CS150 Database and Data Mining
Course Project

Yuting Zheng
ID: 2018533089
zhengyt1@shanghaitech.edu.cn

Shaoting Peng
ID: 2018533245
pengsht@shanghaitech.edu.cn

Guideline

Compared with developing a novel machine learning algorihtm, building a machine learning system is less theoretical but more engineering, so it is important to get your hands dirty. To build an entire machine learning system, you have to go through some essential steps. We have listed 5 steps which we hope you to go through. Read the instructions of each section before you fill in. You are free to add more sections.
If you use PySpark to implement the algorithms and want to earn some additional points, you should also report your implementation briefly in the last section.

1 Explore the dataset

The given data are further explored to find the potential correlation among data. The following are the most useful data feature we've found.
As for the relationship between Correct First Attempt(CFA in the report) and Step Duration (sec) per Problem and apparently, if Step Duration tend to longer, the CFA tend to be incorrect.

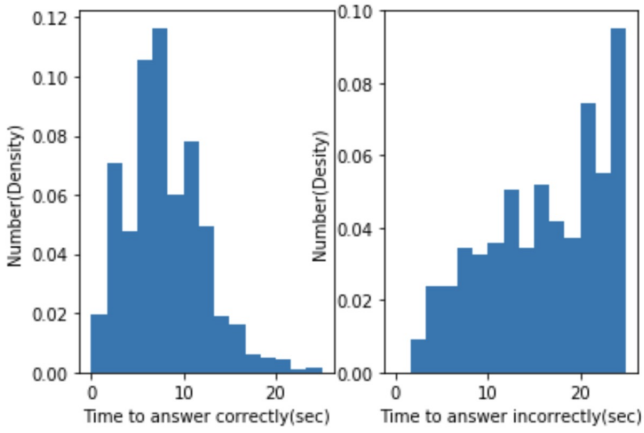

Figure 1: explore time

The relationship between CFA and KC number is also important. we found that

```
when KC number is 0, the correct first attempt rate is 0.8150367057632267, which is 48295 / 59255.
when KC number is 1, the correct first attempt rate is 0.8053092916774393, which is 96559 / 119903.
when KC number is 2, the correct first attempt rate is 0.7997563224446786, which is 24287 / 30368.
when KC number is 3, the correct first attempt rate is 0.5402350008901549, which is 12138 / 22468.
when KC number is 4, the correct first attempt rate is 0.6431535269709544, which is 465 / 723.
when KC number is 5, the correct first attempt rate is 0.5185185185185185, which is 14 / 27.
```

Figure 2: explorekc num

indicating more KC number decline CFA. Also we find there are 110 KC(exclude NaN), and each KC has different CFA, which may influence the final result. Then we explore student, which has a total number of 174, and compute different CFA of each student. The same exploration is also done to 'problem'. For simplicity, we only display one of this kind features.

```
Number of unique student:  174

{'02i5jCrfQK': 0.777126540426811,
 '0GlR30c2Mt': 0.7506134969325153,
 '0KS4yy9G96': 0.8318335208098988,
 '0nF0z1401O': 0.7986798679867987,
 '12M70dm49w': 0.7751493428912783,
 '162lfGskK2': 0.828125,
 '16zN4n2fFO': 0.8068571428571428,
 '183ppvOXp8': 0.8537782139352306,
 '1bJbgQ32E3': 0.7335038363171356,
 '1k3GIfwBw6': 0.8444444444444444,
 '23T7Up1oQ6': 0.875,
 '242fhbKRf9': 0.7093660890546173,
 '276487SwvF': 0.7806513409961686,
 '29U6L12O3k': 0.6891891891891891,
 '2kb387yOiG': 0.7867892976588629,
 '2x0Z9q6jD0': 0.6624365482233503,
 '3008m5T78s': 0.6294363256784968,
```

Figure 3: explore person

Through observe the opportunity, Problem View, Problem Hierarchy, Problem name, we find it relate little toward CFA, so they are ignored in our implementation.

2 Data cleaning

It's observable that NaN may occurs when computing CFA for each step time. By viewing it manually, we find most of them simple and correct, so we fill these NaN with median step time. Other features encounting NaN, we fill them with mean value of other not-NaN value. For example, Problem 'BH1T18' in the testdata is not in the traindata, so it's filled with mean CFA.

More details about data cleaning can be found in the next section.

3 Feature engineering

In addition to the original features, more features are calculated and feed into the learning model. Generally, 4 features are newly calculated, which are CFAR(CFA divided by the total number of lines) for each 'Anon Student Id', CFAR for each 'Step Name', CFAR for each 'KC(Default)' and CFAR for each 'Problem Name'.
The way to calculate them are as follows. KC are used as an example. As we iterate through the training dataset, we maintain a dictionary, where the keys are each unique KC, and the value is a tuple containing the sum of CFAs and the sum of transactions, which has the following form:

$$KC\_dic[' EditAlgebraick'] = (174, 248)$$

For those who have more than one KC, we split them into a list of KC units, and multiply the probabilities together.

Many marginal situations are also taken into consideration. Besides the CFAR for each columns above, we also calculated the **average** CFAR, which can be used when KC equals to NaN; Observed that many 'Step' are only encountered once or twice, which would give a huge bias to our model, **Laplace smoothing** with k = 2 has been implemented. As a result, those steps that are only

encountered once now have a probability of 0.8, which approximates to the average CFAR calculated before. Thus potential overfitting problem is avoided.

To feed into our model, the original training dataset are split into training set with size 220000, and validation set with size 12744. Training labels and validation labels are the corresponding CFA in each transaction.

4 Learning algorithm

Logistic Regression is taken into consideration first because the problem is two classification problem. However, maybe because the complexity or the dirtiness of the data, the final LogicalClassifier result define by $np.sqrt(mean\_squared\_error(y\_pred, yy))$ is 0.36332021103646334. But it's good compare to other models.

We use Random Forest for a try because rf is usually a good model for various data and a great baseline for many scientific research. The RandomForest result define by $np.sqrt(mean\_squared\_error(y\_pred, yy))$ is 0.3583585863954912

Gradient Boosting Decision Tree is also implemented for try. Among the traditional machine learning algorithms, it is one of the best algorithms for fitting real distribution. Before deep learning becoming popular in previous years, GBDT was brilliant in various competitions. Most importantly, GBDT can Screening features, so we can gain the useful feature and try it in other model. After experiment result define by $np.sqrt(mean\_squared\_error(y\_pred, yy))$ is 0.35528424518488205

We also tried AdaBoost, but with an accuracy of 0.4022577513799062, It performs bad.

We separately use different features to train the model and get accuracy in validation set. There are many feature we find important in data exploration: CFAR

We find many features like opportunity, Problem View, Problem Hierarchy, Problem name have little relation with the RMSE, so we remove some of it and try many times.

A neural network implemented by PyTorch has been considered as our primary option to learn the parameters. We have tried many structures for our network, such as the number of hidden layers and perceptrons, different loss functions (MSELoss, CrossEntropyLoss, etc.) and different optimizer (Adam, SGD, Adagrad, etc.). Finally a structure of 4 linear layers has been chosen. As for loss function, we use **MSELoss** because of the correlation to evaluation target RMSE, and use Adam as optimizer. More details about our network are described below.

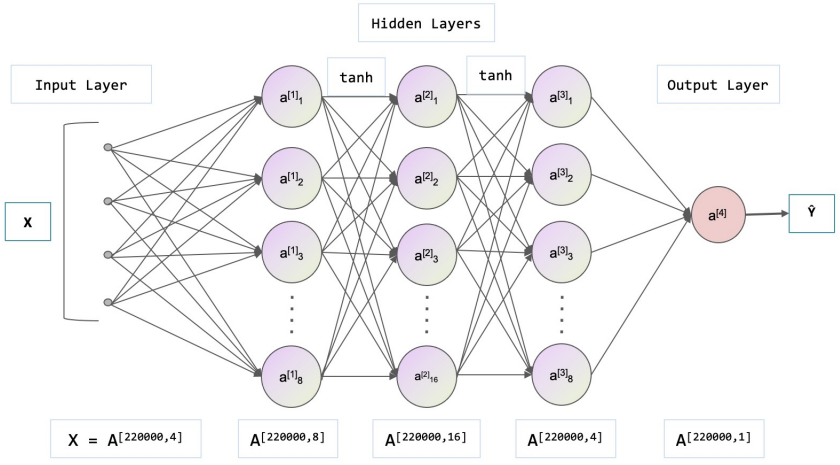

Figure 4: neural network

The first layer receives a [220000, 4] input feature, and extend its dimension to [220000, 8], which can receive a better underlying correlation between the features. Then, a *tanh* non-linear activation function has been implemented, followed by another linear layer of dimension [220000, 16]. The final output of the neural network should have dimension [220000, 1], so 2 additional linear layers are added, which have dimension [220000, 4] and [220000, 1]. All of them are connected using *tanh* non-linear activation function.

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

5 Hyperparameter selection and model performance

Beside the parameters learned by the network shown above, 3 additional parameters(including but not limited to hyperparameters) are taken into consideratoin: learning rate *lr*, number of training epoch, and the threshold to map the output probability into 0 or 1, which will be discussed separately. As for the first hyperparameter, *lr*, 0.01 has been chosen for the reason that 0.001 is too slow for training loss to converge and 0.1 may cause the model to diverge. 3000 epochs are chosen experimentally, which can reach a relatively low MSELoss and avoid overfitting in the meantime.

The threshold is used to map the output probability into 0 or 1. The output of the network is one dimension, and hopefully, it's between 0 and 1. So we iterate through the possible thresholds *th* with a step length of 0.05, from 0.1 to 0.9, and add 0 to the answer list if $th \geq output$, add 1 otherwise. In out experiments, a threshold of 0.54 can get the best result of an accuracy of **86.4%** on the 666 transactions with given CFA in the test set. And for evaluation, RMSE:

$$\sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y_i})^2}$$

equals to **0.36** if $\hat{y_i}$ is 0 or 1 instead of the output of network.

6 PySpark implementation (optional)

PySpark is a Python API for Spark, and is widely used in big data computing. PySpark shines when it comes to deal with operations that are **simple**, **repetive**, and **requires lots of data**. In our implementation, we use PySpark to deal with CFA operations(such as CFA of Person, CFA of Problem, CFA of Step). Computing CFA is quite simple, and needed to be done 4×232744 times, which suits for PySpark well. To start with, Sparkcontext is the entry point to any spark function. When we run any spark application, we start a driver with the main function and Sparkcontext is started here. The driver then runs operations within the executor on the work node, which make computing CFA faster.