CS150A Database
Course Project

Pengyu Ji
ID: 2019533156
jipy@shanghaitech.edu.cn

Haoran Chu
ID: 2019533163
zhuhr@shanghaitech.edu.cn

Abstract

In our project, we basically followed the guideline provided by instructor and walked through Data Exploration and Cleaning, Feature Engineering, Model Choosing, Hyperparameter Tuning and Performance Comparison. Additionally, we use PySpark to make our Feature Engineering more efficient.

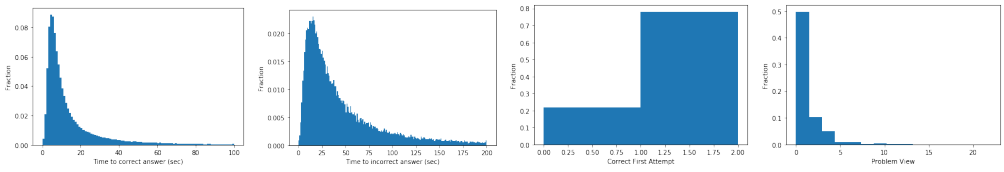
1 Explore the dataset

Because the dataset is huge, we focus mainly on several important columns as following: Correct_Step_Duration(sec), Error_Step_Duration(sec), Correct First Attempt and Problem View. We will provide the distributions in Table 1 and corresponding visualization in Figure 1.

Table 1: exploration

	count	mean	std	min	25%	50%	75%	max
Correct_Step_Duration(sec)	181599.0	17.9	35.2	0.0	5.0	8.0	17.0	1067.0
Error_Step_Duration(sec)	50853.0	60.5	89.3	0.0	16.0	32.0	67.0	1888.0
Correct First Attempt	232744.0	0.8	0.4	0.0	1.0	1.0	1.0	1.0
Problem View	232744.0	1.6	1.5	1.0	1.0	1.0	2.0	21.0

Figure 1: exploration



(a) correct answer (b) incorrect answer (c) correct first attempt (d) problem view

2 Data cleaning

In this part, the following work will be done:

Check whether missing data(NaN) exists For important columns: Anon Student Id, Problem Hierarchy, Problem Name, Problem View, Step Name, Correct First Attempt, there is no NaN exists except for NaN Correct First Attempt in test.csv for test. Some columns have NaN values but that makes sense. For example, if the student correctly finished the step at first attempt, the Error Step Duration should be NaN.

Check whether the special variable is unique The primary key of the data should be the combination of Anon Student Id, Problem Name, Step Name, Problem View, Step Duration (sec). Given such index, each row in train and index is unique.

Check and drop duplicate data For train and test, each row is unique with no duplicates.

Check whether the data and variable types are consistent and reasonable In this part, we focus on numerical_type columns. I set up a naive abnormal detection for train and test data. If the value is extremely higher than tempmax or lower than tempmin, then I will output an error. After my detection, every value can pass this naive test. It makes sense as well. The data in this project should be some kind of consecutive. For example, Problem View should increase each time by 1.

3 Feature engineering

In this part, our feature engineering procedure mainly contains four portions: Removing, Separating, Encoding and Calculating.

Removing Because several columns such as Incorrects, Hints, Corrects and so on are not provided by test, they should not be features to be considered, so we simply remove them. Moreover, according to the project description, Correct First Attempt should be train_label.

Separating In this part, we will divide column 'Problem Hierarchy' into its two components: Problem Unit and Problem Section.

Encoding Because of the existence of categorical features, we need to enocde them in order to generate the data matrix for further use. We use the simple encoding(start from 1 and increase 1 each time we see a new value). The columns need to be encoded are Anon Student Id, Problem Name, Problem Unit and so on. What needs our attention is that we should union train and test set together first, otherwise we will generate different encoding for the two.

Calculating In this part we will divide Calculating into Feature Compression and Feature Extraction.

Feature Compression In this part, we will compress columns KC(Default) and Opportunity(Default). Instead of simply encoding them, we will first separate them by `~~`, then correspondingly we will calculate the number of separated Knowledge_Components and the average of separated Opportunities.

Feature Extraction In this part, we will generate more high-level features for predicting. The new added features are as follows: Personal Correct First Attempt Count(Personal CFAC), Personal Correct First Attempt Rate(Personal CFAR), and by similar naming rule, Problem CFAR, Unit CFAR, Section CFAR, Step CFAR, KC CFAR are extracted by us. What needs our attention is that for generalization, we will consider step_name instead of (problem_name,step_name) pair when figuring out Step CFAR and for simplicity, we will consider original KC instead of the minimun KC CFAR of separated KC elements when figuring out KC CFAR. For any groupby_keys which don't show in train, we will use the average to estimate it.

Finally, we will have features as Table 2 and materialize two files: `train_pyspark.csv` and `test_pyspark.csv` based on our Feature Engineering work above.

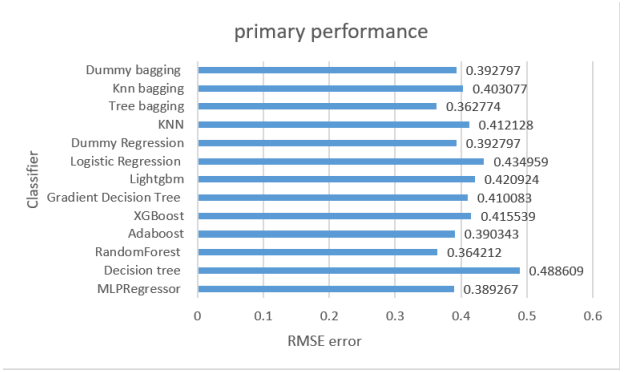
Table 2: features

Problem View	Correct First Attempt	Anon Student Id	Problem Name
Problem Unit	Problem Section	Step Name	KC Count
Opportunity Average	Personal CFAC	Personal CFAR	Problem CFAR
Unit CFAR	Section CFAR	Step CFAR	KC CFAR

4 Learning algorithm

We choose MLPRegressor, Decision tree, RandomForest, Adaboost, XGBoost, Lightgbm, Gradient Decision Tree, Logistic Regression, KNN, Bagging as models. When choosing the model, we consider the more the better and the wider the better. So our model includes parametric methods and nonparametric methods, individual models and ensemble learning models, Nerual Network and traditional methods. The primary performance without hyperparameter tuning is in Figure 2:

Figure 2: Primary Performance

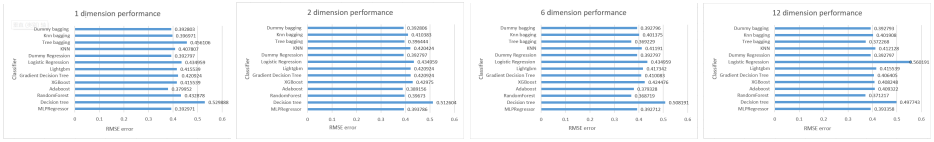


5 Hyperparameter selection and model performance

5.1 Dimension Reduction

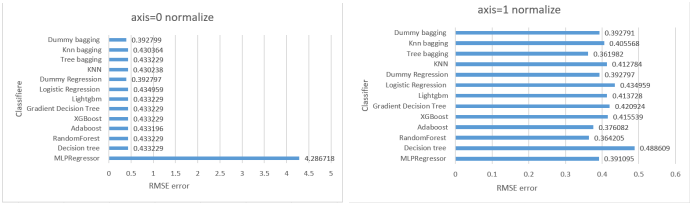
we first tried to use PCA and normalization to process data again in order to improve performance. However, we find that reducing the data into 1 dimension has already contained 99% of the information. After our consideration, we think that it may caused by several features such as Step CFAR which are too representative. We provide model performances under different kind of PCA and normalization by Figure 3 and Figure 4.

Figure 3: PCA performance



(a) 1 dimension (b) 2 dimension (c) 6 dimension (d) 12 dimension

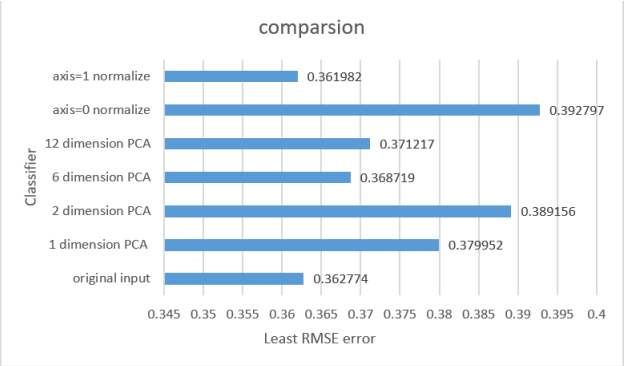
Figure 4: normalization performance



(a) axis=0 (b) axis=1

The final performance after dimension reduction or normalization is in Figure 5. We can find that only one kind of normalization is slightly better than using origin input, so considering that simple is best, we directly use origin input for further trainning.

Figure 5: Performance



5.2 Hyperparameter choosing

We can find that under most situations, Tree Bagging and RandomForestRegressor is better than other models, so we will do hyperparameter choosing on these two models. We will take the optimization of RandomForestRegressor in Table 3 as an example. The tool we mainly use is `GridSearchCV()`

Table 3: hyperparatmeter choosing

		range	best_parameters	optimzed error	better or not
step 1-4	n_estimators	range(10, 200, 10)	190	0.360495	yes
	max_depth	range(5, 21)	15	0.35687	yes
	max_leaf_nodes	range(100, 1000, 100)	900	0.353475	yes
	min_samples_split	range(2, 52, 2)	22	0.353875	no

After our hyperparameter choosing, we provide the comparison between the performance under our chosen hyperparameters and under sub-optimal hyperparameters in Table 4.

Table 4: final performace

	original	optimized
RandomForestRegressor	0.364212	0.353475
Tree Bagging	0.362774	0.353

Because their losses are almost the same and BaggingRegressor has a bad efficiency, we use optimized RandomForestRegressor to make our output.

6 PySpark implementation (optional)

Our project mainly applied PySpark in Feature Engineering. Due the large size of trainning data, using distributed processing methods such as PySpark will greatly improve the efficiency. We use `pyspark.sql.SQLContext` to create dataframe objects and do further process on them. The general steps for our PySpark implementation are as follows: 1. Choose column which needs to be modified. 2. Design specific modify function. 3. Use `pyspark.sql.function` to convert our function in step 2 into user_defined_function(udf) 4. Apply udf function to corresponding column. Additionally, some union and groupby work also need to be done.

7 Reference

[1] Hsiang-Fu Yu et al. (2010). Feature Engineering and Classifier Ensemble for KDD Cup 2010.

[2] Yongming Shen et al. (2010). Predicting Student Performance: A Solution for the KDD Cup 2010 Challenge.