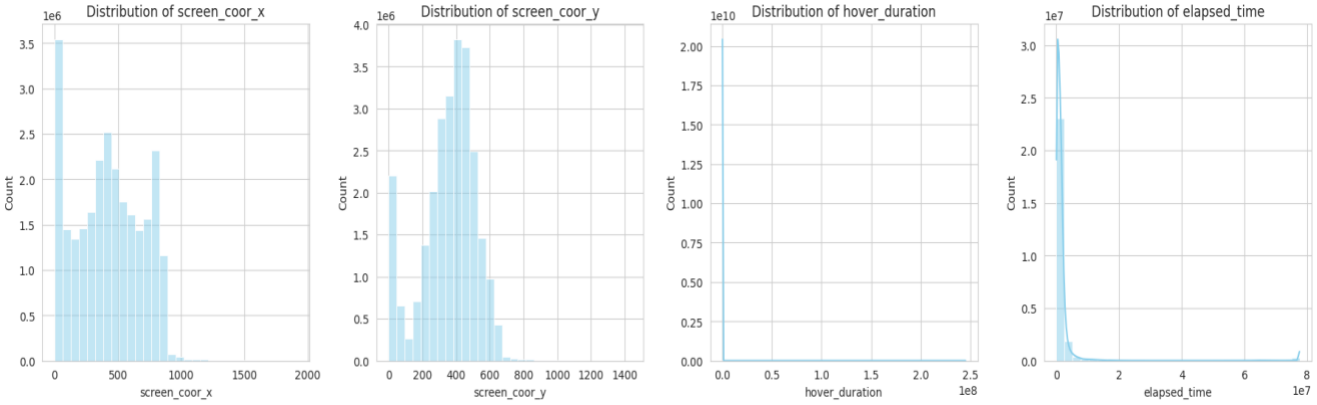


Figure 2.2.1 Distribution of variables 'screen_coor_x', 'screen_coor_y', 'hover_duration' and 'elapsed_time'

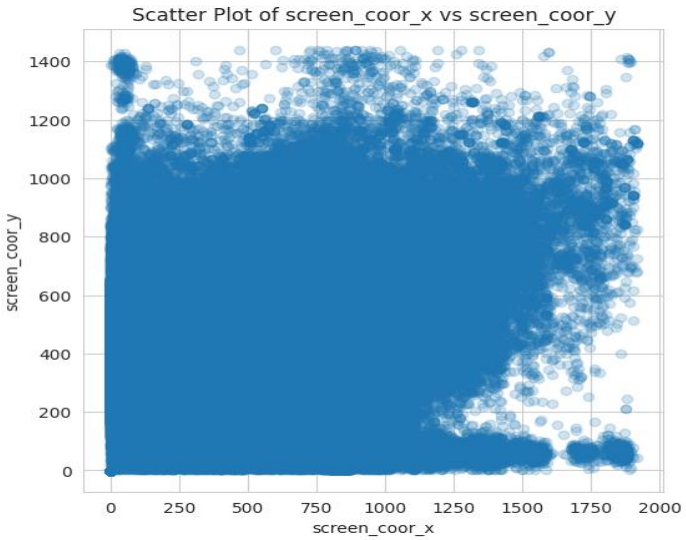


Figure 2.2.2 Scatter plot of screen_coor_x vs screen_coor_y

From Figure 2.2.1 it shows that the screen coordinates where players click 'screen_coor_x' and 'screen_coor_y'are spread widely indicating varied user activity across different parts of the player's screen. Based on this initial insight, Figure 2.2.2 gives a spatial representation of where players click on the screen. The data points are spread throughout the plot which suggests that players interact with different areas on the screen. This factor was considered when implementing the feature engineering function as it can capture spatial patterns of user interactions on the screen that are more frequently interacted with and can be used to infer the ease or complexity of the game interface.

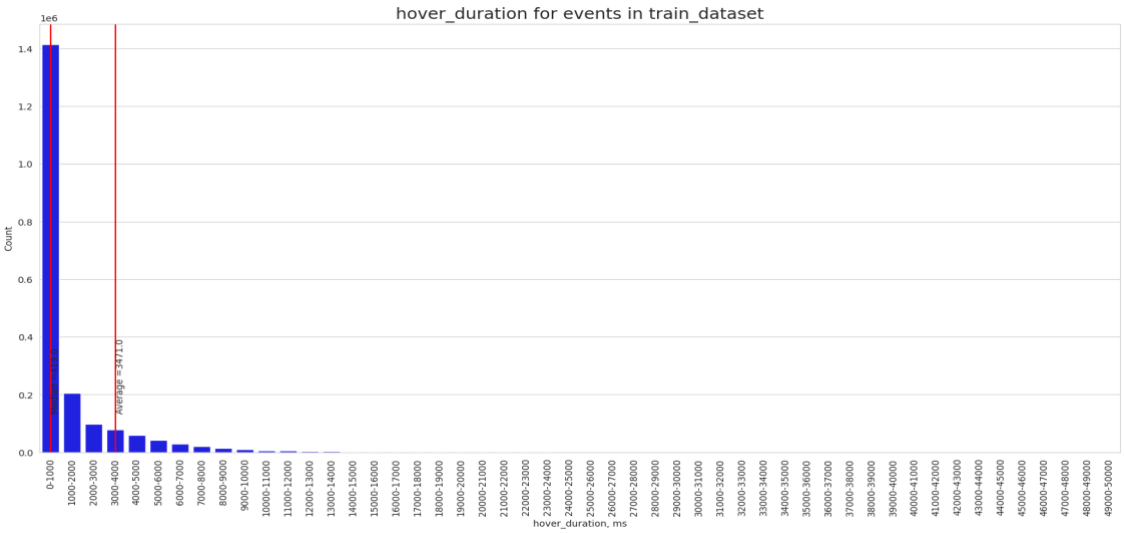


Figure 2.2.3 Histogram of hover_durations

Figure 2.2.3 provides a clear understanding of the distribution of hover durations in the dataset as well as their central tendencies. The mean and median hover durations are relatively low suggesting that most hover events are of short duration. Capturing the 'hover_duration' feature can be used to be served as a measure of user engagement, longer hover durations potentially indicative of higher interest levels, which may potentially affect performance on a user's ability to answer questions correctly. Incorporating the average or median hover duration per user into the feature set could potentially yield insightful results and binning "hover_duration" into distinct intervals can potentially capture important data patterns.

2.2.2 Feature Engineering Implementation

We employed 4 distinct version of feature engineering where each function was iteratively building on the previous version to enhance model learning.

Version	*Features Engineered*	*Evaluation*	*Transformation*
1 (feature_engineer)	- Number of unique entries for categorical variables - Mean and standard deviation of numerical variables	- Initial Version	- None
2 (feature_engineer_ver2)	- Aggregated statistics (mean, std) of categorical variables - Binning for specific numerical variables, New Features: 'screen_coor', 'total_hover_duration	- Performed preliminary modelling and examined improvements in model performance	- None
3 (feature_engineer_ver3)	- Number of unique categories - Dummy variables for top N most frequent events and names	- Checked feature importance and model performance. - Monitored model performance over different data subsets	- Yeo-Johnson power transformation
4 (feature_engineer_ver4)	- Label encoding for categorical features - Expanded aggregates for numerical features to include sum, mean, std	- Conducted an ablation study. Ensured model didn't overfit or underfit	- Discretization of numerical features using quantile strategy (KBinsDiscretizer)
5 (feature_engineer_ver5)	- Label encoding for categorical features, - Discretization (binning) of numerical features and calculation of mean and std for binned features, - Interaction between screen coordinates using Euclidean distance, - Calculation of aggregated total_hover_duration	- Iteratively built based on previous versions	Label encoding - Filling missing values with median - Binning using KbinsDiscretizer

Preliminary Analysis and Insights

The exploratory analysis demonstrated significant variations in user activity across different parts of the player's screen, which influenced the feature engineering function to capture these spatial patterns (Figure 2.2.1 and Figure 2.2.2). Similarly the distribution of 'hover_durations' indicated that most hover events to be of short duration, prompting the inclusion of the feature 'hover_duration' to measure the user engagement and possibly infer about their interest levels (Figure 2.2.3).

Evaluation and Effectiveness

To evaluate the effectiveness of the feature engineering function, we performed preliminary modelling and examined improvements in model performance (e.g. increased accuracy, reduced error rate) over each iteration of the feature engineering function. Features from the later versions of the function that exhibited high feature importance or significantly improved model performance was deemed valuable and were incorporated into the final model.

Our comprehensive feature engineering process was designed to translate complex data patterns into a format usable by machine learning algorithms. For thorough evaluation, we employed a RandomForestClassifier for each question in the dataset. We filtered the data by the 'level_group' of each question, and trained the model on these specific subsets. We then evaluated model performance using accuracy as the metric on a validation set. Additionally, we recorded the feature importances to determine which features were most influential in the model's predictions.

3. Models

3.1 Baseline model (SVM)

3.1.1 Model Selection

The team chose Support Vector Machine (SVM) as our baseline model. SVM is a powerful and widely used supervised machine learning algorithm for classification tasks. Particularly, it is well suited for solving problems with a clear margin boundary between classes. While our data consist of many features, SVM performs well in high-dimensional feature spaces, and it is very robust against overfitting. In this binary classification problem, SVM aims to find a hyperplane that maximize the margin between the two classes, where the margin is the distance between the hyperplane and the nearest data points from each class. However, as the support vectors are the critical data points that influence the position and orientation of the decision boundary, that also cause SVM to be more sensitive to outliers and noise (Cortes and Vapnik, 1995).

3.1.2 Model Implementation

The implementation of SVM utilised the SVC module of scikit-learn. It is confident that there will not be an identifiable linear decision boundary between the two classes based on the data structure, so the default kernel (Radial Basis Function Kernel) is selected. Moreover, SVM standalone does not produce an appealing performance in comparison to the XGB model, and it only participates in very small portions of the model stacking; consequently, SVM hyperparameter tuning is minimal.

3.1.3 Model Performance

The results of the 5-fold cross validation loop on three feature engineer versions are summarised in Table 1. This phase aims to assess the impact of various data engineering variants on the model, with the optimal variant producing the highest mean F1 value. Based on our findings, V3 is the best method for SVM feature engineering, and we will use this version to train the model that will be used for model stacking later. Notably, the mean F1 difference between different feature engineer variants is less than 0.001, indicating that the effect of the feature engineer on this model requires additional investigation.
In addition, Table 2 demonstrates that SVM performs poorly on questions with a balanced distribution of true and false labels; model stacking will enhance this performance.

Baseline performance based on 5-fold cross validation

Model	*Feature Engineer Version*	*Mean F1*	*F1_1*	*F1_2*	*F1_3*	*F1_4*	*F1_5*
SVM	V1	0.649	0.652	0.649	0.649	0.648	0.650
	V2	0.649	0.652	0.649	0.649	0.648	0.649
	V3	0.650	0.651	0.649	0.650	0.651	0.648

Best threshold based on 5-fold cross validation

	V1	*V2*	*V3*
F1_1	0.65	0.65	0.66
F1_2	0.63	0.64	0.63
F1_3	0.64	0.64	0.63
F1_4	0.65	0.66	0.66
F1_5	0.64	0.63	0.63

3.2 LSTM

3.2.1 Model Selection

Long Short-Term Memory (LSTM) is a special kind of Recurrent Neural Network that can store information over long periods by introducing a "Memory Cell". Each memory cell is controlled by gates that determine how information is stored, modified and deleted (Hochreiter and Schmidhuber, 1997). Considering our datasets comprise sequences of student learning, a student's level might depend on their learning history. LSTM has potential to understand those sequences and perform a good result.

3.2.2 Model Implementation

In the context of this, we have engineered a distinct LSTM model for each question and combined their F1 score to derive a comprehensive measurement of our models' performance. After experimenting with various parameters, we determined the optimal parameters Implemented for those models included a learning rate of 0.001, a decay step of 100, a decay rate of 0.9 and 3 epochs. In addition, we discovered that specific models for questions like 1, 5, 10, 11, 13, 15 required more epochs to yield improved performance while avoiding underfitting and overfitting. Thus, we increase those models to 5 epochs. Furthermore, we implemented a function that able to generate data as sequences.

3.2.3 Model Performance

we have tried four different feature engineering techniques to refine the dataset and prepare as sequence for LSTM. Below are the result F1 Score for LSTM in different versions of feature engineering. We can notice that version 3 is best for training LSTM even it is similar to V1. The reason of V3 is best because the best threshold of more stable than other versions.

5-fold cross validation

Model	*Feature Engineer Version*	*Mean F1*	*F1_1*	*F1_2*	*F1_3*	*F1_4*	*F1_5*
LSTM	V1	0.65	0.651	0.649	0.650	0.650	0.65
	V2	0.6496	0.651	0.649	0.650	0.650	0.648
	V3	0.65	0.651	0.649	0.650	0.651	0.649

Best threshold based on 5-fold cross validation

	V1	*V2*	*V3*
F1_1	0.65	0.65	0.64
F1_2	0.61	0.59	0.59
F1_3	0.6	0.58	0.6
F1_4	0.6	0.66	0.65
F1_5	0.61	0.58	0.6

3.3 XGB (Extreme Gradient Boosting)

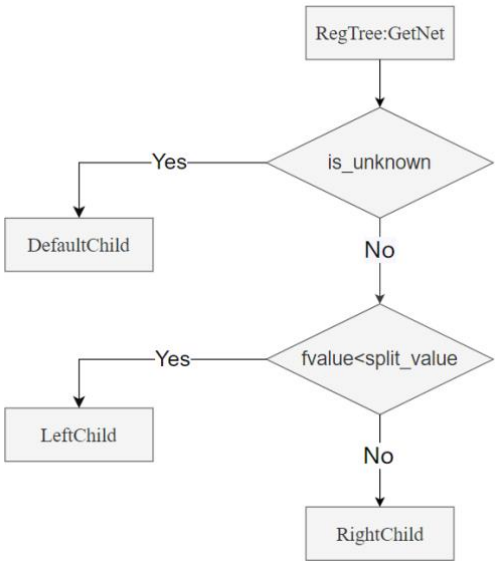
3.3.1 Model Selection

XGB is an ensemble of weak learners sequentially. It fits each weak learner to the negative gradient of the objective function. After the previous generated weak learner is fixed, new weak learner is added into the model. The added weak learner is focused on the questions that previous learner has poor performance. Through this way, the overall performance can be improved. The basic loss function of XGB can be written as:

$$L^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i{}^{(t-1)} + f_t(x_i) + \Omega(f_t)$$

Compared with other gradient boosting model, XGB has some comparative advantages which make it more appropriate for this project.

In the Kaggle competition, there are large amounts of data and some columns have missing values. XGB has built an inner function which can deal with missing values. Therefore, XGB can have good performance without pre-processing of lost data in the database. The logic for dealing with missing data is displayed in the diagram below.



Feature	*Descriptions*
Overfitting Prevention	XGBoost incorporates regularization into the loss function, providing an effective mechanism to control model complexity and prevent overfitting.
Learning Rate Application	XGBoost applies the learning rate to the weight of the leaf node after each iteration. This method attenuates the impact of a single leaf node, enabling broader learning space for subsequent iterations.
Column Subsampling	XGBoost employs column subsampling to mitigate overfitting, reduce computational complexity, and enhance training efficiency. By using random column subsampling, XGBoost can deliver superior performance when handling new data.

3.3.2 Model Implementation

We used XGBClassifier to implement the xgb model. The process is listed below:

Step	*Action*
1	Data Preparation: Apply the original data engine to generate raw data for the model.
2	Data Splitting: Use K-Fold Cross-Validation and Group K-Fold Cross-Validation to partition data into training and validation sets.
3	Initial Model Training: Generate a base model using the categorized data and generate the threshold and F1-score.
4	Hyperparameter Tuning: Conduct parameter tuning to find the parameter set with the best performance.
5	Model-Data Engine Combination: Combine the optimal model with different data engines to determine the best engine for the XGBoost model.

3.3.3 Hyper parameter decision and tuning

After generating original model based on XGB, we think adjusting parameters can improve the f1-score. Our main parameters are listed in the table below.

Parameter	Description
objective	Based on the Kaggle competition, the objective is to correctly predict the correctness of given questions. We choose the 'binary logistic' function as our objective function due to the binary classification nature of the problem.
eval_metric	We use the logloss function as the evaluation metric. This can help evaluate the difference between the true probability and predicted probability in a binary classification problem.
n_estimators	This parameter affects the complexity and efficiency of the model. To avoid overfitting and to train the data thoroughly, we set n_estimators to 1000.
early_stopping	To avoid overfitting and maintain the best model parameters, we implement early stopping after 50 iterations without improvement.
subsample	This parameter defines the proportion of training data used in each training iteration. We set subsample to 0.8, meaning only 80% of parameters are used in the training process, which helps avoid overfitting and makes the model more generalized.
colsample_tree	This parameter is used to define the feature proportion used in training. By setting colsample_tree to 0.8, the model can be more generalized, and the model efficiency can be improved.
max_depth	As one of the most important hyperparameters, it defines the maximum depth of the decision tree used in XGBoost. A higher max_depth leads to a more complex model and increases the risk of overfitting. We set max_depth to 4.
learning_rate	The learning rate determines the contribution of every single decision tree to the final model. A higher learning rate helps the model converge quicker but also increases the risk of overfitting.

Figure 3.3.3 shows the relationship between learning_rate and f1-score. In our case, we have set the learning rate to 0.02.

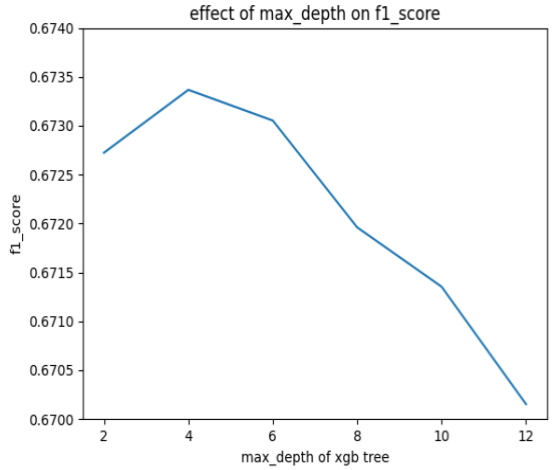


Figure 3.3.2

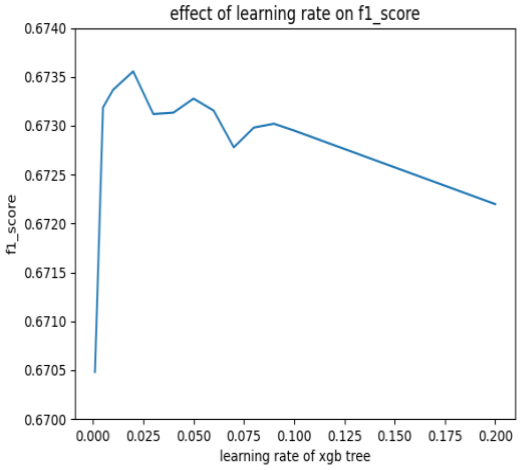


Figure 3.3.3

Since max_depth and learning rate are most important parameters in our case, we conducted grid search to tune the values. The value we tried for the max_depth and learning_rate is listed in the table below.

Hyper Parameter	Number set
max_depth	2,4,6,8,10,12
learning rate	0.001 ,0.005, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.2

After the parameter tunning, the hyper parameter setting is listed in the following table:

Hyper Parameter	Value for parameter
objective	binary_logistic
eval_metric	logloss
n_estimators	1000
early_stopping	50
subsample	0.8
colsample_tree	0.8
max_depth	4
learning_rate	0.02

3.3.4 Model Performance

The table 3.3.1 and table3.3.2 show the threshold and f1-score for first 5 cross-validation step. From the table, it is easily to find with the increase of step, the model has better performance. This is because during each cross-validation step, the model evaluates the performance of current model and adjust parameters in next step to achieve better balance between overfitting and accuracy.

Model	*Feature Engineer Version*	*Mean F1*	*Fold_1*	*Fold_2*	*Fold_3*	*Fold_4*	*Fold_5*
XGB	Original Version	0.538	0.375	0.475	0.551	0.614	0.673
	V2	0.540	0.376	0.476	0.553	0.617	0.677
	V3	0.544	0.378	0.478	0.557	0.622	0.683
	V4	0.541	0.377	0.477	0.554	0.618	0.679

Table 3.3.1 f1-score for each cross-validation process with different feature engineering versions

	Original Version	*V2*	*V3*	*V4*
Fold_1	0.410	0.42	0.46	0.45
Fold_2	0.530	0.52	0.52	0.52
Fold_3	0.570	0.57	0.56	0.57
Fold_4	0.610	0.61	0.61	0.60
Fold_5	0.630	0.63	0.63	0.63

Table 3.3.2 Threshold for each cross-validation process with different feature engineering versions

The cross-validation set for final model is 10. Table 3.3.3 illustrates the final f1-score for xgb model with different data engineering functions.

Model	*F1-score*	*Best Threshold*
Basic XGB	0.673	0.62
XGB with Data Engineering v2	0.678	0.63
XGB with Data Engineering v3	0.684	0.62
XGB with Data Engineering v4	0.679	0.63

3.4 Ensemble Model

3.4.1 Ensemble Process and results

To further improve the f1-score, we decided to create an ensemble using weighed voting mechanism based on existing model. We first combine each model with best feature engine respectively and generate the prediction data. Then we go through all the possible combination of weights to generate the ensemble with bestf1-score. The graph 3.4.1 shows the relationship between different weights and f1-score. It is easy to find that with the increase of xgb portion, the total f1-score increases.

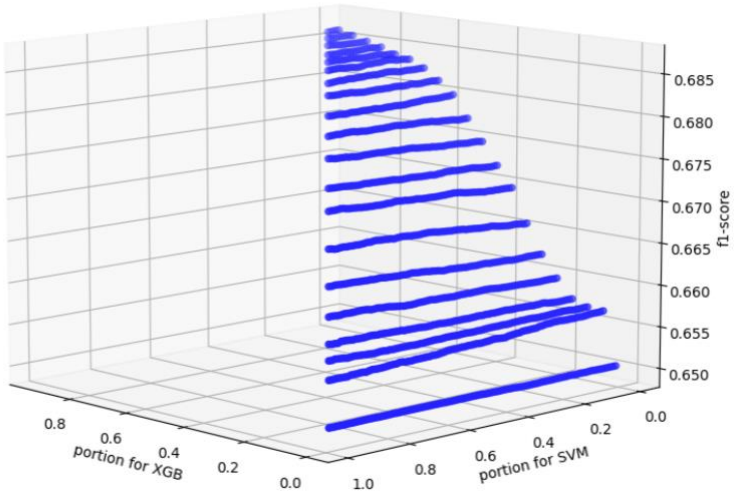


Figure 3.4.1 f1-score with different weights

Based on the above graph, the result of the ensemble model is listed below.

Weight for xgb	*Weight for svm*	*Weight for lstm*	*f1-score*
0.95	0.01	0.04	0.686

3.4.2 Limitation of Current Ensemble Model

For the ensemble model weighted voting mechanism, there are several limitations:
1. All the models used in the ensemble is trained on the same dataset, after ensemble, there may be overfitting problem.
2. The basic assumption of weighed voting mechanism is that each model in ensemble is independent. Based on our model, it is hard to find whether there are relationships between any two models.
3. For our final model, it is highly relied on the accuracy of xgb. If xgb has error or uncorrect prediction outcome, it is hard for the ensemble to fix it.

4. Discussion and Conclusions

During the span of this project, we have experimented with a variety of machine learning and data processing methods. While some of these methods did not yield the desired results, the final model we developed achieves an F1 score of 0.6858 on our private test set. Even though we did not submit our final prediction to the public submission, based on the score difference between the private and public test sets of other participants, our final score of 0.6858 is expected to put us in the top half of the leaderboard.
Early in the development process, we observed that the feature engineer method we adopted from the suggestion solution utilised the group by function on the data frames. While attempting to forecast how a user will respond to the questions, each question's data can be viewed as a separate set of data. Consequently, each question's data has its own data distribution, as shown in the figure 6.1, and using the group by function on their level would almost undoubtedly lead to a loss of information. However, because of the limitation of computing capacity, we were still bounded to use this method. The original dataset contains more than 20 million rows of data, but after feature processing, the size of the dataset is reduced to 400 thousand rows. However, even the modest size of the data, each model mentioned in our report still requires approximately 1 hour to complete its training. As Google Colab's execution time is strictly limited, it would be impracticable to train the model using the complete dataset.

Our approach to utilising various models is also constrained by our limited computational capacity. Initially, the teams intended to implement Transformer for this competition, not only because it is the state-of-the-art, but also because we observed that many of the best ensembled models on leaderboard implemented Transformer. We have an experiment with Transformer using Bidirectional Encoder Representations from Transformers(Bert) in the team's Github repository. When we evaluate the model, however, it takes longer than six hours to train on a single question. Given the duration of the the task, we determine that it would not be optimal for the team to pursue this path. However, implementing Transformer for this topic remains desirable for future research, and if time permits, we can attempt running the model in a local environment to avoid Colab's limitations and continue working along the path.

When examining the data distribution in Figure 6.1, it is evident that some questions have data imbalance issues, i.e., there are significantly more correct labels than incorrect labels. The team employed SMOTE (Chawla, 2018), an oversampling technique based on KNN, in an effort to address the imbalance problem. However, that did not produce an improved outcome. The team also attempted to construct sub-models within a singular model. Each sub-model will incorporate a portion of the majority label group's data with the minority label group's data. This can be thought of as an under-sampling technique, but the advantage of using sub-model is that no information will be lost. We will then derive a weighted class probability by predicting the test set using all the sub-models. Unfortunately, that did not produce an improved outcome. Notably, the team observes that the model achieves a low F1 score on questions with relatively balanced data. Coupled with the fact that the best team on the leaderboard achieves a 1.5% higher score than our model, the team speculates that the current data may not be adequate to produce a more accurate prediction. For instance, a user who is overconfident and a user who truly knows the answer may have very similar data values but very different results. Therefore, we believe that it would be difficult to bring about significant improvements if only screen data were presented; data from other areas, such as the user's personality, might be needed for enhanced predictions.

5. References

Schmucker, R., Wang, J., Hu, S., & Mitchell, T. M. (2022). Assessing the Performance of Online Students - New Data, New Approaches, Improved Accuracy. Available at: <**https://arxiv.org/abs/2109.01753v2**>

Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. Neural computation, 9(8), pp.1735-1780. Available at: < https://ieeexplore.ieee.org/abstract/document/6795963/authors#authors >

Chawla, N.V. *et al.* (2002) 'Smote: Synthetic minority over-sampling technique', *Journal of Artificial Intelligence Research*, 16, pp. 321–357. doi:10.1613/jair.953.

Cortes, C. and Vapnik, V. (1995) 'Support-Vector Networks', *Machine Learning*, 20(3), pp. 273–297. doi:10.1007/bf00994018.

Kumar, N. (2019) 'Advantages of XGBoost Algorithm is Machine Learning', Medium, 9 March, Available at: The Professionals Point: Advantages of XGBoost Algorithm in Machine Learning (Accessed: 02 August 2023)

Or, B. (2021) 'Is F1 the appropriate criterion to use? What about F2, F3,…, F beta?', *Medium*, 3 April. Available at: https://towardsdatascience.com/is-f1-the-appropriate-criterion-to-use-what-about-f2-f3-f-beta-4bd8ef17e285 (Accessed: 01 August 2023).

Tianqi Chen, Carlos Guestrin, 16 August 2016, XGBoost: A Scalable Tree Boosting Sysetem. Available at: arxiv.org/pdf/1603.02754.pdf (Accessed: 02 August 2023)

6. Appendix

Figure 6.1 Label distribution for each question

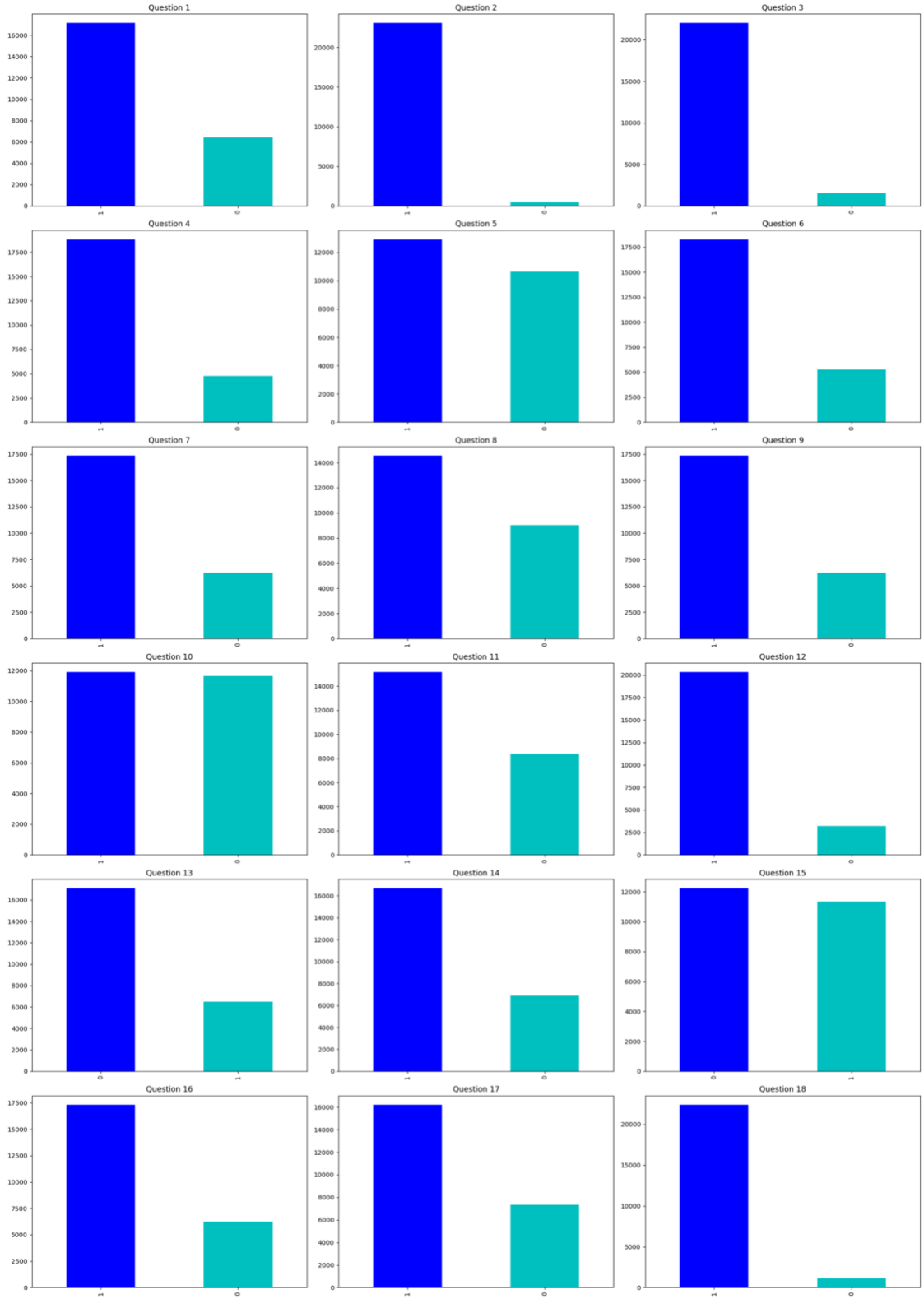


Figure 6.2 Score of F1 of each question based on SVM model

question 1

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1262
1	0.73	1.00	0.85	3451
accuracy			0.73	4713
macro avg	0.37	0.50	0.42	4713
weighted avg	0.54	0.73	0.62	4713

question 2

	precision	recall	f1-score	support
0	0.00	0.00	0.00	115
1	0.98	1.00	0.99	4598
accuracy			0.98	4713
macro avg	0.49	0.50	0.49	4713
weighted avg	0.95	0.98	0.96	4713

question 3

	precision	recall	f1-score	support
0	0.00	0.00	0.00	306
1	0.94	1.00	0.97	4407
accuracy			0.94	4713
macro avg	0.47	0.50	0.48	4713
weighted avg	0.87	0.94	0.90	4713

question 4

	precision	recall	f1-score	support
0	0.00	0.00	0.00	981
1	0.79	1.00	0.88	3732
accuracy			0.79	4713
macro avg	0.40	0.50	0.44	4713
weighted avg	0.63	0.79	0.70	4713

question 5

	precision	recall	f1-score	support
0	0.52	0.07	0.13	2094
1	0.56	0.95	0.70	2619
accuracy			0.56	4713
macro avg	0.54	0.51	0.42	4713
weighted avg	0.54	0.56	0.45	4713

question 6

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1003
1	0.79	1.00	0.88	3710
accuracy			0.79	4713
macro avg	0.39	0.50	0.44	4713
weighted avg	0.62	0.79	0.69	4713

question 7

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1197
1	0.75	1.00	0.85	3516
accuracy			0.75	4713
macro avg	0.37	0.50	0.43	4713
weighted avg	0.56	0.75	0.64	4713

question 8

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1729
1	0.63	1.00	0.78	2984
accuracy			0.63	4713
macro avg	0.32	0.50	0.39	4713
weighted avg	0.40	0.63	0.49	4713

question 9

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1139
1	0.76	1.00	0.86	3574
accuracy			0.76	4713
macro avg	0.38	0.50	0.43	4713
weighted avg	0.58	0.76	0.65	4713

question 10

	precision	recall	f1-score	support
0	0.50	0.88	0.64	2295
1	0.59	0.17	0.26	2418
accuracy			0.51	4713
macro avg	0.54	0.52	0.45	4713
weighted avg	0.54	0.51	0.44	4713

question 11

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1647
1	0.65	1.00	0.79	3066
accuracy			0.65	4713
macro avg	0.33	0.50	0.39	4713
weighted avg	0.42	0.65	0.51	4713

question 12

	precision	recall	f1-score	support
0	0.00	0.00	0.00	612
1	0.87	1.00	0.93	4101
accuracy			0.87	4713
macro avg	0.44	0.50	0.47	4713
weighted avg	0.76	0.87	0.81	4713

question 13

	precision	recall	f1-score	support
0	0.72	1.00	0.83	3375
1	1.00	0.00	0.00	1338
accuracy			0.72	4713
macro avg	0.86	0.50	0.42	4713
weighted avg	0.80	0.72	0.60	4713

question 14

	precision	recall	f1-score	support
0	0.18	0.00	0.00	1274
1	0.73	1.00	0.84	3439
accuracy			0.73	4713
macro avg	0.46	0.50	0.42	4713
weighted avg	0.58	0.73	0.62	4713

question 15

	precision	recall	f1-score	support
0	0.51	1.00	0.68	2405
1	1.00	0.00	0.00	2308
accuracy			0.51	4713
macro avg	0.76	0.50	0.34	4713
weighted avg	0.75	0.51	0.34	4713

question 16

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1181
1	0.75	1.00	0.86	3532
accuracy			0.75	4713
macro avg	0.37	0.50	0.43	4713
weighted avg	0.56	0.75	0.64	4713

question 17

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1397
1	0.70	1.00	0.83	3316
accuracy			0.70	4713
macro avg	0.35	0.50	0.41	4713
weighted avg	0.50	0.70	0.58	4713

question 18

	precision	recall	f1-score	support
0	0.00	0.00	0.00	228
1	0.95	1.00	0.98	4485
accuracy			0.95	4713
macro avg	0.48	0.50	0.49	4713
weighted avg	0.91	0.95	0.93	4713