Sai Raj Reddy

Definition

Project Overview

Carnegie Learning's Math Tutor (Mathia Software ITS) is intented to help high school students learn Math better. PSLC Datashop platform is meant for Education Data Scientists to look for learning analytics' data. Founded by noted researcher Ryan Baker, PSLC hosted an open challenge in 2010 called KDD EDM Challenge which was sponsored by Facebook and IBM Research. They released the Mathia ITS data which consisted of both Training and Testing set.

From a research point of view, the methods to achieve low prediction error on unseen education data is still at early stages. Sophisticated ML 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. This project aims to experiment with such an approach.

Problem Statement

Students who use the Mathia Tutor to solve problems -- their interactions are recorded as logs. This log data can be mined to get insights. My aim is to predict students' correct first attempt (CFA) while solving the math. CFA is a feature in the dataset which is recorded in terms of bit or boolean (0 or 1). Therefore, precise problem domain involved is classification.

Probability can be calculated by sklearn. We can find some models or methods at Sklearn Home and sklearn.svm.libsvm.predict_proba() So, we can get model's performance metric like log loss on unseen data for measurement.

Metrics

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).

We will compare the predictions we provided against test portion(part of training data) true values and calculate the difference as log loss.

• y’: Predicted target probability of Correct First Attempt(CFA=1) • y: Target’s original CFAR -log P(y’|y) = -(y’log(y) + (1 - y’) log(1 - y))

The use of log loss is very common and it makes an excellent general purpose error metric for numerical predictions. We will dedicate our best to acquire the lowest log loss as possible.

Analysis

Data Exploration

There are 8918055 records in our algebra_2008_2009_train.txt file. There are 23 columns in data.

Column Name	Description
‘Row'	Row number
‘Anon Student Id'	Anonymised Student ID stripped of personal data.
‘Problem Hierarchy'	The hierarchy of curriculum levels containing the problem. For example, a problem might be contained in a Unit A, Section B hierarchy.
‘Problem Name'	Name of the problem
'Problem View'	The number of times the student encountered the problem so far. This counter increases with each instance of the same problem.
'Step Name'	Name of the step
'Step Start Time'	Timestamp of starting time of particular step.
'First Transaction Time'	Timestamp of interaction with ITS feature.
'Correct Transaction Time'	Timestamp of interaction with ITS feature when the answer is correct.
'Step End Time'	Timestamp of ending time of particular step.
'Step Duration (sec)'	Step Duration is the total length of time spent on a step.

Column Name	Description
'Correct Step Duration (sec)'	The step duration if the first attempt for the step was correct. Correct Step Duration might also be described as "reaction time" since it's the duration of time from the previous transaction or problem start event to the first correct attempt.
'Error Step Duration (sec)'	The step duration if the first attempt for the step was an error (an incorrect attempt or hint request).
'Correct First Attempt'	Boolean of wether the answer is attempted correctly for the first time.
'Incorrects'	Number of times the step is answered incorrectly.
'Hints'	Number of hints used.
'Corrects'	Number of correct attempts
'KC(SubSkills)'	Referral to Subskill of children in hierarchy for a KC.
'Opportunity(SubSkills)'	Referral to Subskill of children in hierarchy for an Opportunity.
'KC(KTracedSkills)'	Referral to KTracedSkill of child nodes in hierarchy for a KC.
'Opportunity(KTracedSkills)'	Referral to KTracedSkill of child nodes in hierarchy for a Opportunity.
'Opportunity(Rules)'	Referral to Riles of child nodes in hierarchy for a Opportunity.

Number of unique students: 3310

Number of unique problems: 188368

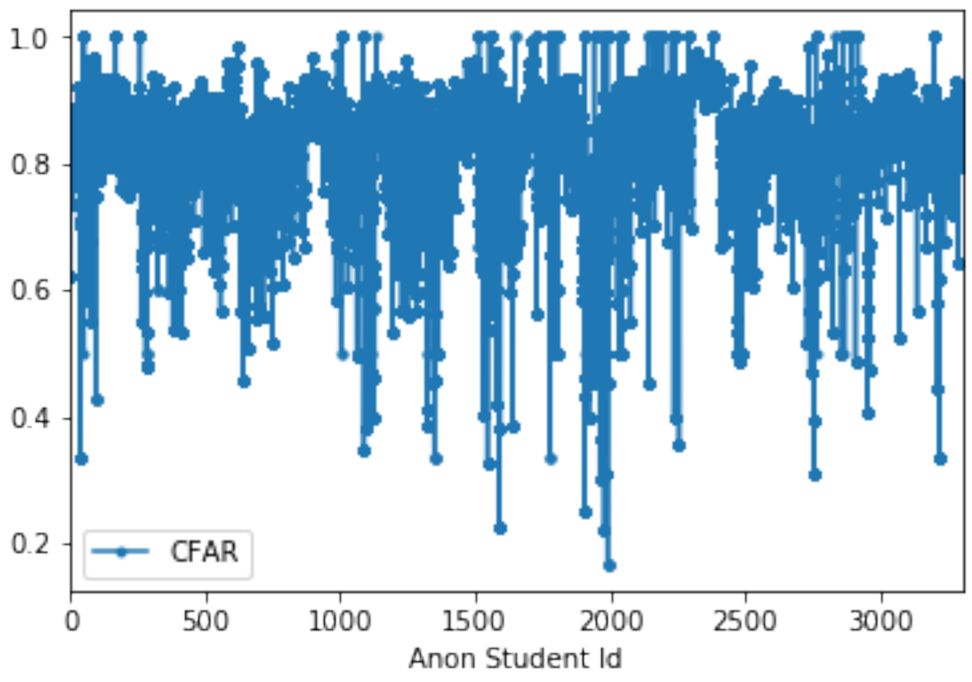
Number of unique KC(SubSkills): 1829

Number of unique KC(KTracedSkills): 922

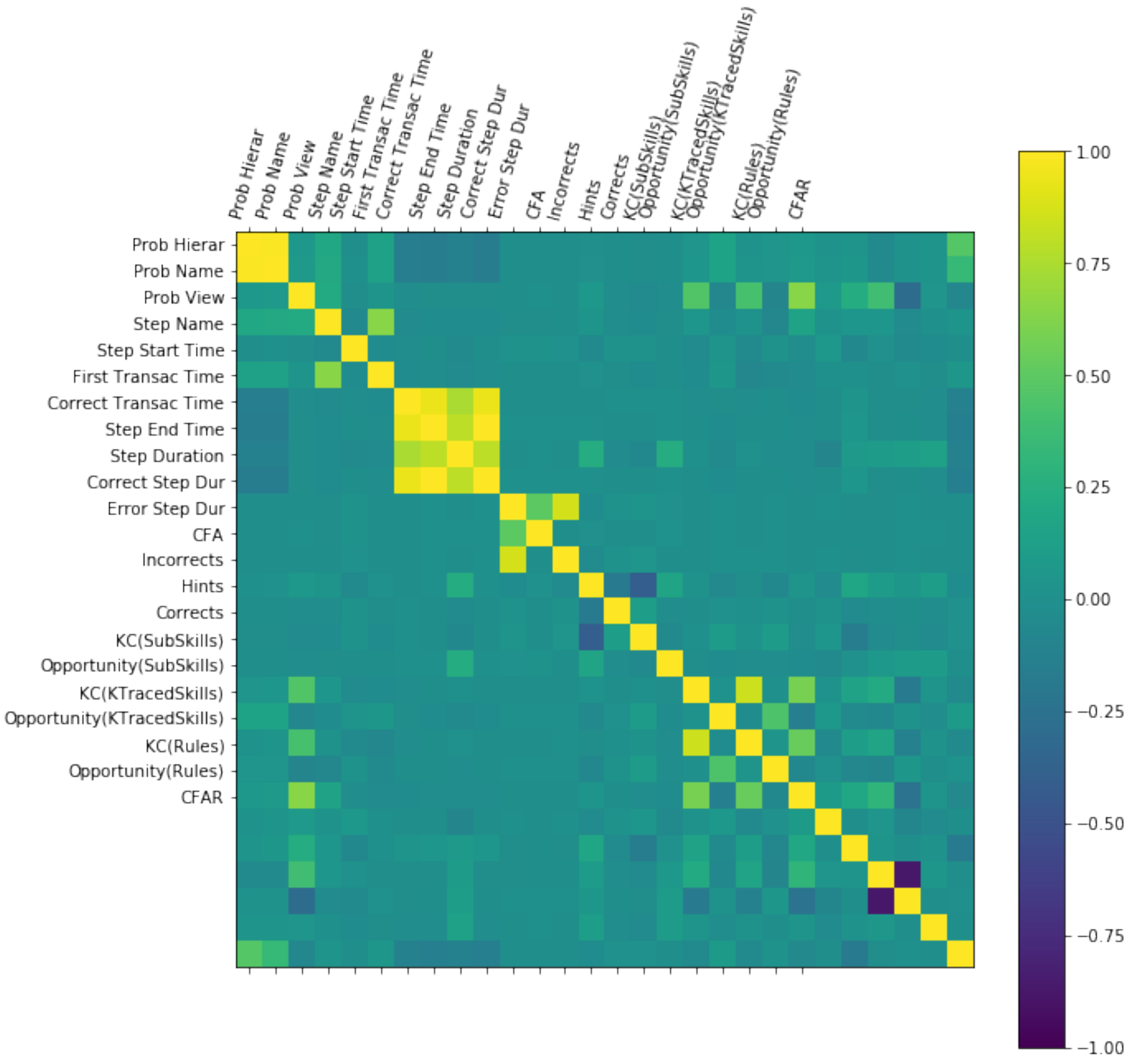
Number of unique KC(Rules): 2979

Number of total records: 8918054

Exploratory Visualisation



Most of the students have high CFAR, but observing that some students' CFAR < 0.6, and only a few students have CFAR value between 0.2 and 0.4. CFA value of NaN is 0.5 which should be unbiased and valid enough.



'Step End Time', 'Step Duration (Sec)', 'Correct Step Duration (Sec)', 'Correct Transaction Time' have strong correlation between each other. More emphasis on these features to look for more insights.

Algorithms and Techniques

CatBoost is an open-source gradient boosting on decision trees library with categorical features support out of the box for Python. It supports multi CPU cores' computation. My GPU doesn't support high performance training. Therefore, I started using algorithms such as LightGBM, XGBoost and Catboost. I was convinced of Catboost's faster performance (https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db) This was also suggested by Capstone Proposal Reviewer. Hence I am using Catboost for this project.

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.

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.

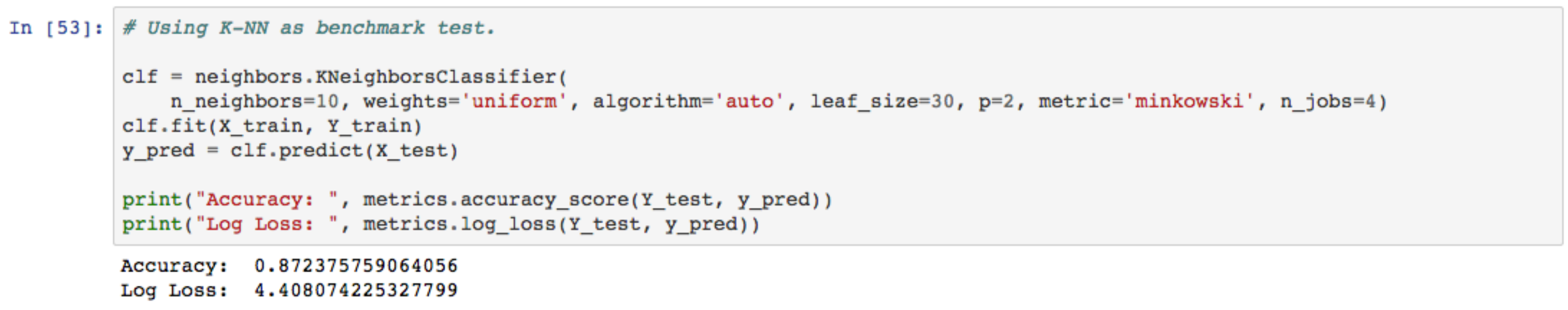
Distance between these points can be measured by standard Euclidean distance and so on.

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.

Benchmark

I used KNN as the benchmark as it is popular and is known for high accuracy results.

The code that I used for benchmarking and printing the results :



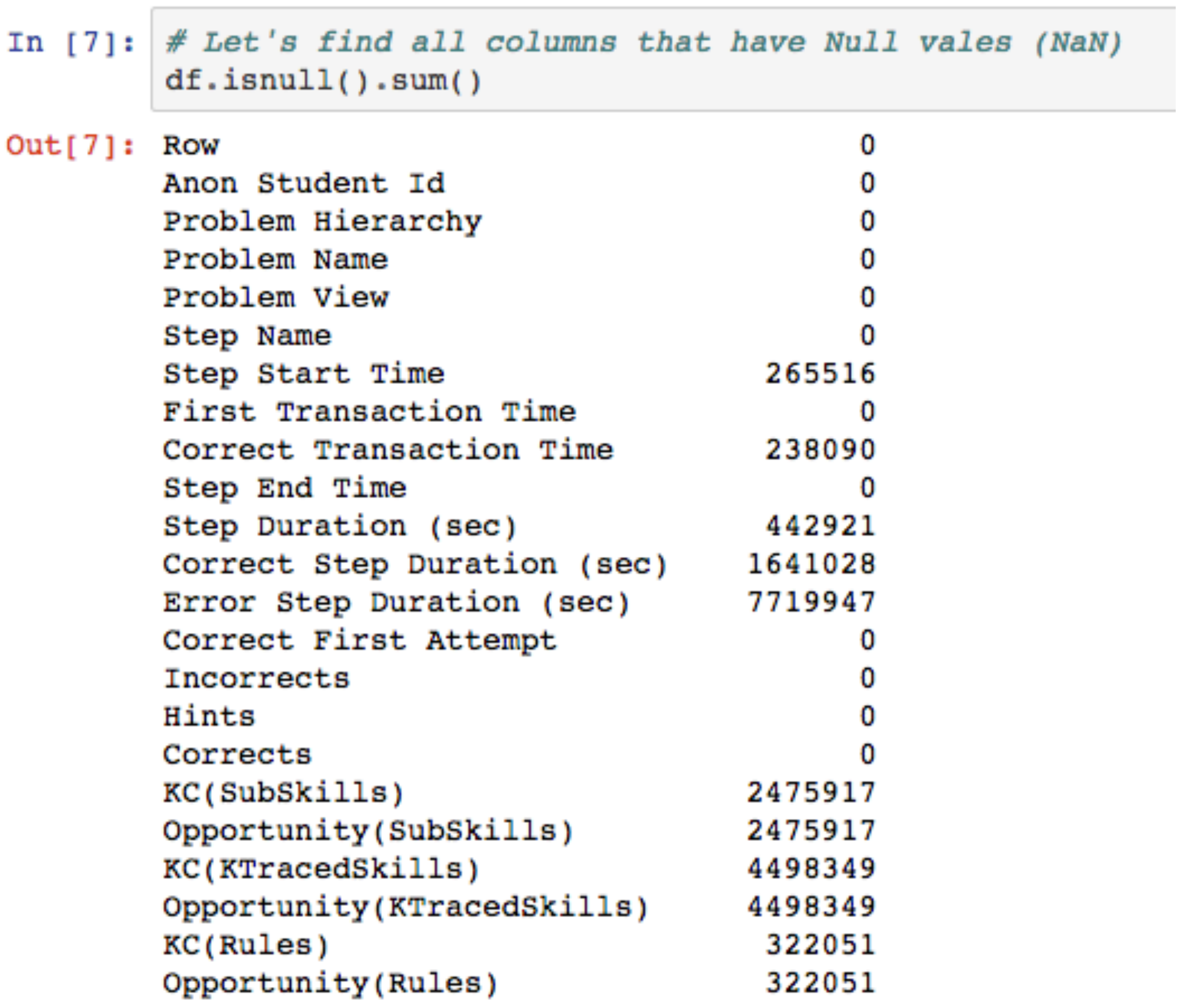
```
In [53]: # Using K-NN as benchmark test.

         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))

         Accuracy:  0.872375759064056
         Log Loss:  4.408074225327799
```

Methodology

Data Pre-Processing

Wrangle NaN and Null values.

We get the list of NaN allowed features. See the screenshot below.



```
In [7]: # Let's find all columns that have Null vales (NaN)
        df.isnull().sum()

Out[7]: 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
```

In order to avoid errors, we need to process the null values to ensure consistency in data. Duration features' NaN is converted to zero seconds. I will talk about KC and Opportunity features sometime later in this document.

Column ‘Correct Transaction Time’ and column ‘Step Start Time’ are Timestamps. NaN values are replaced with ‘2008-09-06 21:32:28’ as it is the first transaction being recorded. In other words, It is from the row with earliest Timestamp value.

Correct First Attempt Rate (CFAR).

As explained in 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)

Therefore, CFAR = N/T

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

One-Hot Encoding.

This is implemented manually as I found it simpler thus saving lot of memory space as compared to sklearn's One-Hot Encoding feature. The data pattern observed is heirarchical and classification process is efficient when applied One-Hot Encoding. 'Anon Student Id', Problem Hierarchy', 'Problem Name', 'Step Name', 'KC(Sub Skills)', 'KC(KTracedSkills)', 'KC(Rules)', 'Opportunity(SubSkills)', 'Opportunity(KTracedSkills)', 'Opportunity(Rules)' are encoded.

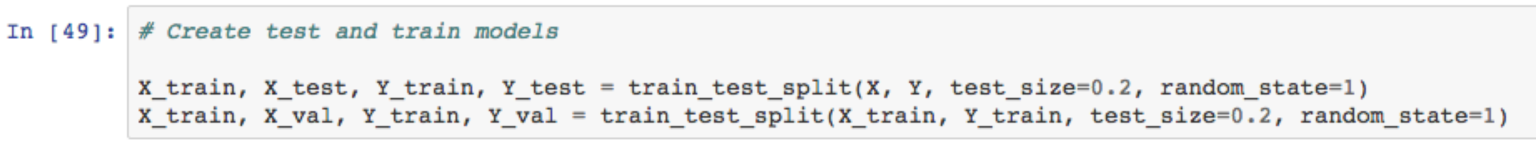
Miscellaneous.

Post processing of data, the data is saved as Hierarchical Data Format (.hdf) using the pandas method df.to_hdf() so that loading of data is much faster and efficient.

Implementation

Data Splitting.

Test and train data is created using sklearn’s inbuilt features. Check the screenshot below.



```
In [49]: # Create test and train models

         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)
```

Training of model and Evaluation.

As mentioned earlier, I am using Yandex's Catboost GBM

```
model = CatBoostClassifier(iterations=200, learning_rate=0.1, l2_leaf_reg=20, loss_function='Logloss', max_bin=255,
                           thread_count=4, leaf_estimation_iterations=40, ctr_leaf_count_limit=10)

model.fit(X_train, Y_train)

preds_class = model.predict(X_test)
preds_proba = model.predict_proba(X_test)

print("class = ", preds_class)
print("proba = ", preds_proba)

print("Accuracy: ", metrics.accuracy_score(Y_test, preds_class))
print("Log Loss: ", metrics.log_loss(Y_test, preds_proba))
```

Refinement

Parameter tuning explained for the parameters below.

- **iterations=200** , as the iterations increase — better the accuracy. However, there's no significant increase in accuracy levels after about 100th iteration.
- **learning_rate=0.1** , minimal learning rate works better and safer for higher number of iterations.
- **l2_leaf_reg=20**, L2 regularization coefficient. Used for leaf value calculation. This number was able to complete the process. Smaller co-efficients < 10 won't suffice higher order iterations.
- **loss_function='LogLoss'**, quite obvious. Aim is to achieve the minimum loss.
- **max_bin = 255**, It's the max number of bin that feature values will bucket in. A small bin number may reduce training accuracy but may increase general power (deal with over-fit).
- **thread_count=4**, my hardware is 4 core CPU.
- **leaf_estimation_iterations=40**, An important param which is the number of gradient steps when calculating the values in leaves. 40 seemed to be a reasonable number after testing out with different values.
- **ctr_leaf_count_limit=10**, The maximum number of leafs with categorical features. If the quantity exceeds the specified value a part of leafs is discarded. The heir across all levels could have more leaves, but then it affects the performance and can lead to overfitting for larger count limit. 10 was perfect.

Now let's see how different parameters for tuned to find the best attempt having best set.

Attempt 1

iterations=100, learning_rate=0.1, l2_leaf_reg=10, loss_function='Logloss', max_bin=255, thread_count=4, leaf_estimation_iterations=80, ctr_leaf_count_limit=10

Accuracy: 0.931646490301514
Log Loss: 0.10264734498875275

Attempt 2

iterations=200, learning_rate=0.1, l2_leaf_reg=10, loss_function='Logloss', max_bin=255, thread_count=4, leaf_estimation_iterations=80, ctr_leaf_count_limit=10

Accuracy: 0.937248321079221
Log Loss: 0.10264734498714356

Attempt 3

iterations=200, learning_rate=0.1, l2_leaf_reg=10, loss_function='Logloss', max_bin=255, thread_count=4, leaf_estimation_iterations=40, ctr_leaf_count_limit=10

Accuracy: 0.9473119410192225
Log Loss: 0.10264734448721254

Attempt 4

iterations=200, learning_rate=0.1, l2_leaf_reg=20, loss_function='Logloss', max_bin=255, thread_count=4, leaf_estimation_iterations=40, ctr_leaf_count_limit=10

Accuracy: 0.9473119418976447
Log Loss: 0.10264626302875719

Therefore, the fourth attempt is the best set we can use for passing the benchmark and with achieving significant accuracy combined with appropriate minimum log loss.

Results

Model Evaluation and Validation

Catboost Model Output :

Accuracy is **0.9473119418976447**
Log Loss is **0.10263836999384292**

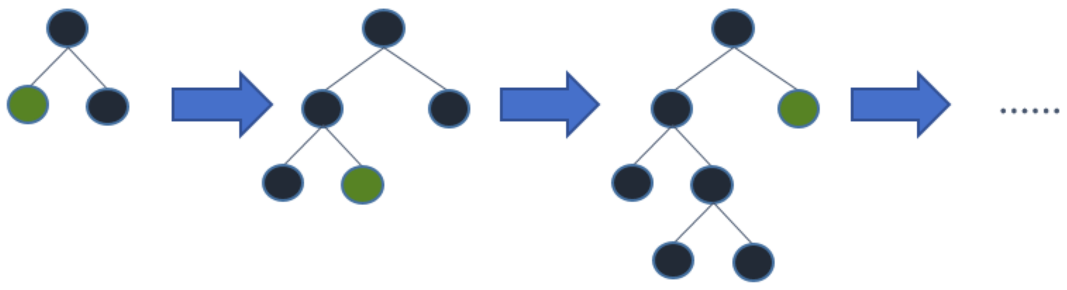
Benchmark K-NN Output :

Accuracy is **0.872375759064056**
Log Loss is **4.408074225327799**

Catboost model works way better than benchmark as shown by results. :)
However, Catboost takes longer time, but its ok as we are achieving higher accuracies.

Justification

K-NN is used for clustering whereas Decision Tree is used for classification.K-NN determines neighborhoods, so there must be a distance metric. This implies that all features must be numeric. Distance metrics might be effected by varying scales between attributes and also high-dimensional space. So, if we want to find similar examples we could use K-NN. If we want to classify examples we could use Decision Tree. In this case, Decision tree is the perfect choice as it also has non-numerical values.



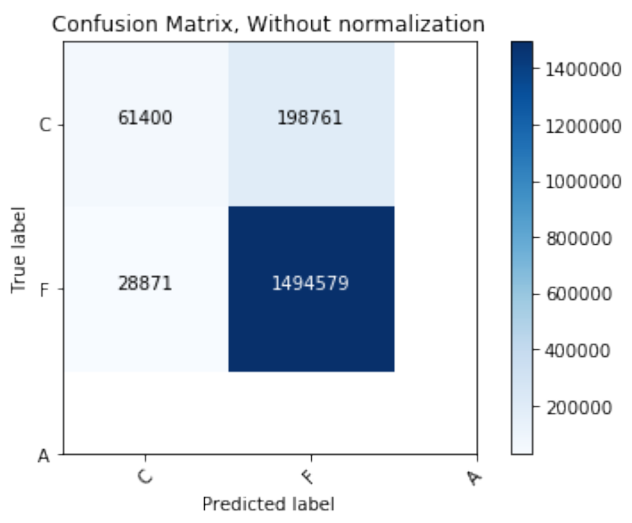
Leaf-wise tree growth

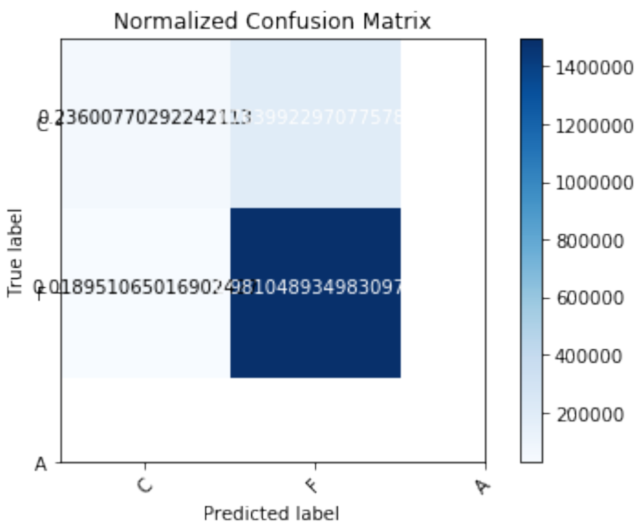
Catboost uses decision tree with leaf-wise, leaf-best tree growth in its internal implementation as shown in above diagram. It is a boosting algorithm. Works faster than other boosting algos (https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db)

It uses histogram based algorithms, which involves bucketing continuous feature(attribute) values into discrete bins, to speed up training procedure and to optimize memory usage in the best manner.

Conclusion

Free-Form Visualization





The confusion matrix is the benchmark model performance. The higher the diagonal values of the confusion matrix the better, indicating many correct predictions.

Reflection

Using my work's prediction features, If integrated — Mathia Learning software can automatically allot questions based on these performances so that students can perform better when they are aware of their confidence levels on solving the math problem. This kind of feedback also promotes Mastery Based Learning. Coupling with re-inforcement learning can also help result in better performance of the Intelligent Tutoring Systems.

Improvement

We can still improve our model in an optimised manner :

- Create multi columns for both KC and Opportunity i.e. seperate these multi-columns as features.
- CFAR's NaN was assumed to be 0.5. There could other equivalent values (Example : 0). Needs careful design.
- We can explore other algorithms. GridSearch CV can help in fine-tuning hyperparameters.

I had my hardware limitations to experiment with other algorithms, however I am happy with the results!