VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY

CS114.P21.KHTN

Báo Cáo Đồ Án Môn Học

Sinh Viên :
Đồng Quốc Thắng - 23521421
Nguyễn Thiện Nhân - 23521083
Nguyễn Đình Thiên Quang-23521285

Giảng viên :
Võ Nguyễn Lê Duy

Ngày 21 tháng 5 năm 2025

Mục lục

0.1 Code and Slides

Our code and slide for this project can be found at `https://github.com/LowTechTurtle/CS114-ML/tree/main/Final_proj`

Chương 1

Data and Contest Overview

Predict the Severity Impairment Index (SII) for problematic internet use by identifying biological markers to improve diagnosis and treatment of mental health and learning disorders

1.1 Contest Description

In today's digital age, problematic internet use among children and adolescents is a growing concern. Better understanding this issue is crucial for addressing mental health problems such as depression and anxiety.

Current methods for measuring problematic internet use in children and adolescents are often complex and require professional assessments. This creates access, cultural, and linguistic barriers for many families. Due to these limitations, problematic internet use is often not measured directly, but is instead associated with issues such as depression and anxiety in youth.

Conversely, physical and fitness measures are extremely accessible and widely available with minimal intervention or clinical expertise. Changes in physical habits, such as poorer posture, irregular diet, and reduced physical activity, are common in excessive technology users. We propose using these easily obtainable physical fitness indicators as proxies for identifying problematic internet use, especially in contexts lacking clinical expertise or suitable assessment tools.

This competition challenges you to develop a predictive model capable of analyzing children's physical activity data to detect early indicators of problematic internet and technology use. This will enable prompt interventions aimed at promoting healthier digital habits.

1.2 Data Overview

The **Healthy Brain Network (HBN)** dataset is a clinical sample of approximately 5,000 individuals aged 5–22 years who have undergone both clinical and research screenings. The objective of the HBN study is to identify biological markers that can enhance the diagnosis and treatment of mental health and learning disorders from an objective biological standpoint. This competition uses two components of the study: physical activity data (including wrist-worn accelerometer data, fitness assessments, and questionnaires) and internet usage behavior data. The aim is to predict a participant's *Severity Impairment Index (sii)*, which is a standard measure of problematic internet use.

Note: This is a *Code Competition*, meaning the actual test set is hidden. This public version includes a sample dataset in the correct format to assist in developing solutions. The full test set includes approximately 3,800 instances.

The competition data is divided into two types: `parquet` files that contain accelerometer (actigraphy) time series data, and `csv` files containing the remaining tabular data. Many of the measures are missing for most participants. Notably, the target variable `sii` is missing for some participants in the training set. Therefore, non-supervised learning techniques may be useful. The `sii` value is present for all instances in the test set.

For more information, please visit the contest on kaggle

`https://www.kaggle.com/competitions/child-mind-institute-problematic-internet-use`

1.2.1 Challenge and Opportunities

Key Challenges

- **Lots of missing data** – Many entries in the tabular data are missing, including the target SII score for some rows.
- **Tricky time-series data** – The actigraphy data is long, detailed, and noisy. It takes work to clean and summarize it well.
- **Feature engineering is tough** – It's not easy to turn raw signals and survey answers into useful model inputs.

Opportunities

- **Clean up the data** – Drop rows with too many NaNs or use smart ways to fill them in.
- **Use models to fill gaps** – Try training small models to guess missing values instead of filling with averages.
- **Mix signals & survey data** – Combining movement data with internet usage and other features could boost results.

1.3 Data Preprocessing

- We encountered several challenges due to missing values and the need for accurate imputations. Here's how we did the preprocessing:
- Firstly, we resolve the issue of missing target values. Since the Severity Impairment Index (SII) is our primary variable of interest, any rows lacking this information were excluded from the training set. This ensures that our models learn from complete and reliable data.

- For the season variable, which had some missing entries, we used a straightforward imputation. By analyzing the distribution of known seasons in the dataset, we randomly assigned missing values based on this existing distribution. This method maintains the natural variability and avoids introducing bias.
- For physical measures like height and weight, we used the K-Nearest Neighbors (KNN) algorithm for imputation. KNN considers multiple related features—such as age, gender, and season—to predict missing values. This approach captures the complex relationships between variables and preserves the dataset's integrity.
- For waist circumference, we noticed a linear relationship with Body Mass Index (BMI) and weight. Leveraging this, we used linear regression to predict missing waist measurements, using BMI and weight as predictors. This method provided accurate estimations based on existing data patterns.
- For the computer/internet hours per day variable presented a categorical challenge. To impute missing values here, we used logistic regression. By analyzing patterns in related features, this model effectively predicted the likelihood of each category, ensuring that our dataset remained comprehensive.

Chương 2

Model

2.1 Choosing Model

We focus on selecting appropriate methods and provide partial justification for our choices. Given the dominance of the Gradient Boosting Decision Tree (GBDT) framework in the domain of tabular data, we adopt two implementations based on different design philosophies: **XGBoost** and **LightGBM**.

Gradient boosting libraries such as XGBoost and LightGBM are widely utilized for structured data tasks. Both aim to accelerate training processes and enhance the predictive accuracy of tree ensemble models.

In this work, we examine the novel algorithmic strategies and mathematical foundations underlying each library. Special emphasis is placed on the unique contributions of each implementation as well as their performance differences.

2.2 Model: Deep Dive

2.2.1 XGBoost

- **Ensemble of trees:** predictions updated as
$$\hat{y}^{(t)} = \hat{y}^{(t-1)} + f_t(x).$$
- **Use gradients** $g_i = \frac{\partial L}{\partial \hat{y}_i}$ and Hessians $h_i = \frac{\partial^2 L}{\partial \hat{y}_i^2}$ for optimization.
- Each new tree f_t minimizes an approximate objective (see block below).
- Splitting decisions maximize reduction in loss using accumulated gradients in leaves.

General Objective
$$\text{Obj} = \sum_i L(y_i, \hat{y}_i) + \sum_{t=1}^{T} \Omega(f_t).$$

- **Regularized Objective:** uses
$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_j w_j^2$$

(where T is the number of leaves and w_j are the leaf weights) to penalize model complexity.

- **Shrinkage (Learning Rate):** multiplies new tree outputs by $\nu < 1$ to reduce the impact of each individual step.
- **Column Subsampling:** samples a fraction of features per tree or level to speed up training and reduce feature correlation.
- **Sparsity-aware:** handles missing or zero values by learning optimal default directions for splits.

XGBoost Objective (t-th Tree)

$$\text{Obj}^{(t)} = \sum_i L(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t),$$

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_j w_j^2.$$

- **Weighted Quantile Sketch:** approximates feature quantiles for fast split finding on large/weighted data.
- **Split Gain Formula:** computes loss reduction using gradient sums

$$G = \sum g_i, \quad H = \sum h_i.$$

- **Second-order Optimization:** uses Hessians to compute optimal leaf weights (Newton step).
- **Distributed/GPU Training:** optimized C++ implementation with multi-threading and GPU support for large-scale learning.

Split Gain (XGBoost)

$$\text{Gain} = \frac{1}{2}\left(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right) - \gamma,$$

where G_L, G_R and H_L, H_R are gradient/Hessian sums of the left/right child.

2.2.2 LightGBM

- **Leaf-wise Growth:** splits the leaf with maximum loss reduction (best-first), potentially producing deeper trees.
- **Histogram Binning:** discretizes continuous features into buckets, reducing memory usage and accelerating the split-finding process.

- **Exclusive Feature Bundling (EFB):** merges mutually exclusive sparse features into a single feature to reduce dimensionality.
- **Gradient-based One-Side Sampling (GOSS):** retains instances with large gradients (details provided in the next slide).
- **GOSS Sampling:** keep the top $a\%$ of instances with the largest gradients, and randomly sample $b\%$ of the remaining ones.
- **GOSS Weighting:** multiply gradients of sampled small-gradient instances by $\frac{1-a}{b}$ to maintain an unbiased gradient distribution.
- **Leaf-wise vs Level-wise:** more accurate splits but deeper trees; behavior is controlled by parameters `num_leaves` and `max_depth`.
- **Efficient Implementation:** optimized for speed with multi-threading, GPU support (histogram-based), and native handling of categorical features.

GOSS Sampling Weights:

Keep top $a\%$ by $|g|$, sample $b\%$ of others; multiply sampled gradients by $\frac{1-a}{b}$.

- **Taylor Expansion:** For a new tree f_t, the objective function is approximated using a second-order Taylor expansion of the loss function:

$$\sum_i \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2 \right] + \Omega(f_t),$$

  where g_i and h_i are the first and second-order gradients of the loss function with respect to the prediction.
- **Optimal Leaf Weight:** For a leaf with gradient sum G and Hessian sum H, the optimal weight w^* is given by:

$$w^* = -\frac{G}{H + \lambda},$$

  where λ is the regularization parameter.
- **Learning Rate:** The leaf outputs are scaled by a learning rate ν, resulting in the final weight update:

$$w^* \leftarrow \nu w^*.$$

- **Gain-based Splitting:** XGBoost uses the gain from the above loss reduction formula to decide how to split nodes in the tree.

Optimal Leaf Weight Formula

For a set of instances $i \in I$ in a leaf, the optimal weight w^* is computed as:

$$w^* = -\frac{\sum_{i \in I} g_i}{\sum_{i \in I}(h_i + \lambda)}.$$

- **Histogram Splitting (LightGBM):** Continuous features are discretized into bins. For each bin, the gradient and Hessian statistics are aggregated as follows:

  $$G_b = \sum_{i \in \text{bin}} g_i, \quad H_b = \sum_{i \in \text{bin}} h_i,$$

  where g_i and h_i are the gradient and Hessian of instance i, respectively.

- **GOSS Reweighting:** After sampling, the gradients are reweighted. Specifically, scaled gradients $\frac{1-a}{b}$ are used in the aggregated sums for determining splits, where a and b denote sampling proportions.

- **EFB Concept (Exclusive Feature Bundling):** Features that are mutually exclusive (i.e., non-overlapping in non-zero entries) are combined into a single feature. This reduces dimensionality and improves efficiency.

- **Leaf-wise Gain:** The tree grows by always splitting the current leaf with the highest gain, achieving a global maximum reduction in loss.

Histogram Aggregation

Gradients and Hessians are aggregated per bin as follows:

$$G_b = \sum_{i \in \text{bin}} g_i, \quad H_b = \sum_{i \in \text{bin}} h_i.$$

These aggregated statistics are then used to evaluate and select the best possible split.

2.3 Tuning

Manual Tuning and Feature Selection

For this project, we manually tuned the hyperparameters by watching how our models performed on the Kaggle leaderboard.

Some of the most important hyperparameters we adjusted included the learning rate, which we set to 0.001, and the number of training epochs, which ranged between 150 and 200. We also experimented with the regularization strength, λ, and found values between 5 and 10 worked best.

After a lot of testing and iteration, we narrowed things down to five input features that consistently gave strong results:

- **feature_1** — `SDS_Score_Weighted`

- **feature_2** — `Internet_Hours_Age`
- **feature_3** — `PhysicalHeight_Age`
- **feature_4** — `Basic_Demos-Sex`
- **feature_5** — `PreInt_FGC_CU_PU`

These features captured a mix of psychological, behavioral, and demographic information that seemed most predictive of the target.

2.4 Result

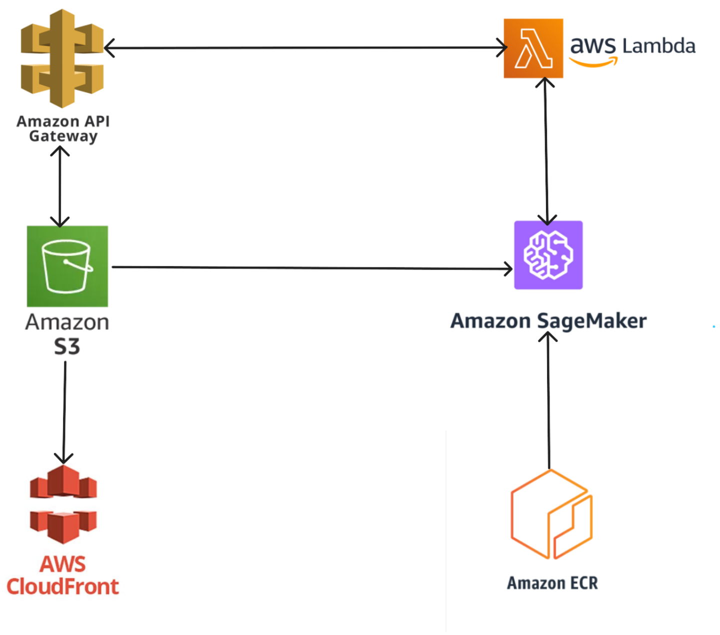
Model	**Implementation**	**Private Score**	**Runtime**
LightGBM	Library	0.476	2m 34s
XGBoost	Library	0.472	2m 22s
LightGBM	From scratch	0.466	1h 31m 23s
XGBoost	From scratch	0.455	38m 52s

Custom implementations provide deeper insight but are computationally expensive.

Chương 3

Bonus: Deploy Method

In the model deployment part, we have used AWS to deploy our model



First we created the container image that provide the suitable environment for our model and save it on AWS ECR

Then we upload our model artifact on AWS S3 bucket

After that we create our model on AWS SageMaker AI. In this project, i've created a serverless inference model, it will only incur charges when we use it(charge base on requests). It used the container image we uploaded to ECR previously and the model artifact we uploaded on S3 to create an inference endpoint

AWS Lambda is used to parse the request comes from AWS API gateway and send the request to sagemaker endpoint, after it got back the result, it return the result to API gateway

API gateway could take API calls from various sources, but in this project, i will only whitelist the request from my own website and block request from everywhere else.

AWS S3 bucket also host a static website and will have a form for user to input and call the API to API gateway

AWS cloudfront is a CDN, it will provide TLS(S3 host website on http), caching and possibly WAF(if we enable it, in this project we don't because it will charge us by the hours, not by request), and more resilient to attack like DDoS.