Machine Learning Engineer Nanodegree

Capstone Project

Aaron Moss
December 14th, 2020

Accepted Proposal Review: https://review.udacity.com/#!/reviews/2594028

I. Definition

Project Overview

This project is based on the Riiid AIEd Challenge 2020[1] Competition on Kaggle. This challenge is aiming to build AI that can act as an online tutor to help provide differentiate learning for a student on a given topic. This idea could lead to a solution that can quickly scale to bring more students to the learning environment globally and help bring more equality of access to education. In addition to equality, this solution could help student learning to occur more in their proximal zone of learning which may lead to an increase in attendance, engagement, and personalized learning.

The solution is to predict the likelihood of a student producing the correct answer, on learning topic within in their individualized learning sequence. To help the Kaggle community accomplish this solution, Riiid Labs have released EdNet, which is the world's largest open database containing over 100 million student interactions with +300k students.

I have chosen this competition due to its close nature to my current position at an EdTech company. We provide content, assessments, and facilitate the interactions between educator & nursing student within their institution via SaaS.

Problem Statement

This competition is to develop a model that can predict the likelihood of a student getting an item correct. This competition inference dataset will have new users outside of the training set but not new question items.

The proposed solution to this problem is to apply tree-boosting classification model within Kaggle notebook or RNN time series within AWS Sagemaker instance. For

train/test split, I will be simulating the batch inference by splitting the training set by last 100 interactions per user and, within the 100 observations, the last four data points will become the test set. In addition, I will need to build robust code to transform incoming batch to update user/item dataframes and merge back to original batch rows for inference. The expected output should be a range between 0.00 and 1.00 as it is a probability.

An outline of tasks needed to implement a solution:

- Due to the size of the dataset, the first task will be to properly import the data by either reduction the data types or pulling a subset of the full data.
- The second task will be to transform and feature build on supplement datasets. This competition has dimensional data on questions and lectures the student is likely to engagement. Topic tags, type of question structure, unique of question bundles can be transformed before merging to training set.
- The third task would be Exploratory Data Analysis on the training set which could include analysis for target, user, time component, and question-to-user interactions. Also, one could create a list of potential features to build based on the intuition for the data or the industry. For example, it is possible that users who have an above-average streak of answering questions correctly, might indicate higher future performance or allow the model to quickly split them in the predictive state.
- The fourth task would be splitting the dataset into train and test that appropriately fits the nature of the competition. This competition is a times series prediction and, as such, would require data splitting within that time constructure. My task will be to pull the last 100 user:question pair interactions for each user. Then parse the last four of the 100 interactions as the test set to simulate the evaluation task of predicting the performance on the next question bundle.
- The last step will be to build a robust API call to receive, transform, and infer predictions. This task will allow to model to ingest the newest performance of the user, question, and custom features as it is coming in via the competition Test API. The competition describes the Test API as performing batch inference and requires a called prediction function before going to the next batch.

Metrics

The evaluation metric for this competition is be Area under the Curve, which indicates the model's ability to correctly identify a random observation to the actual class group (0= Miss, 1= Correct). However, I will be optimizing the model and features for binary LogLoss metric to help enhance model's target metric.

II. Analysis

Data Exploration & Exploratory Visualization

There are three datasets for this competition: training, questions, and lectures.

Riiid Labs description of datasets[2]:

train.csv

- `row_id` : (int64) ID code for the row.
- `timestamp` : (int64) the time in milliseconds between this user interaction and the first event completion from that user.
- `user_id` : (int32) ID code for the user.
- `content_id` : (int16) ID code for the user 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 batch of questions or lectures. For example, a user might see three questions in a row before seeing the explanations for any of them. Those three would all share a `task_container_id`.
- `user_answer` : (int8) the user's answer to the question, if any. Read -1 as null, for lectures.
- `answered_correctly` : (int8) if the user responded correctly. Read -1 as null, for lectures.
- `prior_question_elapsed_time` : (float32) The average time in milliseconds it took a user to answer each question in the previous question bundle, ignoring any lectures in between. Is null for a user's first question bundle or lecture. Note that the time is the average time a user took to solve each question in the previous bundle.
- `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. The value is shared across a single question bundle, and is null for a user's first question bundle or lecture. Typically the first several questions a user sees were part of an onboarding diagnostic test where they did not get any feedback.

questions.csv: metadata for the questions posed to users.

- `question_id` : foreign key for the train/test content_id column, when the content type is question (0).
- `bundle_id` : code for which questions are served together.
- `correct_answer` : the answer to the question. Can be compared with the train `user_answer` column to check if the user was right.
- `part` : the relevant section of the TOEIC test.
- `tags` : one or more detailed tag codes for the question. The meaning of the tags will not be provided, but these codes are sufficient for clustering the questions together.

lectures.csv: metadata for the lectures watched by users as they progress in their education.

- `lecture_id` : foreign key for the train/test content_id column, when the content type is lecture (1).
- `part` : top level category code for the lecture.
- `tag` : one tag codes for the lecture. The meaning of the tags will not be provided, but these codes are sufficient for clustering the lectures together.
- `type_of` : brief description of the core purpose of the lecture

The training set has 393,656 unique users with ~99 million student-questions. There were 2 million interactions with online lectures. Also, the training set contains 13523 unique question items and 10000 unique question bundles. The training set is originally

sorted by user_id and timestamp in ascending order. In prior_question_elapsed_time, the ranges of milliseconds are 16000 (25th percentile) and 29666 (75th percentile) with ~24% of observations above 75th percentile & below the 25th percentile.

	row_id	timestamp	user_id	content_id	content_type_id	task_container_id	user_answer	answered_correctly	prior_ques
0	0	0	115	5692	False	1	3	1	NaN
1	1	56943	115	5716	False	2	2	1	37000.0
2	2	118363	115	128	False	0	0	1	55000.0
3	3	131167	115	7860	False	3	0	1	19000.0
4	4	137965	115	7922	False	4	1	1	11000.0
5	5	157063	115	156	False	5	2	1	5000.0

Figure 4: Train.csv

In timestamp variable, we convert the millisecond unit of time into days and identify that majority of users have a max total of days under one day of activity from first event to last. We can assume the test dataset and the competition hidden set may have more observations with additional time from this point.

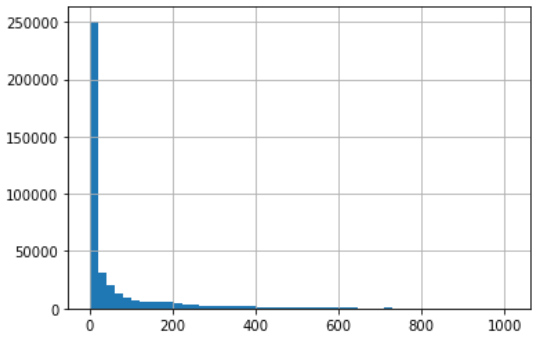

Figure 5: Users' max timestamp by hours

After reviewing the train dataset for memory usage, we see that prior_question_had_explanation is significantly higher memory for a Boolean column. We will need to either reformat or remove from dataset entirely.

```
Index                          128
row_id                   809842656
timestamp                809842656
user_id                  404921328
content_id               202460664
content_type_id          101230332
task_container_id        202460664
user_answer              101230332
answered_correctly       101230332
prior_question_elapsed_time  404921328
prior_question_had_explanation  3594972816
dtype: int64
```

Figure 5: Train Memory Usage

Analyzing the missing values

The train set has missing values for only the prior_question_elapsed_time variable since it is a lagging indicator the first row for each user is NaN. The questions dataset is missing a single tag for question so we will need to impute with the most common tag. The lectures dataset did not have any missing values.

Explore Target

The target variable showed that users answered the questions correctly 64% of the time. They watched lectures 2% of the time (shown by '-1'). My assumption is that lectures will not contain much prediction power as it does not have much user interaction and, after reviewing the Kaggle discussion board, most users have not used lectures to feature development. As a result of my institution and confirmed by Kaggle community, the lectures dataset will not be explored in detail.

At the intersection of task_container_id and target variable, we see that bundle groups have normal distribution across train set. This distribution shows that this variable may be a good enhancement to user performance and a combining variable with other variables for feature engineering.

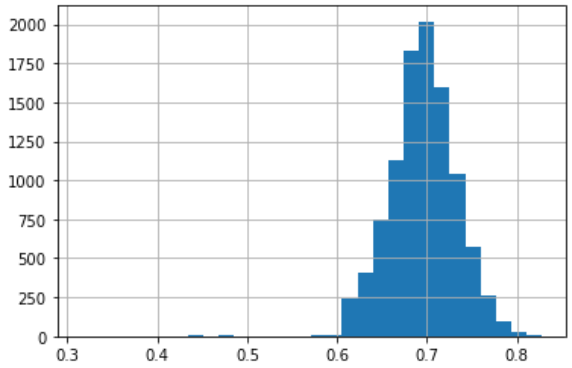

Figure 6: Bins of task_container_Id by answered_correctly mean

We need to investigate the interactions between the number of questions attempted and the percentage correct the user ultimately achieved. This analysis may point to a relationship between the count of questions and performance, which could be help the model split users. The histogram of count per user (Figure 7) showed that most users have answered 500 items or less. The resulting scatter plot (Figure 8) shows a positive trend between the user's count of items attempted and the user's overall average correct. This relationship indicated that count of items per user is positively related with user performance.

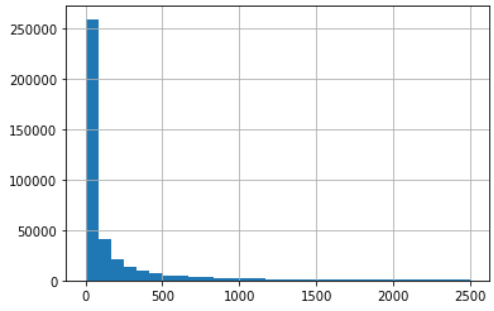

Figure 7: Count of items answered per users

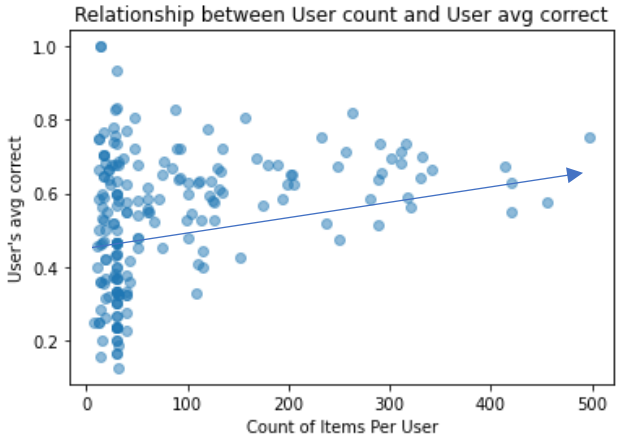

Figure 8: Relationship between User count & User avg correct

Explore Questions

The questions dataset has 13523 unique items with 188 unique tags or topics and 9765 unique bundles. Each question has "part" with 7 levels that connect to relevant section of the TOEIC test. Specifically, Part 1-4 includes Listening Section and Part 5-7 includes Reading Section.

	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

Figure 9: Questions.csv

Algorithms and Techniques

We will use LightGBM to find the solution to our classification problem. LightGBM is a great out-of-the-box algorithm for classification that is robust against outliers, can handle null values, allows for interaction due to splitting, and performs quite well without heavy hyperparameter tuning. Most importantly, it is boosts faster training speed with higher efficiency, which allows for quicker iteration.

Benchmark

The benchmark performance was accomplished by using the sample prediction given by the Kaggle competition, which used a fixed probability of passing item as .50 for all inference rows. This fixed probability technique yielded a .5 on the ROC-AUC metric. In addition, we developed a more-realistic benchmark where we used each individual user's average performance on the training set as the prediction and for new users in the test set, we will use the .5 probability for future items.

III. Methodology

Data Preprocessing

We need to filter out lecture items to only user-question interactions as they will be the main focus on this project. Next, we will focus on the feature engineering process.

My feature engineering process includes listing out all potential ideas first and then review the data for possible implementation of these features. Also, when building features for time series problem, I have found that constructing a simulated memory for users or questions within the model in the form of running count, cumulative sum, or mean usually has significant predictive power.

Of the hypothesized features, I posited that running mean of user's performance would be predictive to future performance regardless of next question. However, if we gave the model the running mean of each content_id then it would be able to split on both user and content_id performance. Overall, there are three groups of hypothesized features that can be developed: time-based, user-based, and question-based performance.

Within time-based features, one feature could be to include the length of time the user took to answer the previous question bundle group, which may explain a degree of "certainty" of answer/s for that bundle. In addition, we may be able to build a feature

that compares a group of top performing users' time to correct response (75th percentile) from the current user's previous time on the same question bundle. This feature could capture the "pace" of a user arriving at a response and allowing the model to split on this difference in time and if correct answer was given using the running sum per user.

Within performance-based features, it may be predictive to allow the model to understand the longest streak of correctly answered questions or running streak within time series by user. Another feature would be called "strength of schedule". This feature would take the user's running average of questions' difficulty that they have completed. In addition, the difference from the user's running average of correctly answered questions over this "strength of schedule" may allow the model to place the user's performance in the historical question-difficulty context.

Lastly, the question-based features would come from the questions.csv data file. The categorical "Part" question dimension (seven levels) indicates the type of question using the TOEIC test format and may help the model split by difficulty and structure of question presented to user. Another question dimension is the "tag" of the question topic. However, the competition host does not indicate what the tags specifically mean. There are 188 unique tags in training set. Each question has one or an array of tags. The feature could be built to treat the array as a unique combination using simple label encoding or split the tags in each combination so that each combination is a column (e.g., first tag, first two tags, first three, etc). Another feature would be to take the top performing users' final mean on each question then see the difference in the full, final mean for each question to differentiate top performing users from the rest of the group or to identify easy or hard questions based on the high performing group.

The data splitting step will split the training set into the last 800 observations per user and then split the last four observations per user for the test set to simulate the batch inference in the Kaggle competition.

Implementation

Reduce training set memory allocation

The first step will be to get the appropriate question interactions from users in the train.csv. Secondly, due to its overall size, we will need to reassign dtypes to lower memory demand in train.csv (shown below).

```
## Memory reduction
train = train.astype({'row_id': 'int64',
        'user_id': 'int32',
        'content_id': 'int16',
        'content_type_id': 'int8',
        'task_container_id': 'int16',
        'answered_correctly': 'int8',
        'prior_question_elapsed_time': 'float32'})
```

Transform and feature build on supplement datasets

We need to transform questions.csv which to be to dummy specific variables or label encode topic sequences for items.

EDA and Feature Engineering

Merge new supplement dataframes to training set**.** Once complete training set is produced, I will use descriptive statistics and EDA to understand dataset. Also, we will build proposed features.

Data Splitting

Since this is a time-series problem, I will pull the last 800 user:qts pair interactions. Next, I will use the last four of the 100 interactions as the test set to simulate the evaluation task.

Model training and evaluation

Build both the simple inference with fixed probabilities: 50% on all items and user's final average item correct percentage. Test out the evaluation metrics and record the metric scores from Kaggle. Results:

Baseline Inference	AUC-ROC score
Fixed probability @ .5	0.5
User-Specific Mean	0.623

Next step is to build the model with the constructed features and test out the proper amount of feature set to gain the best evaluation metrics.

Time Series API for Test set

Build a robust API call to receive, transform, and infer predictions. Debug issues once the Test API is running through my inference loop. The competition describes the Test API as it will perform batch inference and requires a prediction before going to the next batch.

Refinement

After obtaining initial results, we removed some earlier observations to modify the training set to the last 600 observations by user to speed up training. We used the train function within lgbm package and defined basic parameters for initial training where objective is binary and the learning_rate is .10 ('binary logloss' is the default metric). We iterated through final feature list to get the fewest features for the most metric improvement (as shown in bold format).

Boosting Round	Train Metric	Value	Validation Metric	Value	ROC_AUC Metric
45	training's binary_logloss:	0.544198	valid's binary_logloss	0.554317	0.755
58	training's binary_logloss:	0.542914	valid's binary_logloss	0.536882	0.775
2258	training's binary_logloss:	0.544765	valid's binary_logloss	0.536882	0.778
1961	**training's binary_logloss:**	**0.544554**	**valid's binary_logloss**	**0.536882**	**0.779**
****115*	*training's binary_logloss:*	*0.55481*	*valid's binary_logloss*	*0.515873*	*0.794*

***Includes two additional features

We used via feature_importance plot to visualize how many times each feature was used to split the training observations (figure 9). The final features from training model, which gave ROC_AUC of 0.779, was selected. Next, the strength of schedule feature and run_diff_time feature were added to the last model for an improved AUC of .794.

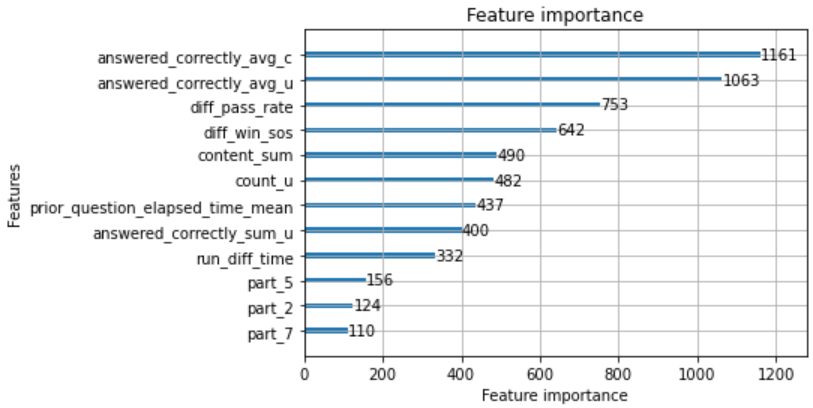

Figure 9: Lightgbm Feature_Importance

Final features

answered_correctly_avg_u, answered_correctly_sum_u, & count_u	User-based features
content_sum, diff_pass_rate, & answered_correctly_avg_c	Question-based features
prior_question_elapsed_time_mean & **run_diff_time	User by Time features
**diff_win_sos	Strength of Schedule feature
part_2, part_5, & part_7	Specific Levels in 'Part' variable

** 2nd final model features

We use Optuna package to perform Bayesian optimization of lgbm hyperparameters with 10 trials using 'binary logloss' as optimized metric[3]. We applied the parameters from Optuna package to the final features and called the lgbm train function. We set the ceiling for the boosting rounds and tell train function to stop the rounds if the metric is not improved within 10 rounds.

```
model = lgb.train(
                    params,
                    lgb_train,
                    valid_sets=[lgb_train , lgb_valid],
                    verbose_eval=100,
                    num_boost_round=1000,
                    early_stopping_rounds=10
                )
```

Figure 11: Lightgbm train function

IV. Results

Model Evaluation and Validation

This project's validation will take place within Kaggle inference. The approach will be to build replicate the models for my top two scoring iterations (AUC: 0.779 & 0.794).

For this competition, we will be making predictions on which questions each student is get correct. Inference task will have to loop through each batch of incoming questions, make a prediction, and then be granted access to the next batch. There are 2.5 million rows in the test set and it will not contain new questions but will have new users.

To aid in this inference batching, one will have to import the riiideducation package to use the iter_test and predict functions. Also, one has to restart their notebook to rerun inference batching if changes need to be made or problems debugged as to deter cheating.

Competition's detailed introduction[4]:

'… loop through all the remaining batches in the test set generator and make the default prediction for each. The iter_test generator will simply stop returning values once you've reached the end.

When writing your own Notebooks, be sure to write robust code that makes as few assumptions about the iter_test/predict loop as possible. For example, the test set contains question IDs that have not been previously observed in train.

You may assume that the structure of sample_prediction_df will not change in this competition. The lecture rows in test_df should not be submitted.'

```
In [9]: for (test_df, sample_prediction_df) in iter_test:
            test_df['answered_correctly'] = 0.5
            env.predict(test_df.loc[test_df['content_type_id'] == 0, ['row_id', 'answered_correctly']])
```

Figure 11: Sample inference code

For the first model (test AUC= 0.779), it submitted an AUC_ROC of **.763** on Kaggle and, at the time of submission a public leaderboard position in the **top 10% submission** (170th of 2283 submissions). This model used feature dataframes to merge into test_df and then predict function using model object. User performance was dynamically updated using functions but the rest of the features were the latest values for each user

about to future-based question interactions. This inference model has some weakest as it does not update continually as new information is presented.

The second model (test AUC= *0.794*), which is similar to the first inference submission but had functions for continuous update of SOS feature by batch, was limited by Kaggle API time constraints (>9 hrs). Improvements on speed were added to inference submission (i.e., concatenate with indexing between test_df and feature dataframes[5]), yet still unable to properly submit. If more time was allowed for this project, numpy would be explored to speed up and cut out the overhead inherit in pandas.

Justification

The model with the successful Kaggle submission was a substantial improvement from both baseline models. Our model was able to correctly assign a random observation into the correct class ~76.3% of the time and achieved a Kaggle leaderboard position in the top 10% of submissions.

Model	Type	AUC Score
Baseline 1	All predictions @ .50	.50
Baseline 2	Predictions set by user correct percentage	0.623
Model	Final Features w/ user dynamic updates	**.763**

V. Conclusion

Reflection

For this project, I felt confident in my abilities to intuitively sketch out feature ideas that would matter prediction based on my current work in this field. However, there were new problems that I needed to solve.

Firstly, the training data size was substantial. The solution that really helped my exploring and iteration was using Google Collab with a bigger instance than Kaggle. Also, reformatting the data before initially loading it into the notebook helped reduce file size. In addition, the usage of datatable package to save files in jay format

dramatically increased speed to upload and download files. Using datatable, I was able to reach a checkpoint in my feature development then merge them into a modified training set and remove those variables to free up notebook memory space.

The second problem was developing the inference API code that would be robust enough to transform the incoming batch to make predictions for the Kaggle competition. The first attempt was successful in giving predictions for the example test_df that was given. The attempt would time out, which in my estimation, relied too much on pandas. However, I was able to build on work from a published Kaggle notebook[6] that used numpy but relied on dictionary format to house and run loops to update the user's performance as it occured.

Improvement

Due to the limited time that I allotted for this project and Kaggle competition requirements, there were a couple of things that, time permitted, could take this project further. The first step would be to implement numpy architecture for Kaggle inference of strength of schedule feature and amount of time spent vs top performers on previous question bundle. The second step would be to attempt to recreate the SAINT+ paper (https://arxiv.org/abs/2010.12042) that uses RNN with self-attention layers. Based on Kaggle discussions, the leaders in the competition either are using these transforms or heavily modified Lightgbm to achieve AUC scores roughly around 0.80. With that being said, I feel relatively good about my solution being implemented in a real-life scenario as it beat the baselines and is within ~5 percentage points of the top Kagglers in the world.

Citations

1. Kaggle Riid Competition Overview: https://www.kaggle.com/c/riiid-test-answer-prediction/overview

2. Kaggle Riid Data Overview: https://www.kaggle.com/c/riiid-test-answer-prediction/data

3. Code taken from https://www.kaggle.com/mamun18/riiid-lgbm-lii-hyperparameter-tuning-optuna

4. Kaggle detailed introduction for inference: https://www.kaggle.com/sohier/competition-api-detailed-introduction

5. Kaggle fast pandas left join: https://www.kaggle.com/tkm2261/fast-pandas-left-join-357x-faster-than-pd-merge

6. Kaggle Tito : https://www.kaggle.com/its7171/lgbm-with-loop-feature-engineering