Kaggle Competition - Problematic Internet Use

Chan, Lok Hei (20693700, lhchanar) SZE, Fung Ming (20866474, fmsze)
Kwong Chi Yan (20852978, cykwongaf) ZHAO, Mingjie (21161845, mzhaoau)

1 Background

In today's digital age, problematic internet use among children and adolescents is a growing concern. A better understanding of this issue is crucial for addressing mental health problems such as depression and anxiety. Child Mind Institute announces a Kaggle Competition to challenge competitors to develop a predictive model capable of analyzing children's physical activity data to detect early indicators of problematic internet and technology use. Our objective is to develop a predictive model using pyspark.

2 Cloud Computing Platforms

2.1 Kaggle Notebook

Kaggle Notebooks are cloud-based interactive coding environments that allow users to write and run Python. They provide easy access to various datasets, enable collaboration by allowing users to share notebooks, and come with pre-installed libraries. However, it does not have good support for pyspark.

2.2 Databricks

Databricks is a cloud-based data platform that integrates big data analytics and machine learning in a collaborative environment. Built on Apache Spark, enabling fast processing of large datasets.

3 Dataset Description

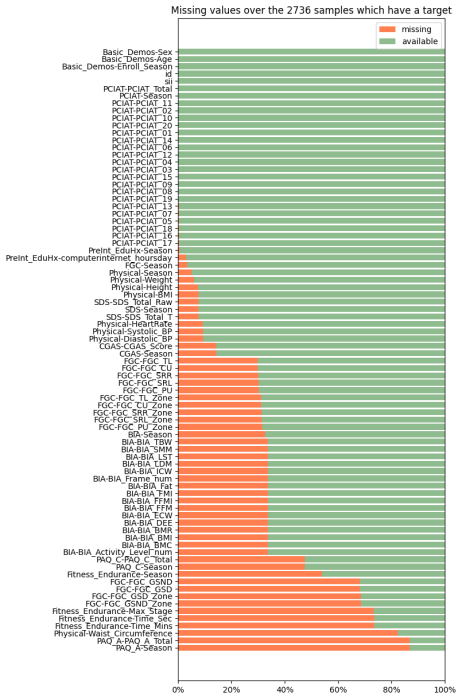
The competition data is compiled into two sources, parquet files containing the accelerometer (actigraphy) series and csv files containing the remaining tabular data. The majority of measures are missing for most participants. In particular, the target Severity Impairment Index (SII) is missing for a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

4 Data Visualization

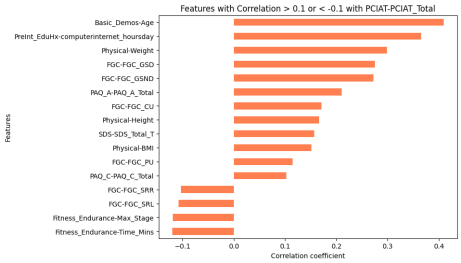
Data visualization allows us to assess data quality, structure, and distribution using intuitive graphs and figures. By using pre-installed libraries, we can easily identify patterns and insights within the data. This process enhances our understanding and enables effective data preprocessing and feature engineering in later stages.

4.1 Data Quality

The dataset contains 3960 training examples, out of which only 2736 are predictable with the target variable sii. Over half of the features have different levels of missing values. 13 features even have 50% missing values. Furthermore, 16 features have more than 0.1 correlation. Overall, the data quality is low. We have to do column removal and data imputation in the later stage.



(a) Percentage of Missing Value of Each Feature

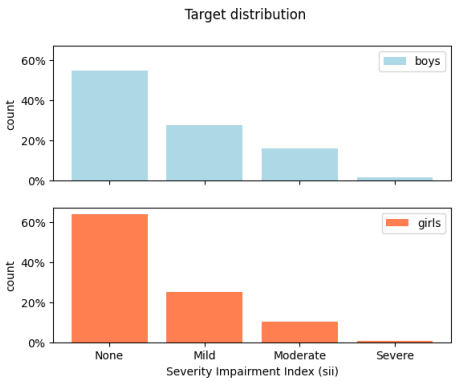


(b) Correlation

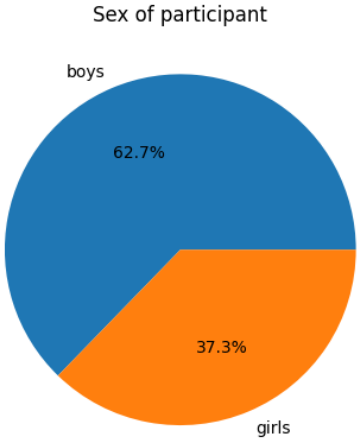
Figure 1: Data Quality

4.2 Data Distribution

The distribution of data shows that most of the data is imbalanced. Approximately 60% of the training data has SII of 0, and only about 1% of the data has SII of 3. This distribution is highly skewed. It might cause the model to tend to predict SII as 0, leading to underperformance in predicting higher SII levels. Besides that, most of the numeric columns are right-skewed, concentrating on low values. Last but not least, the male-female ratio is about 1.75:1. Overall, the data is imbalanced.



(a) Target Distribution



(b) Sex Distribution

Figure 2: Data Distribution

5 Data Preprocessing

We addressed missing values using KNN imputation. High-correlation features were dropped to mitigate multicollinearity. Imbalanced target distribution was handled using oversampling techniques

to ensure better model performance on minority classes.

6 Feature Engineering

The feature engineering process consisted of several key steps to prepare the dataset for machine learning models:

6.1 Handling Missing Data and Removing Redundant Features

- Columns containing the string "PCIAT" were dropped, as these features were not present in the test dataset.
- Columns with more than 40% missing values were identified and removed from both training and test datasets.
- Rows with excessive missing values (fewer than five non-null columns) were removed.
- Duplicate rows were removed from the dataset to avoid redundancy.

6.2 Feature Creation from Actigraphy Data

Actigraphy data, captured in time-series format, was transformed into meaningful statistical features:

- Non-wear flag data was filtered to include only instances where participants were wearing the watch.
- Missing values in actigraphy data were replaced with zeros.
- Daily summary statistics were computed for accelerometer readings (`X`, `Y`, `Z`, `enmo`, `anglez`) for each participant:
  - **Mean, Standard Deviation, Minimum, Maximum, Sum**: Calculated for each variable.
- Moderate-to-Vigorous Physical Activity (MVPA) time was computed based on the `enmo` column.
- Additional features such as **sleep efficiency** and **activity variability** were created:
  - **Sleep Efficiency**: Defined as `daily_mvpa_time / 86400`.
  - **Activity Variability**: Defined as `std_enmo / mean_enmo`.
- Aggregate statistics, such as overall mean, standard deviation, and total MVPA time, were computed for each participant across all days.

6.3 Generating New Features

Additional features were created to capture interactions between existing variables:

- **BMI_Age**: Product of Body Mass Index (BMI) and Age.
- **Internet_Hours_Age**: Product of daily internet usage hours and Age.
- **BMI_Internet_Hours**: Product of BMI and daily internet usage hours.

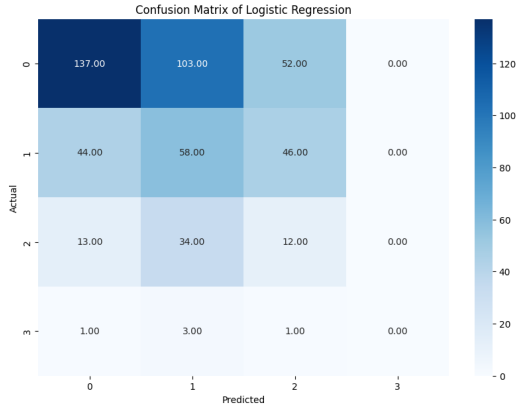
These features provided insights into the relationship between physical attributes, behavior, and internet use.

6.4 Final Output

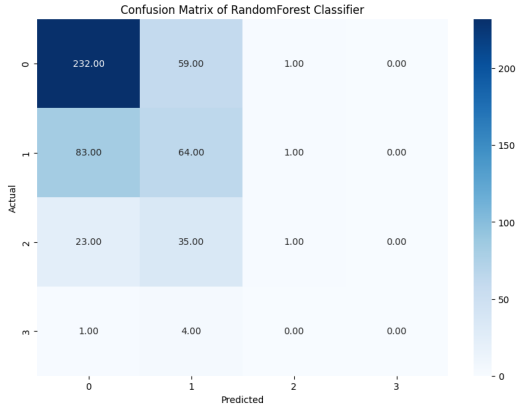
The final datasets (training, validation, and test) consisted of a combination of statistical features derived from actigraphy data, engineered features capturing interactions, and pre-existing demographic variables. These features were combined into a format suitable for machine learning models.

7 Model Training and Evaluation

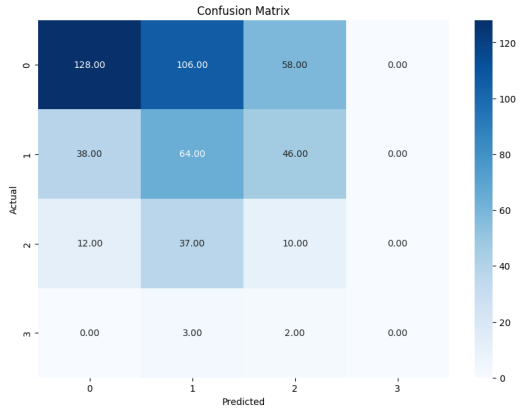
We experimented with several machine learning models to predict the Severity Impairment Index (SII) using PySpark. The models included Logistic Regression, Random Forest Classifier, One-vs-Rest Logistic Regression, and SparkXGBClassifier. Model performance was evaluated using the Quadratic Weighted Kappa (QWK) score, which measures the agreement between predicted and actual values while accounting for ordinal nature.



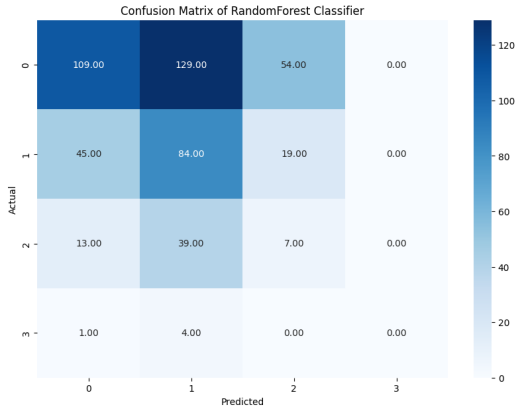
(a) Confusion Matrix of Logistic Regression



(b) Confusion Matrix of Random Forest Classifier



(c) Confusion Matrix of OVR Logistic Regression



(d) Confusion Matrix of SparkXGBClassifier

Figure 3: Confusion Matrices of All Models. (a) Logistic Regression, (b) Random Forest Classifier, (c) OVR Logistic Regression, (d) SparkXGBClassifier.

7.1 Logistic Regression

Logistic Regression was implemented as a baseline model using PySpark's MLlib. The model was trained on the processed training dataset and evaluated on the validation set. The Quadratic Weighted Kappa (QWK) score achieved by the Logistic Regression model was **0.1554**, indicating limited predictive power. The confusion matrix for Logistic Regression is shown in Figure 3a.

7.2 Random Forest Classifier

A Random Forest Classifier was implemented using PySpark to leverage its ensemble learning capabilities. The model achieved a higher QWK score of **0.2807**, demonstrating improved performance over Logistic Regression. The confusion matrix for the Random Forest Classifier is shown in Figure 3b.

7.3 One-vs-Rest Logistic Regression

The One-vs-Rest (OVR) Logistic Regression model was implemented to handle the multi-class nature of the target variable. Despite the additional complexity, the QWK score achieved was **0.1461**,

lower than both Logistic Regression and Random Forest. The confusion matrix for OVR Logistic Regression is shown in Figure 3c.

7.4 SparkXGBClassifier

The SparkXGBClassifier was implemented to explore gradient boosting as a potential method for improved prediction. However, the QWK score was **0.0298**, which was significantly lower than other models, likely due to overfitting or hyperparameter tuning challenges. The confusion matrix for SparkXGBClassifier is shown in Figure 3d.

8 Result

The performance of the models varied significantly, as shown by the QWK scores. The Random Forest Classifier outperformed other models with a QWK score of **0.2807**, making it the most effective model in this study. Logistic Regression, while simpler, provided a baseline with a QWK score of **0.1554**. The One-vs-Rest Logistic Regression and SparkXGBClassifier struggled to achieve high performance, highlighting the challenges posed by imbalanced and noisy data.

The confusion matrices provide further insights into the model predictions, showing the distribution of true and predicted labels. While Random Forest achieved the best results, there remains significant room for improvement through hyperparameter tuning and addressing data imbalances.

9 Discussion and Findings

9.1 Databricks Community Edition and Kaggle Notebook

When we look at Databricks Community Edition and Kaggle Notebooks, we see that each platform has its own strengths and limitation.

Databricks Community Edition is great for learning PySpark because of its strong connection to Apache Spark. However, we face some challenges. The file system are complex, requiring the use of Databricks Utilities (dbutils) to manage files. Additionally, we have to cold start the cluster each time we use it, which can take about half an hour, slowing down our training progress. Plus, the community edition runs on a single node, which limits its ability to fully use Spark's features.

In contrast, Kaggle Notebook is designed for data science competitions and offers a simple and user-friendly Jupyter Notebook environment. This makes it easy for users to quickly retrieve and work with files. However, it has a major limitation: Kaggle Notebooks aren't optimized for Spark, which makes it harder to train PySpark models and handle large datasets effectively. We have even faced memory issues and failed to train the PySpark models.

In summary, Databricks Community Edition is better for doing PySpark project, while Kaggle is easier to use for smaller projects that not using PySpark.

9.2 Comparison of Spark DataFrame and Pandas DataFrame

When we compare Spark DataFrames and Pandas DataFrames, we see that each has its own strengths and limitations.

Pandas DataFrames are great for small to medium-sized datasets that can fit into memory. They work quickly because they operate entirely in memory, allowing for fast data manipulation. Pandas provides many functions for filtering, grouping, and merging data, all within a simple and user-friendly interface. Also, it is well compatible with other libraries like Sklearn etc. However, a major drawback is that it can only handle data that fits within the available RAM.

In contrast, Spark DataFrames are designed for big data processing in distributed computing environments. They can manage large datasets that exceed memory limits by spreading the data across multiple machines. This is especially useful when working with big data tools like Hadoop

and various data sources like HDFS and S3. However, Spark has fewer built-in functions than Pandas, and not compatible with many libraries. Some functions like StratifiedKFold cannot be used.It makes the training complicated.

Overall, both DataFrames are powerful tools. However, Pandas DataFrame is ideal for individual users like us, as Colab, Kaggle Notebook provide the coding environment with 16 GB RAM and we normally do not process very large datasets. A well-compatible DataFrame can make the training easy.