Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: 16 December 2019
Project Report

OXFORD

Expression Analysis

Analysis on Children Learning Performance by Educational Game App

Zhichao Zhang [1]*, Ruixin Liu [1]*,Yanjun Liu [1]*

[1] Electrical Engineering, Columbia University

*To whom correspondence should be addressed.

Abstract

Motivation: Education takes up a large portion is today's life. How to accurately analyze children's learning behavior and achievement becomes crucial. Thus, we tried to build machine learning models based on children's history performance, such as their task accuracy, event code (processing steps) on game app to determine children's game assessment result based on a game APP called the PBS KIDS Measure Up!
Results: We gave a quick EDA about how the data distributed. Then the data preprocessing is carried out to unique information for each installation id, mapping the data with train label.After giving a baseline model based on lightGBM, we tried different models, and implemented feature engineering to determine the children's performance on the last assessment test. The model performs well with the highest evaluation kappe score up to 0.7 based on our four groups of unused test. Additionally, the front-end website with Flask framework can show ROC, loss and feature importance results generated by back-end under four different test set.
Availability and implementation: Scripts and data are available at GitHub.
Contact: zz2668@columbia.edu, rl3063@columbia.edu, yl4227@columbia.edu

1 Introduction

Game and children education are taking larger and larger market in today's world. Especially the combination of these two, thus we focus on data from an educational game app called the PBS KIDS Measure Up!
In the PBS KIDS Measure Up! App, children navigate a map and complete various levels, which may be activities, video clips, games, or assessments. There are five assessments games: Bird Measurer, Cart Balancer, Cauldron Filler, Chest Sorter, and Mushroom Sorter. We need to use the gameplay data to forecast how many attempts a child will take to pass a given assessment (an incorrect answer is counted as an attempt). Each application install is represented by an installation id, and this is typically corresponding to one child. In the data table, we have the full history of gameplay data. Then, we need predict the number of attempts for a single and random chose assessment based on the history performance. We think the information inside the database are very useful and worth analyzing, like the actions within different children, the classification of children and accuracy in assessment. We would like to analyze the information in database to show the relationship between history performance and determine the accuracy in final assessment. In addition, we want to classify users based on their history information in different game sessions.

2 Related works

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It's a widely used framework on various machine learning problems.
The Gradient Boosted Decision Tree (GBDT) method predicts the company's failure and explains how to better analyze each financial variable[1]. Its advantages include but are not limited to: fast training speed and higher efficiency, lower memory usage, good accuracy, supporting parallel and GPU learning, capable of handling large-scale data.
In the paper of random decision forests [3], the author introduced a method for tree based classifiers, which can arbitrarily expand capacity for increase in accuracy in both training and testing data.In the paper of Random Forests for Big Data, the author proposes an idea that how to expand random forest into big data issues [4]. Besides, in the paper named Selecting a representative decision tree from an ensemble of decision-tree models for fast big data classification, the author represents a method to select a single representative model among multiple decision tree models to reduce the

Table1: List of parameters and their descriptions in the initial train and test dataset

Feature name	**Type**	**Description**
installation_id	string	Randomly generated unique identifier grouping game sessions within a single installed application instance
game_session	string	Randomly generated unique identifier grouping events within a single game or video play session
event_id	string	Randomly generated unique identifier for the event type. Maps to event$_i$$dcolumninspecstable$
timestamp	timestamp	Client-generated datetime
event_data	json	Semi-structured JSON formatted string containing the events parameters. Default fields are event_count, event_code, and game_time; otherwise fields are determined by the event type.
event_count	numerical	Incremental counter of events within a game session (offset at 1). Extracted from event$_d$$ata$
event_code	numerical	Identifier of the event 'class'. Unique per game but may be duplicated across games. E.g. event code '2000' always identifies the 'Start Game' event for all games. Extracted from event$_d$$ata$
game_time	numerical	Time in milliseconds since the start of the game session. Extracted from event$_d$$ata$
title	string	Title of the game or video
type	string	Media type of the game or video. Possible values are: 'Game', 'Assessment', 'Activity', 'Clip'
world	string	The section of the application the game or video belongs to. Helpful to identify the educational curriculum goals of the media. Possible values are: 'NONE' (at the app's start screen), TREETOPCITY' (Length/Height), 'MAGMAPEAK' (Capacity/Displacement), 'CRYSTALCAVES' (Weight)

time complexity. The representative model they chosen by the proposed method provides higher speed and easier interpretation for classification in big data [5].

Among all these literature work, random forest as well as decision trees are common in big data nowadays.

3 Materials

3.1 Data

The data used in this project is anonymous, tabular data of interactions with an app called PBS KIDS Measure Up! These datasets are collected from Kaggle, including train set.csv$trainlabel.csv$ and $test.csv$. The app is a game-based learning tool for kids, and the dataset includes users' assessment scores and their records through the game. PBS KIDS is committed to creating a safe and secure environment that family members of all ages can enjoy. Children navigate a map and complete various levels, which may be activities, video clips, games, or assessments. All these processes contribute to event data. Each assessment is designed to test a child's comprehension of a certain set of measurement-related skills.

The train dataset we used includes 11341,042 rows of different event data in Json format from 303,319 different game sessions of 17000 installation id. In the original data, it contains many $installationids$ which never took assessments, whereas every installation id in the test set made an attempt on at least one assessment.

For additional information on data see the part 1 EDA in Method section. The tables below show the data types in the data files, which containing the gameplay events. After checking the number and percentage of missing values in each column, we find that the missing value is 0

3.2 Exploratory Data Analysis

There are 17K different installation id in train and 1K in test sets, and no overlap between train and test datasets. Besides, one installation id can have one more game session, and a lot of installation id do not have an "Assessment" type at the train dataset: from the total of 17,000 installation id, only 4242 installation id have an "Assessment" type.

To get a clearer overview about train and test datasets, the number of event tittles and number of event code are collected in the both train and test datasets. The figures are shown below:

From figures above, the top event titles and event codes are quite similar in train and test set. This indicates that the event title as well as event code for both train and test have similar pattern, even though the total number of events is different. Therefore, the training model can represent the test model well, and sampling them to get the similar pattern is not required in this problem.

Besides, the distribution of accuracy group is being collected. The result is shown as below:

In this figure 3, the highest count for accuracy group is 3, which is around 7000, followed by accuracy group with 0, which is around 4000. This indicates that the dataset at least in the training part is some extent unbalanced, which will be considered in choosing validation schema for training.

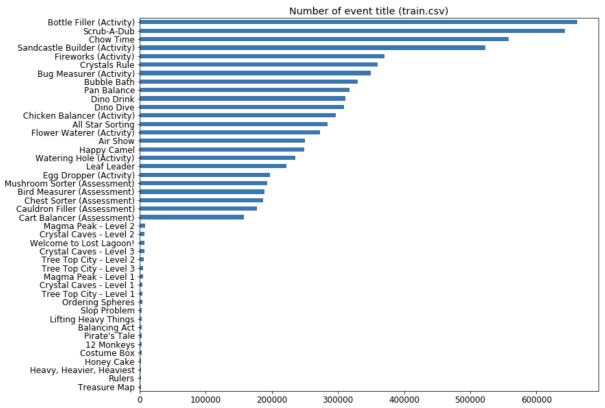
Moreover, the game time difference between different types and worlds are tested in the EDA. The result is shown in figure 4 below:

From the result, we discover that the spending time between types for activity, games and assessment is similar, but time for clip is limited, then it can be ignored in model feature. Also, the time cost in world for MAGMAPEAK, TREETOPCITY and CRYSTALCAVES is similar, but the time for none world is rather limited, so it can also be ignored.

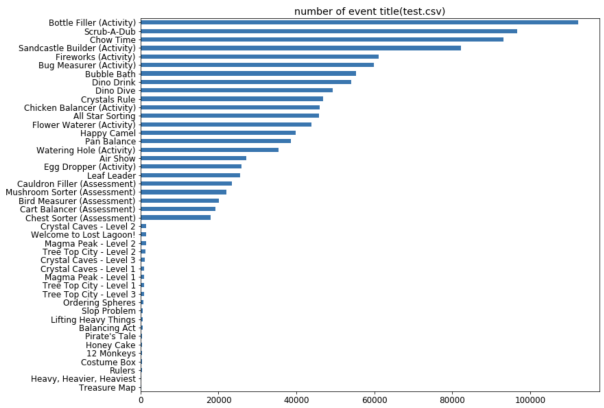
Besides, all I mentioned above, we also explore various distribution data for different event types in both train set and test set, finding that in both sets,

Table2: List of features and their descriptions in the train_label_dataset

Feature name	Type	Description
installation_id	string	Randomly generated unique identifier grouping game sessions within a single installed application instance.
game_session	string	Randomly generated unique identifier grouping events within a single game or video play session.
title	string	Only indicates "Assessment" type in train_label: five different titles for 5 different games in assessment
num_correct	numerical	Correct attempts in each assessment, only has 0/1.
num_incorrect	numerical	Incorrect attempts in each assessment
accuracy	numerical	The portion of correct attempts in total attempts.
accuracy group	numerical	It is described by the total attempts to achieve success. i.e., there are four groups: 3 (the assessment was solved on the first attempt), 2 (the assessment was solved on the second attempt), 1 (the assessment was solved after 3 or more attempts), 0 (the assessment was never solved)

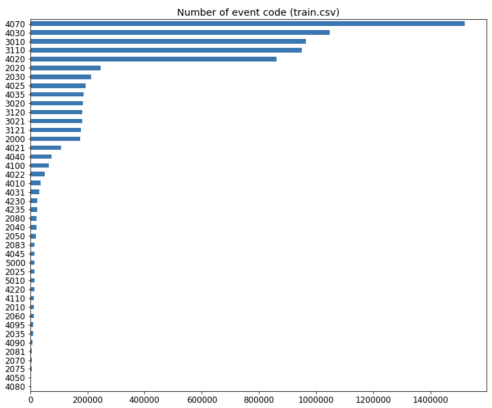


(a) Number of event title (train.csv)

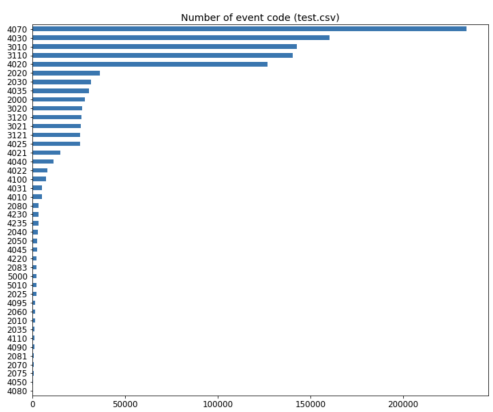


(b) Number of event title (test.csv)

Fig. 1: Statistical number of event tittle in train (left) and test (right) set separately.



(a) Number of event title (test.csv)



(b) Number of event title (test.csv)

Fig. 2: Statistical number of event tittle in train (left) and test (right) set separately.

children took participant more in games, followed by activities, assessment and clips.

3.3 Data Preprocessing

In this part, we want to do some data preprocessing to find the original features which may be related to the children's performance in final assessment. The steps to achieve task are shown as follows.

Firstly, taking the intersection of the installation id in the train set and the train label set. It is possible that some children just install this app, and do not take part in the assessment session, thus this should not be considered in our model, and we only focus on the intersection part.

Secondly, the number of unique kind of categories for each installation id

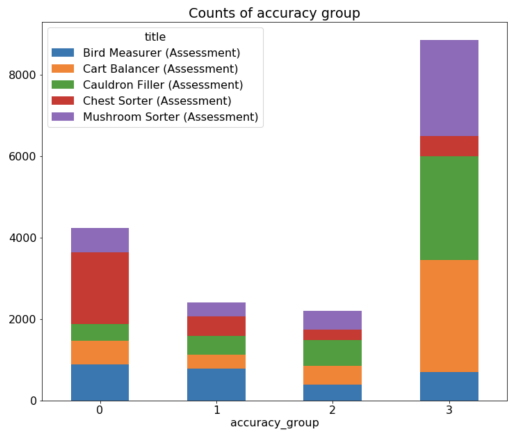
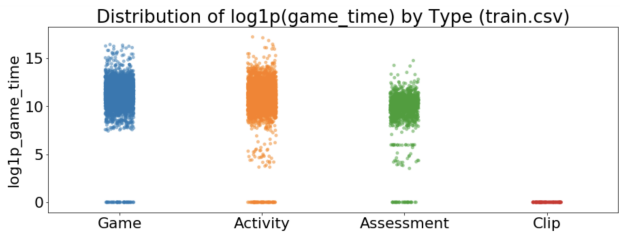
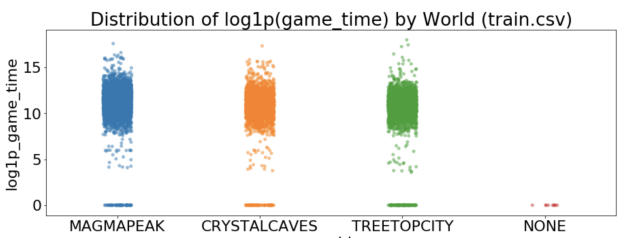


Fig. 3: Distribution of accuracy group in train label



(a) Distribution of game time by type



(b) Distribution of game time by world

Fig. 4: Distribution of game time

should be counted, and there are 5 categories, such as event count, event code, title, type and game session count. After this process, the number of game session as well as other related information the player has can be get.

Thirdly, more detailed information inside the large categories mentioned above will be added as features. For instance, there are four different types of the game session, such as activity, game, assessment and clip, and the number of each type should be counted, and set them separately as features. It is the same process for other categories. The important thing is that make sure to delete the NA in the result. This will exist because if the user just installed the app without playing it, the duration time is 0, and the returning result is NA for this feature.

Finally, accuracy groups information in the $train_label$ need to be added in features. The problem is that children have different performance in different games, and we have got five different games. The method we took to solve such problem is that expand an installation id into 5 with different games names, and other features remain the same. All features we got in data preprocessing is shown in table below:

4 Methods

4.1 Baseline

- Matching features and labels
  After data preprocessing, to define our baseline model, the matches between existing features and labels should be determined, which is also the trickiest part in this project. As the data section mentioned, the features are concluded from a series of game records from each installation instance. However, each installation instance has several assessments with different accuracy group, and the only different between those assessments for the same installation instance is Assessment type, which is a feature attached with each assessment in train label.csv. Even for the same installation instance and doing the same type of assessment, the accuracy group could be different because some degree of randomness could be involved. In order to prevent overfitting and optimize the performance, after some round of experiments, this project matches the features and labels by randomly selecting one assessment instance as the accuracy group label, and the feature Assessment type is also included in features. Another strategy to define the label is majority vote for each installation id.

- Model training
  We use lightGBM as our model framework. LightGBM is a popular gradient boosting framework that uses tree based learning algorithms, and this framework is capable of achieving machine learning algorithm with lower memory usage in large scale data. The model we choose is random forest classifier defined in lightGBM. The input data is the summarized feature with its installation id in the dimension of 3614 rows and 196 columns. After that, each installation id is attached by a label generated from $trainlabel.csv$ randomly. Because it is an ongoing contest, we could not obtain the real test set, so we split 80 percentage of installation instances as training set and 20 percentage of installation instances as test set.
  In training process, we also use cross validation with 5 folds to reduce the degree of overfitting. The training error and validation error in each iteration are recorded to monitor the level of overfitting. The loss curve of baseline model could be plotted in Fig.5(f). It could be found that in the around 670 iterations, the validation loss reaches its lowest point and the iteration stops. The train loss is still far lower than validation loss, but the cross validation prevents the further train set overfitting.
  As a tree-based model, the feature importance could be obtained directly after training. Fig.6 shows the top 10 most important features. The most important feature for baseline model is $eventcode2010$, representing the number of game starting events, and the second important feature is $Assessmenttype(input)$, which is the assessment type in trainlabel.csv. The feature importance provides both a further insight of the features and instruction on the following feature reduction.

- Evaluation
  For evaluation, the contest regards Quadratic Weighted Kappa Matrix as the criteria of performance evaluation. It is a popular method to evaluate the amount of agreement. The formula is defined as:

. Firstly, the weight is calculated by the square of index difference divided by a coefficient related to classifier dimension. After that, the kappa matrix is obtained by the outer product between the true histogram vector of outcomes and the predicted histogram vector,

Table3: List of features and their descriptions after preprocessing

Feature name	**Type**	**Description**
Title_count	numerical	The number of unique titles for each installation_id
Event code count	numerical	The number of unique event code count for each installation_id
Event id count	numerical	The number of unique id count for each installation_id
World count	numerical	The number of unique world count as well as the number of counts for each world for each installation_id, i.e., the number of counts for world TREETOPCITY
Type count	numerical	The number of unique type count for each type for each installation_id
Game count	numerical	The number of unique games for each installation_id
Max duration time	numerical	The max duration time for each installation_id
Type activity count	numerical	The number of activities for each installation_id
Type assessment count	numerical	The number of assessments for each installation_id
Type clip count	numerical	The number of clips for each installation_id
Type game count	numerical	The number of games for each installation_id
Event code_xxxx	numerical	The number of counts for each event_code for each installation_id separately, i.e., event_code_2010
Accuracy group	numerical	The distribution of accuracy group for each installation_id

normalized such that E and O have the same sum.
Also, ROC curve is also used to validate the efficacy of our model. For baseline, the performance of ROC curve is plotted in Fig.5(a). Also, the average quadratic weighted kappa score for baseline model is 0.663, which could be further improved by feature engineering.

- front-end result:
The user interface is shown as the figure below. User can choose one of the four unused datasets we have already generated. After that one of three models will be chosen by user. The 'use and run' button will trigger the program in the backend to generate the ROC curve as well as a Kappa score. The ROC curve and score will be then sent to the web page and be shown to the user. Below the ROC curve, every model's feature importance figure and loss function figure is shown as well. User can compare the performance of each model by checking the Kappa score.

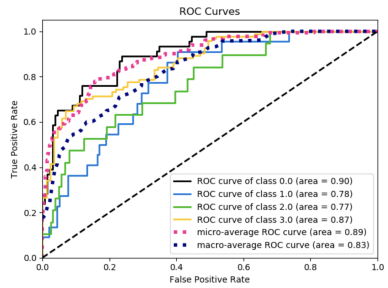
Experiments

In order to improve the performance, we further explore the features and label match strategy. There are three algorithms with different performances explored.

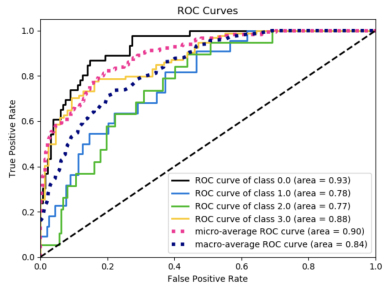
- Using random selection as feature-label match strategy
Instead of using majority vote, we apply some randomness to feature-label match. For each installation id, the model randomly picks one of the rows and extracts the input features and labels from those selection. However, due to more randomness are introduced to the model, the performance of the model is no longer highly related to the seed we set, and the variance of kappa score is more stable and variance is smaller.

- Using several random selections (bagging) as feature-label match strategy
In this part, we introduce another feature-label match strategy similar to the strategy introduced in 2.1. Similar to bagging, we could build several weak predictors and combine them as a strong predictor with iterative random selections. In order to use the train label.csv more comprehensive than baseline, the whole information could be applied in our model by iterative selecting one random choice for each installation instance, then combine those models to predict the result. The average kappa score is 0.638, and the ROC plot is shown in Fig.5(b).
It could be found that the overall performance is worse than the baseline. This could be caused by the confusing labels. Even for the same features, the label could be different because the children performances may contain some randomness and instability. With this algorithm, the instability is exaggerated by repeating deny the previous decisions.

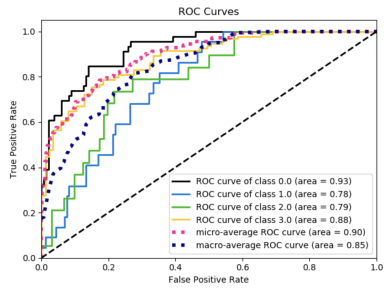
- Feature selection based on Feature importance
This feature engineering method is mainly implemented by involving the series of most important features in the baseline. By only including those important features, the redundant and dependent features are deleted and the model could obtain a better performance. The number of features included in this model is 30. We also implement an experiment to identify the number of included features.



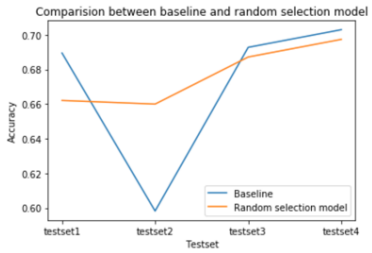
(a) Baseline model



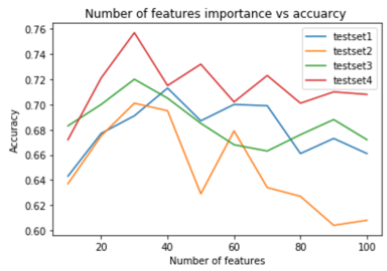
(b) Baseline model with random selection



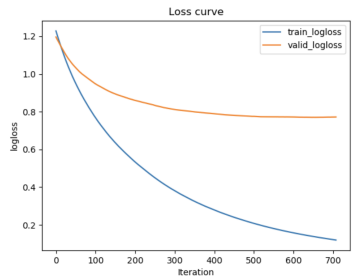
(c) Best model with feature engineering



(d) Comparison between baseline and random selection on four datasets



(e) Experiment on number of features includes



(f) Loss curve

Fig. 5: Experimental results

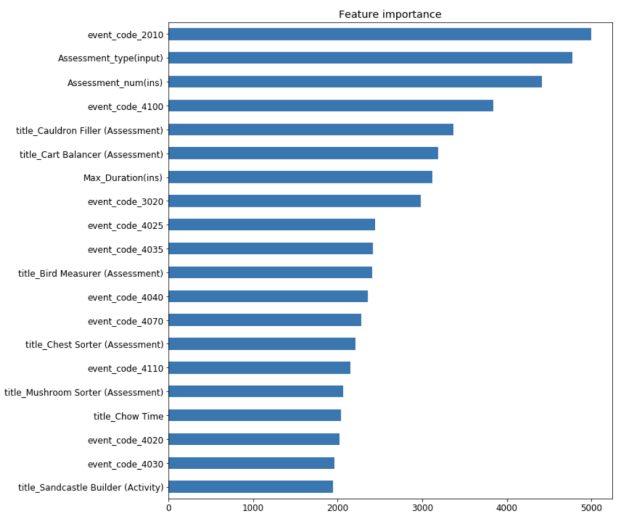


Fig. 6: Feature Importance

It could be found that the overall performance is worse than the baseline. This could be caused by the confusing labels. Even for the same features, the label could be different because the children performances may contain some randomness and instability. With this algorithm, the instability is exaggerated by repeating deny the previous decisions.

System Overview

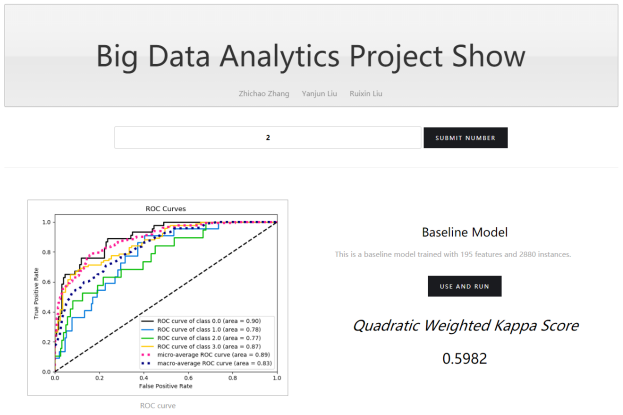
The system of our project basically consists of three parts: front-end web representation, back-end data processing and a Flask web framework to connect front-end and back-end.

Flask is a micro web framework written in Python. The reason why we use Flask is that it does not require particular tools or libraries pre-installed before using it. Although small in size, it supports extensions that can add application features as if they were implemented in Flask itself.

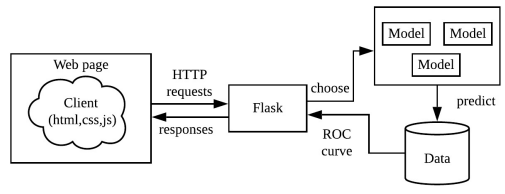
AJAX, shorted for Asynchronous JavaScript and XML, is a set of web development techniques using many web technologies on the client side to create asynchronous web application. Using AJAX, a web page can be updated without being reload, and requesting, receiving, sending data via server in the background is supported. Because the advantages AJAX has, in our project we use AJAX as the method of transmitting data between front-end and back-end.

HTTP method was used in our project. It defines a set of request methods to indicate the desired action to be performed for a given resource. Relative methods include GET, POST, PUT and DELETE, etc. In our project, most of the methods are POST method. In the respect of program language choosing, HTML, CSS, JavaScript were used in the front-end, and Python was used in the back-end.

The detailed process is as following: Users can pick one of the four unused datasets we have already generated, as well as one of our three mathematical models. The reason why we choose to have four datasets is that the use of a single dataset may cause the problem of overfitting, thus we made four datasets to generate randomness. The choices user made will be sent to the back-end using HTTP method via Flask framework in AJAX manner. After receiving the choices, a ROC curve as well as a Kappa score will be generated using that model on that data set. The ROC curve and the score will be sent back to the web page and be shown in the user interface.



(a) Front End UI example



(b) System Overview

Fig. 7: Front end

Conclusion

Using the model of Gradient Boost, we successfully predicted the performance of children in PBS KIDS Measure Up! APP. The highest Kappa score is up to 0.7 in one of the datasets. As a result, we achieved the goal of predicting the performance of children using machine learning algorithm.

Acknowledgements

We would like to thank Professor Ching-Yung Lin for his excellent lecture and helpful comments.

References

[1] Liu, J. and Wu, C. (2017). A gradient-boosting decision-tree approach for firm failure prediction: an empirical model evaluation of Chinese listed companies. The Journal of Risk Model Validation, 11(2), pp.43-64.

[2] Guolin Ke, Qi Meng (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. NeurIPS 2017.

[3] Ho, Tin Kam (1995). Random Decision Forests . Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282. Archived from the original (PDF) on 17 April 2016. Retrieved 5 June 2016.

[4] R. Genuer, J. Poggi, C. Tuleau-Malot and N. Villa-Vialaneix, "Random Forests for Big Data"Big Data Research, vol. 9, pp. 28-46, 2017. Available: 10.1016/j.bdr.2017.07.003.

[5] A. Weinberg and M. Last, "Selecting a representative decision tree from an ensemble of decision-tree models for fast big data classification", Journal of Big Data, vol. 6, no. 1, 2019. Available: 10.1186/s40537-019-0186-3.

Appendix

- Yanjun Liu (30%)
  Yanjun Liu contributes to explore data analysis as well as data preprocessing to find the original features which may be related to the children's performance in final assessment.

- Ruixin Liu (35%)
  Ruixin Liu contributes to design the front-end user interface as well as web paging building, build the framework connecting back-end and front-end using Flask and use model already generated to make prediction on dataset to get ROC curve in the back-end.

- Zhichao Zhang (35%)
  Zhichao Zhang contributes to machine learning models and feature engineering, providing baseline model as well as improved models to generate prediction on this problem. Also contributes to the Latex typesetting.