Capstone Project

Machine Learning Engineer Nanodegree

Definition

Project Overview

KDD Cup 2010 is an educational data mining competition in which participants are challenged with predicting students' algebraic problem performance from logs of student interaction with Intelligent Tutoring Systems. It is hosted by PSLC DataShop in 2010. I took their dataset because I am interesting in discovering fundamentals in education and ultimately optimizing the student learning process using machine learning technology. There are two kind of datasets available: Algebra I and Bridge to Algebra from year 2005 to 2009. I chose the dataset of Algebra I 2005-2006 which contains 575 students and 813661 interaction steps. This dataset have both training and testing sets, but some features are included in both sets, such as student ID, problem hierarchy including step name, problem name, unit name, section name, as well as knowledge components (KC) used in the problem and the number of times a problem has been viewed. However, some features are not available in testing set, but only appears in training set. For example: whether the student was correct on the first attempt for this step (CFA), number of hints requested (hint) and step duration information. More detail can be find in this website:

http://pslcdatashop.web.cmu.edu/KDDCup/downloads.jsp

Problem Statement

In this competition, CFA is regarded as label in the classification task. The incorrect attempt on the first attempt is denoted as 0 while the correct one is

denoted as 1. My task is use the other features to predict the CFA, the features are like student ID, problem hierarchy including step name, problem name, unit name, section name, as well as knowledge components (KC). Given that the problem is one of supervised learning, and more specifically one of binomial classification, I decided to use the supervised classification techniques, such as Gaussian Naive Bayes, Decision Trees, Logistic Regression and some ensemble method.

As mentioned above, the training set includes known CFA, but a testing set of unknown CFA in the testing set is left for evaluation. The testing set is used for the competition, the CFA in that set is only available in the holder. In this way, I cannot use it for testing. In order to validate the classifiers, I split the original training set into training set and testing set. Usually, cross validation is a common technique applied in document classification problems. However, for this competition, cross validation is likely not a suitable approach due to the special property of this data. This is means: a student's knowledge is growing as he/she solved more and more problem and will have a higher chance to correct in the next problem. In this way, I cannot randomly split my dataset. Instead, I collected last problem of each student as the testing dataset.

Metrics

This case is a binary classification (0 or 1). In order to measure its accuracy, I would like use F1 score as the metric since it is a weighted average of the precision and recall. It has the value between 0 to 1 where 0 represent the wort score and 1 is the best score. Here is the equation:

F1 = 2 * (precision * recall) / (precision + recall)

Analysis

Data Exploration

The following table is a raw sample of my dataset:

Row	Anon Stuc	Problem H	Probl	Prob	Step Nam	Step Sta	First Tra	Correct	Step End T	Step Du	Corr	Error S	Correct Fi	Incorrects	Hints	Corrects	KC(Default	Oppo
1	0BrbPbwC	Unit ES_04	EG4-F	1	3(x+2) = 15	24:35.0	24:49.0	25:15.0	25:15.0	40		40	0	2	3	1	[SkillRule:	1
2	0BrbPbwC	Unit ES_04	EG4-F	1	x+2 = 5	25:15.0	25:31.0	25:31.0	25:31.0	16	16		1	0	0	1	[SkillRule:	1~~1

The dataset contain 19 features. All the details of this dataset are listed as follows:

- **Row:** the row number Update (04-20-2010): for challenge data sets, the row number in each file (train, test, and submission) is no longer taken from the original data set file. Instead, rows are renumbered within each file. So instead of 1...n rows for the training file and n+1..m rows for the test/submission file, it is now 1...n for the training file and 1...n for the test/submission file.

- **Anon Student Id:** unique, anonymous identifier for a student

- **Problem Hierarchy:** the hierarchy of curriculum levels containing the problem.

- **Problem Name:** unique identifier for a problem

- **Problem View:** the total number of times the student encountered the problem so far.

- **Step Name:** each problem consists of one or more steps (e.g., "find the area of rectangle ABCD" or "divide both sides of the equation by x"). The step name is unique within each problem, but there may be collisions between different problems, so the only unique identifier for a step is the pair of problem_name and step_name.

- **Step Start Time:** the starting time of the step. Can be null.

- **First Transaction Time:** the time of the first transaction toward the step.
- **Correct Transaction Time:** the time of the correct attempt toward the step, if there was one.
- **Step End Time:** the time of the last transaction toward the step.
- **Step Duration (sec):** the elapsed time of the step in seconds, calculated by adding all of the durations for transactions that were attributed to the step. Can be null (if step start time is null).
- **Correct Step Duration (sec):** the step duration if the first attempt for the step was correct.
- **Error Step Duration (sec):** the step duration if the first attempt for the step was an error (incorrect attempt or hint request).
- **Correct First Attempt:** the tutor's evaluation of the student's first attempt on the step—1 if correct, 0 if an error.
- **Incorrects:** total number of incorrect attempts by the student on the step.
- **Hints:** total number of hints requested by the student for the step.
- **Corrects:** total correct attempts by the student for the step. (Only increases if the step is encountered more than once.)
- **KC(KC Model Name):** the identified skills that are used in a problem, where available. A step can have multiple KCs assigned to it. Multiple KCs for a step are separated by ~~ (two tildes). Since opportunity describes practice by knowledge component, the corresponding opportunities are similarly separated by ~~.
- **Opportunity(KC Model Name):** a count that increases by one each time the student encounters a step with the listed knowledge component. Steps with multiple KCs will have multiple opportunity numbers separated by ~~.

The properties of the dataset are as follows:

1. The data matrix is sparse: not all students are given every problem, and some problems have only 1 or 2 students who completed each item. So, the contestants need to exploit relationships among problems to bring to bear enough data to hope to learn.

2. There is a strong temporal dimension to the data: students improve over the course of the school year, students must master some skills before moving on to others, and incorrect responses to some items lead to incorrect assumptions in other items. So, contestants must pay attention to temporal relationships as well as conceptual relationships among items.

3. Which problems a given student sees is determined in part by student choices or past success history: e.g., students only see remedial problems if they are having trouble with the non-remedial problems. So, contestants need to pay attention to causal relationships in order to avoid selection bias.

In total, there are 575 students and 813661 interaction steps. Due to the special properties of this dataset, the statistical numbers for other features have no meaning in solving the problem, so I did not include this.

Exploratory Visualization

Visualization is a very useful technical for most cases in order to have a direct view of the relationship between the features. However, in this case, our task is predicting the future success based on the study history where the data is related as time goes by. Namely, the performance of the next step should be a cumulated results from all the steps in the past. In this way, using visualization to present the relationship between features is trivial. A more important thing is to find out a way to represent this cumulative property of the data which I will state in the data preprocessing part.

Algorithms and Techniques

There are 813661 data point in total and the number of features after processing is 112. Given that the problem is one of supervised learning, and more specifically one of binomial classification, I decided to use the following supervised learning model:

- **Gaussian Naive Bayes (GaussianNB):** It is chosen since I can use it to process the data in a sufficient way and get some basic idea of the data properties. With its help, I can modify the data to a more suitable form for the other classification.
- **Decision Trees:** A decision tree splits a set of data into smaller and smaller groups by one feature at a time. It requires little data preparation. The parameters which are able to tune include:

  1, Criterion: The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.

  2, splitter: The strategy used to choose the split at each node. Supported strategies are “best” to choose the best split and “random” to choose the best random split.

  3, min_samples_split: The minimum number of samples required to split an internal node.

  4, max_leaf_nodes: The minimum number of samples required to be at a leaf node.
- **Logistic Regression:** Logistic regression is a linear model for classification. Due to the large size of the training set, I can use it to save the training time without sacrifice too much time. The parameters which are able to tune include:

  1, penalty: Used to specify the norm used in the penalization. The ‘newton-cg’, ‘sag’ and ‘lbfgs’ solvers support only l2 penalties.

  2, C: Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.
- **Ensemble Methods (AdaBoost):** This 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 casesIt is a method which adaptively improves the performance by a series of weak classifiers. I would like to use it to find a better performance classifier. The parameters which are able to tune include:

1, n_estimators: The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early.

2, learning_rate: Learning rate shrinks the contribution of each classifier by learning_rate. There is a trade-off between learning_rate and n_estimators.

Benchmark

Ideally, the F1 score should be 1. However, considering the real situation, the students may not be correct for the first attempt just by carelessness or other situation which happens very randomly, the F1 score could not be 1. Moreover, we can not expect a classifier works perfectly in predicting future performance with all the information of past performance. In this case, I would say if the F1 score is more than 0.8 should be good enough for this case.

Methodology

Data Preprocessing

The dataset contains 19 features：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 (Default), Opportunity (Default). And based on my knowledge about study, I only selected the feature of Student ID, CFA, Incorrects, Hints, Corrects, KC and Opportunity. These features can be categorized into two types: Student ID and KC are categorical features, while others are numerical features.

I chose to expand a categorical feature into a set of binary features. There are 575 students in dataset. Therefore, the feature vector contains 575 binary

features to indicate the student who finished the step. For the feature of KC, because the Multiple KCs for a step are separated by ~~ (two tildes). I split the KC strings with ~~, and expand them into a set of binary features. In this way, there are 112 binary features to indicate KC. All the feature processing is in my **feature_process.py** file.

Because learning is a process of skill-improving over time, temporal information should be taken into consideration. I would like to have the correct rates for each KC. The equation I use is correct rates = 1 – (cumulated incorrects)/(cumulated incorrects+cumulated corrects). Since cumulated corrects is equal to opportunity, so the final equation is:

Correct_rate = 1 – (cumulated incorrects) / (cumulated incorrects+ opportunity)

Since the correct rates for all the KC can override the role of binary features to indicate KC, I only included this correct rates for these 112 binary features. In this way, I totally had 575+112 = 687 features. It exceeded the limit of most algorithm in sklearn kit. I had to reduce it. I worked out an idea that I can use categorical data to represent Student ID, in this way, I will only have 112+1=113 features. The following table is an illustration of my dataset.

Student ID	Knowledge 1	Knowledge 2	Knowleage 3	CFA
A	0	0.9	0.6	0
A	0.8	0	0.7	1
B	0	0.7	0.9	1
B	0.5	0.8	0	0

In this table, 0 represents that this knowledge is not required for the problem while the number between 0~1 is the correct rate of that knowledge for the student studying from the beginning to this problem. From this table I realized that in most of the case the cumulative results for knowledge should be almost no difference for any human being. In the meantime, most of the classifier cannot work with

categorical data. Therefore, I removed the Student ID feature, with only 112 numerical features left.

After this, I split the data into training set and testing set as mentioned in the problem statement part above. The code for splitting can be find in **split_data.py file.** I also split out the validating set using **validating_data.py.**

Implementation

In **Project5_Predict_future_success.ipynb** file, I implemented 4 classifiers to predict the results. At first, I used the GaussianNB to get the rough idea of how the data will perform. With the full training dataset, I got F1 score of 0.8285. It is not quite bad. Considering there are over 800000 data points which will make the training time very long, I reduced the data sets to ¼ by choosing every 4^{th} row in this dataset as a sample dataset. The code can be find in **smaller_data.py.** After doing this, the GaussianNB give me a F1 score of 0.8213 which changes very small amount comparing to that of original data. I realized that the limitation of F1 score to a higher number is due to the bias. This means my data might not be complex enough to capture all the relationship of the features. But over 0.8 is good enough for my expectation, so I decided to keep these features. Since large data size is not necessary in this case, I decided to use the smaller data set for all the other classifier. The following table list the primary result of these classifier:

	F1 score	Time (s)
GaussianNB	0.8213	1.8210
Decision Tree	0.8402	9.9120
AdaBoost	0.8449	32.0410
Logistic Regression	0.8425	3.4310

Refinement

From the table above, I found that the F1 score for Decision Tree, AdaBoost, Logistic Regression is similar with AdaBoost with the highest score. However, AdaBoost took about 32 second to train and predict the results which is 10 folds than Logistic Regression.

In order to improve them, I tune the parameters for these three classifier using GridSearchCV method. The following is the table for the tuned parameters:

	Parameters
Decision Tree	{'criterion': ('gini','entropy'), 'splitter':('best','random'), 'min_samples_split':[2,10,20], 'max_leaf_nodes':[5,30,100]}
AdaBoost	{ "n_estimators": [1, 50, 100]}
Logistic Regression	{'penalty':('l1','l2'), 'C':[0.01, 0.1, 1.0, 10, 20]}

Results

Model Evaluation and Validation

The following table shows the result with GridSearchCV method:

	Parameters	F1 score
Decision Tree	(class_weight=None, criterion='gini', max_depth=None, max_features=None, max_leaf_nodes=100, min_samples_leaf=1,	0.8514

	min_samples_split=10, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='random')	
AdaBoost	(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0, n_estimators=100, random_state=None)	0.8478
Logistic Regression	(C=0.1, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1, penalty='l1', random_state=None, solver='liblinear', tol=0.0001, verbose=0, warm_start=False)	0.8420

Justification

From the table above, I got the best F1 score of 0.8541 from the validating data set. with decision tree. The classifier's parameter is: (class_weight=None,

criterion='gini',

max_depth=None,

max_features=None,

```
max_leaf_nodes=100,
min_samples_leaf=1,
min_samples_split=10,
min_weight_fraction_leaf=0.0,
 presort=False,
 random_state=None,
 splitter='random')
```

It reaches my original bench mark.

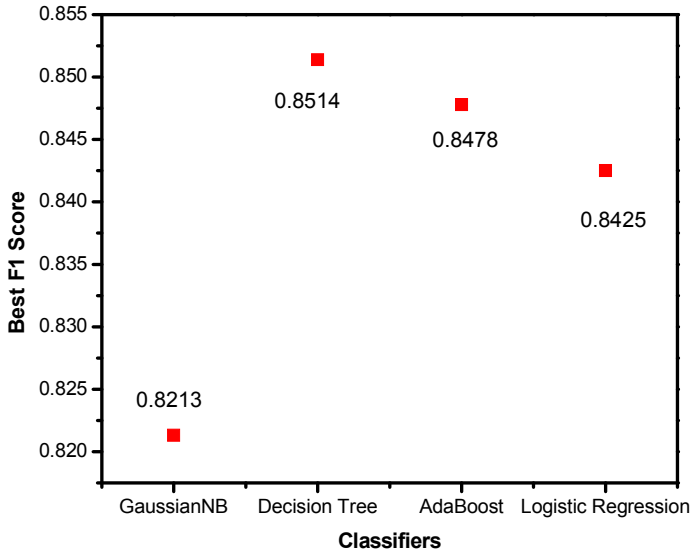
In order to justify the robustness of this classifier, I also tried the testing set to get the F1 score. The following table is the F1 score from training dataset, validating dataset and testing dataset which shows that the classifier is robust.

Dataset	F1
Training	0.8765
Validating	0.8541
Testing	0.8630

Conclusion

Free-Form Visualization

From the figure below, we can see after the parameters tuning, the best F1 score for predicting correct in first attempt (CFA) is 0.8514 from decision tree. This results reaches my expectation in the beginning of the project.



I also created the confusion matrix for all the classifiers:

GaussianNB	negative	positive
true	40	462
false	41	131

Decision Trees	negative	positive
true	9	493
false	32	140

Adaboost	negative	positive
true	18	484
false	37	135

Logistic Regression	negative	positive

true	10	492
false	19	153

From this table we can calculate the precision and recall for all these classifiers which are showed in the following table. From this table we can see Decision tree gives the best recall score while Adaboost's precision is slightly better than the others.

	Precision	Recall
GaussianNB	0.779	0.920
Decision Trees	0.779	0.982
Adaboost	0.782	0.964
Logistic Regression	0.762	0.980

Reflection

The project began with the understanding of the data set. It is different with most of the data set because the learning is a process of skill-improving over time, temporal information should be taken into consideration. However, the temporal information is not given in the data set. So I used the equation: Correct_rate = 1 – (cumulated incorrects) / (cumulated incorrects+ opportunity) to store these information in skill names (KC). Also, due to the special property of the data set, cross validation did not work with this case. Instead, I collected the last problem of each student as the testing dataset.

Next, several classifiers were trained, using grid search (GridSearchCV in sklearn). Once several classifiers were trained, the F1 score of the classifiers on the test data set are used to determine which classifier is the best.

Finally, I got the best F1 score of 0.8541 with decision tree. The parameters of this decision tree is stated in the results parts.

The most difficult part is how to understand the data set and figure out the way to present the cumulative performance information. Since it is different from the most data set I worked with, it took me most time to figure it out.

Improvement

As I mentioned early in this report, the limitation for F1 score to be a higher number is the bias which means my data is not complex enough to capture all the relationship of the features. This is due to the discard of the 575 binary data of the Student ID. I believe if I can implement a classifier which is able to take these 575 binary data, the result will be better.

A second way I can come up to improve my result would be to implement the code in a much faster language, like C, so that the parameter space can be further mined for optimal values. Python is a great prototyping language, but the run times for GridSearchCV increase very quickly when adding even just a few new parameters to search. With this limitation, the number of combination of parameter is not enough to find the best performed classifier.