Why did we fail

1st Le Xuan An
22028131@vnu.edu.vn

2nd Nguyen Thanh Phat
22028108@vnu.edu.vn

3rd Pham Cong Minh
22028249@vnu.edu.vn

***Abstract*—Problematic Internet Use is a Kaggle competition organized by the Child Mind Institute that involves predicting problematic internet use (PIU) in children using actigraphy data, which is plagued by missing values, noise, and variability. We have implemented approaches based on publicly available notebooks and discussions. Overall, the results from cross-validation and evaluations on the public leaderboard indicate promising outcomes. However, the results from the private leaderboard suggest a discrepancy, indicating that the optimistic outcomes observed during cross-validation and on the public leaderboard may not generalize effectively. This leads to our observation that most of the approaches try to fit the leaderboard too much, therefore they will mostly overfitted, including us. This report will explain that problem.**

I. Introduction

Problematic Internet Use (PIU), which affects approximately 20–45% of individuals worldwide [1], is characterized by excessive internet engagement and is associated with depression, anxiety, ADHD, aggression, reduced physical activity, and diminished life satisfaction [1]–[5]. This issue is particularly concerning for children and adolescents, as their developmental stage makes them more susceptible, potentially impacting their long-term health and social well-being [6], [7]. Assessing PIU often necessitates professional evaluations, which may be inaccessible to many families. However, physical fitness metrics, such as activity levels measured by accelerometers, provide a more accessible alternative. Leveraging these indicators to predict PIU facilitates early interventions and supports the development of evidence-based health and education policies.

The Child Mind Institute (CMI) Kaggle competition on PIU prediction garnered substantial interest from the AI/ML community due to the challenge's complexity and importance [8]. The competition dataset integrates tabular data with time-series actigraphy files, containing accelerometer readings collected over several days. However, the dataset presents significant challenges, including missing labels in approximately 30% of the training data, noisy and heterogeneous information, and biases stemming from both systemic and random factors. These challenges reflect real-world psychological research's inherent complexities, posing considerable data processing and modeling difficulties. The challenge involves predicting the Severity Impairment Index (SII) using a provided dataset that includes time-series data. The dataset combines temporal patterns with other potential features, requiring robust modeling techniques to effectively capture and predict the target variable. Since the competition's launch, nearly 3,500 teams have participated, publishing hundreds of notebooks. Although some high-ranking notebooks achieved impressive public leaderboard scores (e.g., 0.494 and 0.497), they have been criticized for their lack of reliability, with issues such as data leakage, data drift, and overly complex or redundant code making their approaches questionable. These challenges highlight the competition's core difficulty: designing a robust pipeline to handle missing data, mitigate noise, and prevent overfitting while effectively leveraging both tabular and time-series components of the dataset. Participants have employed various techniques, including null value handling, imputation, and autoencoders, to address the missing data problem. Despite these efforts, the substantial proportion of missing data poses a heightened risk of data leakage, leading to discrepancies between cross-validation and leaderboard scores and increasing the likelihood of overfitting. For time-series data, some teams have opted for simpler approaches, such as extracting statistical features (e.g., mean, standard deviation, trend analysis). However, these methods often oversimplify the temporal structure of the data, limiting their effectiveness in capturing the intricate patterns within actigraphy signals and potentially reducing model performance.

We developed a solution by integrating insights from public approaches and conducting our own experiments. While the results from cross-validation and the public leaderboard appeared promising, the private leaderboard scores revealed a notable discrepancy. Why did this happen? After reviewing other winning approaches and consulting with peers, we identified two potential explanations for the strong performance of these approaches. Data processing and strict cross-validation process play a key role in their results.

II. Dataset

The dataset used in the competition Child Mind Institute - Problematic Internet Use is derived from the The Healthy Brain Network (HBN) dataset, which is a clinical sample of about five-thousand 5-22 year-olds who have undergone both clinical and research screenings. The objective of the HBN study is to find biological markers that will improve the diagnosis and treatment of mental health and learning disorders from an objective biological perspective. Two elements of this study are being used for this competition: physical activity data (wrist-worn accelerometer data, fitness assessments and questionnaires) and internet usage behavior data. The goal of this competition is to predict from this data a participant's Severity Impairment Index (sii), a standard measure of problematic internet use.

The competition data is compiled into two sources, parquet files containing the accelerometer (actigraphy) series and csv files (train.csv and test.csv) containing the remaining tabular data. The majority of measures are missing for most participants. In particular, the target sii is missing for a portion of the participants. The sii value is present for all instances in the test set.

A. Tabular data

The tabular data in train.csv and test.csv comprises measurements from a variety of instruments. The fields within each instrument are described in data_dictionary.csv. These instruments are as follows.

1) **Demographics** - Information about age and sex of participants.
2) **Internet Use** - Number of hours of using computer/internet per day.
3) **Children's Global Assessment Scale** - Numeric scale used by mental health clinicians to rate the general functioning of youths under the age of 18.
4) **Physical Measures** - Collection of blood pressure, heart rate, height, weight and waist, and hip measurements.
5) **FitnessGram Vitals and Treadmill** - Measurements of cardiovascular fitness assessed using the NHANES treadmill protocol.
6) **FitnessGram Child** - Health related physical fitness assessment measuring five different parameters including aerobic capacity, muscular strength, muscular endurance, flexibility, and body composition.
7) **Bio-electric Impedance Analysis** - Measure of key body composition elements, including BMI, fat, muscle, and water content.
8) **Physical Activity Questionnaire** - Information about children's participation in vigorous activities over the last 7 days.
9) **Sleep Disturbance Scale** - Scale to categorize sleep disorders in children.
10) **Actigraphy** - Objective measure of ecological physical activity through a research-grade biotracker.
11) **Parent-Child Internet Addiction Test** - 20-item scale scored from 0-5 that measures characteristics and behaviors associated with compulsive use of the Internet including compulsivity, escapism, and dependency.

Note in particular the field PCIAT-PCIAT_Total (Total score of Parent-Child Internet Addiction Test). The target sii is derived from this field as described in the data dictionary: 0 for None, 1 for Mild, 2 for Moderate, and 3 for Severe. Additionally, each participant has been assigned a unique identifier id.

B. Actigraphy data

During their participation in the HBN study, some participants were given an accelerometer to wear for up to 30 days continually while at home and going about their regular daily lives.

series_train—test.parquet/id=id - Series to be used as training data, partitioned by id. Each series is a continuous recording of accelerometer data for a single subject spanning many days.

1) **id** - The patient identifier corresponding to the id field in train/test.csv.
2) **step** - An integer timestep for each observation within a series.
3) **X, Y, Z** - Measure of acceleration, in g, experienced by the wrist-worn watch along each standard axis.
4) **enmo** - As calculated and described by the wristpy package, ENMO is the Euclidean Norm Minus One of all accelerometer signals (along each of the x-, y-, and z-axis, measured in g-force) with negative values rounded to zero. Zero values are indicative of periods of no motion. While no standard measure of acceleration exists in this space, this is one of the several commonly computed features.
5) **anglez** - As calculated and described by the wristpy package, Angle-Z is a metric derived from individual accelerometer components and refers to the angle of the arm relative to the horizontal plane.
6) **non-wear_flag** - A flag (0: watch is being worn, 1: the watch is not worn) to help determine periods when the watch has been removed, based on the GGIR definition, which uses the standard deviation and range of the accelerometer data.
7) **light** - Measure of ambient light in lux. See here for details.
8) **battery_voltage** - A measure of the battery voltage in mV.
9) **time_of_day** - Time of day representing the start of a 5s window that the data has been sampled over, with format %H:%M:%S.%9f.
10) **weekday** - The day of the week, coded as an integer with 1 being Monday and 7 being Sunday.
11) **quarter** - The quarter of the year, an integer from 1 to 4.
12) **relative_date_PCIAT** - The number of days (integer) since the PCIAT test was administered (negative days indicate that the actigraphy data has been collected before the test was administered).

III. Evaluation

IV. Exploratory Data Analysis

A. Overview

The train.csv file contains 3960 samples, however there are only 996 parquet files in series_train folder, which means we are missing about 3/4 of series data for training set. The full test set according to the competition description contains 3800 samples, of which about 38% are used for calculating public score, the rest 62% are used for calculating private score.

Analyzing the train.csv file, we can see that there are 82 columns which comprise of id, sii and 80 measures from the instruments. Meanwhile the test.csv file (which store the

20 samples for testing format) only has 59 columns with no sii column and all 22 PCIAT-related measures columns, which implies that the test set will not have any PCIAT-related feature.

The target variable sii is defined as:

- **0: None** (PCIAT-PCIAT_Total from 0 to 30)
- **1: Mild** (PCIAT-PCIAT_Total from 31 to 49)
- **2: Moderate** (PCIAT-PCIAT_Total from 50 to 79)
- **3: Severe** (PCIAT-PCIAT_Total 80 and more)

This makes sii an ordinal categorical variable with four levels, where the order of categories is meaningful.

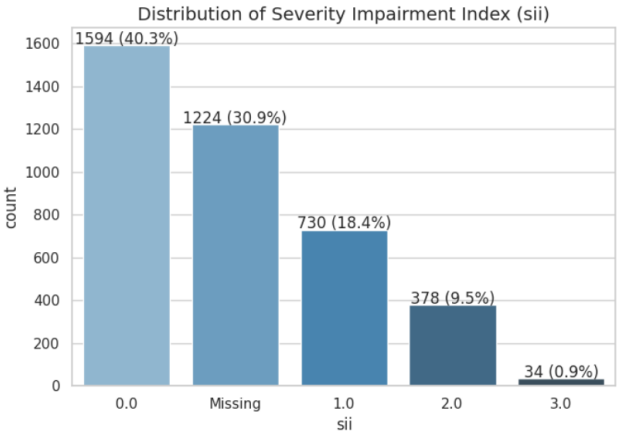

Fig. 1. Distribution of target sii

Overall, the dataset is heavily skewed with approximately 40% of participants not affected by Internet use but only under 1% are severely impaired. Besides, the dataset has a lot of missing target with almost 1/3 without sii value.

B. Features

1) Missing data: Not only is the target value missing, a significant proportion of the data contains missing values in fields, up to 40.5% . Additionally, 77 out of 80 feature columns have missing values.

Features with similiar missing rate tend to come from the same instrument and there are some features with high missing rate like Fitness_Endurance_Time Sec, Fitness_Endurance_Time_Mins, Fitness_Endurance_Max Stage, PAQ_A-Season,..., suggesting that the data collection process was flawed or the data was not recorded fully for those samples. Some with a missing rate of more than 80% may make it difficult to apply data imputation strategies and may require the removal of rows if recovery is impossible.

sii is derived from PCIAT columns but in some samples with target sii, there are still some missing PCIAT values, indicating that the sii value can be incorrect and noisy. Further analysis on recalculating sii can be found in this notebook [**?**].

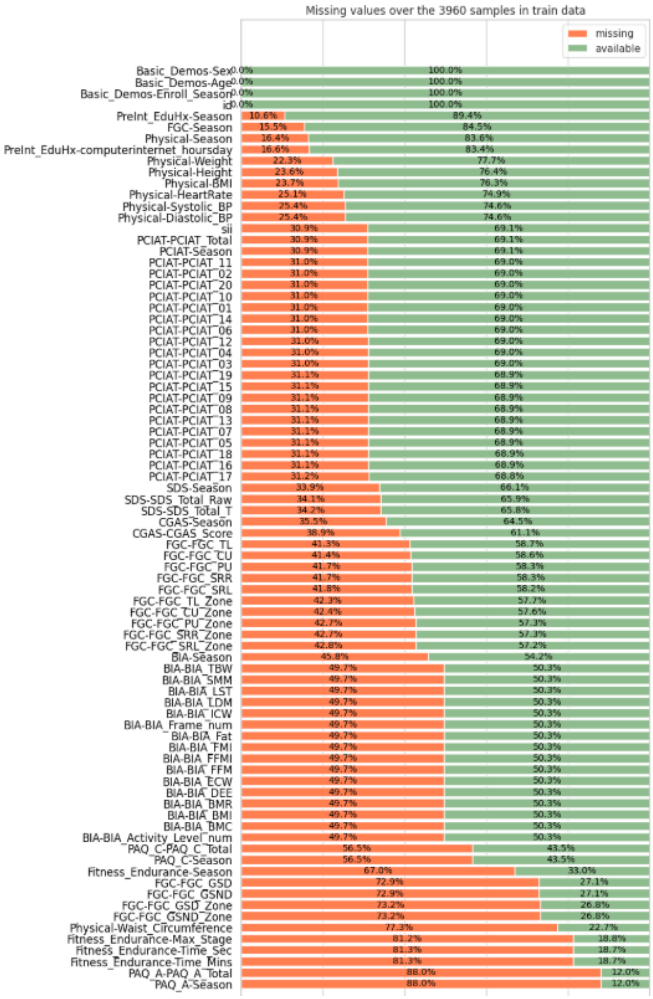

Fig. 2. Missing values in train data

C. Find Outliers

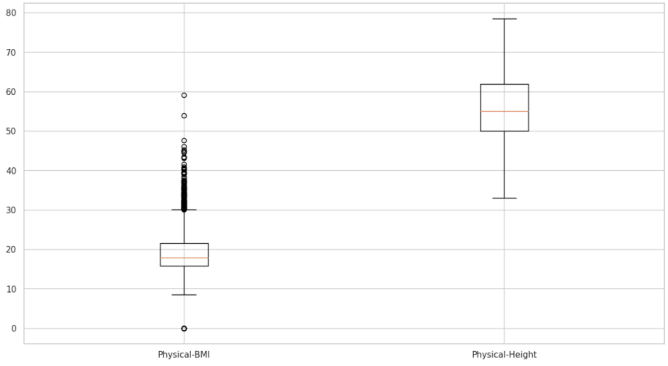

Fig. 4. Physical BMI and Height Boxplot

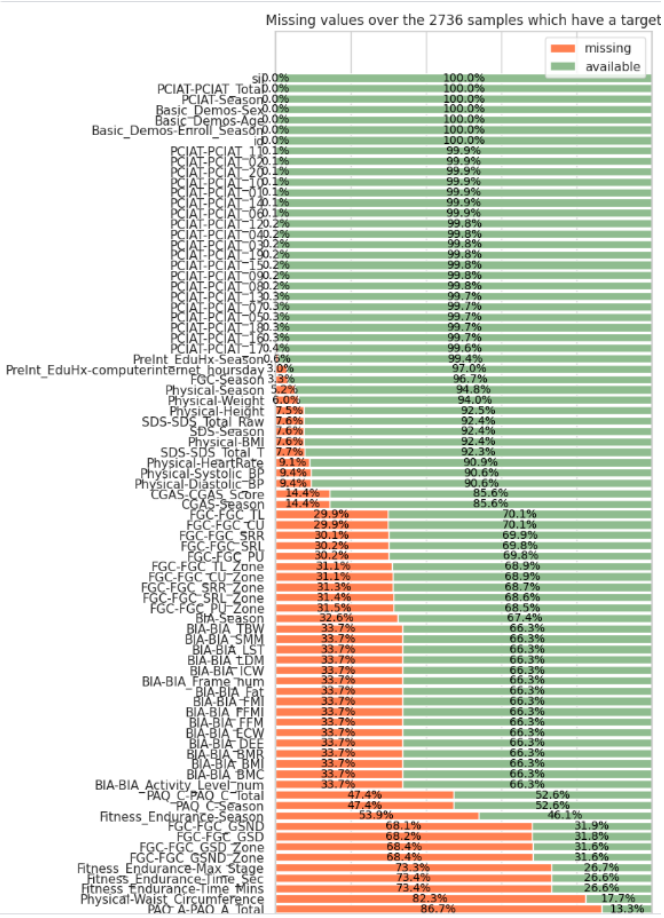

Fig. 3. Missing values in train data with target

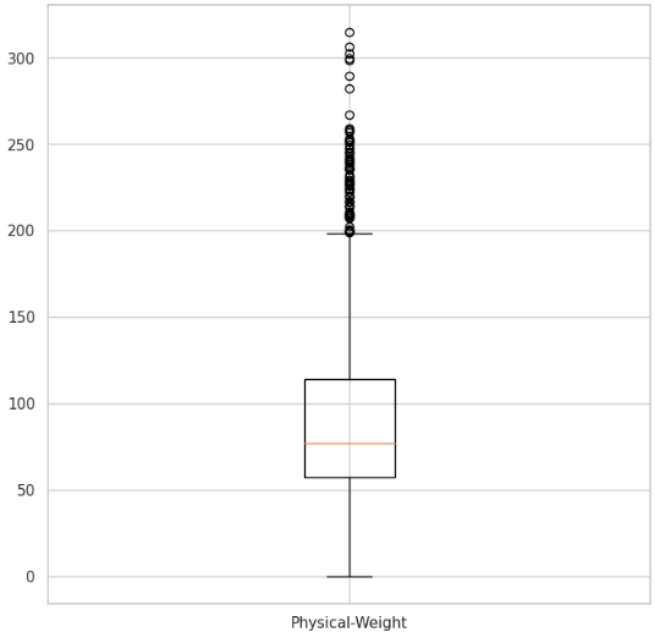

Fig. 5. Physical Weight Boxplot

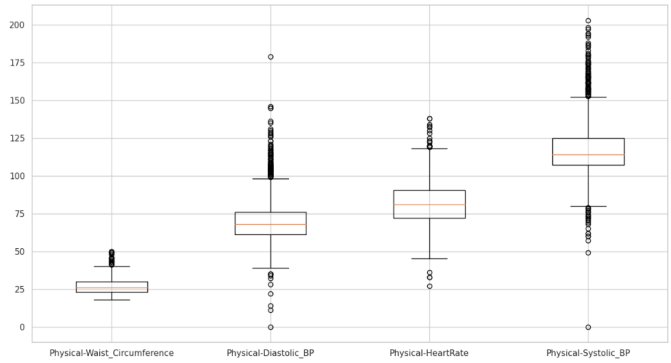

Fig. 6. Physical Boxplot

1) Physical Measures: From the boxplots, we can see that there many outliers in BMI, Weight, Waist_circumfernce, Diastolic_BP, HeartRate, Systolic_BP.

For example, Diastolic Blood Pressure, Body Mass Index and Systolic Blood Pressure in training set have zero value although they are real values and often vary within a range and can not be zero.

Dialostic_BP is usally around 80mmHg for a normal person yet there are many values above 100 here,

Systolic_BP is usually less than 120mmHg but there are many values even above 175mmHg

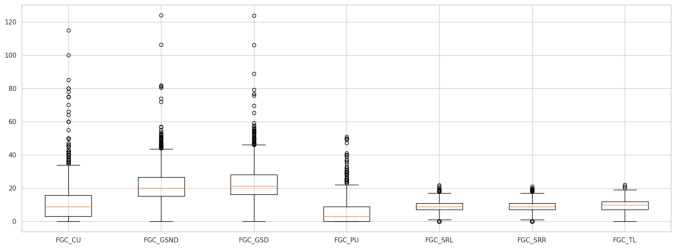

Fig. 7. FitnessGram Boxplots

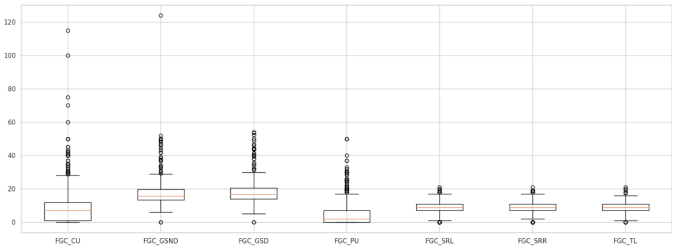

Fig. 8. FitnessGram Boxplots for children (age ¡= 12)

2) FitnessGram: The FitnessGram continuous features comprise of CU (Curl up total), GSND (Grip Strength total (non-dominant)), GSD (Grip Strength total (dominant)), PU (Push-up total), SRL (Sit & Reach total (left side)), SRR (Sit & Reach total (right side)), TL (Trunk lift total)

For children age 12 and below, it is odd that they can curl up more than 100, or their grip strength is higher than 120, the highest out of all participants, which is likely an error.

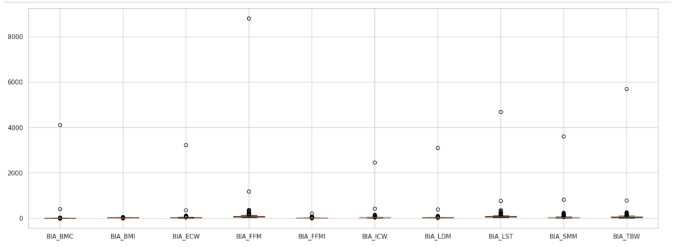

Fig. 9. BIA Boxplots 1

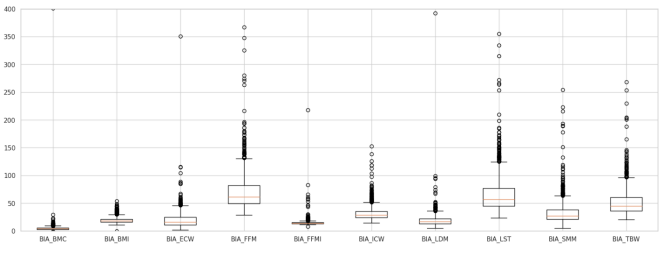

Fig. 10. BIA Boxplots 1 without extreme outliers

Fig. 11. BIA Boxplots 2

Fig. 12. BIA Boxplots 2 without extreme outliers

3) Bio-electric Impedance Analysis (BIA): Bio-electric Impedance Analysis gives data about BMC (Bone Mineral Content), BMI (Body Mass Index), BMR (Basal Metabolic Rate), DEE(Daily Energy Expenditure), ECW(Extracellular Water), ...

There are many outliers here for BIA values, many are extremely higher than the rest and no real human can have those value, for example one DEE value is more than 120000, or one BMR value is more than 800000, one BMI value is 0 when this is impossible, one BMC value is higher than 4000, one ECW value is higher than 3000,... Hence we need to remove these outliers to denoise our data.

D. Correlation

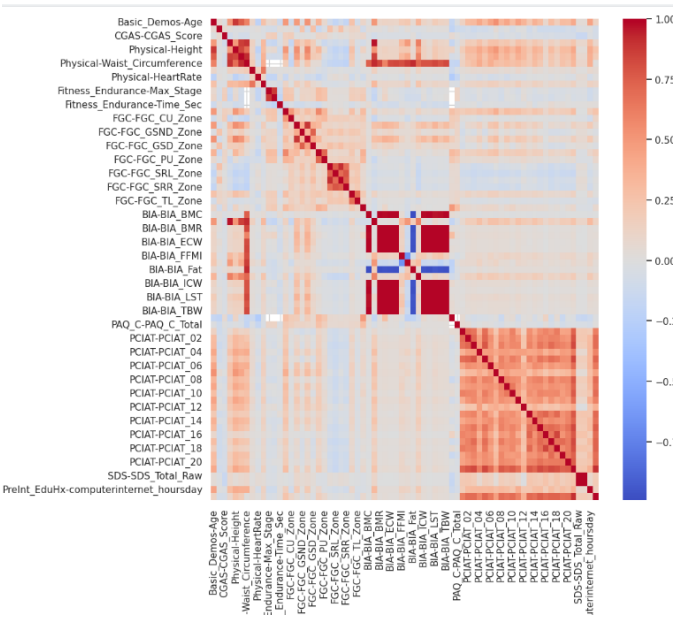

Fig. 13. Heatmap of filtered features

From the heatmap of filtered features (we decided to drop out the season columns because they seem not important), it is clear that the measures from BIA, Physical and FitnessGram have high interrelations.

Fig. 14. Heatmap of physical measures

From the heatmap, some features like BMI, Weight, Waist_Circumference, Height have corelation all about 0.5 or greater, so one or two measures here can be dropped.

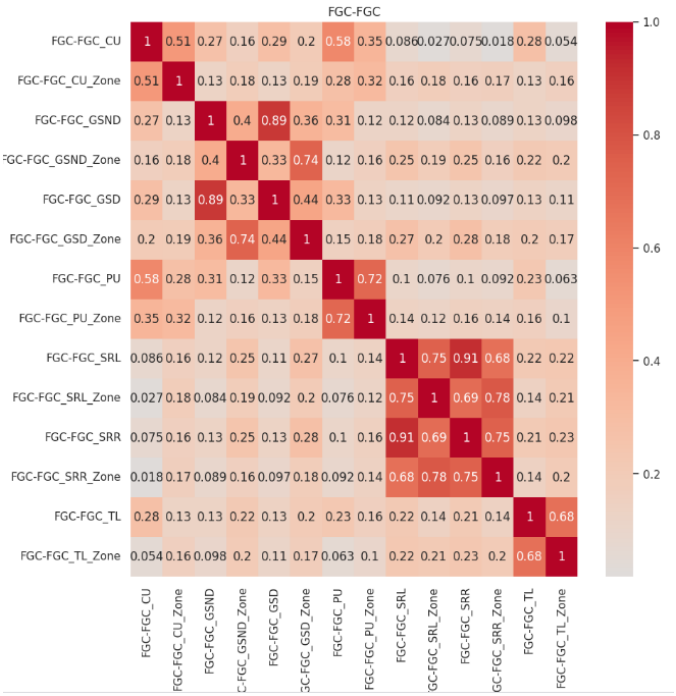

Fig. 15. Heatmap of FGC measures

From the heatmap, some features like GSND and GSND_Zone or GSD and GSD_Zone have high correlation, which is expected because of their descriptions but overall the data here should be kept.

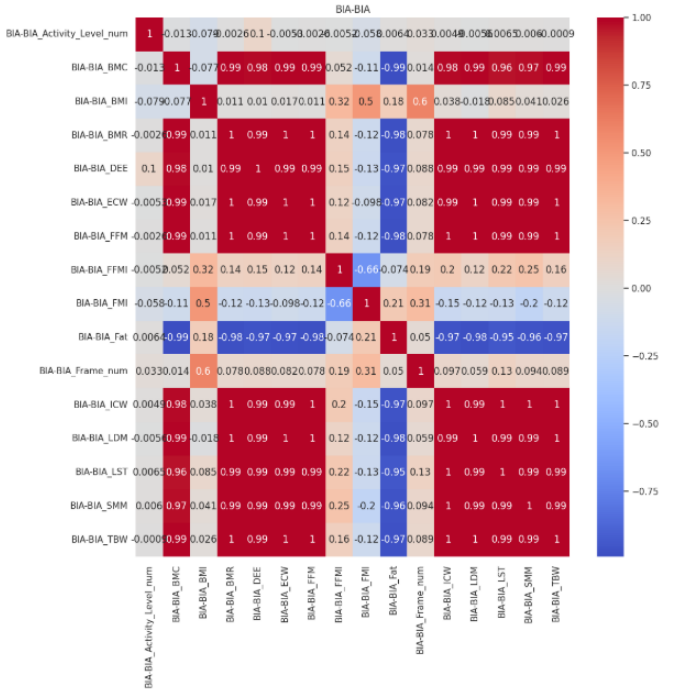

Fig. 16. Heatmap of BIA measures

From the heatmap, some groups of BIA features are highly correlated (which is understandable because many measures in BIA test can be calculated by equations), so we can decide to keep one feature from each group.

E. Actigraphy data

Based on PIU EDA which makes sense kaggle notebook and CMI-PIU: Actigraphy data EDA kaggle notebook, the actigraphy data is rather noisy and contain many odd scenarios.

V. Approaches

A. Overview

We implemented a range of techniques, which are detailed in VIII-A. Below, we will discuss an overview of how each algorithm operates.

1) Gradient Boosting: Gradient Boosting is a machine learning technique used for both classification and regression tasks. It belongs to the family of ensemble methods and is designed to convert a set of weak learners, typically decision trees, into a strong learner. Gradient Boosting has gained significant attention due to its ability to handle complex patterns in data and deliver high predictive performance, especially on structured/tabular datasets. Its applications span various domains, including finance, healthcare, and competition platforms such as Kaggle.

a) Key features: The general idea of Gradient Boosting is train a model F over M interactions, where at each iteration, an imperfect model F_m is improved by adding a component $h_m(x)$, where $h_m \in \mathcal{H}$ represents the set of base learners. This leads to:

$$F_{m+1}(x_i) = F_m(x_i) + h_m(x_i) = y_i,$$

which is equivalent to:

$$h_m(x_i) = y_i - F_m(x_i).$$

Thus, the algorithm fits h_m to the *residual* $y_i - F_m(x_i)$. This idea can be generalized by using other loss functions instead of Mean Squared Error (MSE).

b) Pseudocode: Assuming we have the training data: $\{(x_i, y_i)\}_{i=1}^N$, where x_i are the input features and y_i are the target values, M are the interactions, $L(y, F(x))$ is the chosen loss function and differentiable, $\eta \in (0, 1]$ is the learning rate.

1) Initialize the model:

$$F_0(x) = \arg\min_c \sum_{i=1}^{N} L(y_i, c)$$

2) For $m = 1$ to M, repeat:

a) Compute the pseudo-residuals:

$$r_i^{(m)} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, N$$

b) Fit a weak learner $h_m(x)$ (e.g., a decision tree) to the residuals:

$$h_m(x) \approx r_i^{(m)}$$

c) Update the model:

$$F_m(x) = F_{m-1}(x) + \eta h_m(x)$$

3) Return the final model:

$$F_M(x)$$

e input data into a lower-dimensional latent space and then reconstruct it back to its original form. This process makes autoencoders a powerful

c) Usage: For this problem, we utilize three Gradient Boosting frameworks—XGBoost, LightGBM, and CatBoost—for predicting the Severity Impairment Index (SII) [9]–[11].

2) Nelder-Mead Algorithm: The Nelder-Mead algorithm [12] is a derivative-free optimization technique used to minimize an objective function $f(x)$ in d-dimensional space. It is particularly suitable for non-differentiable, noisy, or discontinuous functions. The algorithm operates on a geometric structure called a simplex, which consists of $d+1$ vertices in d-dimensional space. By iteratively applying operations such as reflection, expansion, contraction, and shrinkage, the algorithm gradually converges to the minimum.

a) Key concepts:

- Simplex: A set of $d+1$ points $\{x_1, x_2, \ldots, x_{d+1}\}$ in d-dimensional space.
- Objective Function: The function $f(x)$ to be minimized.
- Operations:
  - Reflection: Reflect the worst vertex across the centroid to explore better regions.
  - Expansion: Expand further in the reflected direction if it improves the result.
  - Contraction: Contract the simplex towards the centroid if reflection and expansion fail.
  - Shrinkage: Reduce the simplex size if all other operations fail.

After this we will achieve the vertex x_1 as the approximate solution.

b) Pseudocode: Assuming we have an Objective function $f(x)$, the Initial simplex is $\{x_1, x_2, \ldots, x_{d+1}\}$, a Reflection coefficient: $\alpha > 0$ (typically $\alpha = 1$), a Expansion coefficient: $\gamma > 1$ (typically $\gamma = 2$), a Contraction coefficient: $\rho \in (0, 1)$ (typically $\rho = 0.5$), a Shrinkage coefficient: $\sigma \in (0, 1)$ (typically $\sigma = 0.5$) and a Stopping criteria: Tolerance $\epsilon > 0$.

1) Initialize:
   - Evaluate $f(x)$ at all simplex vertices: $f(x_1), f(x_2), \ldots, f(x_{d+1})$
   - Sort vertices such that: $f(x_1) \leq f(x_2) \leq \cdots \leq f(x_{d+1})$
2) Repeat until convergence:
   a) Compute the centroid:

$$x_c = \frac{1}{d} \sum_{i=1}^{d} x_i$$

   b) Reflect: Compute the reflection point:

$$x_r = x_c + \alpha(x_c - x_{d+1})$$

   Evaluate $f(x_r)$: If $f(x_1) \leq f(x_r) < f(x_d)$, replace x_{d+1} with x_r.
   c) Expand: If $f(x_r) < f(x_1)$, compute the expansion point:

$$x_e = x_c + \gamma(x_r - x_c)$$

   Replace x_{d+1} with x_e if $f(x_e) < f(x_r)$, otherwise replace it with x_r.
   d) Contract: If $f(x_r) \geq f(x_d)$, compute the contraction point:

$$x_c' = x_c + \rho(x_{d+1} - x_c)$$

   Replace x_{d+1} with x_c' if $f(x_c') < f(x_{d+1})$, otherwise proceed to shrinkage.
   e) Shrink: Reduce the simplex towards x_1:

$$x_i = x_1 + \sigma(x_i - x_1) \quad \text{for } i = 2, \ldots, d+1$$

3) Stopping criteria: Stop when the difference in $f(x)$ values among simplex vertices is less than ϵ, or the simplex size becomes sufficiently small.
4) Output: Return the best vertex x_1 as the approximate solution.

c) Usage: This algorithm is employed to optimize the Quadratic Weighted Kappa metric, ensuring the model's performance aligns with the evaluation criteria.

3) Hyperparameter Tuning: We utilize the Optuna framework [13] for this task to efficiently optimize hyperparameters. Optuna is an open-source hyperparameter optimization framework designed for efficient and scalable tuning of machine learning models. It employs a flexible and user-friendly interface while offering powerful optimization algorithms, including Tree-structured Parzen Estimator (TPE) and other state-of-the-art techniques. Optuna integrates seamlessly with popular machine learning libraries such as Scikit-learn, TensorFlow, PyTorch, and XGBoost, making it a versatile tool for improving model performance.

a) Key Features:

- Define-by-Run Interface: Optuna allows users to dynamically define the search space as the optimization process proceeds, offering greater flexibility compared to traditional grid or random search.
- Pruners: Efficiently terminates unpromising trials early, saving computational resources.
- Sampler: Uses advanced sampling techniques like TPE for efficient exploration of the search space.
- Integration: Supports major machine learning libraries and can be easily incorporated into existing workflows.

The optimization process in Optuna involves the following steps:

1) Define an objective function to be minimized or maximized. This function takes a set of hyperparameters as input and returns a performance metric.
2) Specify a search space for the hyperparameters.
3) Use Optuna's sampler to propose hyperparameter configurations.

4) Evaluate the objective function with the proposed hyperparameters.
5) Update the sampler based on previous evaluations and repeat the process until a stopping criterion is met (e.g., a maximum number of trials or a time limit).

b) Pseudocode:

1) Import Optuna and define the objective function:

   Objective function: $f(\text{trial}) \rightarrow$ performance metric.

   Example:

   trial.suggest_uniform('learning_rate', 0.01, 0.1)

2) Create a study to manage the optimization process:

   study = optuna.create_study(direction='minimize')

3) Start the optimization loop:

   study.optimize(objective, n_trials=100, timeout=600)

4) Retrieve the best parameters:

   best_params = study.best_params

5) Use the best parameters for model training and evaluation.

c) Usage: Optuna is utilized to optimize the hyperparameters of the Voting Regressor model, ensuring improved performance and fine-tuning of its parameters.

4) Data Imputation: Data imputation is the process of replacing missing or incomplete data with estimated values to enable robust analysis and modeling. Missing data is a common challenge in machine learning and statistical analysis, and improper handling of these gaps can lead to biased results or reduced model performance. Imputation helps retain the dataset's integrity by approximating the missing values based on the available information.

a) Key Techniques: Three popular imputation methods are discussed below:

1) KNNImputer (K-Nearest Neighbors Imputation):
   - This method imputes missing values based on the values of the k-nearest neighbors in the feature space.
   - For a given missing value, the algorithm identifies k samples with similar characteristics (nearest neighbors) and uses the mean (or median) of their values to replace the missing entry.
   - Suitable for datasets where correlations among features can inform the imputation.
2) SimpleImputer:
   - A straightforward imputation strategy that replaces missing values with a single value such as the mean, median, or mode of the respective feature.
   - Efficient and fast for datasets with a high percentage of missing values or when computational simplicity is required.

b) Usage: This approach facilitates the systematic filling of missing values, ensuring the dataset remains complete and suitable for further analysis and modeling.

5) Feature Engineering: Among various feature engineering methods, we employ autoencoders for this problem to effectively extract meaningful and compact representations from the data. An autoencoder is a type of neural network designed for unsupervised learning, where the goal is to encode input data into a lower-dimensional latent space and then reconstruct it back to its original form. This process makes autoencoders a powerful tool for feature engineering, as the latent space captures the most salient and compressed representations of the input data. These representations can be used to generate meaningful and noise-free features for downstream tasks.

a) Structure of an Autoencoder: An autoencoder consists of two main components:

- *Encoder:* Transforms the input data $x \in \mathbb{R}^n$ into a latent representation $z \in \mathbb{R}^m$, where $m < n$. This transformation is achieved using one or more layers of neural networks:

$$z = f_{\text{encoder}}(x).$$

- *Decoder:* Attempts to reconstruct the input x from the latent representation z, ensuring the latent space retains the essential information:

$$\hat{x} = f_{\text{decoder}}(z).$$

The autoencoder is trained to minimize the reconstruction loss, which measures the difference between the original input x and the reconstructed output $\hat{x}$. Common loss functions include Mean Squared Error (MSE) and Binary Cross-Entropy (BCE):

$$L_{\text{reconstruction}} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2.$$

b) Workflow for Using Autoencoders in Feature Engineering:

1) Normalize and preprocess the input data to ensure compatibility with neural network architectures.
2) Train the autoencoder on the dataset, ensuring it effectively minimizes the reconstruction loss.
3) Extract the latent representation z from the encoder part of the autoencoder.
4) Use z as the engineered features for training or improving machine learning models.

c) Usage: We employ this method to perform feature engineering on tabular data combined with actigraphy, enabling the extraction of meaningful representations for prediction.

6) K-Fold Cross-Validation: K-fold cross-validation is a widely used technique in machine learning for evaluating model performance and reducing the risk of overfitting. It ensures the model is tested on different subsets of the data, providing a robust and reliable estimate of its generalization ability.

a) How It Works:

1) *Partitioning the Data:* The dataset is divided into k equally (or nearly equally) sized folds. Each fold serves as a test set once, while the remaining $k-1$ folds form the training set.
2) *Training and Testing:* The model is trained k times, each time using a different fold as the test set and the other $k-1$ folds as the training set. The model's performance is evaluated on the test fold during each iteration.
3) *Aggregation:* The performance metrics (e.g., accuracy, precision, recall, mean squared error) are averaged across all k iterations to provide a comprehensive evaluation of the model.

b) Special Cases:

- *Leave-One-Out Cross-Validation (LOOCV):* A special case where $k = n$, and each data point is used as a test set once. It provides exhaustive evaluation but is computationally intensive.
- *Stratified K-Fold Cross-Validation:* Ensures each fold has a similar distribution of target classes, making it particularly useful for imbalanced datasets.

c) Mathematical Formula: Let the dataset be $D = \{(x_i, y_i)\}_{i=1}^{n}$, where n is the number of samples. For each fold $j \in \{1, 2, \ldots, k\}$:

1) Split D into:
   - Training set: $D_{\text{train}} = D \setminus D_j$,
   - Test set: $D_{\text{test}} = D_j$, where D_j is the j-th fold.
2) Train the model f_j on D_{train}.
3) Evaluate f_j on D_{test} using a metric M.

The final performance is given by:

$$\text{Performance} = \frac{1}{k} \sum_{j=1}^{k} M(f_j, D_{\text{test}}).$$

B. Detailed

1) Overall pipeline: We will mostly follow this pipeline throughout all other approaches. The pipeline consists of four main steps: data preprocessing, feature engineering, training loop, and ensembling. We lock all the random states for reproducibility.

a) Data Preprocessing: Data preprocessing is a critical step in ensuring the quality and consistency of the dataset for effective model training. For handling missing data, we experimented with two popular imputation techniques: KNNImputer and SimpleImputer. Other than that, with features that are considered an outlier, we remove them from the dataset. KNNImputer estimates missing values based on the k-nearest neighbors in the feature space, leveraging the similarity between samples to provide imputations. The number of neighbors (k) was treated as a hyperparameter, and we evaluated its impact on the overall model performance with $k = 5$. SimpleImputer, on the other hand, employs straightforward strategies such as replacing missing values with the mean, median, or mode of the respective feature. We use $strategy$ = mean in our experiments. Both imputation methods were evaluated based on their ability to improve downstream model performance, and the choice of imputer was guided by empirical results from cross-validation experiments.

b) Feature Engineering: Feature engineering plays a pivotal role in enhancing model performance by transforming raw data into more informative representations. For this task, we experimented with autoencoders, utilizing both a standard architecture and a modified version designed by our team. Both autoencoders were performed on the dataset combined with actigraphy data. We also perform manual feature engineering based on the results from our EDA.

c) Training Loop: To ensure robust model evaluation and mitigate the risk of overfitting, we have to design a training strategy. Our training loop use k-fold cross-validation with five folds. In this approach, the dataset is partitioned into five subsets of equal size. For each iteration, one subset is designated as the validation set, while the remaining four subsets form the training set. We also use Nedler-Mead algorithm as the optimizer for the threshold of SII.

d) Ensembling: We perform what we called ensemble of ensemble on approaches with ensembling. The main ensemble has three smaller models, each consisting of three or more base models, such as XGBoost, LightGBM, and similar algorithms with a Voting Regressor for ensembling.

2) Baseline: We got our baseline from a public Kaggle notebook [14]. The approach in this notebook will mostly follow the pipeline. Models in the ensemble are:

- LightGBM, XGBoost, CatBoost and TabNet.
- LightGBM, XGBoost and CatBoost.
- LightGBM, XGBoost, CatBoost, RandomForest and Gradient Boosting.

Each subensemble has different parameters to prevent the repetition.

3) Our approach: Our approach consists of several key steps to effectively preprocess the data, engineer features, and build robust predictive models. For data preprocessing, we begin by removing outliers to ensure the dataset is clean and free from extreme values that could negatively impact model performance. Following this, we employ our modified autoencoder alongside manual feature engineering to generate meaningful and compact representations of the data. The next stage involves creating an ensemble of models, with each ensemble relying on a different imputation strategy and combination of base models. The details are as follows:

- Impute data using KNNImputer, followed by an ensemble of LightGBM, CatBoost, and TabNet.
- Impute data using SimpleImputer, followed by an ensemble of LightGBM, XGBoost, and RandomForest.
- Impute data using SimpleImputer, followed by an ensemble of XGBoost, RandomForest, and TabNet.

By utilizing diverse imputation methods and model combinations, our approach ensures robustness and reduces the potential for overfitting, ultimately improving predictive performance. Figure 17 illustrates the workflow of our approach, detailing the steps.

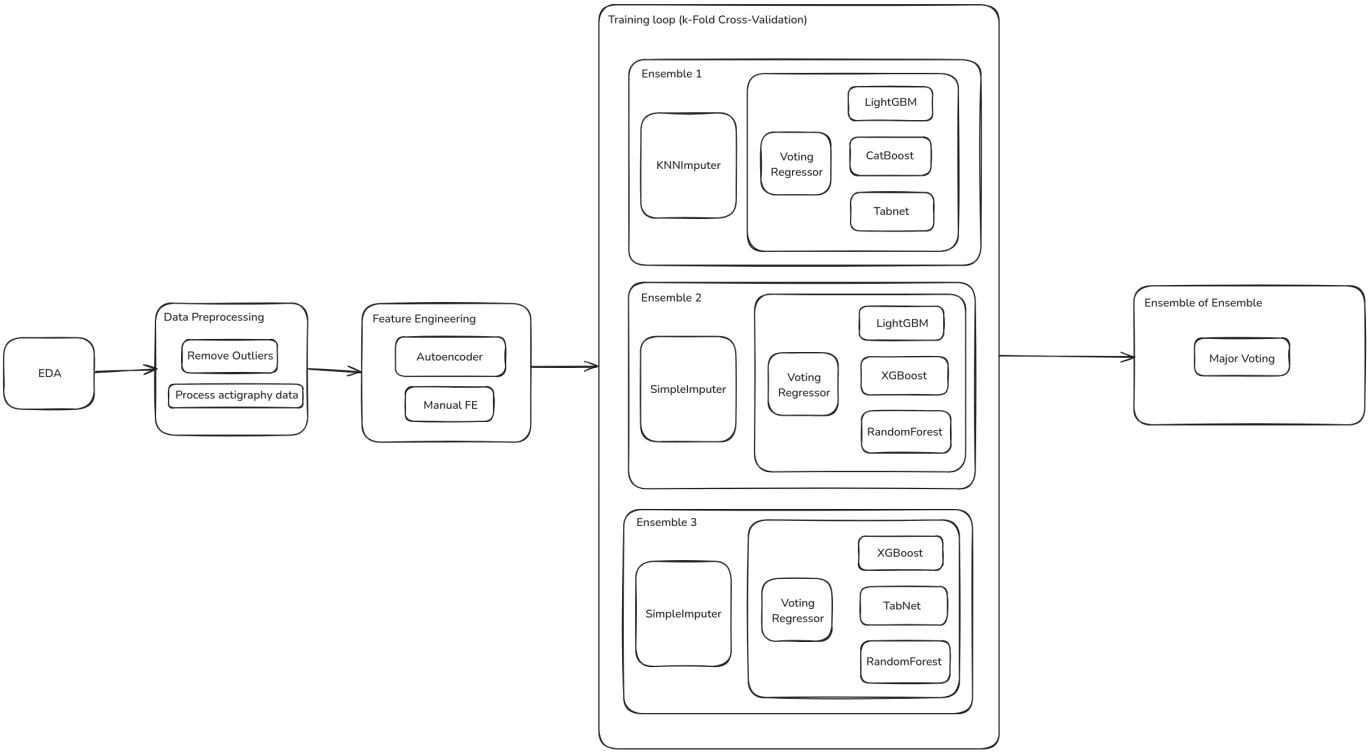

Fig. 17. Proposed approach

TABLE I
RESULTS FROM OUR EXPERIMENTS

Approach Architecture	Train	Validate	Optimized Threshold	Private Score	Public Score
Our approach	0.8049	0.3840	**0.4660**	0.3570	0.4530
	0.7202	0.3641	0.4520		
	0.6979	0.3607	0.4480		
LightGBM + XGBoost	**0.8898**	0.3855	0.4460	**0.4410**	0.4630
LightGBM	0.7293	**0.4042**	0.4590	0.4310	0.4460
XGBoost	0.8898	0.3855	0.4480	**0.4410**	0.4580
CatBoost	0.5458	0.3788	0.4610	0.3710	0.4510
TabNet	0.1703	0.1680	0.1830	0.1960	0.2980
LightGBM + XGBoost + CatBoost + Optuna Optimizer	0.6931	0.3953	0.4560	0.3970	0.4420
Baseline				0.407	**0.497**
LightGBM + Autoencoder	0.7407	0.4003	0.4540	0.3990	0.4440
Convolutional 1D + Manual Feature Engineering + Autoencoder	0.2329	0.2221	0.3210	0.0600	0.2210
Ensemble of ensemble				0.4050	0.4350
KNN + Ensemble	0.8074	0.3857	0.4550	0.3430	0.4360

VI. EXPERIMENTS

The results of the conducted experiments are presented below, accompanied by insights derived from the analysis.

Table I presents the results of our conducted experiments. This table has been trimmed for brevity; the full results can be accessed via the provided link in VIII-B. The column ”Approach Architecture” provides a brief description of the approach used in each submission. The ”Train” column indicates the QWK metric on the training set, while ”Validate” represents the QWK metric on the validation set. ”Optimized Threshold” refers to the QWK metric after tuning the threshold for the SII. The ”Private Score” and ”Public Score” columns represent the private and public leaderboard scores, respectively.

We selected our submissions based on the public score and cross-validation. We also want the solution to be sophisticated hence we selected the baseline and the current approach. But the private score results proved our philosophy wrong. We can achieve greater scores with more simple model such as LightGBM and XGBoost ensemble which is one of two approaches to achieve our highest score.

First, we need a point to start from hence we chose the baseline which is a public notebook that achieved a high leaderboard score. After that, we observe and analyze the notebook and the cross-validation. We came up with some foundings. Public notebooks only change the seed or the

parameters of models with lots of duplicated code and magic numbers. We knew that we could not use this to win the contest. We also notice a problem with public notebooks when handling actigraphy data, which is they used *df.describe()* to process it. We tried to come up with methods to process it, but after some time and public discussions about the contribution of actigraphy data to the final prediction, we decided just use that method. From the baseline we have a vague roadmap about how to attack the problem. We will try to tune the hyperparameters with Optuna.

With Optuna, we tried to tune the weights of the Voting Regressor model, but that worsened the results and was very unstable. We came up with the reason behind the failure of Optuna is the selection of $pbound$ value, since we need to tune this parameter manually. That is why we need to rethink our approach, we need to experiment from scratch to see which component contributes to the final results. Our target back then was the public leaderboard score. Furthermore, the Voting Regressor's complexity contributed to these challenges. As an ensemble model, the Voting Regressor combines predictions from multiple models, and the final performance heavily depends on the appropriate weighting of these models. If certain models in the ensemble are less stable or weaker predictors, assigning them even slightly higher weights can degrade performance. This phenomenon could amplify errors and make the optimization process erratic. Another potential issue is the instability of the objective function itself. Since Optuna relies on the objective function to evaluate the performance of each parameter configuration, fluctuations due to randomness in training or evaluation can lead to misleading signals. For instance, if the public leaderboard score—our target metric—varied significantly across runs due to external factors, Optuna may have struggled to converge on an optimal solution. To address these challenges, we concluded that we need to rethink our approach and experiment from scratch. By dissecting the contribution of each component in the Voting Regressor to the final result, we aim to identify the sources of instability and improve performance systematically.

First, we began our experiments by working with the raw data without applying any feature engineering or preprocessing steps. Our initial approach involved training a single model on this unprocessed dataset, and the results of this experiment are detailed in VIII-B. The performance of the model in this setup was unsatisfactory and clearly highlighted the need for preprocessing. Without proper data preprocessing, including the removal of noisy data points and outliers, the model struggled to generalize and produced results that could not be used for meaningful insights. Following this, we analyzed the data in greater depth and identified outliers that were skewing the distributions of certain features, as discussed in II. Given that our training data was already limited in size, these outliers had a disproportionate impact on the overall data quality. Once we removed these outliers, the results improved significantly, underscoring the importance of data cleaning in our pipeline. Outlier removal directly affected the performance of the model by reducing noise and allowing it to better capture the underlying patterns in the data. Following this, we analyzed the data in greater depth and identified outliers that were skewing the distributions of certain features, as discussed in II. Given that our training data was already limited in size, these outliers had a disproportionate impact on the overall data quality. Once we removed these outliers, the results improved significantly, underscoring the importance of data cleaning in our pipeline. Outlier removal directly affected the performance of the model by reducing noise and allowing it to better capture the underlying patterns in the data. Autoencoder used in the baseline has a very simple architecture, we modified it and tried with a simple model. The new architecture includes *BatchNorm* and *DropOut* to prevent the model from overfitting. Why au.Results yielded from both entries show no difference and the public score was bad. Autoencoders often encounter challenges when applied to tabular data for latent space representation due to several inherent characteristics of such data. Tabular datasets typically comprise heterogeneous features, including both continuous and categorical variables, which complicates the learning process for autoencoders. The intricate relationships and dependencies among these diverse feature types can hinder the model's ability to effectively capture and reconstruct the underlying data distribution [15]. Moreover, the high dimensionality and sparsity common in tabular data can lead to overfitting, where the autoencoder learns to memorize the training data rather than generalizing to unseen instances. This issue is exacerbated when the dataset contains noise or irrelevant features, further impeding the model's performance [16]. Additionally, the lack of inherent spatial or sequential structure in tabular data, unlike in image or text data, poses a significant challenge. Autoencoders excel in capturing patterns in data with such structures, but the absence thereof in tabular data makes it difficult for the model to learn meaningful latent representations [17]. We also tried TabNet as a model to predict the SII directly, but it failed miserably. This aligns with some community observations, as noted in a Reddit discussion where users reported difficulties in using TabNet effectively for certain prediction tasks [18].

After the experiments,

VII. Conclusion

In this report, we reviewed various approaches employed in the Problematic Internet Use competition on Kaggle and conducted an in-depth analysis of our methodologies and their methodologies. Our investigation revealed insightful findings regarding the mechanisms behind these approaches and the factors contributing to their effectiveness. Although we did not win the competition, the experience provided valuable insights and enhanced our understanding of advanced methodologies and practical challenges in addressing the Problematic Internet Use problem.

VIII. Supplements

A. List of techniques

We have implemented these algorithms/methods:

- Nelder-Mead Optimization for Kappa metric. [12]

- Autoencoder for feature engineering.
- Optuna for hyperparameter tuning [13].
- KNNImputer, SimpleImputer, IterativeImputer for data imputation [19].
- For the prediction models, we used the below:
  - XGBoost [9].
  - LightGBM [10].
  - CatBoost [11].
  - TabNet [20].
  - Convolutional 1D Neural Network.

B. Resources of the report

Below are the resources we used to conduct experiments:

- Our GitHub repo: ML-CMI.
- Our results: Result sheet.

REFERENCES

[1] Z. Cai, P. Mao, Z. Wang, D. Wang, J. He, X. Fan, Associations between problematic internet use and mental health outcomes of students: A meta-analytic review, Adolescent Research Review 8 (1) (2023) 45–62.

[2] A. Restrepo, T. Scheiniger, J. Clucas, L. Alexander, G. Salum, K. Georgiades, D. Paksarian, K. Merikangas, M. Milham, Problematic internet use in children and adolescents: associations with psychiatric disorders and impairment, BMC Psychiatry 20 (2020) 1–11.

[3] S. Li, Z. Wu, Y. Zhang, M. Xu, X. Wang, X. Ma, Internet gaming disorder and aggression: A meta-analysis of teenagers and young adults, Frontiers in Public Health 11 (2023).

[4] J. Liu, S. Riesch, J. Tien, T. Lipman, J. Pinto-Martin, A. O'Sullivan, Screen media overuse and associated physical, cognitive, and emotional/behavioral outcomes in children and adolescents: An integrative review, Journal of Pediatric Health Care 36 (2) (2022) 99–109.

[5] A. Al-Amri, S. Abdulaziz, S. Bashir, M. Ahsan, T. Abualait, Effects of smartphone addiction on cognitive function and physical activity in middle-school children: a cross-sectional study, Frontiers in Psychology 14 (2023).

[6] S. Lakkunarajah, K. Adams, A. Pan, M. Liegl, M. Sadhir, A trying time: Problematic internet use (piu) and its association with depression and anxiety during the covid-19 pandemic, Child and Adolescent Psychiatry and Mental Health 16 (1) (2022) 49.

[7] Y. Yu, Y. Wu, P. Chen, H. Min, X. Sun, Associations between personality traits and problematic internet use among chinese adolescents, Journal of Adolescence (2023).

[8] K. Conyngham, J. M. Luckey, F. Pawelczyk, [proposal-ML] relating physical activity to problematic internet use, in: Submitted to Tsinghua University Course: Advanced Machine Learning, 2024, under review.
URL https://openreview.net/forum?id=DZI0v9KRU0

[9] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, CoRR abs/1603.02754 (2016). arXiv:1603.02754.
URL http://arxiv.org/abs/1603.02754

[10] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.-Y. Liu, Lightgbm: a highly efficient gradient boosting decision tree, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, Curran Associates Inc., Red Hook, NY, USA, 2017, p. 3149–3157.

[11] A. V. Dorogush, V. Ershov, A. Gulin, Catboost: gradient boosting with categorical features support, CoRR abs/1810.11363 (2018). arXiv:1810.11363.
URL http://arxiv.org/abs/1810.11363

[12] J. A. Nelder, R. Mead, A simplex method for function minimization, The Computer Journal 7 (4) (1965) 308–313. arXiv:https://academic.oup.com/comjnl/article-pdf/7/4/308/1013182/7-4-308.pdf, doi:10.1093/comjnl/7.4.308.
URL https://doi.org/10.1093/comjnl/7.4.308

[13] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: A next-generation hyperparameter optimization framework (2019). arXiv:1907.10902.
URL https://arxiv.org/abs/1907.10902

[14] P. Chotepanit, CMI— Tuning — Ensemble of solutions — kaggle.com, https://www.kaggle.com/code/batprem/cmi-tuning-ensemble-of-solutions, [Accessed 22-12-2024] (2024).

[15] X. Chen, Others, Heterogeneous data processing for tabular autoencoders, ArXiv (2023).
URL https://arxiv.org/abs/2410.10463

[16] Y. Xu, Others, Robust autoencoders for noisy tabular data, ArXiv (2023).
URL https://arxiv.org/abs/2301.00802

[17] H. Liu, Others, Structure-aware autoencoders for tabular data, ArXiv (2024).
URL https://arxiv.org/abs/2401.02013

[18] R. user, Challenges with tabnet for prediction tasks, https://www.reddit.com/r/MachineLearning/comments/wfbj9e/comment/iiu2ucf/?utm_source=share&utm_medium=web3x&utm_name=web3xcss&utm_term=1&utm_content=share_button, accessed: 2024-12-22 (2024).

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[20] S. Ö. Arik, T. Pfister, Tabnet: Attentive interpretable tabular learning, CoRR abs/1908.07442 (2019). arXiv:1908.07442.
URL http://arxiv.org/abs/1908.07442