Kaggle Competition

Child Mind Institute — Problematic Internet Use

Team member: 張原鳴, 馮紹哲, 方俊平, 吳孟維, 黃子軒

Final score (ranking): 45.7 (85/3629)

Introduction

Our goal is to predict the level of problematic internet usage exhibited by children and adolescents, based on their physical activity. The dataset contains a main field “sii”, which stands for “Severity Impairment Index”. This value ranges from 0 to 3. 0 is None and 3 is Severe. In this work, we tried many data preprocessing, (e.g. feature aggregation and analysis) and machine learning methods (e.g. Random Forest, the ensemble models). With lots of discussion and improvement, our work finally gets a good result.

Related Works

We found some public notebooks on Kaggle, which gives us some ideas about our implementation. KLeeDG, one of these notebook authors in this Kaggle competition, implements Stacking Regressor and Voting Regressor. This method ensembles multiple models and generates the final submission results with majority voting. The base models he considered to train include LGBM, Random Forest, CatBoost, XGB, TabNet, ExtraTrees, HistGradientBoosting, Ridge, and SVR. Though the execution time is very long, it combines the advantages of these models and shows good performance in this competition, which inspires us to get deep on it.

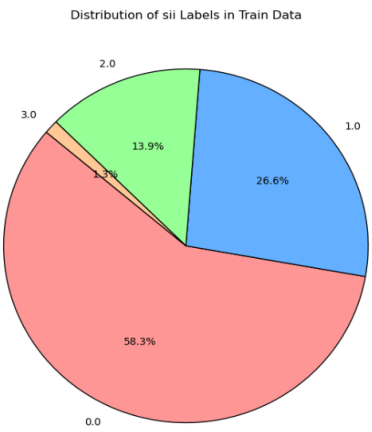
In the early stage we used **SMOTE, Synthesized Minority Oversampling Technique**. This is a sampling method for solving imbalanced dataset problems. Its main idea is to synthesize some sample around the “rarely-appeared” data and its neighbors, so you can make the dataset more balanced and uniformed. We found that this method surely improves our performance a little, but we did some research and used a better way to handle the imbalanced data, which is K-Fold. We will introduce K-Fold in the next part.

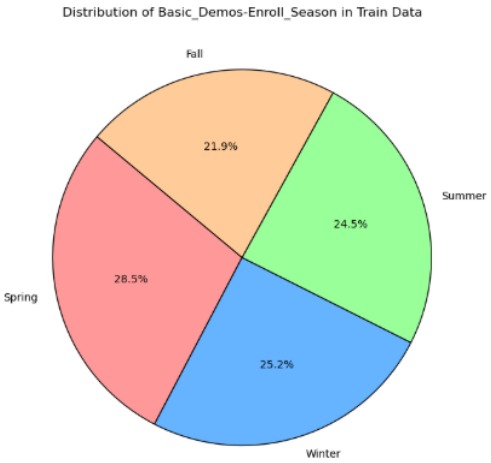
SimpleImputer and **KNNImputer** are classes in the **sklearn.impute** module designed to handle missing values in datasets based on different strategies. SimpleImputer files missing values with a constant value (ex: fill zero) or a statistic (mean, median, or most frequent) from the other values in the column. KNNImputer fills missing values using the mean (or weighted mean) of the nearest k-neighbors and considers the similarity between rows. The row data has a lot of missing values so handling this kind of issue is critical. By using the imputer we don't need to throw away lots of data. However, the drawback of using the imputer is that the estimation will affect the results significantly. If we cannot find a good approximation method to impute the data, our model performance might probably get low.

Methodology

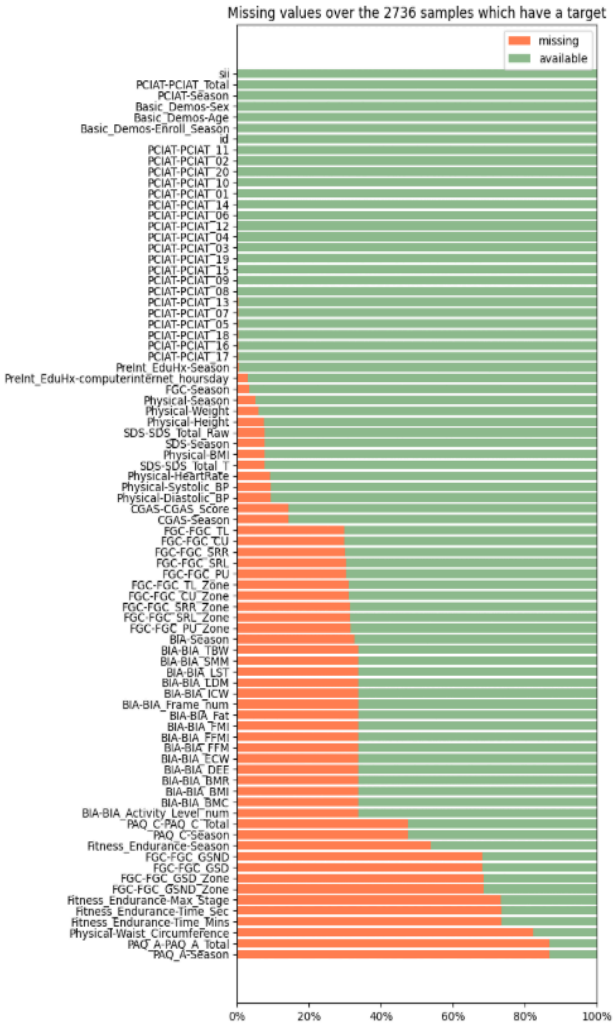
Data Visualization

We try some visualization tasks to better understand the data. Pie charts help us to observe the proportion distribution of the statistics.





The left figure shows the distribution of ‘sii’ labels in training data; we observe that class 0 takes the majority of the label, 58.3%. In a data imbalance issue, the excessively dominated by one class would affect the performance.



The target sii in the training dataset is determined based on the Parent-Child Internet Addiction Test total score (PCIAT-PCIAT_Total). The PCIAT score ranges are used to categorize participants into different classes for the sii labels.

So, We can directly predict sii (this is the value we have to submit), or we can predict PCIAT-PCIAT_Total and then transform this prediction to a sii prediction for submission.

sii	PCIAT-PCIAT_Total min	PCIAT-PCIAT_Total max	count
i64	i64	i64	u32
null	null	null	1224
0	0	30	1594
1	31	49	730
2	50	79	378
3	80	93	34

Data Preprocessing

The data contains two parts: csv file and parquet file. For the csv file, the tabular data in **train.csv** and **test.csv** comprises measurements from a variety of instruments. For the parquet file, there is a time step recording from the HBN study that some participants were given an accelerometer to wear for up to 30 days continually while at home and going about their regular daily lives.

In this part, we try some methods to preprocess the input file to improve the training performance. We merge the mutually exclusive columns, **'PAQ_A' (Physical Activity Questionnaire for adolescents)** and **'PAQ_C' (Physical Activity Questionnaire for children)**, into a single column named **PAQ (Physical Activity)**. Doing the feature merging not only enhances feature representation but also reduces dimensionality. Besides, we observe that the target feature 'sii' is calculated by the sum of the sequences of the **Parent-Child Internet Addiction Test (PCIAT).** Therefore, we do the following operations. First, we **remove rows (participants) with missing value in the 'sii' column** because we do not guarantee that we can fill the completely correct class label for them. Besides, we find that there are some participants whose PCIAT is not complete, which may influence the final score to determine the 'sii' result. We delete the rows whose 'sii' class would change if their missing 'PCIAT' scores all fill in 5 points in order to eliminate the uncertainty of the result.

Model

We have used some machine learning models commonly used in classification tasks. **The Random Forest model** is a widely used learning method designed for both classification and regression tasks. It builds multiple decision trees and combines their predictions to improve accuracy and robustness, offering superior generalization performance. We used this model at an early stage of the competition and obtained basic results. We also tried the **Tabnet (Attentive Interpretable Tabular Learning),** which is a deep learning model specifically designed for tabular data, it uses a sequential attention mechanism to select which features to focus on at each decision step, making it adaptive to the most relevant parts of the input data.

Finally, we use the **ensemble model** as our final version. The ensemble model includes **LightGBM**, **XGBoost**, **CatBoost**, **Random Forest** and **Gradient Boosting**. Each base model is wrapped in a pipeline that includes Simple Imputer for handling missing values and Regressor for machine learning. The ensemble method combines the strengths of multiple models, leveraging their complementary advantages, and enhances the stability.

Additionally, we use some techniques and measurements to optimize the model. We use **Quadratic Weighted Kappa**, which is a popular metric in Kaggle competitions and is especially useful for classification tasks. We calculate QWK between true

labels (y_true) and predictions (y_pred), and measure the agreement between two outcomes.

Besides, we use the concept of **cross validation**. Cross validation is a statistical method used to evaluate the performance of a machine learning model by splitting the dataset into multiple parts. **Stratified K-Fold Cross-Validation** is a type of cross validation and it is useful for imbalanced datasets, we use it to split the data into several folds and ensure that the distribution of target classes in each fold is the same as in the entire dataset. In the training phase, we divide several folds into a training set and a validation set in each iteration and training the model. This approach not only reduces the variance of the model but also uses the data efficiently. The cross validation is usually more reliable in Kaggle competition than public leaderboard (LB). Since your model will be biased toward public LB if you only take its score into consideration.

The following models are the model we use in ensemble model:

- **Random Forest**: Random forest combines the output of multiple decision trees to reach a single result. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the output is the average of the predictions of the trees.
- **Gradient Boosting** combines several weak models, and builds a strong prediction model. Instead of directly focusing on errors (residuals), it improves the model step-by-step by targeting the difference between the actual and predicted values. When decision trees are used as weak models, the method is called **gradient-boosted trees**, which often performs better than random forests.
- **LightGBM** (Light Gradient-Boosting Machine) is a machine learning tool developed by Microsoft, which is designed for tasks like ranking, classification, and regression, and it can handle large datasets efficiently. It builds trees leaf-by-leaf for better accuracy and speed, using a memory-saving histogram-based algorithm. LightGBM is ideal for managing big data since it uses techniques like GOSS (focusing on key data points for faster training) and EFB (combining features to simplify and improve efficiency).
- **XGBoost** (eXtreme Gradient Boosting) is a gradient-boosting library used for structured data. It manages sparse data, allows distributed and parallel training, incorporates L1/L2 regularization to lessen overfitting, and provides adjustable loss functions for a range of tasks. It is a popular option for data science competitions because of its tree pruning.
- **CatBoost** is a gradient-boosting library designed for efficient handling of **categorical data** without extensive preprocessing. It uses techniques like Ordered Boosting to reduce bias and includes regularization to prevent overfitting.

Experiments and Results

We tried several submissions. In the beginning version, we had a poor score because we hadn’t resolved the problem of imbalance data. It is inspiring that after we used SMOTE to handle imbalanced data, associated with the Random Forest model, the score significantly improved. However, the version executed without considering the parquet data and it seems that there must be more improvement when we try the more strongly model. For the improved version, we use the ensemble model and add the parquet data into training. We scored **0.457** as our current version.

Discussions and/or Future Works

Now we have two issues to discuss and try to solve.

1.Deep research of parquet data

For the preprocessing parquet data, we first remove timestamps where the patient isn’t wearing the accelerometer, based on the 'non_wear_flag'. We then drop columns irrelevant to the SII value and pass the x, y, and z accelerometer data through a low-pass filter for noise reduction. Feature extraction is performed on the data to capture important patterns.

For a parquet data with n record, we compute features without a sliding window, such as the low-pass filtered accelerometer data and the length of the acceleration vector. We then apply a sliding window of length 6 (approximately 30 seconds) with a stride of 1, extracting additional features like the acceleration vector change, max-min values of the accelerometer data, and inclination angles, resulting in n-6 records.

To ensure consistency, we use the **describe()** function to summarize the features—count, mean, standard deviation, min, max, and percentiles—leading to an 11x8 dimensional vector. Finally, to further reduce dimensionality, we use an autoencoder to embed the data into a 60-dimensional vector, following a similar approach to others.

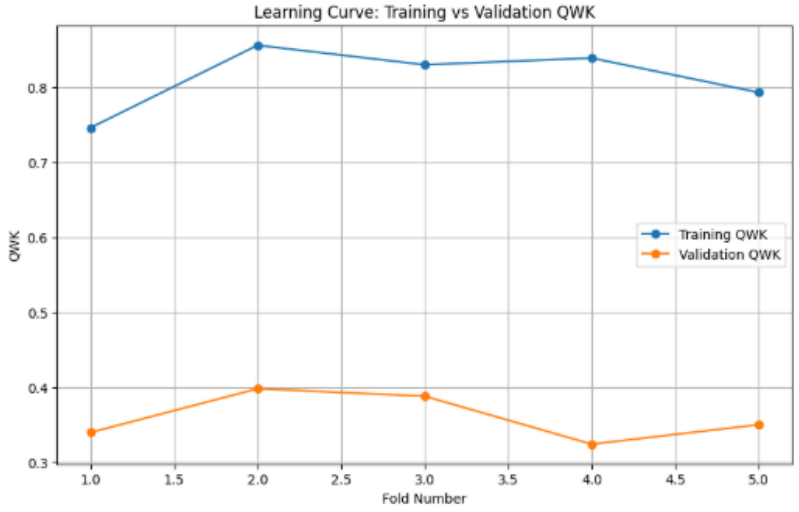
We concatenated the preprocessed CSV vector data with the Parquet 60-dimensional vector from model to train and test. The submission results are shown below:

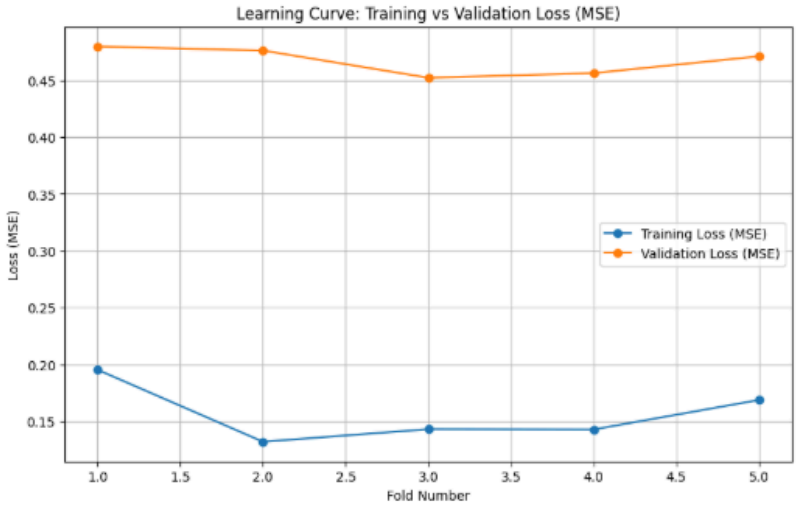
2.Try for feature engineering

For the feature engineering part, we especially concern about the following 3 parts: **Age, SDS** (columns with prefix ‘sds’), and **FGC** (columns with prefix ‘fgc’), since these 3 parts involve multiple columns that seem to have high redundancy and can be combined. With a lot of effort, we do some calculations and combine multiple columns in these 3 aspects, trying to make the data more interpretable, reduce the redundancy, and help the model learn the features contained in these data. However, from our experiment results, the performance didn’t improve significantly, and some even decreased the score. There may be some underlying reasons for that, but we can’t clearly figure out the logic. Due to the limited time, we leave it as future work.

3.Evaluation of the result

We use line plots to visualize the QWK and loss over iterations during both training and validation steps. Following figure is the result, the learning curves in the results demonstrate the comparison between **Training vs. Validation QWK** and **Training vs. Validation Loss (MSE)** across the cross-validation folds.





We observe that although the model performs well in both QWK and loss in the training step, the performance degrades significantly in the validation step. We speculate that the model might be overfitting.

We previously know that one of the benefits of the cross validation is that it could avoid overfitting problems because it provides a more robust way to evaluate a model's performance and ensures that the model generalizes well to unseen data. However, it's weird that there is a contradiction between the understanding and the reality.

For conclusion, we discussed some issues and tasks as our future work:

- Simply the ensemble model because overfitting might be caused by the complexity of our ensemble model.
- Experiment with other cross-validation strategies for potential improvement of the model.
- Try to adjust the configuration and parameters of each model for higher performance.

4.Transformation Target from sii to PCIAT_PCIAT_Total

We switch from predicting direct target 'sii' to predicting a more granular and informative target, PCIAT-PCIAT_Total, which is later transformed into sii. This change likely improves the model's ability to capture subtle relationships in the data due to the more continuous nature of the new target.

5. Hyperparameter Tuning of the LightGBM model using Optuna

The parameters such as **n_estimators, learning_rate, max_depth**, and **num_leaves** were optimized to improve model performance. By combining discussion 4 (using the target PCIAT-PCIAT_Total) and 5 (Hyperparameter Tuning), the model achieved a public score of 0.453 (Second Highest). Although this is not higher than our highest public score model, it resulted in a better mean validation QWK score of 0.3936. This improved validation performance indicates better generalization, which is typically more desirable in most scenarios.

```
# Optimized LightGBM model
optimized_lgb_model = LGBMRegressor(
    n_estimators=107,
    learning_rate=0.1370718043598246,
    max_depth=4,
    num_leaves=68,
    colsample_bytree=0.7238253848440545,
    subsample=0.5543321896833934,
    reg_alpha=7.854917349399392,
    reg_lambda=8.769761580128085,
    min_child_weight=1.105089752114477,
    random_state=42,
    force_col_wise=True,
    verbose=-1
)
```

```
Training Folds: 100%|██████████| 5/5 [10:36<00:00, 127.28s/it]

Fold-wise QWK Scores:
Fold 1: Train QWK = 0.6134, Validation QWK = 0.4186
Fold 2: Train QWK = 0.7134, Validation QWK = 0.3850
Fold 3: Train QWK = 0.7291, Validation QWK = 0.4030
Fold 4: Train QWK = 0.6979, Validation QWK = 0.3357
Fold 5: Train QWK = 0.7048, Validation QWK = 0.4256

Mean Train QWK --> 0.6917 ± 0.0405
Mean Validation QWK ---> 0.3936 ± 0.0321

Best thresholds: [30.63849172 37.99271247 79.        ]
Best QWK score after optimization: 0.4521
----> || Optimized QWK SCORE ::  0.452
```