




Riiid! Answer Correctness Prediction

Team ‘:)’

Vy Tran & Rachel Peng & Selena Lin



COSI 123A - Statistical Machine Learning
Perforessor Pengyu Hong

Dec 11th, 2020

TABLE OF CONTENTS

1. Introduction
2. Technologies
3. Exploratory Data Analysis (EDA)
 - a. train.csv
 - b. question.csv and lecture.csv
4. Experimentation
 - a. Data pre-processing
 - b. Feature engineering
 - c. Primary method: LightGBM
 - d. Secondary methods: Keras and XGboost
5. Result Prediction
6. Summary and Lesson Learned
7. Future Improvement on Model
8. Member Contributions
9. References
10. History of Submission

1. Introduction

In this report, we present the process of deriving the result of a month-long investigation on Kaggle's Data Science Competition *Riiid! Answer Correctness Prediction* hosted by Riiid AIED Challenge. The report will include different stages of the experimentation from exploratory data analysis, pre-processing of data, models for classification, and results with analysis.

a. The competition

The idea of the competition is generated from the lack of educational resources available and the lack of dynamically personalized learning. The special case of COVID-19 makes education an even less approachable resource since most countries are forced to temporarily close the schools. Moreover, school learning is unable to detailedly keep track of each student's performance and mastery of each skill and concept. Thus, in order to prevent intellectual development from further delaying and before the equity gap, as a result, becomes larger, the society is encouraged to update the conventional methodology of evaluating performance, arousing engagement, and personalized teaching. Riiid Labs has already made some footprints in building a creative educational system. In the competition, we are provided with more than 100 million student interactions from the EdNet, which was released this year and now is the world's largest open database for AI education. In the competition, participants will build algorithms for tracking students' performance over the course of studying interactions made in EdNet. The overall goal of the model is to predict students' accuracy in answering the next question.

b. The host: Riiid Lab



Riiid Lab is developed to provide leading solutions of data-driven AI technology. They connect with experts in education, skill training, and technology in a global perspective to innovate better learning algorithms and provide learning with effectiveness and efficiency. Moreover, the competition is written and based on the Kaggle's times-series API which we will discuss in later sessions.

c. Data Source and overview

Riiid Lab has launched EdNet, the world's largest open database for AI education. This competition asks participants to predict whether users are able to answer the next question correctly with the given history of each user's interaction in EdNet and the detailed information about the questions and more.

For data, four files are given: `train.csv`, `questions.csv`, `lectures.csv`, and `example_test_rows.csv`.

`train.csv`:

- `row_id`: (int64) ID code for the row.
- `timestamp`: (int64) the time between students' first event completion and following interactions (in milliseconds).
- `user_id`: (int32) User ID, unique for each student.
- `content_id`: (int16) ID code for the user interaction, unique for each interaction
- `content_type_id`: (int8) 0 if the event was a question being posed to the user, 1 if the event was the user watching a lecture.
- `task_container_id`: (int16) ID code for the group of questions or lectures. If there are three lectures or questions in a row, then they will have the same `task_container_id`.
- `user_answer`: (int8) either 0, 1, 2, or 3 for answers, -1 for lectures.
- `answered_correctly`: (int8) 0 if students responded incorrectly, 1 if students responded correctly, and -1 if students watched lectures.
- `prior_question_elapsed_time`: (float32) The average time for students to answer each question in the previous bundle, null for the students' first question bundle or lecture.
- `prior_question_had_explanation`: (bool) Whether or not the user saw an explanation and the correct response(s) after answering the previous question bundle, ignoring any lectures in between. Typically the first several questions a user sees were part of an onboarding diagnostic test where they did not get any feedback.

`questions.csv`:

- `question_id`: foreign key for the train/test `content_id` column
- `bundle_id`: ID code that indicates which questions are grouped together.
- `correct_answer`: the correct answer to every question. This feature is used to assay the students' answer accuracy by comparing this with the `user_answer` feature from `train.csv`.
- `part`: the relevant section of the TOEIC test.
- `tags`: one or more detailed tag codes for the question. This is used to cluster the questions together.

`lectures.csv`:

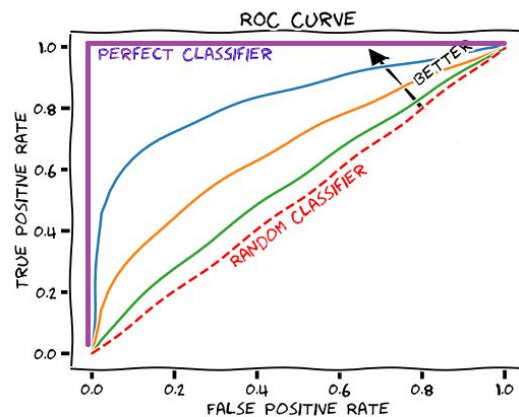
- `lecture_id`: foreign key for `content_id` feature in `train.csv` and test file
- `part`: category code for lectures
- `tag`: one tag code for the lecture. This is used to cluster the lectures together.
- `type_of`: a brief description of the core purpose of the lecture

`example_test_rows.csv`: this file is the same as `train.csv`, but with the two extra columns below. This file also has hidden users that are not presented in `train.csv`.

- `prior_group_responses` (string): provides all of the `user_answer` entries for the previous group in a string representation of a list in the first row of the group.
- `prior_group_answers_correct` (string): provides all the `answered_correctly` fields for the previous group, with the same format and caveats as `prior_group_responses`.

d. Competition evaluation metric: ROC curve

The evaluation metric for this competition is the area under the ROC (receiver operating characteristic) curve. The ROC curve illustrates the diagnostic ability of a binary classifier with its threshold. In the competition, we will use the history of user interaction to build a model to predict whether certain users will answer the next question correctly or not. Thus, the result of the prediction will be the possibility of getting the correct answer. As the threshold value varies, we will label the result as positive (p) or negative (n) and compare it with the ground truth given by the Kaggle system after the submission. Since there are two choice for prediction, then result will have four types of outcome, which are:



1. True positive: the truth is “positive” and we predict “positive” as well.
2. False positive: the truth is “negative” but we predict “positive”.
3. True negative: the truth is “negative” and we predict “negative” as well.
4. False negative: the truth is “positive” but we predict “negative”.

In order to draw the ROC curve, we only need the true positive rate (TPR) and false positive rate (FPR) and plot TPR against FPR with various threshold values.

In this competition, the leaderboard ranks the participants’ performance by the area under the ROC curve that their prediction produces after comparing with the ground truth. We can think of the area under the ROC curve (AUC) as the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one. If we compute the AUC, we get:

In the classification process, with the given threshold parameter T , which is decided at the best point where there are many true positives and less false positives., each interaction will be classified as true if the estimated possibility made from our prediction based on a continuous random variable, X , is greater than T ; otherwise, negative. Then

we can compute the probability density $f_1(x)$ if the user interaction belong to the “answer correctly” and $f_0(x)$ if otherwise. The true positive rate and false positive rate are given as following:

$$\begin{aligned} \text{TPR}(T) &= \int_T^\infty f_1(x) dx \\ \text{FPR}(T) &= \int_T^\infty f_0(x) dx. \end{aligned}$$

$$\text{TPR}(T) : T \rightarrow y(x)$$

$$\text{FPR}(T) : T \rightarrow x$$

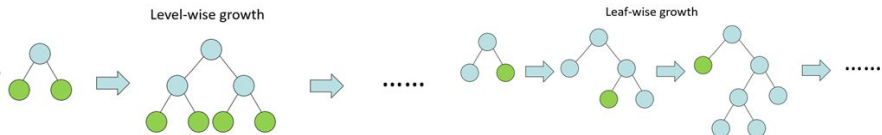
$$A = \int_{x=0}^1 \text{TPR}(\text{FPR}^{-1}(x)) dx = \int_{-\infty}^{\infty} \text{TPR}(T) \text{FPR}'(T) dT = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} I(T' > T) f_1(T') f_0(T) dT' dT = P(X_1 > X_0)$$

where X_1 is the score for a positive instance and X_0 is score for a negative instance and f_0 and f_1 are probability densities we have given above. The AUC score ranges between 0 and 1, as the TPR and FPR both range between 0 and 1.

2. Technologies

<p>a. Data Visualization</p>	<p>Data visualization enables data scientists to discover and explore any valuable trends or possible correlation between features and thus further understand the data before getting their hands dirty with the model building. Thus, the choices of tools to present the visualization need to best serve the purpose of information that we manage to convey. In the exploratory data analysis section, we employed two visualization tools commonly used in Python: <i>matplotlib</i> and <i>plotly</i>.</p> <p><i>Matplotlib</i> allows users for comprehensive customization and makes the report look more consistent with the style. On the other hand, <i>plotly</i> is also a power tool. One noticeable advantage that we had discovered of <i>plotly</i> over <i>matplotlib</i> is that <i>plotly</i> generally have more aesthetically pleasing and interactive plots.</p> <p>The reason we relied more on <i>matplotlib</i> was because it is a relatively quick and straightforward tool; <i>plotly</i> would definitely be a better choice if we move into a more sophisticated stage of our study.</p>
<p>b. Data Management</p>	<p>In our experimentation, we conducted the use of two essential libraries for computation, <i>NumPy</i> and <i>Pandas</i>. <i>NumPy</i> provides numbers of straightforward and optimized implementations in multi-dimensional array-oriented computation which will be involved in our feature engineering.</p>

	<p>Unlike <i>NumPy</i>, <i>Pandas</i> employs 2-d table objects called Dataframe with rows and columns. Thus, pandas enables more manipulations such as computation between columns, which was heavily used in this competition.</p>
c. Parameter Tuning - Optuna	<p>Optuna is an automatic hyperparameter optimization software framework. It is designed for machine learning to tune for the best parameters for a model with automation and acceleration. The framework has an <u>imperative</u>, <u>define-by-run</u> style user API as a feature.</p> <p>Imperative programming is a programming paradigm that uses statements that change a program's state and this programming paradigm often compares with Declarative programming which is a programming paradigm that expresses the logic of a computation without describing its control flow. If declarative programming describes that one person demands the other person to complete a task but does not care about the process as long as the task is completed, then imperative programming describes one person providing detailed instructions for completing the task for the other person.</p> <p>The define-by-run framework is often compared with the define-and-run framework when discussing deep learning frameworks. Basically, the difference between these two frameworks lies in the order of defining the network and mini-batch data. For define-and-run, the network is first defined and fixed and then mini-batch data comes into the network; whereas, for the define-by-run, the network is defined with the actual flow of computation in a more dynamic way and this enables conditionals and loops into the network in a more easy way.</p> <p>Because of the define-by-run API feature, Optuna provides code with high modularity and enables coders to create search spaces for the hyperparameters in a dynamic way.</p> <p>When implementing Optuna for hyperparameter tuning, we employed the terms study and trial. The purpose of constructing a study was to find the set of hyperparameters with the values that show the best performance in modeling, by running through numbers of trials.</p> <p>The <i>study</i> module implements Study objects and related functions. In our experiment, we use <code>suggest_int()</code> and</p>

	<p><code>suggest_uniform()</code> to set up a range of values for each parameter. Then we use the <code>create_study()</code> in the <i>study</i> module to create a new study of the data.</p> <p><code>Suggest_int(name, low, high)</code> => methods inside the <i>trial</i> module which suggests a value for the integer parameter, specified by <i>name</i>.</p> <p><code>Suggest_uniform(name, low, high)</code> => similar with <code>suggest_int()</code> and it suggests a value for the continuous parameter, specified by <i>name</i>.</p>																		
d. Models	<p><i>LightBGM</i> is a comparably new algorithm in the machine learning field. It is a gradient boosting framework that employs a tree based learning algorithm and performs powerfully in the computation of large datasets. LightGBM differentiates itself from other algorithms by its direction of tree growth; it grows leaf-wise (vertically) whereas others grow level-wise (horizontally).</p> <div></div> <p>Not only in this competition, also in the entire machine learning field, LightBGM is popular and it is because it spends less memory to deal with larger amounts of data and has a stronger computation power to produce the results faster. However, the high sensitivity causes LightBGM to easily overfit. We circumvented this situation since we train on a large dataset and also made use of the parameters that mainly aim for overfitting prevention.</p> <p><i>XGboost</i> is another gradient boosting framework that always gets compared to LightBGM. They both work well towards building predictive models from structured data as for this competition. Although XGboost is the baseline of LightBGM and LightBGM is designed to perform better in training speed and the size of the dataset handleable, it may still be worth a try in our experimentation.</p> <table><tr><th>Case</th><th>XGBoost (GBT)</th><th>TensorFlow (NN)</th></tr><tr><td>Automatic feature extraction from images, sequences and text</td><td>X</td><td>X</td></tr><tr><td>Run efficiently on CPU</td><td>✓</td><td>X</td></tr><tr><td>Large scale training</td><td>X✓</td><td>✓</td></tr><tr><td>Missing values handling</td><td>✓</td><td>X</td></tr><tr><td>Transform any inputs shape to any output shape</td><td>X</td><td>✓</td></tr></table> <p>In the early stage of model picking, we noticed that Gradient Boosting Decision Trees (GBDT) are often compared with Neural Networks. Although, many shows that GBDT generally are more</p>	Case	XGBoost (GBT)	TensorFlow (NN)	Automatic feature extraction from images, sequences and text	X	X	Run efficiently on CPU	✓	X	Large scale training	X✓	✓	Missing values handling	✓	X	Transform any inputs shape to any output shape	X	✓
Case	XGBoost (GBT)	TensorFlow (NN)																	
Automatic feature extraction from images, sequences and text	X	X																	
Run efficiently on CPU	✓	X																	
Large scale training	X✓	✓																	
Missing values handling	✓	X																	
Transform any inputs shape to any output shape	X	✓																	

preferably in running time and accuracy. We still wanted to use this algorithm as a different perspective in deep learning and a chance for us to learn new things.

Keras is one of the leading high-level neural networks frameworks in Python. Since Keras is designed to be user-friendly, impressive modularity, extensibility and compatibility with Python, we took advantage of them and employed the framework into our experimentation.



3. Exploratory Data Analysis (EDA)

It is better to understand the data before going into the coding process without any insights. In data science, we refer to this pre-coding investigation as exploratory data analysis (EDA). The purpose of this initial exploration of data to observe any existing patterns, to detect any anomalies and to discover any correlations between or within features by plotting figures or statistics. Note that the EDA was done based on the entire train dataset.

I. `Train.csv`

a. Overview

We examined each dataset and possible methods for data preprocessing and feature engineering.

`train.csv`:

	row_id	timestamp	user_id	content_id	content_type_id	task_container_id	user_answer	answered_correctly	prior_question_elapsed_time	prior_question_had_explanation
0	0	0	115	5692	0	1	3	1	NaN	<NA>
1	1	56943	115	5716	0	2	2	1	37000.0	False
2	2	118363	115	128	0	0	0	1	55000.0	False
3	3	131167	115	7860	0	3	0	1	19000.0	False
4	4	137965	115	7922	0	4	1	1	11000.0	False

`train.csv` size: (101230332, 10)

As mentioned before, in the `train.csv` dataset, each row represents one interaction between a user with the EdNet system. With 10 features, we were able to define which user at what time has done what action with what result.

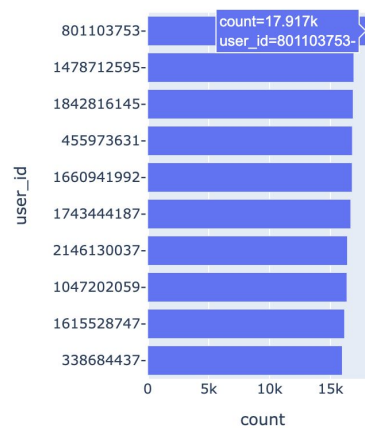
The train.csv is large: 5.45G. Simply reading data with `pd.read_csv` will take large portions of the RAM. Additionally, with further manipulation of data, the kernel will terminate due to an out-of-memory error on Kaggle Notebooks. Thus, in data preprocessing, we needed to find a way to reduce the memory usage in data reading. There are a total of 101,230,333 rows with 10 features in the `train.csv` dataset. From these 10 features, more features can be engineered from potential correlations and clusterings to train models and predict test data.

b. Users interaction

There are 393656 unique users who have made 101230332 interactions. Thus, on average the interactions of a user should be 257.15. However, if we count the number of interactions for each user, we get:

user_id	Interactions
801103753	17917
1478712595	16914
1842816145	16851
455973631	16789
1660941992	16777
...	...
1032404821	1
1071441751	1
121434886	1
505065481	1
607601423	1

Top 10 users by number of interactions

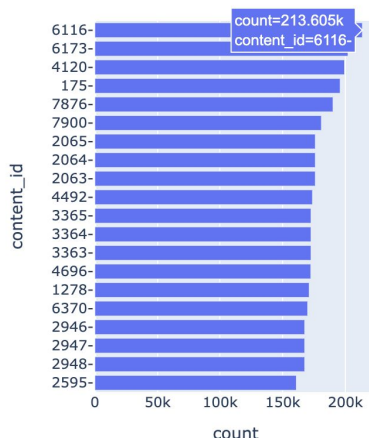


This means every user has interacted with the system, but some users have way more interactions than others. After some calculations, we found that 87 users who only have one interaction, and 320516 users have less than the average number of interactions: 258. The number of interactions per user is highly skewed: 60% users have less than 63 interactions. Thus, we may want to eliminate those users with significant limited interactions and focus on the ones with more dynamics in order to better observe their overall performance.

c. Top Content ID

The top 20 most useful `content_id` suggests certain questions of certain contents are more likely to be given to the users.

Top 20 most useful content_id



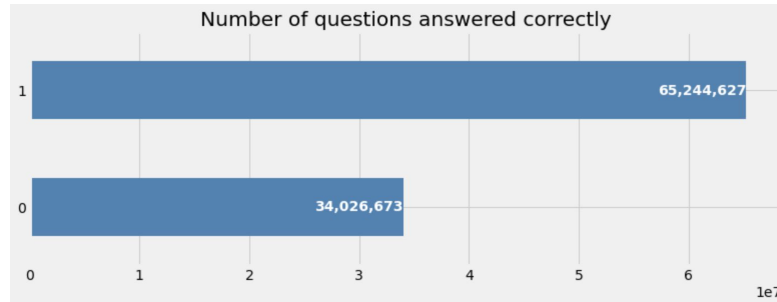
d. User answer counts

user_answer	counts
0	28186489
1	26990007
3	26084784
2	18010020
-1	1959032

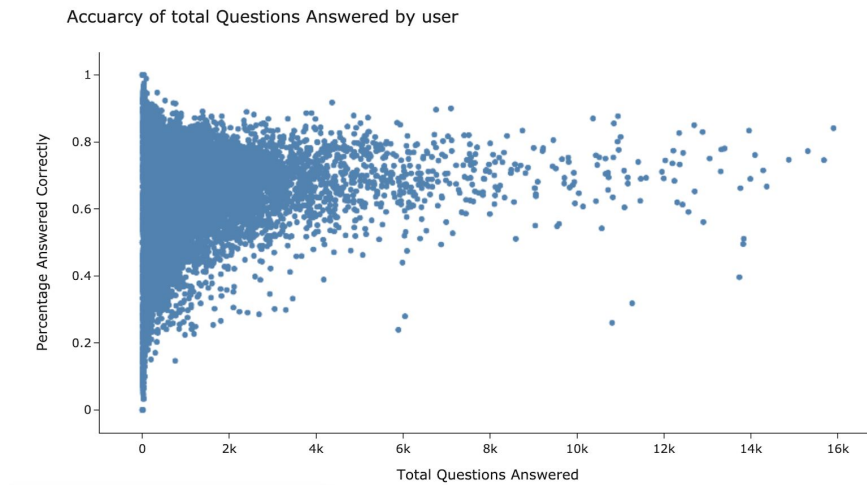
The questions in EdNet are multiple choices with four options: 0, 1, 2, and 3. -1 means the user is watching the lecture. The counts for each answer option are fairly even. It is unlikely that correlation between answer options and accuracy will be found, therefore, user answer options and counts are not the best features to predict accuracy.

e. Question answered accuracy

The goal for this competition is to preprocess the data and create features to models to predict the possibility of accuracy, so the relation between `answer_correctly` and other features should be closely examined. Among all the interactions made by answering questions, 65.73% of the questions were answered correctly by the users.

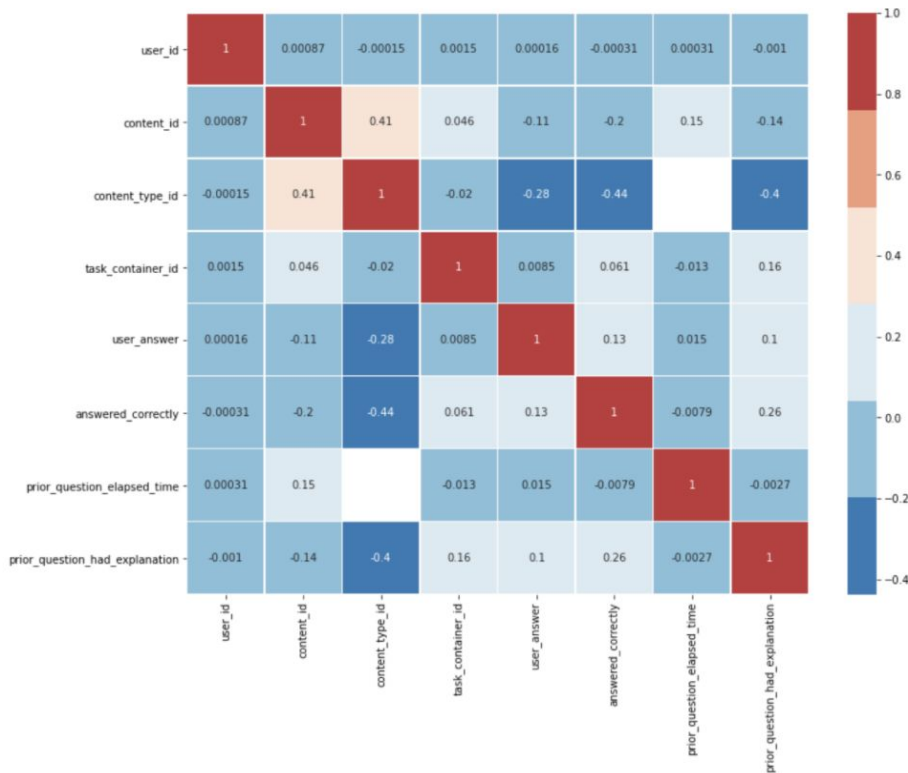
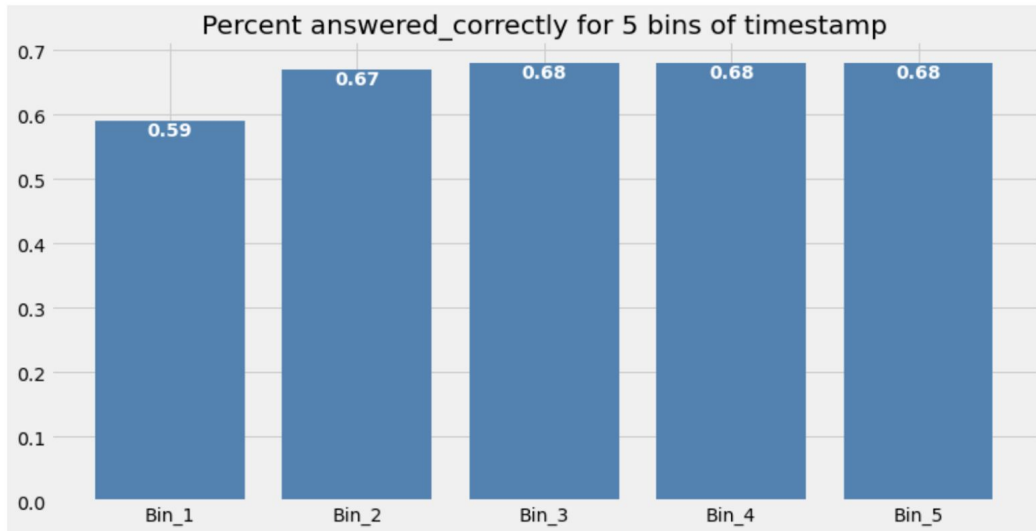


If we plot the percentage of questions answered correctly against the total number of questions, we discover that as more questions are answered the average performance becomes better (around 0.7 to 0.8) and the variance of the average score decreases. This finding suggests that accuracy is correlated with the total question answered, i.e. the number of interactions per users.



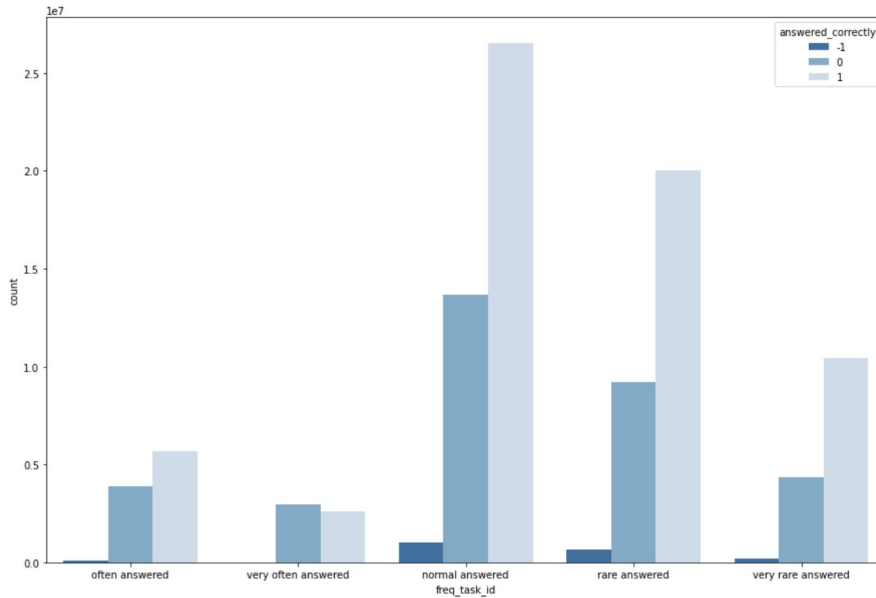
f. Accuracy and timestamp

Accuracy also correlates with timestamp: the time between users' first event completion and following interactions. Splitting the timestamp into 5 bins in ascending order and for each timestamp range compute the average percentage of questions answered correctly, we find that the users who just register recently perform a little worse than the users who have been active for longer. This may suggest that EdNet does help improve user performance in the TOEIC test.



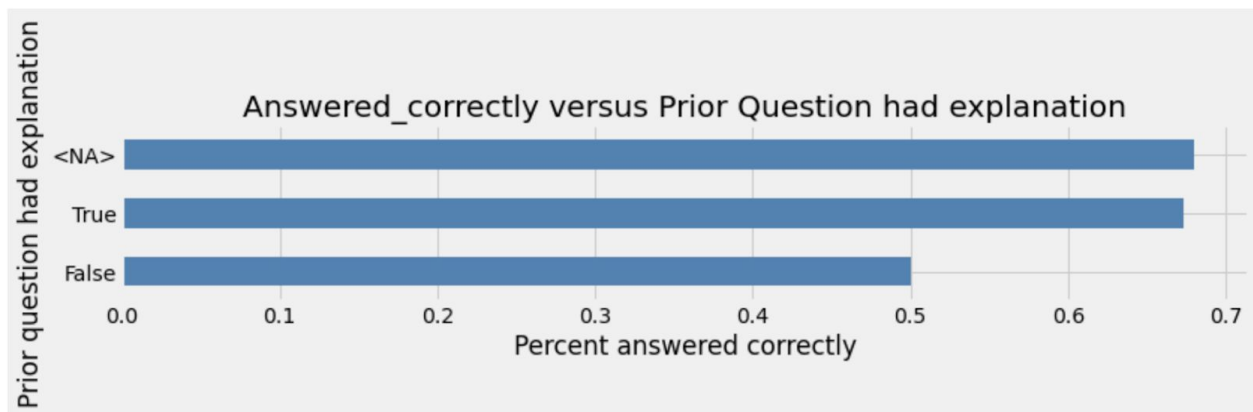
Heatmap for each feature in `train.csv` is plotted to see the correlation between feature and accuracy (anwser_correctly). The heatmap suggests that `task_container_id` and `prior_question_had_explanation` carry higher significance in terms of accuracy.

g. Accuracy and task container ID



If we categorize the task container id by its frequency that it gets presented to the users and plot it against the number of correct and wrong answers, also lectures watched, we see that the relatively less often tested task container ids actually have more questions and the disparity between the number of right and wrong responses are larger.

h. Accuracy and explanation on prior question



For `prior_question_had_explanation`, when we plot the percent of questions answered correctly with whether the user sees an explanation and the correct response(s) after answering the previous question bundle, we see the percentage of accuracy is about 17% higher when the user sees an explanation. Moreover, what is interesting is that the missing values have a similar percent of accuracy with the True rather than with the False. From the introduction, we know the missing values appear because typically the first several questions a user sees are part of an onboarding diagnostic test, therefore, they did not get any feedback.

II. Question.csv and Lecture.csv

a. Overview

Question.csv

	question_id	bundle_id	correct_answer	part	tags
0	0	0	0	1	51 131 162 38
1	1	1	1	1	131 36 81
2	2	2	0	1	131 101 162 92
3	3	3	0	1	131 149 162 29
4	4	4	3	1	131 5 162 38
...
13518	13518	13518	3	5	14
13519	13519	13519	3	5	8
13520	13520	13520	2	5	73
13521	13521	13521	0	5	125
13522	13522	13522	3	5	55

Question size: (13523, 5)

Lecture.csv

	lecture_id	tag	part	type_of
0	89	159	5	concept
1	100	70	1	concept
2	185	45	6	concept
3	192	79	5	solving question
4	317	156	5	solving question
...
413	32535	8	5	solving question
414	32570	113	3	solving question
415	32604	24	6	concept
416	32625	142	2	concept
417	32736	82	3	concept

Lecture size: (418, 4)

b. Question ID and lecture ID

`Content_id` is the union of `question_id` and `lecture_id` and can be distinguished in the company of `content_type_id`. Thus, `content_id` is the foreign key and can be used to join train datasets with questions and lecture datasets for any further feature engineering.

c. Weights of questions compared to lectures

content_type_id	row_id
	count
0	19610123
1	389877

Majority of the interactions are questions answered and less than 2% of interactions are from watching lectures. Thus, although we will do analysis for all datasets, we choose to focus mainly on `train.csv` and `question.csv` in the model and leave the lecture portion for future investigation and improvement if time allows.

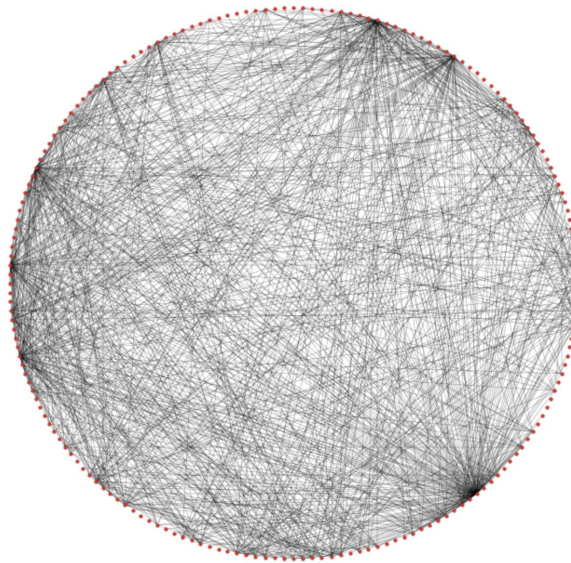
d. Tags

Since tag feature was actively discussed in class, we should definitely take a closer look into it. Each question may have one or more tags associated with it representing certain skills; in other words, it is not a singular value. Thus, intuitively, we want to label encoding each combination with a numeric presentation. We converted `'tags'` from object dtype to category using `.astype('category')` and then did label encoding on it using `.cat.codes`.

	content_id	part	tags	tags_label	mean_content_accuracy
0	0	1	51 131 162 38	981	0.907721
1	1	1	131 36 81	307	0.890646
2	2	1	131 101 162 92	251	0.554281
3	3	1	131 149 162 29	287	0.779437
4	4	1	131 5 162 38	317	0.613215

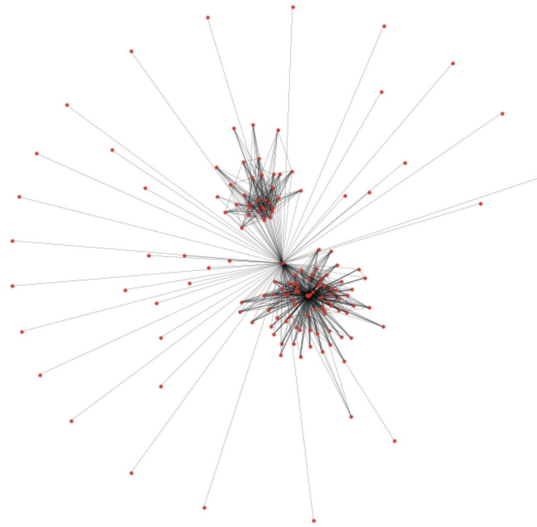
On the other hand, an alternative hypothesis we can make is that some tags are more likely to be grouped together with each other. We can cluster them into different groups with labels and use labels as a new feature to replace 'tag' for model building. We will encode the tags by dividing them into different communities based on their clusters.

To start off, we built a graph using the Networkx module. Each tag is associated with a node in the graph and the edges between nodes indicates tags occurring together. The weight on each edge is the number of questions in which two adjacent tags occurred together.

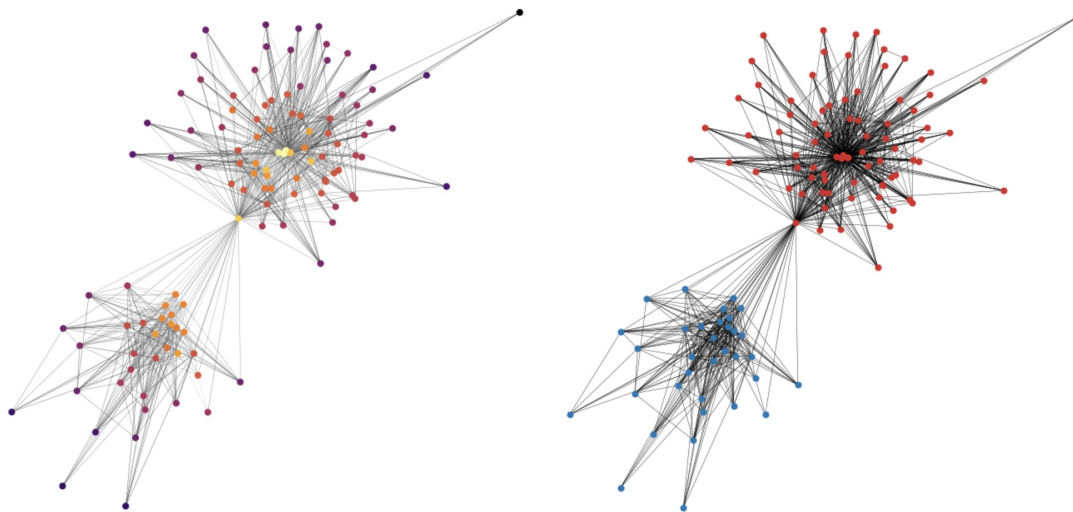


From the graph, we are able to see that some nodes have a denser end meaning tags associated with them appear more often than others. On the other hand, if we take a closer look at the graph, some nodes have no edge attached to them; in other words, some tags only appear by themselves in the questions.

Therefore, we take out all the tags that always appear by themselves and collect them in a list. Then, we continue to observe the clustering patterns with the remaining tags by plotting the graph again.



With the second graph, we can see that there are some nodes only connected to the node in the middle. There are the tags that either appear by themselves or pair with the tag that is represented by the node in the middle which we find out to be tag 162. Then, we list these tags with a single connection out and count how many times they pair with tag 162. With the remaining nodes which have more intrinsic pairing in between, we plot them out again (graph on the left).



We discover that there are two obvious clusterings. To prove our observation is correct, we run a community finding algorithm to separate the nodes into two clusters and color the nodes with red and blue for visualization. The two clusterings indeed exist.

So far, we have already categorized all the tags into 4 communities:

- A list with all tags that only appear by themselves (community 0)
- A list with all tags that only appear by themselves or pair with tag 162 (community 1)
- Two well-separated clusters of all remaining tags (communities 2 and 3)

We would further generate a new column based on this categorization later in our feature engineering.

	question_id	bundle_id	correct_answer	part	tags
10033	10033	10033	2	6	NaN

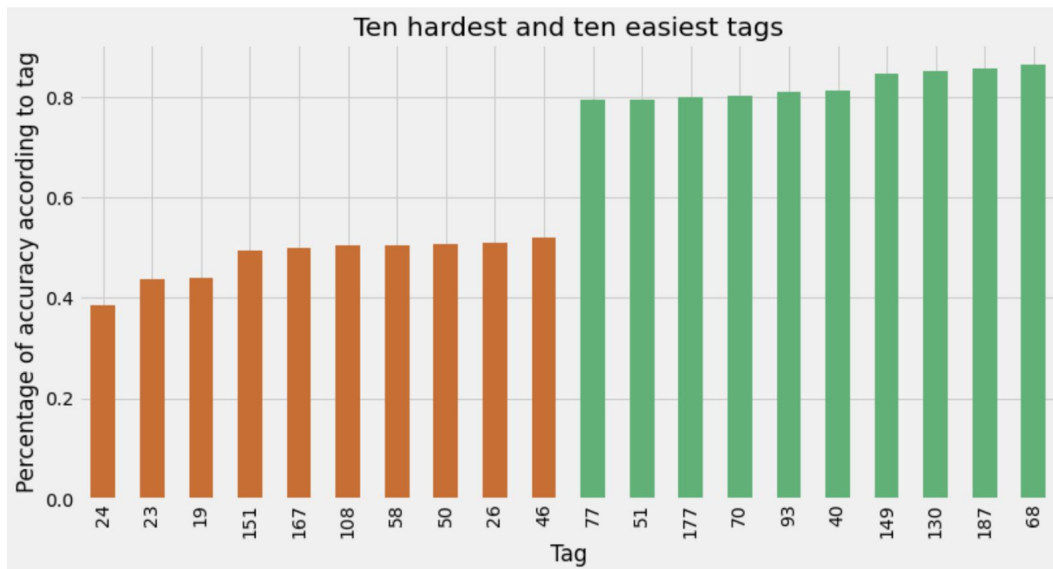
	row_id	timestamp	user_id	content_id	content_type_id	task_container_id	user_answer	answered_correctly
62750278	62750278	1286282597	1333688829	10033	False	1127	2	1

Furthermore, we realize that there is one `question_id` that does not have a tag associated with it and in the `train.csv`, this question has only appeared once. This suggests that if we use tags as a feature for model building, we need to impute the none value with a presentation or just eliminate this question from `question.csv` and the interaction related to it from `train.csv`.

	question_id	bundle_id	correct_answer	part	tags	Wrong	Right
0	0	0	0	1	[51, 131, 162, 38]	637	6266
1	1	1	1	1	[131, 36, 81]	809	6589
2	2	2	0	1	[131, 101, 162, 92]	20015	24890
3	3	3	0	1	[131, 149, 162, 29]	5067	17906
4	4	4	3	1	[131, 5, 162, 38]	12275	19461

If we calculate the number of questions answered correctly or not by the `question_id` and accumulate the number by tags, we are able to obtain the question tags that are the easiest and hardest to the vast users. Even though the number of right or wrong answers for each tag may be double counting, this does not influence us to get a sense of the difficulty of the tags and how it has an impact on the user's accuracy.

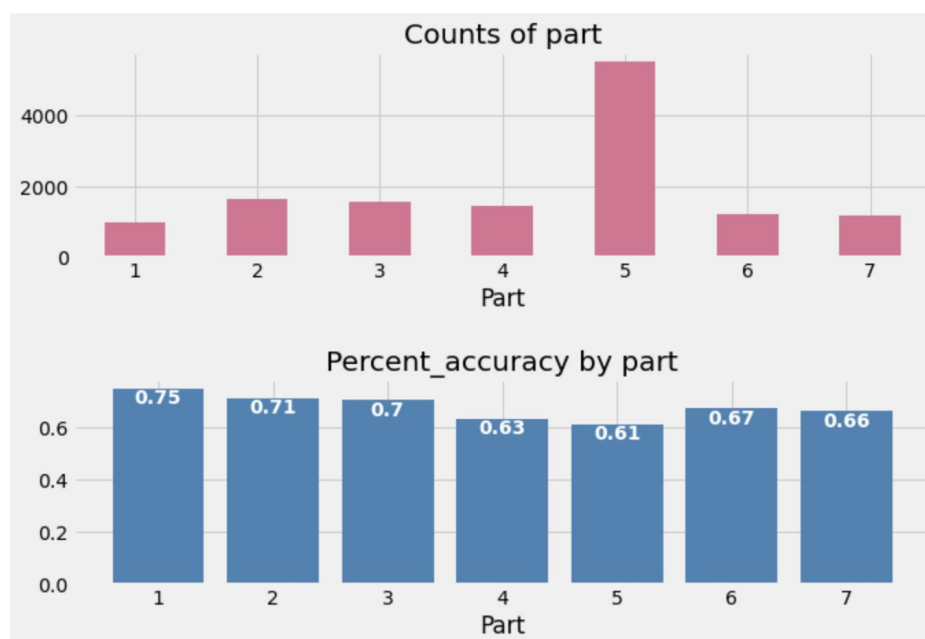
	Wrong	Right	Total_questions	Question_ids_with_tag	Percent_correct
tag					
24	157631	98982	256613	17	0.385725
23	261611	204293	465904	11	0.438487
19	173602	136367	309969	57	0.439938
151	272267	264913	537180	16	0.493155
167	170386	170681	341067	11	0.500432



Additionally, `tag` points to its associated lecture, which may help us understand the improvement on user's performance overtime by watching which kind of lecture.

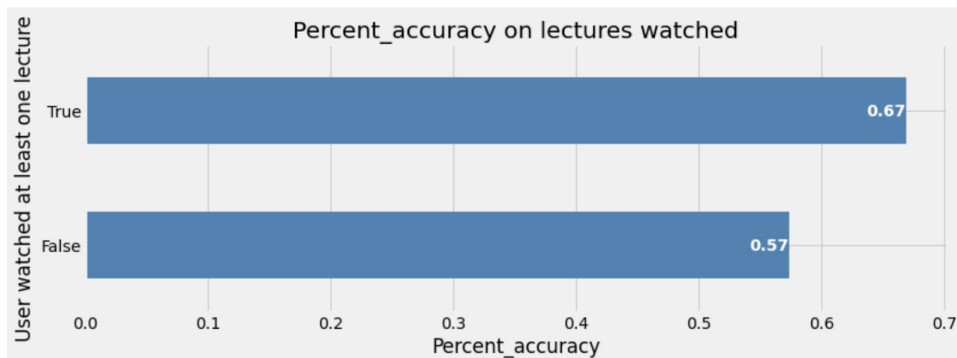
e. Parts

There are 7 distinct parts that relate to the TOEIC test content. In the TOEIC test, there are two sections, listening and reading. The listening section consists of Part 1-4 which provides approximately 45 minutes to finish 100 questions and the reading section consists of Part 5-7 and provides 75 minutes to finish 100 questions.



If we plot the number of `question_id` in each part and the average percent of accuracy of questions answered from each unit, as shown in the diagram, part 5 contains the most questions which are also the most difficult ones.

f. Effect of watching lecture on Accuracy



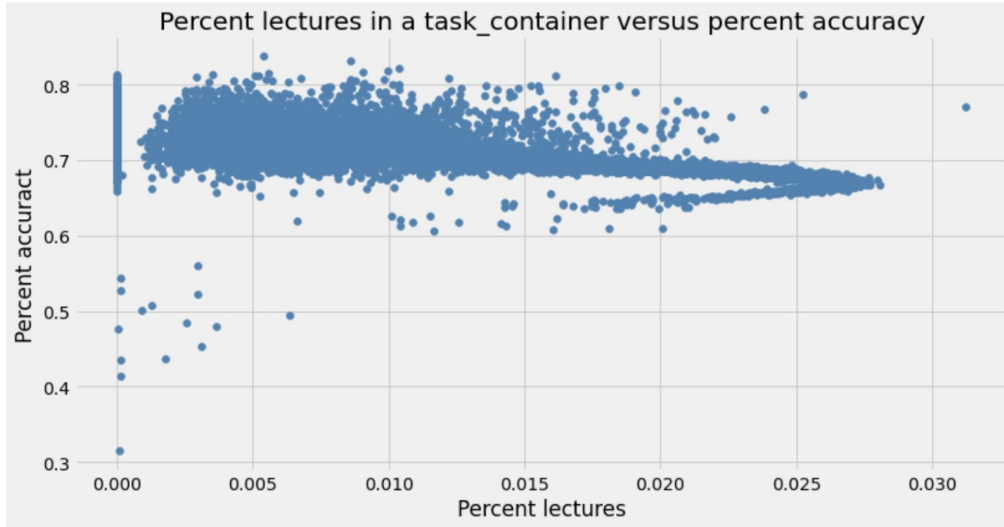
The plot of average percent accuracy of the users who watch at least one lecture or not at all shows that watching lectures does help improve the performance.

g. `Task_container_id` and `lecture_id`

Recall that the `task_container_id` has correlation with `answer_correctly`, thus we also want to examine the correlation between `task_container_id` with `lecture.csv`. Since each `task_contain_id` shares a batch of interactions including questions and lectures, then we count the number of lectures and right or wrong responses in each `task_container_id`, and the percentage of lectures and accuracy of answers.

	Lecture	Wrong	Right	Percent_correct	Percent_lecture
task_container_id					
9096	8	57	191	0.770161	0.031250
270	2265	26105	52352	0.667270	0.028059
477	1428	16166	33427	0.674027	0.027988
253	2343	27380	54798	0.666821	0.027721
351	1835	21225	43158	0.670332	0.027711

It seems that lectures do not take up a significant portion in each `task_container_id`. Just in case, we will check if there is any correlation by plotting percent accuracy by percent lecture. As shown in the diagram below, there is no relative correlation in between.



Before heading to preprocessing and feature engineering, we want to take a quick look at the `example_test_rows.csv`. `example_test_rows.csv` has the same format with the `train.csv`. However, our model needs to consider that the test data has new users who just registered for EdNet. Among 42 users in `example_test_rows.csv`, one user who has not appeared in our history of interactions: `user_id: 275030867`

4. Experimentation

This section includes different data preprocessing methods and machine learning techniques we experimented with before building the final model. To evaluate the efficacies and accuracies among our data preprocessing methods and feature engineering processes, we will test them and compare the results using our main prediction method with LightGBM using the `answered_correctly` column in the dataset. The steps of constructing the prediction models and determining the hyper-parameters for LightGBM will be discussed in detail later in the report. We consider the `roc_auc_score` provided in the training of the LightGBM models and the `roc_auc_score` (Area Under the Receiver Operating Characteristic Curve) provided by Kaggle in the submission result in the comparison which were introduced earlier.

a. Data Preprocessing:

These were several challenges we encountered during data preprocessing experimentation:

1. Limited memory space on Kaggle and large training dataset
2. Corrupted data: Null, NaN, and missing values in features
3. Categorical values and encoding
4. New users in the test dataset

A. Ways to combat limited memory issue

I. Exploring the best method to read large data efficiently

For better performance, we need to consider not only the accuracy of our model, but also the run time and memory usage. Starting from reading data, there are multiple suggestions regarding how to load all of train.csv into a kaggle notebook:

- Pandas
- Dask
- Datatable
- RAPIDS cuDF - GPU DataFrames
- BigQuery

Since we are familiar with Pandas and it is also the default method on Kaggle, we decided to use Pandas. The strength of Pandas is it has many options and functions for reading and processing data. The challenge of using Pandas is that it requires a lot more RAM for the pandas dataframe. Pandas is also the method used in the official starter notebook of the Riiid competition.

Apart from methods of reading data from the raw csv files, it is also common to convert the dataset into another format which uses less disk space and be proceeded in faster speed for subsequent reads:

- csv
- feather
- hdf5
- jay
- parquet
- Pickle

We chose 'pickle' to be our dataset format because Pandas has inbuilt functions to read and write dataframes as pickle objects. The public notebook from Vopani had converted the dataset into multiple formats and we used his file "riiid_train.pkl.gzip" (3.02GB) as one of our inputs.

II. Manipulation on dataset to reduce memory space

Since we have a dataset with more than 10,000,000 rows, we checked the memory usage to see how all the features take up the memory space:

```
In [6]: train2.memory_usage(deep=True)

Out[6]: Index          794170400
row_id          794170400
timestamp       794170400
user_id         794170400
content_id      794170400
content_type_id 794170400
task_container_id 794170400
user_answer     794170400
answered_correctly 794170400
prior_question_elapsed_time 794170400
prior_question_had_explanation 3535423840
dtype: int64
```

As in the picture, `prior_question_had_explanation` features take up the most memory space as it is in 'object' type. Kaggle has a very limited amount of RAM to run its notebook, therefore, we tried to reduce the memory usage of our data as much as possible by converting it to boolean type and the memory usage did reduce significantly.

```
train2['prior_question_had_explanation'] = train2['prior_question_had_explanation'].astype('boolean')
```

We can free the memory a bit by fitting some columns into smaller types

```
np.int8      1.270000e+02
np.int16     3.276700e+04
np.int32     2.147484e+09
np.int64     9.223372e+18
np.float16   6.550400e+04
np.float32   3.402823e+38
np.float64   1.797693e+308
Name: max value, dtype: float64
```

The chart above tells us the maximum value representable by each of the common numeric types. Based on this, we converted our features with small values and short length from type `np.float64` into `np.float8` such as '`content_type_id`', '`user_answer`', '`answered_correctly`', etc. The medium-length numeric values like '`user_id`', '`content_id`', etc. were rearranged to `np.float16`. For '`prior_question_elapsed_time`', fit to `np.float32` type.

This single reduction based on type conversion was not enough to load & run code on Kaggle notebook. Hence, we implemented multiple approaches to reduce the memory usage:

- a. To reduce the user with the least interaction and outliers:
 - Prior, we calculated the interactive frequency of each user in the training dataset, which showed that 87 users have only one interaction (one row),

and the interaction per user distribution is highly left skewed (60% users have less than 63 interactions, compared to average interaction of 258).

- To remove users with limited interactions, we looked at the distribution for the number of interactions for all users. At 15% quantile, thousands of users have less than 21 interactions, which in total, constitute about 1 million rows of data. To experiment with the accuracy from this outlier elimination, the users below 15% quantile were removed from the `train.csv` dataset.

A Comparison table for outliers removal and non-removal:

Outliers/Anomalies Removal	Outliers/Anomalies Non-Removal
<pre> user_id freq 0 801103753 17917.0 1 1478712595 16914.0 2 1842816145 16851.0 3 455973631 16789.0 4 1660941992 16777.0 332649 856146027 21.0 332650 1361528377 21.0 332651 1322699682 21.0 332652 2056009358 21.0 332653 1952994264 21.0 [332654 rows x 2 columns]</pre>	<pre> Total unique users are 393656 However the length of user_id is 101230332 On average the interactions of a user should be 257.1542971528441 801103753 17917 1478712595 16914 1842816145 16851 455973631 16789 1660941992 16777 ... 1032404821 1 1071441751 1 121434886 1 505065481 1 607601423 1 Name: user_id, Length: 393656, dtype: int64</pre>
<p>The removal of outliers/anomalies (least interactive users) results in the Kaggle's <code>roc_auc_score</code> of 0.759, which is a pretty good score. The difference between removal and non-removal method is not significant.</p>	<p>The non-removal of outliers/anomalies (least interactive users) gives a Kaggle's <code>roc_auc_score</code> of 0.760 - an increase of 0.001 compared to the removal method and the score is also the highest one we could obtain overall.</p> <p>There are two possibilities that explain this increase in accuracy:</p> <ol style="list-style-type: none"> 1. The outliers, which is a group of students with little interaction plays as a fairly important information in the training model 2. LightGBM is a respectable machine learning framework, it can handle pretty

	well the anomalies/outliers itself.
--	-------------------------------------

b. In the model, we took about 50% of the `train.csv` to be the training dataset:

- The train data was taken from the last 400 users from the `train.csv`. Those 400 users constitute about 50% of the `train.csv`, suggesting that each user has high interaction history volume. This approach was simple, yet effective, as more meaningful correlations and patterns with regard to accuracy could be derived from users who spend more time and practice more questions. However, the cons is obvious: within the 400 users, none has small interaction volume. Therefore, it is likely that our model does not perform well for predicting users with low interaction volume.
- Since removing the rows on the dataset leads to the loss of information on our users, we only kept the full dataset to build the users and questions dictionary. However, in order to improve on the performance of the training process when building the model, we implemented it with the 50-million training dataset as it is a sufficient number to build a valid model for prediction.

c. Convert the pandas dataframe to numpy array to save memory space (optional):

- We were informed that the Numpy package in python consumes less memory compared to Pandas. Thus, we decided to convert our Pandas dataframes to Numpy arrays.
- However, there is also a disadvantage of Numpy is that it generally performs better in terms of timing than Pandas only when we have a set of 50,000 rows or less. In our case, we have around 50 millions rows, therefore, Pandas would perform better than Numpy.
- We tried two different ways in this step, one using Pandas and the other using Numpy and obtained the result:

	Numpy Arrays	Pandas Dataframe
Time on each step	<pre>time on each step: 0.06312990188598633 0.03677535057067871 0.03565788269042969 0.043634891510009766</pre>	<pre>time on each step: 0.2528524398803711 0.13791513442993164 0.1341538429260254 0.0998077392578125</pre>
Time for training model	<pre>CPU times: user 9min 42s, sys: Wall time: 3min 14s</pre>	<pre>CPU times: user 46min 13s, sys: Wall time: 12min 16s</pre>

Notebook runtime	Run Time 461.8 seconds Timeout Exceeded False Output Size 2.25 KB Accelerator None	Run Time 1172.8 seconds Timeout Exceeded False Output Size 939 KB Accelerator None
Memory usage	Disk 44.1 MB Max 19.6GB RAM 184.7 MB Max 16GB	Disk 44.1 MB Max 19.6GB RAM 2.3 GB Max 16GB
Submission time	1-2 hours	5-6 hours

- As we could observe from the table, even though the running time of inference using Numpy arrays is faster than Pandas dataframe, when it comes to submission, numpy arrays notebook acquired the public score on Kaggle around 3 times faster than the pandas notebook.
- The Numpy approach's memory usage is roughly 8% of the Pandas's approach (184.7MB vs 2.3GB in RAM). Notice that the output size does not reflect correctly the memory usage of each package. In the Pandas approach, the output includes the model and submission.csv file while the Numpy only contains the submission file so Pandas output size is heavier.

B. Handling null/NaN/unidentified values:

In the `train.csv` dataset, we detected two features that contain the null values: `prior_question_elapsed_time` and `prior_question_had_explanation`. There are 392,506 null values in the total of approximately 101 millions values. Since we did not include the `prior_question_elapsed_time` feature in the model training, we could choose to not handle its null value. However, we tried two approaches to handle null values: ignore it with null values or replace null values with the mean of elapsed time from all the users. For the `prior_question_had_explanation` feature, after converting it to the boolean form, we filled all the null values with False.

Another option is to use LightGBM for its capability to handle null values. LightGBM has an option that it will ignore the null values. However, one thing to take caution is that LightGBM cannot ignore the zero values automatically but we need to set up an *option zero_as_missing* to be True if we would want to do so. (Reference: <http://mlexplained.com/2018/01/05/lightgbm-and-xgboost-explained/>).

C. Handling categorical values:

In `train.csv`, there is no categorical feature so we did not perform any encoding process on this set.

In `questions.csv`, we considered the 'tags' feature as a non-numerical feature. Hence, we tried out two different ways for encoding it:

1. Cat.codes encoding:

As discussed in EDA, label encoding 'tags' resulted in 1520 unique numerical values (NaN = -1) corresponding to each group of tags and easily fit in an `uint16-dtype` new 'tags_label' feature in the training dataset. However, using this amount of unique values can make our training model become overfit, so we did not use them for the training process and moved to another method.

2. Clustering Encoding

Since we find out the label encoding results in overfitting, we then turn to our alternative hypothesis made in EDA, finding any possible clustering of certain tags. We have already categorized all the tags into 4 communities:

- Community 0: A list with all tags that only appear by themselves
- Community 1: A list with all tags that only appear by themselves or pair with tag 162
- Community 2 and 3: Two well-separated clusters of all remaining tags

Using the category mechanism, we create a new column, "community", in the `question.csv` to indicate which community each question belongs to by looking at which community most of the tags in that question belong to. If we check the number of questions in each community now, we discover that there is one question that is placed in community 4. Recall in EDA, we find out there is one question without a tag and that question has only been asked once in the `train.csv`. If we want to use the "community" as a model training feature, in order to prevent confusion, we should eliminate this question and also its related interaction in the `train.csv`. However, after doing that, we found our `roc_auc_score` before submission decreased from 7.299 to 7.194. Thus, we decided to leave the question and the question related to it.

In `lectures.csv`, the feature 'type_of' is in non-numerical form, so we tried to encode it into a numerical one. We applied the same method of encoding as in 'tags' feature to this feature and resulted in 4 unique numerical values. Even though this amount of unique values does not lead to the overfitting problem in our model, we did not use 'type_of' as the lectures are not our main focus in this trial. We only concentrated on the questions for our `content_id` and `tag`.

D. Handling user_id in test dataset but not exist in training dataset:

Prior, when we looked up the `example_test.csv`, there is one `user_id` that does not exist in the `train.csv` dataset. We had two approaches to solve this problem. The first one is to fill in the features that are needed for model training with 0 for numerical values and False for boolean data type. The other approach is to find the overall mean of that feature in the training dataset and fill it in. We will discuss in detail about these approaches in all three feature-engineering methods and how they affect the results in the next part of our report.

b. Feature Engineering

Due to a large amount of rows in the dataset, we reduced it from 10,000,000 rows to less than a half, around 4,400,000 rows and dropped unnecessary features. Our remaining features are `'row_id'`, `'user_id'`, `'content_id'`, `'content_type_id'`, `'answered_correctly'`. By doing so, we could leave memory space for training our model without affecting the outcome heavily. From the EDA part, we found these features are correlated and important ones so we mostly stayed focused on them.

We divided the full 11-million `train.csv` to a 4.4-million-row `train.csv` and four-hundred-thousand-row `test.csv` for model training. The removal of rows leads to the loss of user's information and affects the accuracy of our model. We have discussed this issue and had different approaches on how to handle the user information in the testing dataset but not in our training dataset in the “*Handling user_id in test dataset but not exist in training dataset*” section of data-preprocessing part of this report.

We only engineered the rows with `'content_type_id'` equal to 0, which means a question-type row. We chose to not focus on information from the lecture dataset and its rows in the training set because the lectures dataset only takes up around 2% of the training dataset. We wanted to predict the accuracy of a student's answer to a certain question. Hence, our approach analyzed mostly the 'question' rows in the training dataset.

We have 3 different ways for feature engineering:

- a. Basic Statistics Feature Engineering
- b. Harmonic Mean Feature Engineering
- c. User's Performance Feature Engineering

In order to evaluate the efficacy of each feature engineering process, we tested each of them and compared the results using our main prediction `roc_auc_score` with LightGBM predicting `'answered_correctly'` feature. The steps of 3 feature engineering methods are discussed in details below:

1. Overview for 3 feature engineering methods:

Basic Statistics	In this method, we created a dictionary for users and questions. In each dictionary, we calculated the four common statistical values -
------------------	---

	<p>mean, median, standard deviation and skewness of each user and question id.</p> <p>We then merged the user and question dictionaries to our training dataset and chose them as features to train the prediction model.</p> <p>We also calculated these four statistical values in the testing dataset and used them for our model input and received the prediction on the target columns.</p>
Harmonic Mean	<p>In this method, we created a dictionary for users and questions. In each dictionary, we calculated the mean accuracy of each user and question id.</p> <p>The difference of this method compared to the others is the new variables which represent the correlation of user and question: 'attempt' and 'harmonic mean'. The 'attempt' is the number of times one user has answered a question. Meanwhile, the harmonic mean is between one user's accuracy and a question's accuracy.</p> <p>We also calculated these statistical variables in the testing dataset and used them for our model input and received the prediction on the target columns.</p>
User's Performance	<p>In this method, we created a dictionary for users and questions. In the questions dictionary, we computed the total answers toward that question id, the number of correct answers, and the average accuracy of it. In the users dictionary, we calculated the total questions that the user has answered, the total correct answers of a user, and the average accuracy of each user.</p> <p>Note that the average accuracy in this method is not computed by .mean() function but by the division of cumulative sum by cumulative count. Hence, our average accuracy in this method for a user or a question is time-based and it is fairly different from the mean accuracy of two other methods. The most important variable of this method is the user's 'performance'. When a user U meets a question Q, he/she has a calculated average of how accurate he/she had done in previous questions, and the question Q also has an average accuracy of how other users performed on it. We compared these two averages together by subtracting them and got the 'performance' feature. The positive sign shows that the user has a higher accuracy score than the question's accuracy and the negative sign presents otherwise.</p> <p>Additionally, we considered the question 'tags' as our training feature. We group each question into a 'community' corresponding to their correlation with each other. Questions with highly equivalent tags will be placed within a community.</p> <p>We also calculated these variables in the testing dataset and used them for our model input and received the prediction.</p>

2. Statistics of 3 feature-engineering methods:
 - a. Variables & Calculations:

Mean_content_accuracy: the average of correct answers toward each question
Question_asked: the number of answers toward each question
Answered_correctly_question: the number of correct answers toward each question
Mean_user_accuracy: the average of correct answers of a user
Question_answered: the number of questions each user had answered
Answered_correctly_user: the number of correct answers of each user
Performance: the average accuracy of a user compared to the average accuracy of all the questions
Std_user_accuracy: how far the accuracy of a user deviates from the mean of all users
Median_user_accuracy: the middle point of the accuracy score's range of all users
Attempt: the number of time that user had answered a question
Hmean_user_content_accuracy: the performance of a user compared with other users on a question

In three different methods of feature engineering, we calculated multiple variables for our training process. The table below shows the variables and the calculation to obtain them in our user dictionary, question dictionary, and training features list.

Basic Statistics	Harmonic Mean	User's Performance
User_df: User_id Question_answered Mean_user_accuracy Std_user_accuracy Median_user_accuracy Skew_user_accuracy	User_df: User_id Question_answered Mean_user_accuracy Answered_correctly_user	User_df: User_id Question_answered Mean_user_accuracy Answered_correctly_user Performance
<pre> user_answers_df = grouped_by_user_df.agg({ 'answered_correctly': ['mean', 'count', 'std', 'median', 'skew'] }) </pre>	<pre> mean_user_accuracy = groupby('user_id')['answered_correctly'].mean() Question_answered = train['user_id'].value_counts() Answered_correctly_user = train.groupby('user_id' </pre>	<pre> User_shift = train.groupby('user_id') ['answered_correctly'].shift() Cumulated = train.groupby('user_id') ['user_shift'].agg(['cumsum', 'cumcount']) </pre>

	<pre>)['answered_correctly'] .sum()</pre>	<pre>Cumulated_question =Groupby('user_id')['us er_shift_question'].agg (['cumsum', 'cumcount']) mean_user_accuracy = train.groupby('user_id')['answered_correctly_u ser_average'].last() question_answered = Groupby('user_id').size () - 1 Answered_correctly_user = mean_user_accuracy * question_answered</pre>
Content_df: Content_id Question_asked Mean_content_accuracy Std_content_accuracy Median_content_accuracy Skew_content_accuracy	Content_df: Content_id Mean_content_accuracy Tags_label	Content_df Content_id Question_asked Mean_content_accuracy Answered_correctly_question Tags_community
<pre>content_answers_df = grouped_by_content_df.a gg({ 'answered_correctly': ['mean', 'count', 'std', 'median', 'skew']})</pre>	<pre>mean_content_accuracy = train.groupby(['content _id'])['answered_correc tly'].mean() train['tags_label'] = train['tags'].astype('c ategory').cat.codes</pre>	<pre>mean_content_accuracy = groupby('user_id')['ans wered_correctly'].mean() Question_asked = train['user_id'].size() Answered_correctly_ques tion = question_answered * mean_user_accuracy</pre>
Train Features Mean_user_accuracy Question_answered	Train Features Mean_user_accuracy Question_answered	Train Features Mean_content_accuracy Performance

Std_user_accuracy Median_user_accuracy Skew_user_accuracy Mean_content_accuracy Question_asked Std_content_accuracy Median_content_accuracy Skew_content_accuracy Prior_question_had_explanation Prior_question_elapsed_time	Answered_correctly_user Mean_content_accuracy Part Attempt Hmean_user_user_content_accuracy	community
	attempts =train.groupby(['user_id', 'content_id']).content_id.transform('cumcount') hmean_user_content_accuracy = (2 * ((train['mean_user_accuracy'] * train['mean_content_accuracy']) / (train['mean_user_accuracy'] + train['mean_content_accuracy'])))	Answered_correctly_user_average = cumulated['cumsum'] / cumulated['cumcount'] Average_past_questions =(cumulated_question['cumsum']) / (cumulated_question['cumcount'] + 1) Performance_before = Answered_correctly_user_average - average_past_questions performance = train.groupby('user_id')['performance_before'] .last()
Pros: <ul style="list-style-type: none"> - Simple calculation and computation - Handle the new user in the test dataset by using mean, median, standard deviation, and skewness of both user and question as a ground information to evaluate new user's potential of answering 	Pros: <ul style="list-style-type: none"> - Fewer features for model training → speed up the process - Additional feature 'attempt' provides our model more information to predict. - The 'harmonic mean' feature shows the relationship of a user with a certain question 	Pros: <ul style="list-style-type: none"> - Fewer features for model training → speed up the process - Additional feature 'tag_community' gives our model the additional information that if question A is related to question B. - The average accuracies of both

<p>correctly a certain question.</p> <p>Cons:</p> <ul style="list-style-type: none"> - Many features for the model to train → slower training speed - Do not have a feature that represents the correlation between user and question 	<p>→ model considers fewer feature without losing general information</p> <p>Cons:</p> <ul style="list-style-type: none"> - Not consider many features in the questions dataset 	<p>questions and users are based on the time interval. The 'performance' does consider the improvement of the user's accuracy and the question's accuracy over time → model looks at one feature at a specific time to get the prediction.</p> <p>Cons:</p> <ul style="list-style-type: none"> - Complicated calculation and computation → increase memory usage
--	---	--

3. Time-series API & Batch Looping

a. Time Series API

- This competition is different from other Kaggle Competitions in that:
 - We need to submit from our Kaggle notebooks
 - We must use the **riiideducation** python module to control the flow of information to avoid us from using future data to make predictions.

We had to use the time-series API (iter-test) emulator to receive the test set and make predictions. A time series model is a supervised learning method to forecast the future values of a field based on its previously observed values. It is used to analyze time based data when historical patterns can explain the future behavior.

For declaration, we used:

```
import riiideducation

env = riiideducation.make_env()
iter_test = env.iter_test()
```

For iteration, we used (notice that we cannot recall it once it's called):
`'env.iter_test()'`

For prediction, we used: `'env.predict(sample_prediction_df)'`

b. Batch Looping:

- In the training dataset, we have the feature 'task_container_id' as a code ID for the batch of questions or lectures. Looping through a series of batches of questions in the testing dataset, and in each batch, we obtained the existing features in the test set and calculated the training features needed referencing from our users and questions dictionaries. After that, we updated the user and question in the test dataset to the dictionaries for later use.

Basic Statistics	Harmonic Mean	User's Performance
We merged each batch of questions from the test dataset with our user and questions dictionaries.	<p>For question dictionary:</p> <ul style="list-style-type: none"> - Update the number of times the question is asked - Update the percentage of correct answers for that question <p>For user dictionary:</p> <ul style="list-style-type: none"> - Update the number of questions answered, the number of correct answers that user have for the question - Update the user's accuracy score - Update the attempt of user in the dictionary - Update the harmonic mean of a user with certain question 	<p>For question dictionary:</p> <ul style="list-style-type: none"> - Update the number of times the question is asked - Update the percentage of correct answers for that question <p>For user dictionary:</p> <ul style="list-style-type: none"> - Update the sum of average correctness by adding the average accuracy of each question - Add the number of correct answers and the number of questions. - Update the user's performance: average of answers – average of others on same question
Handle missing data or null/NaN values:	Handle missing data or null/NaN values:	Handle missing data or null/NaN values:
All cases:	Existing users in training	Existing users in training

<ul style="list-style-type: none"> - Fill in the ground number (value = 0.5) for all the null/NaN places in the test dataset after merging 	<p>set:</p> <ul style="list-style-type: none"> - Fill in the null/NaN values in <code>prior_question_had_explanation</code> a value = 23916 (mean of all valid values in this column) <p>New users in the test set that does not exist in training set:</p> <ul style="list-style-type: none"> - Created a new user in dictionary with <code>mean_user_accuracy = 0.68</code> (mean of all users accuracy), the rest = 0 	<p>set:</p> <ul style="list-style-type: none"> - Fill in the null/NaN values with the mean of all users for that feature. <p>New users in test set that does not exist in training set:</p> <ul style="list-style-type: none"> - Created a new user in dictionary with all null/NaN values = 0
---	--	--

After updating the dictionaries and obtaining all the needed features for training, we applied the data to our model and predicted results. Then, we moved on to the next batch of questions and repeated the same process for each batch.

c. Primary Method: LightGBM (used in submission notebook)

I. LightGBM overview

LightBGM is an implementation of gradient boosted decision trees. Boosting is an ensemble technique where it keeps adding new models to correct the errors made by existing models until no further improvement is possible. Furthermore, gradient boosting uses gradient descent algorithm in order to keep the loss to minimum when adding new models to predict errors of existing models, and then, summing the models up for final prediction.

As mentioned above in the Technologies Used, LightBGM is different from other gradient boosting algorithms because it grows leaf wise instead of level-wise. Leaf-wise algorithms can decrease the loss more than level-wise which explains the reason why LightBGM normally achieves much better accuracy than other algorithms. Additionally,

LightGBM is “light” because it spends less memory to deal with larger amounts of data and has a stronger computation power to produce the results faster.

Possible disadvantage that we need to take into consideration is, although the leaf-wise method allows the trees to converge faster, it may easily lead to overfitting. But this can be prevented by training on larger dataset like what we have for this competition and tune on parameters that focus on overfitting prevention which we will elaborate on in the later session.

II. Prevent overfitting for LightGBM

A disadvantage of LightGBM is that the model is prone to overfitting, especially if the data size is small: the suggestive value is around or less than 10,000+ rows. However, in this competition, our model was trained on 44 million rows, which is considerably larger than the suggestive value, so overfitting is unlikely to occur. In addition, LightGBM has parameters to prevent overfitting:

1. `early_stopping_rounds`: one simple and effective way to prevent overfitting is to use early stop, a parameter for LightGBM, that controls the number of rounds the model runs before stopping due to lack of improved performance.
2. `num_leaves`: this is the main parameter to control the complexity of the tree model. If the number of leaves in the tree has a smaller value, then the tree would be smaller in the sense that it will generalize well, thus, prevent overfitting. This parameter needs to be tuned together with `max_depth`, as the number of leaves are dependent on the depth of the tree.
3. `max_bin`: since LightGBM uses a histogram based algorithm to decide how to split the data and create a weak learner. A larger bin value can lead to higher accuracy, but may cause overfitting.
4. `max_depth`: this is used to avoid growing deep trees.
5. `min_data_in_leaf`: a key parameter to prevent over-fitting in a leaf-wise tree. This parameter value depends on the training data size and `num_leaves`. To avoid overfitting, set a larger value to this parameter to avoid having a tree that is too deep. However, there is a balance that needs to be considered between overfitting and under-fitting. In practice, setting it to 100 or 1,000 is enough for a large dataset.
6. `lambda_l1` and `lambda_l2`: parameters to control L1 and L2 for regularization: If `lambda_l2` is 0, it is a LASSO penalty. If `lambda_l1` is 0, it is a RIDGE penalty. If none are zeros, it is an elastic-net penalty.
7. `feature_fraction`: this is used to speed up the training and prevent overfitting through performing column sampling. On each iteration (tree), LightGBM will randomly select a fraction of features. The goal is to not use all features, but only a fraction of features.

8. `bagging_fraction`: this works similarly to `feature_fraction` in that it specifies the fraction of data to be used for each iteration (tree) to prevent overfitting.

Note that there are many trade-offs between increasing accuracy and preventing overfitting, such as for `num_leaves`, `max_bin`, and etc.

III. Parameter tuning result:

Method 1: Basic Statistics

```
Number of finished trials: 5
Best trial:
Value: 0.7305851357810981
Params:
  num_leaves: 174
  max_bin: 892
  max_depth: 6
  min_child_weight: 1
  feature_fraction: 0.6429213820434558
  bagging_fraction: 0.7141760446475449
  bagging_freq: 4
  min_child_samples: 52
  lambda_l1: 0.06726534180070018
  lambda_l2: 4.38741895928788e-05
```

Method 2: Harmonic Mean

```
Best trial:
Value: 0.7605258237155187
Params:
  num_leaves: 470
  max_bin: 821
  max_depth: 10
  min_child_weight: 7
  feature_fraction: 0.7527902562607991
  bagging_fraction: 0.9645024130180766
  bagging_freq: 7
  min_child_samples: 6
  lambda_l1: 0.06432115222474881
  lambda_l2: 5.4961749734305755e-08
```

Method 3: User's Performance

```
Number of finished trials: 5
Best trial:
Value: 0.7327974200600917
Params:
  num_leaves: 446
  max_bin: 778
  max_depth: 11
  min_child_weight: 14
  feature_fraction: 0.4177390539238916
  bagging_fraction: 0.6247455288923738
  bagging_freq: 5
  min_child_samples: 42
  lambda_l1: 2.4626401998820047e-08
  lambda_l2: 4.426327090969216e-07
```

We will compare the result with different feature engineering approaches in section 5.

d. Secondary Method (used for prior experimentation and results comparison)

I. Keras Neural Network

Keras is a powerful and easy free open source in Python library for developing and evaluating deep learning models. It uses efficient numerical computation libraries like TensorFlow that allows users to train neural network models.

A neural network is a series of algorithms that mimics the human brain to try to recognize the relationships between variables in a dataset. “Neuron” is a mathematical function that collects and classifies information based on a defined architecture, while “network” applies statistical methods to analyze.

In Keras, there is an organization of layers, which is a sequence or a graph of different modules that can interact with each other. The Keras Modules consist of:

- Neural layers
- Optimizers
- Regularization schemes
- Cost functions
- Activation functions
- Initialization schemes

There are multiple deep learning backends that can be used to develop Keras models. In this project, we used TensorFlow backend from Google to support. However, we did not use Keras as a our primary model because we expected that the Neural Network algorithm would not outperform the LightGBM due to the competition’s structured data and a few disadvantages comparing with decision tree algorithm LightGBM:

- The speeds of training and classifying in Keras are both slower
- The possibility of overfitting from Keras is higher
- Keras is not friendly for categorical features
- Complicated to observe the decision-making process

Due to these disadvantages of Keras, we did not pay great effort to build and train our model with Keras Neural Network. Nonetheless, we still tried to implement and ran it for the result comparison with LightGBM.

- Data-preprocessing & Feature-engineering:

In the pre-processing data part, we did the same process as in the LightGBM preparation. After having all the statistical features, we chose these features for modeling:

‘mean_content_accuracy’, ‘question_answered’, ‘answered_correctly_user’,
‘mean_user_accuracy’, ‘prior_question_elapsed_time’, ‘prior_question_had_explanation’.

We used the LabelEncoder function to encode the categorical features. For null/NaN value, we filled in the value 0.5 for 'mean_user_accuracy' and 'mean_content_accuracy', overall mean calculated for 'prior_question_elapsed_time', and 0 for the rest. Then, we normalized the data by scaling them using StandardScaler.

- Model Training

We created a model using Sequential and tried to prevent the overfitting problem by having dropout, Conv1D, Flatten, and Dense layers. Firstly, it created Dense layers with pre-defined batch size and dropout layers with rates of 0.1. The output of the Dense layer later applied Rectified Linear Unit along with BatchNormalization activation for the network to converge quickly. The model is compiled with the optimizer Adam at the learning rate of 0.01 and using the metric BinaryAccuracy.

We called fit() function to train our data on the model. The training process occurs over epochs, which is a single pass through all the rows in our training set. Each epoch is split into many batches. The number of batches in each epoch is decided by the chosen batch size, in this case we set batch size to 3072 and decided the epochs number to be 13. After a few epochs, if the roc_auc_score does not improve, the training process is stopped, and we finalize the fold with the best epoch score. We used KFold and splitted randomly for 3 folds. In order to receive the prediction, we proceeded and engineered the test dataset in the same way as the training set and applied it to the model.

- Results:

As expected, we received a pretty low roc_auc_score:
Training accuracy: 0.746282

Hence, Keras Neural Network is considered to be our secondary approach for this project due to its suboptimal accuracy.

II. *XGBoost*

Similar to LightBGM, XGBoost is an implementation of gradient boosted decision trees and it stands for “eXtreme Gradient Boosting”. If we leave the LightBGM aside, XGBoost may perform the best among other implementations of gradient boosting since it is designed for execution speed and model performance. Szilard Pafka, an expert in data science, once compared the performance of XGBoost to other implementations of gradient boosting and bagged decision trees back in May 2015 and he commented that “[xgboost] is fast, memory efficient and of high accuracy” in the result blog he wrote. Regarding the model performance, a number of Kaggle competition winners won using XGBoosting.

Again, from what has been discussed previously, XGBoost may not perform better than LightBGM when it comes to larger datasets. However, we still want to have an experiment with this algorithm just for our own knowledge.

- Data-preprocessing and feature engineering

Since this implementation was done for the sake of experimentation, we had conducted similar feature engineering as basic statistics in LightGBM. We conducted basic statistics computation on questions and trains and chose the following for modeling:

'prior_question_elapsed_time', 'prior_question_had_explanation', 'q_correct', 'q_count', 'q_var', 'q_std', 'part_percent_correct', 'user_answer_mean', 'user_answer_count', 'user_min', 'user_max', 'user_std', 'user_var'. For missing value, we filled in with the mean.

- Model training

Similar to the process of value prediction in LightBGM, we tuned for parameters.

1. Learning rate: it helps make the model more robust by decreasing the weights on each step.
2. Min_child_weight: this indicates the minimum sum of weights of all observations needed in a child. A higher value of min_child_weight avoids the model from learning particular cases which may induce overfitting.
3. Max_depth: a lower value set for the maximum depth of a tree will make sure the tree does not learn very detailed to particular samples.
4. Alpha: Since we have numbers of features to run for the model, we need to implement regularization so that the algorithm will run faster. Alpha is the L1 regularization term.
5. Objective: We set it to 'binary logistics' since 'answer_correctly' has binary output and the goal for this competition is to predict the possibility of the next question answered correctly, thus we need a probability output instead of classes.

- Result

As expected, we received a much lower score than LightBGM. However, this version was only done for the purpose of experimentation. We definitely could produce a higher score if we improve on our feature engineering and tuned more parameters that aim for higher accuracy.

Training ROC: 0.754498

5. Results Prediction

Here is a table that summarizes the results for 3 different methods:

Methods	Basic Statistics	Harmonic Mean	User's Performance
---------	------------------	---------------	--------------------

Predictions table sample (first 15 row_ids)	row_id		answered_correctly	row_id		answered_correctly	row_id		answered_correctly
	0	0	0.349644	0	0.628373	0	0	0.493751	
	1	1	0.831839	1	0.869051	1	1	0.949753	
	2	2	0.889674	2	0.743329	2	2	0.635232	
	3	3	0.855463	3	0.861937	3	3	0.863164	
	4	4	0.514874	4	0.346753	4	4	0.378793	
	5	5	0.589086	5	0.665424	5	5	0.628790	
	6	6	0.777008	6	0.596698	6	6	0.543708	
	7	7	0.836661	7	0.714278	7	7	0.825884	
	8	8	0.620337	8	0.727930	8	8	0.889581	
	9	9	0.615558	9	0.498891	9	9	0.554914	
	10	10	0.844725	10	0.876799	10	10	0.857146	
	11	11	0.803229	11	0.586017	11	11	0.486263	
	12	12	0.694808	12	0.490312	12	12	0.435731	
	13	13	0.886981	13	0.696509	13	13	0.585561	
	14	14	0.514164	14	0.336204	14	14	0.313425	
LGBM training roc_auc_score	0.724754			0.759666		0.78016			
LGBM valid roc_auc_score	0.7306			0.7605		0.7328			
Kaggle public score	0.723			0.753		0.760			

The above table is the best score for 3 different methods. In the meantime of finalizing the report, we still have one more trial, employing the user's performance mechanism in running state. Although in training, harmonic mean seems to produce a higher valid score, from our past experience running the code, we still expect the user's performance to produce a better score since it trains on the more sophisticated features ('performance' and 'tag') that consider the correlation between original features and feature itself.

6. Summary and Lessons Learned:

The model that gives us the best result is the one that applied the user's performance feature engineering method and LightGBM. In this model, we decided to group the data by user's id to compute user's accuracy over time and also group by content's id to compute question's accuracy based on cumulative summation and count. From the accuracy rates of users and questions, we were able to produce a feature that compares the two variables, 'performance'. Additionally, from the 'tags' column in the questions dataset, we clustered

those tags based on their correlations and created the `'community'` feature. The final training set that was used to build our model consists of three features:

`'mean_content_accuracy'`, `'community'`, `'performance' for user'`. We did the BayesianOptimization parameters tuning using Optuna and took the result parameters to build our model with LightGBM. The test dataset is then applied to the model to receive the predictions.

With online learning, education fuses with AI technologies that learn from big data to produce more personalized learning experiences through knowledge tracing and high quality education for children in the post-COVID-19 world. During this one-month experimentation with this competition, we had encountered many challenges and had learned how to implement techniques from classes and also from our own exploration.

The competition is a big-data analysis project in that we were given a goal to predict the learning result for users using massive datasets to engineer features. Because it was our first time in handling big data, we learned a lot during the past month in terms of understanding memory usage, data complexity, and machine learning technologies:

a. Memory usage:

It was definitely our first time dealing with such a large scale of data. We never expected that just for how to read data with a faster speed and less memory usage, which we achieved by converting our data into pickle form, could be a topic that needed investigation. Moreover, it was also our first time to conduct running codin online which we have to take the RAM memory into consideration. Data preprocessing and features engineering for memory reduction brought us a level deeper to practical practice.

b. Data Complexity:

Not only does the scale of the data create headaches, so does the complexity of the data. With three files and more than 20 features intervene, we have spent a long time in the early stage to understand the relationships in between. One good thing for Kaggle competition is that we share knowledge and insights with other thousands of teams. Consulting ideas from people who have more experience or are knowledgeable prevents us from heading in the wrong direction. On the other hand, one big improvement of ourselves is that we have learnt how to use different data visualization tools to observe the hidden correlation between features and generate more accurate ideas for feature engineering. We also found concepts learned from other classes were able to make use in this competition. In feature engineering, we were able to utilize our knowledge on SQL to have a better understanding on the meaning of computations like `'groupby'` and `'join'`.

c. Technology:

All the algorithms, LightBGM, Keras, XGBoost that we have employed in this competition were new for us. Thus, we have definitely learned about these algorithms' mathematical concepts, advantages and disadvantages in different scenarios, and

strategies to prevent overfitting. Moreover, we have also studied a new approach, optuna, to perform faster and more robust parameter tuning.

7. Further Improvements on Model

In machine learning, there are always rooms for improvement. We have considered a few improvements to increase accuracy and efficiency of our model, but due to the time constraints, we unfortunately could not implement them in time. In terms of feature engineering, data selection, data cleaning, and data reading, these are plans for future improvements:

- a. Create a new feature through exploring the `lecture.csv` dataset: prior, in the EDA section, we learned that users who watched lectures have significantly higher accuracy. Therefore, a boolean feature should be constructed to reflect if users watched lectures before answering the next question.
- b. Select data with greater diversity: the training dataset consists of only users with very high volume of interaction, which produced bias in the model. This is not representative since we have no prior knowledge of the users composition in the test data. Conceptually, users who spend long periods of time have different behavioral data than users who spend limited time to practice and answer questions.
- c. Identify and remove outliers by different features: prior, in the EDA section, we learned that a small portion of the users have very limited interaction volume, and a few who have very high interaction volume. If outliers were determined by a single standard, interaction volume, it may not yield the best result as the training data is high dimensional. More EDA should be performed to determine outliers in other features, such as `timestamp`, to eliminate outliers.
- d. Adapt advanced technologies for pre-proceeding data and training models:
 - i. Data Management & Manipulation: we can learn more about BigQuery or RAPIDS (with cuDF/GPU) to handle our 101-million-row dataset. These technologies can speed up the process of computing to 16 times in comparison with Pandas. Since we used less than a half of our training dataset due to the limited memory of Kaggle notebook, these techniques may help us to manipulate on the full training dataset.
 - ii. Model Training: we want to dig through the documentation of CatBoost and try out this new complex technology to obtain a model with higher accuracy and faster speed.

These are the plans that we have had so far. In a couple of weeks until the competition's deadline, we can always experiment more to optimize our result.

8. Member Contribution

Please note that most of our work is done collaboratively due to the small size of the team and the time difference challenges that we are facing from remote learning. We realize the list will be too detailed if we provide contributions by work done. We agree to equally split the contributions.

9. References

We read through notebooks, articles, documentations of technologies used for reference:

<https://www.kaggle.com/markwijkhuizen/riiid-training-and-prediction-using-a-state>

<https://www.kaggle.com/its7171/lgbm-with-loop-feature-engineering>

<https://www.kaggle.com/mamun18/riiid-lgbm-lji-hyperparameter-tuning-optuna>

<https://www.kaggle.com/pratikskarnik/riiid-keras-transformer-starter>

<https://www.kaggle.com/erikbruin/riiid-comprehensive-eda-baseline>


<https://www.kaggle.com/isaienkov/riiid-answer-correctness-prediction-eda-modeling>

<https://www.kaggle.com/datafan07/riiid-challenge-eda-baseline-model>


<https://tech.preferred.jp/en/blog/lightgbm-tuner-new-optuna-integration-for-hyperparameter-optimization/>

<https://towardsdatascience.com/lightgbm-vs-xgboost-which-algorithm-win-the-race-1ff7dd4917d>

10. History of submissions

Submission and Description	Status	Public Score	Use for Final Score
Fork of Riid : modeling tuning ed2692 (version 5/5) 3 minutes ago by Vy Tran From Notebook [Fork of Riid : modeling tuning ed2692]	Notebook Running		<input type="checkbox"/>
Riid : modeling tuning (version 3/3) 9 hours ago by Vy Tran From Notebook [Riid : modeling tuning]	Succeeded	0.760	<input type="checkbox"/>
Prediction EDA Tuning (version 2/2) 21 hours ago by Vy Tran From Notebook [Prediction EDA Tuning]	Succeeded	0.723	<input type="checkbox"/>

Riid : modeling tuning (version 1/3) 21 hours ago by Vy Tran From Notebook [Riid : modeling tuning]	Succeeded	0.760	<input type="checkbox"/>
Riid : modeling 15quantile (version 2/4) 3 days ago by Selena Lin From Notebook [Riid : modeling]	Succeeded	0.759	<input type="checkbox"/>
npz notebook 4 Version 1 (version 2/2) 4 days ago by Vy Tran From Notebook [npz notebook 4]	Succeeded	0.753	<input type="checkbox"/>
npz notebook 2 (version 1/2) 5 days ago by Vy Tran From Notebook [npz notebook 2]	Succeeded	0.752	<input type="checkbox"/>
npz notebook (version 2/2) 6 days ago by Vy Tran From Notebook [npz notebook]	Succeeded	0.753	<input type="checkbox"/>

Riid : modeling (version 1/4) 8 days ago by Selena Lin From Notebook [Riid : modeling]	Succeeded	0.760	<input type="checkbox"/>
notebook3ea4c51a0d (version 1/1) 8 days ago by Vy Tran From Notebook [notebook3ea4c51a0d]	Succeeded	0.760	<input type="checkbox"/>
Trial 2 (goose riid modeling notebook) (version 1/1) 10 days ago by Vy Tran From Notebook [Trial 2 (goose riid modeling notebook)]	Notebook Timeout	Error 	<input type="checkbox"/>
notebook6f92f7557b Version2 (version 2/2) 12 days ago by Vy Tran From Notebook [notebook6f92f7557b]	Succeeded	0.749	<input type="checkbox"/>
Cartoon girl without internet (version 4/4) 15 days ago by Selena Lin From Notebook [Cartoon girl]	Succeeded	0.758	<input type="checkbox"/>