Riiid-test-answer-prediction

# NOTEBOOKS

## <https://www.kaggle.com/sohier/competition-api-detailed-introduction>

* API introduction

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

* EDA
* Super basic baseline
* Fillna, label encoder
* Only using 5 features. Argues other features doesn’t help CV nor LB.
* Uses lgbm
* No hpo

## <https://www.kaggle.com/rohanrao/tutorial-on-reading-large-datasets>

* How to read large datasets
* Rapids cudf is one of the options. Actually it is the fastest.
* It also mentions the format of the file. The type of file drastically affects the speed at which you can read the file.

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

* EDA
* Simple eda of the different files provided
  + Train.csv
  + Questions.csv
  + Lectures.csv
  + Example\_test.csv
* Uses LGBM Classififer
* No HPO
* FE shitty. Only computes momentus of relevant features with data leakage.

## <https://www.kaggle.com/its7171/cv-strategy>

* CV strategy
* Uses 5 last entries of each user as validation set
* 5 fold strategy
* Needs to be run in a local environment, kaggle doesn’t support it.
* <https://www.kaggle.com/its7171/time-series-api-iter-test-emulator> this notebook uses this cv strategy.
* Since the data provided is big enough, people are not using all the folds for training. Using one fold gives the same results than using the 5 of them.

## <https://www.kaggle.com/its7171/time-series-api-iter-test-emulator>

* The competition is evaluated on a time series API
* This notebook emulates the API
* This is useful for checking some things that cannot be observed using the real API:
  + Memory usage
  + Disk size consumed
  + The time it took to inference
  + Handling of New Users
  + Handling of not only questions but also lectures.
  + etc.

## <https://www.kaggle.com/its7171/can-we-trust-pandas-mean>

* Problem with pandas mean()
* If we use mean() using float32 the results are wrong. Need to use float64.
* It is an overflow problem since we have a lot of data

## <https://www.kaggle.com/c/riiid-test-answer-prediction/discussion/198245>

* Explains why it is better to use np arrays before creating lgbm datasets.
* It reduces memory spikes.

## <https://www.kaggle.com/spacelx/2020-r3id-clustering-question-tags>

* Clustering of question tags
* It finds a total of 5 clusters.
* The output is a csv that can be used in other notebooks.
* **I retrained the model setting a higher 'patience' for early stopping and it now converges at 0.7978, 0.7959 with this feature. People say it helps.**

# Kernels E2E

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

* **0.760 public**
* It uses only 1 fold, from the notebook of cv strategy
* The feature engineering is super simple

|  |
| --- |
| FEATS = [  "answered\_correctly\_avg\_u",  "answered\_correctly\_sum\_u",  "count\_u",  "answered\_correctly\_avg\_c",  "part",  "prior\_question\_had\_explanation",  "prior\_question\_elapsed\_time", ] |

* It seems that people are not using many features because of out of memory problems.

## <https://www.kaggle.com/zephyrwang666/riiid-lgbm-bagging2>

* **0.772 public 0.770 valid**
* Ohe of features type and part from lectures
* Generates agg features from groupby user\_id on content type id. We compute sum and count so then we can create features for cumulative sum and avg.
* Imputation seems ok
* Computes max timestamp per user
* Lagtime feature: groupby user\_id, shift timestamp, then subtract wrt current timestamp. It also computes lagtime\_mean.
* Delta\_prior\_question\_elapsed\_time feature: similar than before but with prior question elapsed time.
* It also computes features from the previous targets: user\_correctness user\_correctness\_cumsum, user\_correctness\_cumcount
* Features: explanation mean and explanation cumsum based on prior question had explanation.
* Feature attempt\_no.
* It computes more features related to the target, sum, count and var with different groupby (user id, content id and task container id).
* From the questions data extracts also features. Content correctness mean and var group by part. Content correctness group by bundle id.
* Splits the tags in different columns from tag1 to tag6. I thought there were more tags.
* Computes agg features grouping by the tags. Actually it only does it with tag1, don’t know why, It provides ohe tag features and then mean and std of correctness grouped only by tag 1. **Why not other tags?**
* No hpo
* It also tried tabnet and catboost but didn’t work as well as lgbm.
* When they train, they take a sample of train\_df (why?). Then from this sample, they take 0.025 users for the validation sample and remove that sample from the train. Then we merge questions\_df. Sample again from train to extract valid and append the new users valid to the latter.
* Create a loop to train the lgbm. In each iteration, create lgbm dataframes, train lgbm and plot importance.
* For the inference, first prepares features, then starts with the loop of the generator iter\_test. For each iteration, it does fe to mimic train features and data preprocessing. Once it has the test\_df ready, it uses model.predict. Then it subsample test df to extract the row\_id and the target predict. We call env.predict to this subsample. Once this is done, we can move to the next iteration of the env.
* **If change num to 5 or others num, we can run 5 times lightgbm. it can improve lb 0.002-0.003. But for RAM limit, sample less training data.**
* **cumsum/cumcount/count on target column should shift，other colum like prior\_question\_had\_explanation should not shift.**
* **I think there is data leakage.**

## <https://www.kaggle.com/zephyrwang666/riiid-lgbm-bagging2-1>

* **LB: 0.775 , valid: 0.775**
* Adds new features: 'lagtime2':'float32', 'lagtime3': 'float32',
* It loads all the train data, later it makes samples for training.
* There are some fillnan hardcoded.
* It uses the data generated form the notebook that creates the clusters of the question tags.
* **There are interesting comments that can be applied.**

## <https://www.kaggle.com/ammarnassanalhajali/riiid-lgbm-bagging2-sakt-0-781>

* **0.781 public,**
* It does not incorporate the things from the notebook 2.1, it is a fork from version 2.0
* The main difference is that it uses SAKT instead of lgbm
* After reading the paper of SAKT, it is clear that it is the way to go. Lgbm should not be a good approach for this type of problem.

## <https://www.kaggle.com/satorushibata/optimized-lightgbm-with-optuna-adding-sakt-model>

* **0.781 public,**
* Fork from notebook sakt2 0.781
* It does random sampling from train, every time it trains the models
* It uses optimized parameters obtained with optuna
* The parameters are optimised in this notebook (<https://github.com/satorushibata0627/Publishment/blob/main/Kaggle_Python3_LightGBM_on_GPU_with_Feature_Engineering_Optuna_and_Visualization.ipynb>)
* This notebook downloads the parameters.
* The final result is an ensembling of lgmb and sakt at 0.5 each. **It can be interesting to modify these weights to try to increase the score. We cannot do optimisation because we are using an API.**

## <https://www.kaggle.com/julianguo/fork-of-riiid-lgbm-bagging2-1-471152>

* **0.783 public**
* Seems an improved version of the previous notebook.
* **It uses a specific sample of train for the lgbm. Therefore, this removes the possibility of training more than one lgbm.**
* **The sakt model seems to be trained already. I think we are loading the sakt model trained in another notebook**