Machine Learning Engineer Nanodegree

Capstone Project

Michael Feng
June 8th, 2017

I. Definition

Project Overview

A lot of universities are currently using Artificial Intelligent Tutoring systems. These systems are used to tutoring students on some courses. Students can acquired knowledge from remote and anytime they want. The time and hours students spend on the system are valuable for both students and universities. Analysis of these behaviors can be a good practice, so that Universities can use students' behaviors on tutoring system to optimize their tutoring systems and exercises.

Carnegie Learning systems[1] are mainly focusing on transforming math classroom to both students and teachers. A course on Carnegie Learning system tutoring algebra deployed 2005- 2006 and 2006-2007. It's used to tutoring Algebra knowledge. Our project data comes from Carnegie Learning system tutoring algebra teaching system, which contains about 8918055 records, collected between 2008 and 2009. It's opening to internet.

Stamper, J., Niculescu-Mizil, A., Ritter, S., Gordon, G.J., & Koedinger, K.R. (2010). Algebra I 2008-2009. Challenge data set from KDD Cup 2010 Educational Data Mining Challenge. Find it at http://pslcdatashop.web.cmu.edu/KDDCup/downloads.jsp

From a scientific viewpoint, the ability to achieve low prediction error on unseen data is evidence that the learner has accurately discovered the underlying factors which make items easier or harder for students. Knowing these factors is essential for the design of high-quality curricula and lesson plans (both for human instructors and for automated tutoring software).

Also, improved models could be saving millions of hours of students' time (and effort) in learning. And we have the potential to influence lesson design, improving retention, increasing student engagement, reducing wasted time, and increasing transfer to future lessons.

Problem Statement

The students solve problems in the tutor and each interaction between the student and computer is logged as a transaction. Our data are records of transactions(which we've mention before). A record is a step summary of all of a given student's attempts info for given exercise problem.

Training data are records of different students's transactions on different exercises problems. First, We are gonna use thousands of student-record (a record of a student working on a step) to predict students' correct first attempt on problems. CFA(Correct First Attempt): a binary number, which is 0 or 1. A CFA(Correct First Attempt) value indicates the result of a student's correct first attempt on one problem. Then, the probability values of different students' first correct attempt on different problems are going to be predicted as our final prediction result.

CFA(Correct First Attempt) is our prediction target. Also CFA(Correct First Attempt) is 0 or 1 makes the problem likely to be a classification problem. We will evaluate model's performance metric like log loss on unseen data for measurement.

Metrics

Our the data comes from KDD CUP 2010 competition. Therefore, we don't have an unidentified portion to validate model's generalize performance.

So we will only train on the training portion of each data set, meanwhile, use part of training portion data as validation set, and will then be evaluated on our performance at providing correct first attempt values for the test portion(part of training data).

Accuracy is not always a good indicator because of its yes or no nature. The use of log loss is very common and it makes an excellent general purpose error metric for numerical predictions. Log loss takes into account the uncertainty of your prediction based on how much it varies from the actual label. This gives us a more nuanced view into the performance of our model. Also, Log Loss quantifies the accuracy of a classifier by penalising false classifications. Minimising the Log Loss is basically equivalent to maximising the accuracy of the classifier.

So, We will compare the predictions we provided against test portion(part of training data) true values and calculate the difference as log loss. Use Log loss as our main evaluation metric.

Logarithmic Loss can be represented by the following formula.

$-\log P(y'|y) = -(y'\log(y) + (1 - y') \log(1 - y))$

y': data's predicted label value

y: data's actual label value

Using sklearn calculate log loss with following code snippet:

```
from sklearn import metrics

print("Log Loss: ", metrics.log_loss(y_test, y_pred))
```

For more log loss details, find it at reference[8] and reference[9].

II. Analysis

Data Exploration

We load and explore data with pandas. Try to see what's the data type and value, also what's inside the data. It's about 8918055 records in our algebra_2008_2009_train.txt file.

There are about 23 columns in data. Column names are as follows:

```
['Row', 'Anon Student Id', 'Problem Hierarchy', 'Problem Name',
       'Problem View', 'Step Name', 'Step Start Time',
       'First Transaction Time', 'Correct Transaction Time', 'Step End Time',
       'Step Duration (sec)', 'Correct Step Duration (sec)',
       'Error Step Duration (sec)', 'Correct First Attempt', 'Incorrects',
       'Hints', 'Corrects', 'KC(SubSkills)', 'Opportunity(SubSkills)',
       'KC(KTracedSkills)', 'Opportunity(KTracedSkills)', 'KC(Rules)',
       'Opportunity(Rules)']
```

Some specific columns are described below：
• Step: an observable part of the solution to a problem.
• Transaction: students’ interaction with tutoring system. For e.g., hint request, incorrect at-
tempt, or correct attempt. Every transaction is referred to as an attempt for a step.
• Solution: All steps comprises to figure problem out.
• Problem: A task for a student to perform that typically involves multiple steps.
• Answer: A final step to one problem
• Knowledge Component(KC): A piece of knowledge or information could be used to accomplish a problem. A KC is associated with one or more steps. Also one or more KC can be
associated with one step.
• Opportunity: A chance for student, representing he or she has acquired of the KC that are
needed to accomplish a problem.
• Opportunity count: Steps(containsKC)count for student that are needed to solvea problem. So, student can have multi chances to solve a problem.

Use following code, we can see the data’s variation about students, problems and knowledge modules.

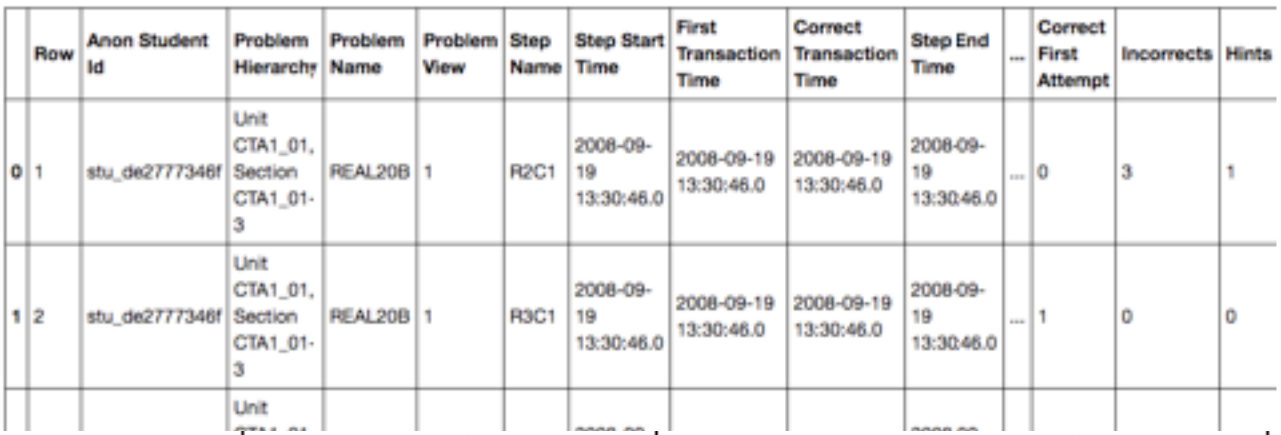
```
# Print the number of unique students' id
print 'Number of unique students: ', len(np.unique(traindata['Anon Student Id']))

# Print the number of unique problems
print 'Number of unique problems: ', len(np.unique(traindata['Problem Name']))
```

```
# Print the number of KC(SubSkills)
print 'Number of unique KC(SubSkills): ', len(np.unique(traindata['KC(SubSkills)']))
# Print the number of KC(KTracedSkills)
print 'Number of unique KC(KTracedSkills): ', len(np.unique(traindata['KC(KTracedSkills)']))
# Print the number of KC(Rules)
print 'Number of unique KC(Rules): ', len(np.unique(traindata['KC(Rules)']))

# Print the number of total records
print 'Number of total records: ', len(traindata)
```

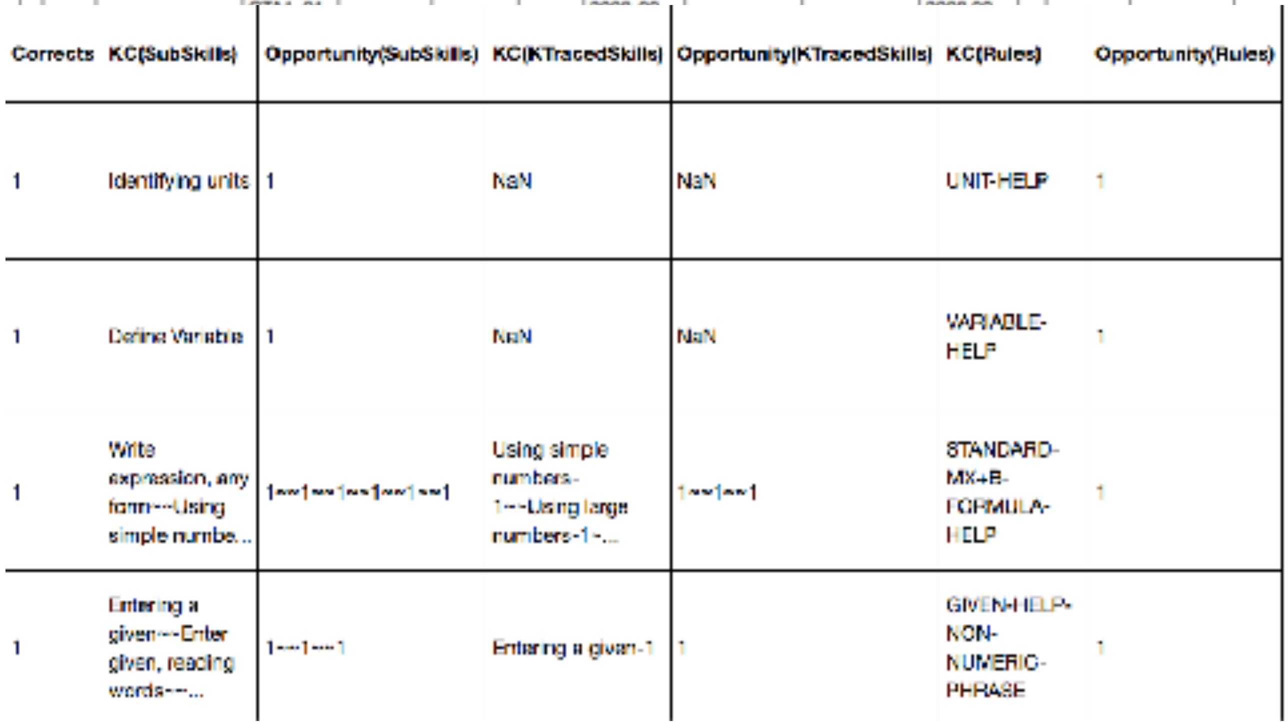
Output:

```
Number of unique students:  3310
Number of unique problems:  188368
Number of unique KC(SubSkills): 1829
Number of unique KC(KTracedSkills): 922
Number of unique KC(Rules):  2979
Number of total records:  8918054
```

Following images are data samples, we use pandas to show us.



	Row	Anon Student Id	Problem Hierarchy	Problem Name	Problem View	Step Name	Step Start Time	First Transaction Time	Correct Transaction Time	Step End Time	...	Correct First Attempt	Incorrects	Hints
0	1	stu_de2777346f	Unit CTA1_01, Section CTA1_01-3	REAL20B	1	R2C1	2008-09-19 13:30:46.0	2008-09-19 13:30:46.0	2008-09-19 13:30:46.0	2008-09-19 13:30:46.0	...	0	3	1
1	2	stu_de2777346f	Unit CTA1_01, Section CTA1_01-3	REAL20B	1	R3C1	2008-09-19 13:30:46.0	2008-09-19 13:30:46.0	2008-09-19 13:30:46.0	2008-09-19 13:30:46.0	...	1	0	0
			Unit											



Corrects	KC(SubSkills)	Opportunity(SubSkills)	KC(KTracedSkills)	Opportunity(KTracedSkills)	KC(Rules)	Opportunity(Rules)
1	Identifying units	1	NaN	NaN	UNIT-HELP	1
1	Define Variable	1	NaN	NaN	VARIABLE-HELP	1
1	Write expression, any form---Using simple numbe...	1~~1~~1~~1~~1~~1	Using simple numbers-1---Using large numbers-1-...	1~~1~~1	STANDARD-MX+B-FORMULA-HELP	1
1	Entering a given---Enter given, reading words---...	1---1---1	Entering a given-1	1	GIVEN-HELP-NON-NUMERIC-PHRASE	1

Using pandas df.types show our data’s type:

```
Row                               int64
Anon Student Id                   int64
Problem Hierarchy                 int64
Problem Name                      int64
Problem View                      int64
Step Name                         int64
Step Start Time                   int64
First Transaction Time            int64
Correct Transaction Time          int64
Step End Time                     int64
Step Duration (sec)             float64
Correct Step Duration (sec)     float64
Error Step Duration (sec)       float64
Correct First Attempt             int64
Incorrects                        int64
Hints                             int64
Corrects                          int64
KC(SubSkills)                     int64
Opportunity(SubSkills)            int64
KC(KTracedSkills)                 int64
Opportunity(KTracedSkills)        int64
KC(Rules)                         int64
Opportunity(Rules)                int64
CFAR                            float64
year                              int64
month                             int64
day                               int64
dtype: object
```

Using pandas df.isnull().sum(), we can see the number of records which contains NaN value is too large. We can't just drop them all.

df.isnull().sum():

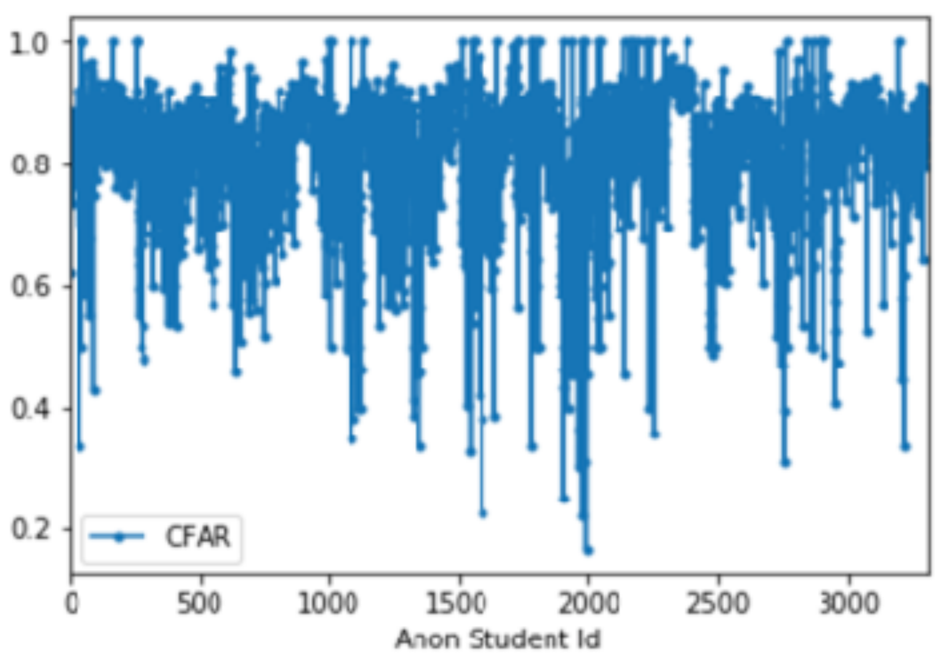
```
Row                                    0
Anon Student Id                        0
Problem Hierarchy                      0
Problem Name                           0
Problem View                           0
Step Name                              0
Step Start Time                   265516
First Transaction Time                 0
Correct Transaction Time          238090
Step End Time                          0
Step Duration (sec)               442921
Correct Step Duration (sec)      1641028
Error Step Duration (sec)        7719947
Correct First Attempt                  0
Incorrects                             0
Hints                                  0
Corrects                               0
KC(SubSkills)                    2475917
Opportunity(SubSkills)           2475917
KC(KTracedSkills)                4498349
Opportunity(KTracedSkills)       4498349
KC(Rules)                         322051
Opportunity(Rules)                322051
dtype: int64
```

Also we can see KC and Opportunity pairs seems to have some count of NaN value. For e.g., KC(SubSkills) and Opportunity(SubSkills) both have 2475917 records contains NaN value.

Exploratory Visualization

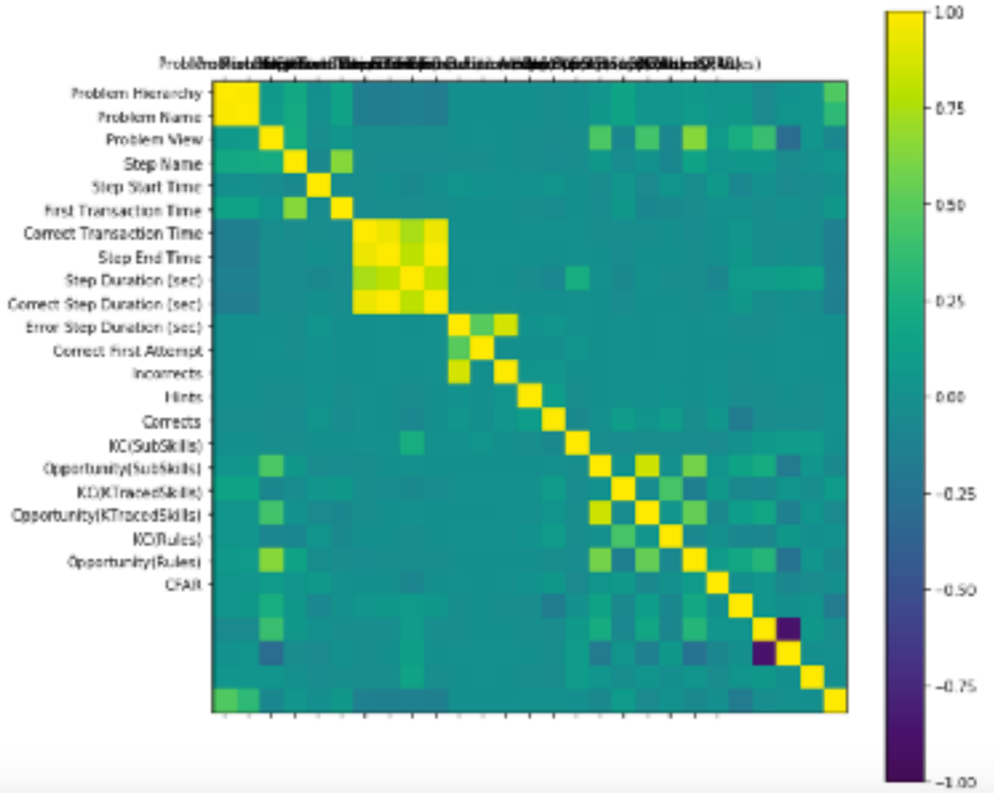
We filter out student id and CFAR to see CFAR on different students' performance.

```
tmp = data[['Anon Student Id','CFAR']]

tmp2 = tmp.drop_duplicates()

tmp2[['Anon Student Id','CFAR']].plot(x='Anon Student Id',y='CFAR',marker='.').figure
```



From the figure above, We can see most of the students have a high CFAR. But also there are some students' CFAR lower than 0.6, and only a few students have CFAR value between 0.2 and 0.4. We set records which CFA value is NaN to 0.5 seems to be a fair choice.

Data columns' correlation



From the figure above, we can see columns named ‘Step End Time’, ‘Step Duration (Sec)’, ‘Correct Step Duration (Sec)’, ‘Correct Transaction Time’ have strong correlation between each other.

So we should process these columns carefully, in case we missing important cues.

Algorithms and Techniques

LightGBM is a fast, distributed, high performance gradient boosting (GBDT, GBRT, GBM or MART) framework based on decision tree algorithms, used for ranking, classification and many other machine learning tasks. It supports multi CPU cores’ computation. My laptop a MBP(2015 Mid) only contains an AMD GPU which is not supported well by NVIDIA’s cuDNN. So I’d like to use some algorithms such as LightGBM or XGBoost. Computation could be accelerated by using multi CPU cores.

There are a bunch of default parameters for the algorithm. Parameters can be find at reference[5]. And there are a lot of turning tricks can be found at reference[6].

The input feature data for training is our processed data which have been one-hot encoding and NaN value fulfilled, without the target predict column ‘Correct First Attempt’. Our labels data is ‘Correct First Attempt’ column.

KMeans clustering aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean. KMeans is popular for cluster analysis in data mining. KMeans is efficient. But it has a loose relationship to the k-nearest neighbor((KNN)) algorithm.

Find more detail about KMeans at reference[12].

KNN is an algorithm to find a predefined number of training samples closest in distance to one point, and predict the label from these. These training samples are regarded as nearest neighbor to that point.

Distance between these points can be measured by standard Euclidean distance and so on.

Both for classification and regression, KNN can be useful to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones

Find more detail about KNN at reference[10].

Benchmark

KNN is one of the most popular algorithms for find the nearest neighbors in data. For our KDD CUP 2010 competition problem, we suppose to find the nearest K students for one student.

So we can use sklearn’s Neighbours Classifier to predict target CFA.

Sklearn code are as follows:

```
from sklearn import neighbors

start_time = time.time()

# we create an instance of Neighbours Classifier and fit the data.

clf = neighbors.KNeighborsClassifier(n_neighbors=10, weights='uniform', algorithm='auto',
leaf_size=30, p=2, metric='minkowski', n_jobs=4)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

print("Accuracy: ", metrics.accuracy_score(y_test, y_pred))

print("Log Loss: ", metrics.log_loss(y_test, y_pred))

print("%s seconds." % (time.time() - start_time))
```

III. Methodology

Data Preprocessing

- Process NaN or Null value

In order to make our model more generalize on unseen data, data with NaN or Null value may interfere our training and rise error. So we need to do some work.

The following three columns are integer type.
1) Step Duration (sec)
2) Correct Step Duration (sec)
3) Error Step Duration (sec)
A first thing comes into our mind is just remove these records which contains NaN or Null value. But we may need to see whether these records are scatters or normal data without some information.

Use python code:
df.isnull().sum()

We can see output below:
```
Row                                0
Anon Student Id                    0
Problem Hierarchy                  0
Problem Name                       0
Problem View                       0
Step Name                          0
Step Start Time               265516
First Transaction Time             0
Correct Transaction Time      238090
Step End Time                      0
Step Duration (sec)           442921
Correct Step Duration (sec)  1641028
Error Step Duration (sec)    7719947
Correct First Attempt              0
Incorrects                         0
Hints                              0
Corrects                           0
KC(SubSkills)                2475917
Opportunity(SubSkills)       2475917
KC(KTracedSkills)            4498349
Opportunity(KTracedSkills)   4498349
KC(Rules)                     322051
```

```
Opportunity(Rules)           322051
dtype: int64
```

The number of records contain NaN or Null value is so large. From the statistic of column ‘Error Step Duration (sec)’ and column ‘Correct Step Duration (sec)’. So we can’t just remove these data, but fill NaN or Null with a default value, such as 0, cause they are integer type. Also it wouldn't take too long for a operation on tutoring system in common sense. Fill these columns with 0 seems to be a fair choice.

Column ‘Correct Transaction Time’ and column ‘Step Start Time’ are date time string. I fill these columns with ‘2008-09-06 21:32:28’. It comes from the first records’ ‘Correct Transaction Time’ value of sort data using ascending. I mean the earliest date time appeared in our data. So that we can minimize the interval between records’ date time and it’s true happened date time.

- CFAR, calculate CFAR for every student and put into data.

Regarded to our proposal document, CFAR can be expressed by:

- CFA: Student's correct first attempt
- N: Total number of one student's all records(CFA = 1)
- T: Total number of one student's all records(both CFA = 0 and CFA = 1)

  CFAR = N/T

The CFAR column will be as a new feature for training then.

- One-hot encoding

Since a fewer columns of our data are string type. One-hot encoding seems to be a required step for model training. Cause transforms categorical features to a format that works better with classification and regression algorithms. Columns like ‘Anon Student Id’, ‘Problem Hierarchy’, 'Problem Name’, 'Step Name’, KC(Module) and Opportunity(Module) are processed using one-hot encoding. We didn’t use sklearn’s one-hot encoding cause it’s takes more memory usage.

- Clustering, add clustering feature.

We clustered our data in order to generate a new feature for better training model. We suppose to use GMM, but it’s slow. Instead, we use KMeans to cluster our data into 3 clusters. Also i’ve tried to calculate silhouette_score for clustering efficient, but it causes memory issue. I think it’s fine to use one more feature than nothing.

After we processed our data, and for quick reload purpose, we save our processed data into hdf format. Using pandas’s method with following code:

df.to_hdf('train_data.hdf','data',mode='w')

Implementation

Data splitting code are as follows:

```
import numpy as np

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2,
random_state=1)
```

Model training and evaluating code are as follows:

```
from pylightgbm.models import GBMClassifier

from sklearn import datasets, metrics, model_selection

from sklearn import cross_validation, metrics

exec_p = "/Users/michaelfeng/code/LightGBM/lightgbm"

import time
```

```
start_time = time.time()

clf =
GBMClassifier(exec_path=exec_p,max_bin=255,learning_rate=0.1,boosting_type='dart',
num_iterations=200, early_stopping_round=10,num_threads=4, num_leaves=40,
min_data_in_leaf=10)

clf.fit(X_train, y_train, test_data=[(X_val, y_val)])

y_pred = clf.predict(X_test)

print("Accuracy: ", metrics.accuracy_score(y_test, y_pred))

print("Log Loss: ", metrics.log_loss(y_test, y_pred))

print("%s seconds." % (time.time() - start_time))
```

Refinement

We set num_threads=4, cause my laptop has 4 core on CPU. It's quite efficient to use multi thread as multi CPU core.

min_data_in_leaf, default value is 20. It's the minimal number of data in one leaf. From LightGBM's official document's introduction, it can be used to deal with over-fit. We will try to use a big number such as 80, then adjust it later based on our prediction result.

boosting_type, default value is 'gbdt'. It's also can be set to 'dart'. Based on my personal experience, i'd like to use 'dart'. But 'gbdt' will also be tested.

max_bin, default value is 255. It's the max number of bin that feature values will bucket in. From LightGBM's official document's introduction, a *s*mall bin number may reduce training accuracy but may increase general power (deal with over-fit). I'll just let it unchanged with default value.

min_samples_split, min_samples_leaf, these two parameters are not supported by pyLightGBM framework(we use pyLightGBM to deal with LightGBM core). So we will not set them.

max_depth, it the maximum depth of a tree in LightGBM's internal implementation. From LightGBM's official document's introduction, It's used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample. From reference[6] we can see it needs to be tuned by GridSearchCV technology. So i will just leave it alone, then use GridSearchCV tuning it later when we need.

num_iterations, early_stopping_round are parameters used very common. Based on my own experience, i just num_iterations to 200, it's default value is 100. early_stopping_round's default value is 10. We just leave it alone.

Parameter Set0

exec_path=exec_p, max_bin=255, learning_rate=0.1, boosting_type='gbdt', num_iterations=200, early_stopping_round=10, num_threads=4, num_leaves=80, min_data_in_leaf=10

Output:

```
Accuracy:  0.947369689916
Log Loss:  1.81778803045
252.233078956604 seconds.
```

Parameter Set1

exec_path=exec_p, max_bin=255, learning_rate=0.1, boosting_type='gbdt', num_iterations=200, early_stopping_round=10, num_threads=4, num_leaves=80, min_data_in_leaf=20

```
Output:
Accuracy:  0.947368007934
Log Loss:  1.81784608905
258.72518587112427 seconds
```

Based on our previous experiments, the performance seems pretty good. So we didn't use a grid search by now, just change some parameters randomly. And just changed num_leaves from 80 to 40 in the next training to see if there's any improvement.
The rule of changing value is multi by 2 or divide by 2. So that performance divergence can be more obvious.

Parameter Set2

*exec_path=exec_p, max_bin=255, learning_rate=0.1, boosting_type=‘gbdt',
num_iterations=200, early_stopping_round=10, num_threads=4, num_leaves=40,
min_data_in_leaf=10*

Output:

```
Accuracy:  0.947348384822
Log Loss:  1.81852385364
228.89791417121887 seconds.
```

Parameter Set3
*exec_path=exec_p, max_bin=255, learning_rate=0.1, boosting_type=‘dart',
num_iterations=200, early_stopping_round=10, num_threads=4, num_leaves=40,
min_data_in_leaf=10*

Output:

```
Accuracy:  0.947368007934
Log Loss:  1.81784608905
280.3851001262665 seconds.
```

Parameter Set3 is our final parameter set.

A best practice of parameter tuning method can be found at reference[6].

IV. Results

Model Evaluation and Validation

LightGBM Model outputs:

```
Accuracy:  0.947368007934
Log Loss:  1.81784608905
280.3851001262665 seconds.
```

KNN classifier's output of evaluation metric using sklearn.

Benchmark outputs:

```
Accuracy:  0.872375759064
Log Loss:  4.40807422533
204.81724405288696 seconds.
```

The two model show their performance on data with outputs. Obviously, LightGBM perform better than KNN classifier. But it takes a little more time.

Justification

KNN is used for clustering, Decision Tree for classification.

KNN determines neighborhoods, so there must be a distance metric. This implies that all features must be numeric. Distance metrics may be effected by varying scales between attributes and also high-dimensional space.

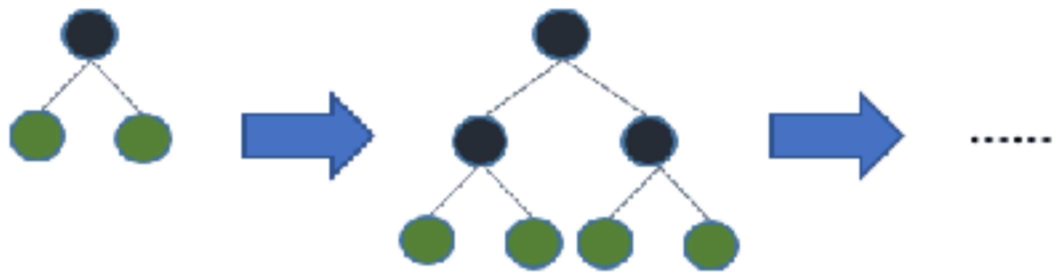
So, if we want to find similar examples we could use KNN. If we want to classify examples we could use Decision Tree.

In our case, i think Decision Tree could be a better choice.

LightGBM use decision tree in its internal implements. It is a boosting algorithm. Also It's been optimized in speed and memory usage. It use histogram based algorithms, which bucketing continuous feature(attribute) values into discrete bins, to speed up training procedure and reduce memory usage.

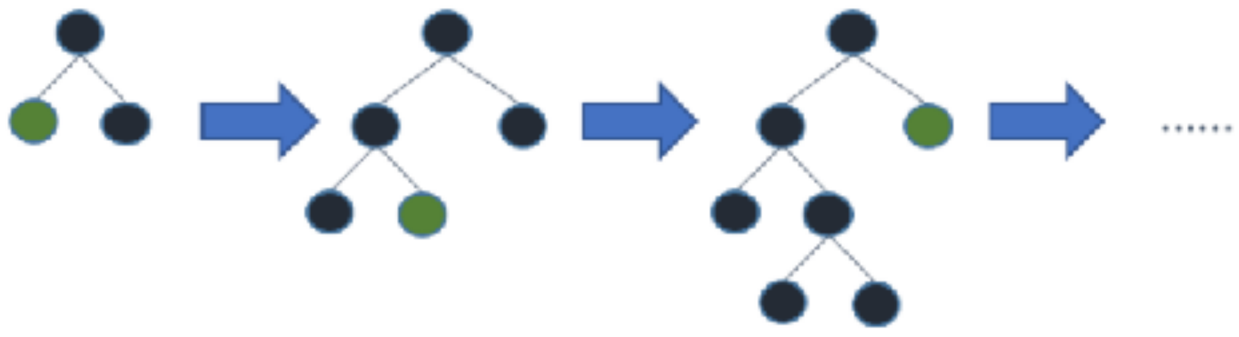
Meanwhile, Most decision tree learning algorithms grow tree by level(depth)-wise, like the following image:

LightGBM grows tree by leaf-wise(best-first)[7]. It will choose the leaf with max delta loss to grow. When growing same #leaf, Leaf-wise algorithm can reduce more loss than level-wise algorithm.



Level-wise tree growth

Leaf-wise may cause over-fitting when #data is small. So, LightGBM can use an additional parameter `max_depth` to limit depth of tree and avoid over-fitting (Tree still grows by leaf-wise).



Leaf-wise tree growth

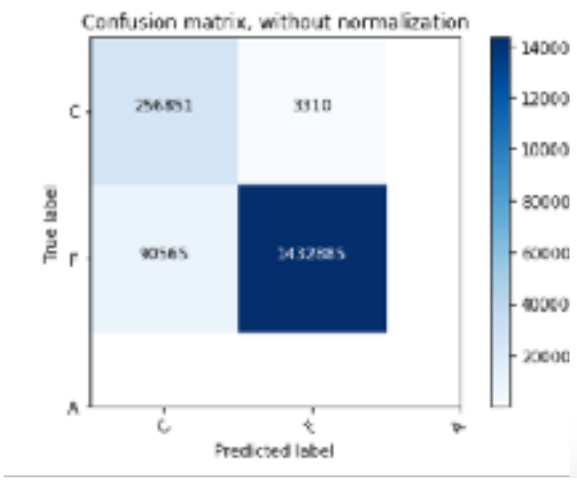
For more LightGBM features detail, find it at reference[7].

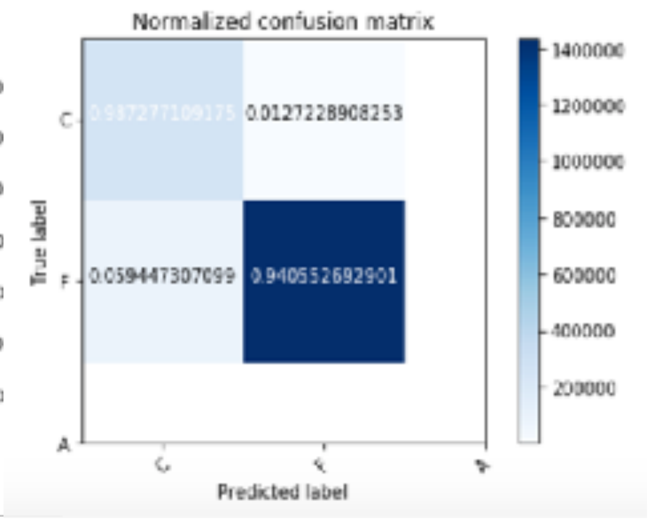
V. Conclusion

From the comparison between benchmark model and our designed model, we can see our designed model works greater than the benchmark model.

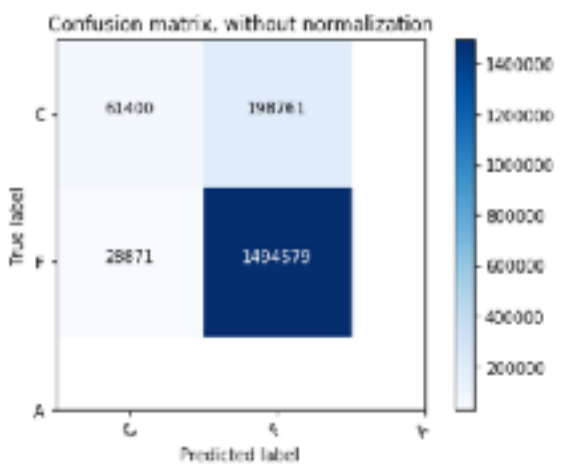
From confusion matrix aspect, it's also show our designed model works better than benchmark model.

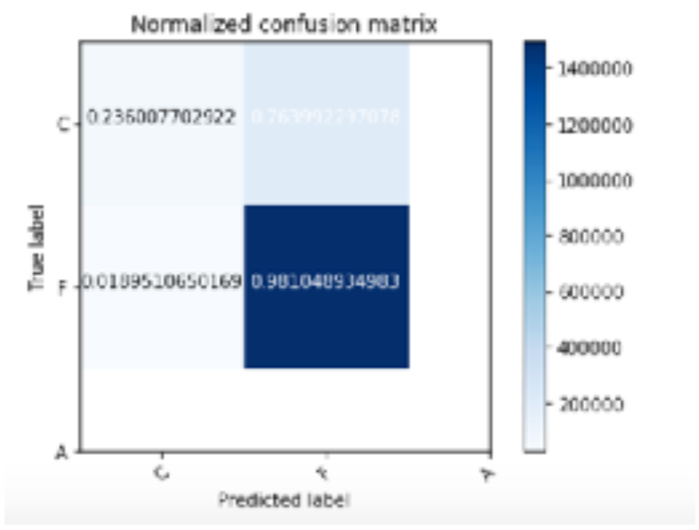
The below confusion matrix are our LightGBM's model performance. It shows the quality of the output of LightGBM classifier. The higher the diagonal values of the confusion matrix the better, indicating many correct predictions.





The below confusion matrix are our benchmark model performance. It shows the quality of the output of benchmark classifier. The higher the diagonal values of the confusion matrix the better, indicating many correct predictions.





Some of the methods have been waived because the limitation of my laptop's memory. But it's still a good experiment.

At the end of the experiment, we've achieved an accuracy about 94.73% for classification, and log loss about 1.8178.

Using the project’s prediction, we can measure different students’ performance on tutoring system’s different problem or exercise questions. So that new tutoring problem can be better scheduled and added to these system for better tutoring experience and effect.

Improvement

Our model and result can be optimized in many different ways, such as:

- Separate KC and Opportunity column modules which separated by '~~' to multi columns
- Fill 'CFAR' columns' NaN data with other options, for e.g., 0
- Other classification algorithms
- Using GMM to replace KMeans cluster for better clustering feature column
- Tuning LightGBM parameter with GridSearchCV technology.
- etc.

Some methods are limited by compute hardware limitation such as RAM on my laptop. So that they are not applied. Anyway, we can do better.

Reference

[1] Carnegie Learning system, http://www.carnegielearning.com/

[2] Addressing the assessment challenge with an online system that tutors as it assesses, http://repository.cmu.edu/cgi/viewcontent.cgi?article=1303&context=hcii

[3] Feature Engineering and Classifier Ensemble for KDD Cup 2010, http://pslcdatashop.org/KDDCup/workshop/papers/kdd2010ntu.pdf

[4] Collaborative Filtering Applied to Educational Data Mining, http://pslcdatashop.org/KDDCup/workshop/papers/KDDCup2010_Toescher_Jahrer.pdf

[5] pyLightGBM: python binding for Microsoft LightGBM, https://github.com/ArdalanM/pyLightGBM

[6] Complete Guide to Parameter Tuning in Gradient Boosting (GBM) in Python, https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/

[7] LightGBM features, https://github.com/Microsoft/LightGBM/wiki/Features

[8] Log Loss, http://wiki.fast.ai/index.php/Log_Loss

[9] Making Sense of Logarithmic Loss, http://www.exegetic.biz/blog/2015/12/making-sense-logarithmic-loss/

[10] *k*-nearest neighbors algorithm, https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[11] Nearest Neighbors, http://scikit-learn.org/stable/modules/neighbors.html

[12] *k*-means clustering, https://en.wikipedia.org/wiki/K-means_clustering