CS150 Database and Data Mining Course Project

Qiu Longtian
ID: 2018533107
qiult@shanghaitech.edu.cn

Shi Qianjing
ID: 2018533194
shiqj@shanghaitech.edu.cn

1 Explore the dataset

Before we train our model, we need to analyze the data. In the original data set, the total 19 columns can be categorized into 2 types: categorical features, such as: Problem Hierarchy, Problem Name, etc, and numerical features, such as: Step End Time, Correct First Attempt and etc.And we will add new features by the original numerical features.

1.1 Data Structure:

For data separating, we found that Problem Hierarchy is a combination of Problem Unit and Problem Section. So it can be seperated to 2 parts. Therefore, there are 20 columns in all. And we have also found that the Opportunity is connected with $\sim\sim$ when it has more than one KC. We can separate them by $\sim\sim$.

1.2 Train Data:

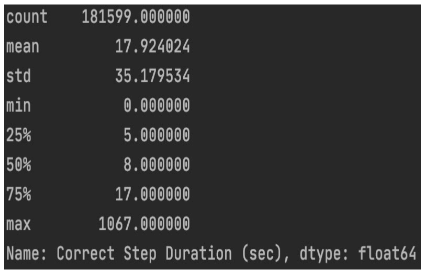
The goal is to predict the answer of Correct First Attempt(CFA) for given data, so we just focus on the relation between CFA and other columns. For example, for CFA = 1, let's look at "Step Duration (sec)", which counts 181599. Mean is about 17.9 ,and std is about 35.2. 3rd quartile(Q3) is 17. So we may consider that if a student's Step Duration is larger than 53, his CFA is very likely be 0.

1.3 Test Data:

Since the training data is very comprehensive while some of them are useless in test data. In test data, only these columns have value, which are Row, Anon Student Id, Problem Hierarchy, Problem Name, Problem View, Step Name, Correct First Attempt, KC(Default), Opportunity(Default).

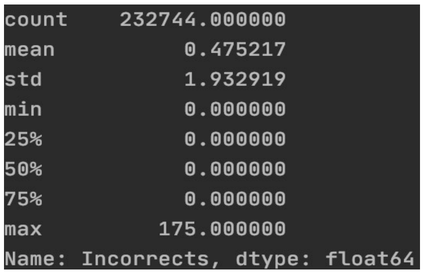
2 Data cleaning

The first thing we do in data cleaning is to remove the meaningless columns and here is a list of columns which marks meaningless in this problem ["Row", "First Transaction Time", "Step Start Time", "Correct Transaction Time", "Step End Time", "Step Duration (sec)", "Error Step Duration (sec)", Hints]. These columns will not be used in following steps. For the rest of meaningful columns ["Anon Student Id", "Problem Hierarchy", "Problem Name", "Problem View", "Step Name", "Correct Step Duration (sec)", "Correct First Attempt", "Incorrects", Corrects, KC(Default)", Opportunity(Default)], we decide to pick out reasonable rows.
According to the describe() function in pandas library, we find there are two columns are meaningful but have outliers["Correct Step Duration (sec)", Incorrects, Corrects]. The result of describe() function is shown in Figure1 and Figure2. So we decide to remove the outlier by the checking whether the delta of the value and it mean value is larger than 10 times its standard deviation. The formula is "if |value - mean of value| > standard deviation of value, then remove value". By this



```
count    181599.000000
mean         17.924024
std          35.179534
min           0.000000
25%           5.000000
50%           8.000000
75%          17.000000
max        1067.000000
Name: Correct Step Duration (sec), dtype: float64
```

Figure 1: Distribution of column CSD



```
count    232744.000000
mean          0.475217
std           1.932919
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max         175.000000
Name: Incorrects, dtype: float64
```

Figure 2: Distribution of column Incorrects

way, we removed about 800 rows in total.
Besides, we also check whether the data in a row is inconsistent. The rule for inconsistent is defined by observation. If the Correct First Attempt value of a row is 1, then the Correct Step Duration (sec) of the row is not a nan value. Any row violate the rule above will be marked as inconsistent and be removed. However, the result shows there is no inconsistent row in the training dataset.
Given the column Problem Hierarchy consist of two subpartSection and Unit. We separate the column Problem Hierarchy into two columnsProblem Section and Problem Unit.
At last, after the training data is processed by our data cleaning, there are 11 columns remaining ['Anon Student Id', 'Problem Section, 'Problem Unit, 'Problem Name', 'Problem View', 'Step Name', 'Correct Step Duration (sec)', 'Correct First Attempt', 'Incorrects', 'KC(Default)', 'Opportunity(Default)'].

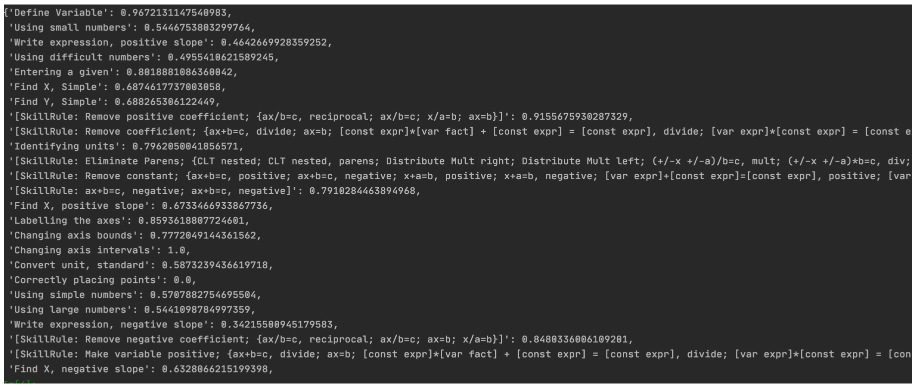
3 Feature engineering

Feature engineering is the most important part in solving the problem. And we are trying to generate the most relevant features to the predicting column Correct First Attempt from the data in training data set.
Here we first deal with the categorial columns first. Nowadays there are several popular schema to encode categorial attributes, such as One-hot Encoding Scheme, Dummy Coding Scheme, Effect Coding Scheme, Bin-counting Scheme and Feature Hashing Scheme. Here we choose to try two of the schemas, which are One-hot Encoding Scheme and Feature Hashing Scheme. The implement of these two encode schema is done by the library in sklearn.feature_extraction and sklearn.preprocessing. Then we explore the unique value of these categorial columns, the result is shown in Table 1.

Anon Student Id	Problem Section	Problem Unit	Problem Name	Step Name
174	138	32	1021	60624

Table 1: Unique number of categorial columns value

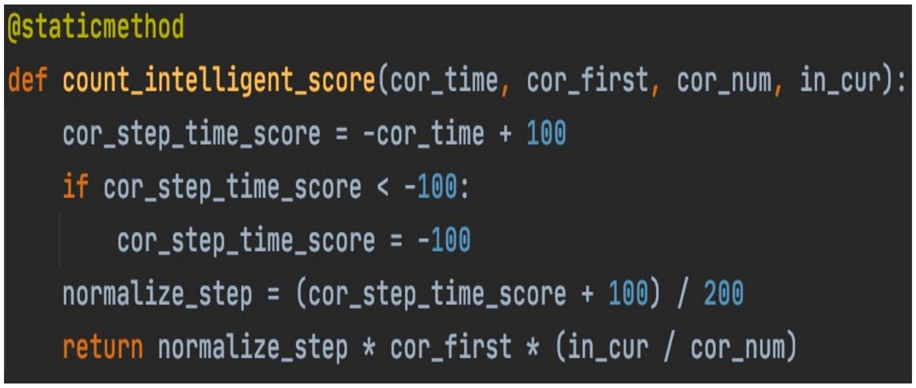
If we choose to do one hot encoding on all of the categorial columns, we will get 174*138*32*1021*60624 = 1.74e16 features, which is obviously too large to train, so we turn to Feature Hashing Scheme to get a fixed number of features. The actual number of feature generated by the hashing encoder is treated as an hyperparameter and will be decided later in Hyperparameter selection and model performance section. After we generate features for categorial columns, we treat column Problem Viewdirectly as an feature since its value is related to the Correct First Attempt. Now we will process more complex columns-KC(Default) and Opportunity(Default). Since there may be more than one knowledge component and corresponding opportunity value in one row. First, we build up a table of difficulty of a knowledge component, the difficulty of a knowledge component is defined by $\frac{the\ number\ of\ rows\ where\ the\ knowledge\ component\ appear\ and\ the\ value\ of\ Correct\ First\ Attempt\ equals\ one}{the\ number\ of\ rows\ where\ the\ knowledge\ component\ appear}$.
The result table is like a dictionary as shown in figure 3.



```
{'Define Variable': 0.9672131147540983,
 'Using small numbers': 0.5446753803299764,
 'Write expression, positive slope': 0.4642669928359252,
 'Using difficult numbers': 0.4955410621589245,
 'Entering a given': 0.8018881086360042,
 'Find X, Simple': 0.6874617737003058,
 'Find Y, Simple': 0.688265306122449,
 '[SkillRule: Remove positive coefficient; {ax/b=c, reciprocal; ax/b=c; x/a=b; ax=b}]': 0.9155675930287329,
 '[SkillRule: Remove coefficient; {ax+b=c, divide; ax=b; [const expr]*[var fact] + [const expr] = [const expr], divide; [var expr]*[const expr] = [const e
 'Identifying units': 0.7962050041856571,
 '[SkillRule: Eliminate Parens; {CLT nested; CLT nested, parens; Distribute Mult right; Distribute Mult left; (+/-x +/-a)/b=c, mult; (+/-x +/-a)*b=c, div;
 '[SkillRule: Remove constant; {ax+b=c, positive; ax+b=c, negative; x+a=b, positive; x+a=b, negative; [var expr]+[const expr]=[const expr], positive; [var
 '[SkillRule: ax+b=c, negative; ax+b=c, negative]': 0.7910284463894968,
 'Find X, positive slope': 0.6733466933867736,
 'Labelling the axes': 0.8593618807724601,
 'Changing axis bounds': 0.7772049144361562,
 'Changing axis intervals': 1.0,
 'Convert unit, standard': 0.5873239436619718,
 'Correctly placing points': 0.0,
 'Using simple numbers': 0.5707882754695504,
 'Using large numbers': 0.5441098784997359,
 'Write expression, negative slope': 0.34215500945179583,
 '[SkillRule: Remove negative coefficient; {ax/b=c, reciprocal; ax/b=c; ax=b; x/a=b}]': 0.8480336006109201,
 '[SkillRule: Make variable positive; {ax+b=c, divide; ax=b; [const expr]*[var fact] + [const expr] = [const expr], divide; [var expr]*[const expr] = [con
 'Find X, negative slope': 0.6328066215199398,
```

Figure 3: Result table

Then we are able to generate a feature called knowledge difficulty based on our knowledge component difficulty table. To be more specific, the value of feature knowledge difficulty equals to the sum of knowledge components appear in the row divided by the number of knowledge component appear in the row. We also treat the number of knowledge component in a row as a feature either.
Opportunity is related to the knowledge component, so we generate a feature called opportunity value. Since there may be more than knowledge component and corresponding opportunity in one row, we defined opportunity value equals the sum of the difficulty of knowledge component times corresponding opportunity number, then divided the result by sum of opportunity numbers. $\frac{sum(KC\_i*OPPO\_i}{sum(OPPO\_i)}$, where KC_i means ith knowledge component and OPPO_i means ith opportunity number. If the knowledge component is not in the knowledge component difficulty table, we will assign the mean value of knowledge component difficulty table to it. Afterwards, we construct two table called person intelligent table and step difficulty table similar to knowledge difficulty table. The person intelligent table is designed to estimate how smart a participate is. The person intelligent table is defined by a function which takes mean values of one participates "Correct Step Duration (sec)", "Correct First Attempt", Incorrects" and Corrects and output a score.



```
@staticmethod
def count_intelligent_score(cor_time, cor_first, cor_num, in_cur):
    cor_step_time_score = -cor_time + 100
    if cor_step_time_score < -100:
        cor_step_time_score = -100
    normalize_step = (cor_step_time_score + 100) / 200
    return normalize_step * cor_first * (in_cur / cor_num)
```

Figure 4: Person intelligent table

With person intelligent table in figure 4, we generate the feature called person intelligent which is defined by the value of person intelligent table given an Anon Student Id. With respect to step difficulty table, it estimate the difficulty of a problem step which is defined by the number of rows one problem step appear and the value of Correct First Attempt equals 1 divided by the number of rows one problem step appear. Then we generate a feature called step difficulty based on the value of step difficulty table given a Step Name.
In conclusion, there are number of hash output * 5 + 5 features listed below.

(1) **Anon Student Id**

(2) **Problem Section**

(3) **Problem Unit**

(4) **Problem Name**

(5) **Step Name**

(6) **Knowledge Difficulty**

(7) **Person Intelligent**

(8) **Knowledge Component Number**

(9) **Step Difficulty**

(10) **Opportunity Value**

4 Learning algorithm

4.1 Learning Algorithm Methods and Description

After we have transformed some data into numerical data, we can start training and testing. After reading many papers, we have tried several methods for our prediction: AdaBoost classifie, GradientBoostingClassifier, BaggingClassifier, RandomForestClassifier, VotingClassifie, HistGradientBoostingRegressor, RandomForestRegressor, HistGradientBoostingRegressor, AdaBoostRegressor to know which one is better.

(1) AdaBoost classifier: An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

(2) GradientBoostingClassifier: GB builds an additive model in a forward stage-wise fashion; It allows for the optimization of arbitrary differentiable loss functions.

(3) BaggingClassifier: A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction.

(4) RandomForestClassifier: A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

(5) HistGradientBoostingRegressor: This estimator has native support for missing values (NaNs). During training, the tree grower learns at each split point whether samples with missing values should go to the left or right child, based on the potential gain.

(6) RandomForestRegressor: A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

(7) HistGradientBoostingRegressor: This estimator is much faster than GradientBoostingRegressor for big datasets.

(8) AdaBoostRegressor: An AdaBoost regressor is a meta-estimator.

(9) RandomForestRegressor: A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

4.2 Methods Result

And the RMSE result is shown in table 2. According to the performance of learning algorithm result , we decide to try combination of different algorithm. The implementation is realized by the function VotingRegressor(). The result is shown in table 3.

Method	RMSE
AdaBoost	0.41222
GradientBoostingRegressor	0.41221
GradientBoosting	0.41404
Bagging	0.44889
RandomForest	0.42302
HistGradientBoostingRegressor	0.39356
HistGradientBoosting	0.40856
AdaBoostRegressor	0.42480
RandomForestRegressor	0.44384

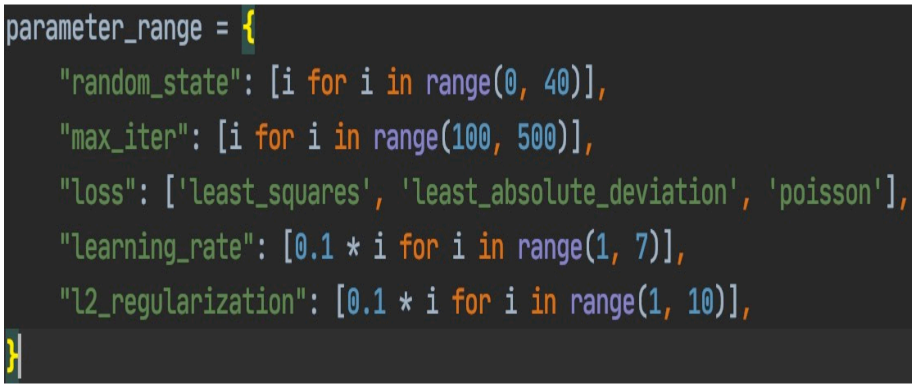
Table 2: The result of nine algorithms

GradientBoostingRegressor&HistGradientBoostingRegressor	0.40671
GradientBoostingRegressor&RandomForestRegressor	0.41585
HistGradientBoostingRegressor&RandomForestRegressor	0.39736

Table 3: The result of each combination

5 Hyperparameter selection and model performance

The learning algorithm we choose is HistGradientBoostingRegressor which has the best performance among all the single or combination learning algorithm. And now we need to find the best hyperparameter for the HistGradientBoostingRegressor. The main parameters are loss function, learning rate, L2 regularization parameter, pseudo-random number generator to control the subsampling, maximum number of iterations. The rest parameter will be remained as default value given by sklearn library. Since the inner relation between the parameters are unknown for us. We adopt an random testing method. First we define the reasonable range for the hyperparameter. The range is shown in figure 5.



```
parameter_range = {
    "random_state": [i for i in range(0, 40)],
    "max_iter": [i for i in range(100, 500)],
    "loss": ['least_squares', 'least_absolute_deviation', 'poisson'],
    "learning_rate": [0.1 * i for i in range(1, 7)],
    "l2_regularization": [0.1 * i for i in range(1, 10)],
}
```

Figure 5: Parameter Range

And the best result is given by parameters in table 4.

For the suboptimal parameters, we can easily find out the performance is worse than the optimal parameters in table 5.

6 PySpark implementation (optional)

Nowadays the scale of data is growing rapidly with the development of digital technology. We decide to use pyspark to accelerate the process of processing data. In this project, given that the operation of processing data is complex such as split the words by and compute the weighted sum value of a row. So we implement a function which compute the mean value of an column in the pandas data frame. The first step is to temporarily store the data frame as a .csv file. Then initialize a SparkSession called the Spark. Use the Spark to read the data from csv file and create an view

random_state	max_iter	loss	learning_rate	l2_regularization	RMSE
1	331	least_squares	0.4	0.2	0.39356

Table 4: Best Result

random_state	max_iter	loss	learning_rate	l2_regularization	RMSE
4	415	least_absolute_deviation	0.4	0.8	0.41585
39	175	least_absolute_deviation	0.3	0.7	0.42124
32	368	least_absolute_deviation	0.5	0.7	0.41404
3	177	least_squares	0.1	0.6	0.40113
19	470	least_squares	0.1	0.8	0.42124
5	177	least_absolute_deviation	0.2	0.1	0.42302
38	267	least_absolute_deviation	0.2	0.6	0.42480

Table 5: Result of Different Hyperparameter

called train. At this point, we use SQL query to acquire the mean value of the columns and output the result to json format. At last, we read the mean value from the json.