**Child Mind Institute Problematic Internet Use Classification**

This repository provides the solution for the Kaggle competition involving the **Child Mind Institute's Problematic Internet Use** dataset. The goal is to build a classification model that can predict the severity of problematic internet use in children using various demographic and behavioral features.

**Overview**

The dataset contains information on children's mental health, specifically focused on their internet usage patterns and the potential correlation with various mental health conditions. The aim is to classify the severity of problematic internet use using machine learning techniques.

**Installation**

**Requirements:**

* Python 3.x
* Libraries: numpy, pandas, scikit-learn, matplotlib (optional for visualization), seaborn (optional for visualization)

You can install the necessary libraries using pip:

pip install numpy pandas scikit-learn matplotlib seaborn

**Cloning the Repository:**

git clone https://github.com/your-username/child-mind-institute-problem.git

cd child-mind-institute-problem

**Dataset:**

The datasets used for this project are available on Kaggle. To access them, go to [Kaggle's Child Mind Institute Problematic Internet Use page](https://www.kaggle.com/), download the data, and place it in the appropriate folder as indicated in the code.

**Dataset**

**train.csv**

Contains the training data with features such as demographic information, behavioral data, and the target column sii (severity of internet use).

**test.csv**

Contains the test data (without target labels) used for making predictions.

**data\_dictionary.csv**

A dictionary that explains each feature in the dataset.

**sample\_submission.csv**

An example file showing the format required for submission.

**Code Breakdown**

The following steps outline the core steps of the code and its purpose.

**1. Data Loading and Exploration**

import pandas as pd

train\_data = pd.read\_csv('/kaggle/input/child-mind-institute-problematic-internet-use/train.csv')

test\_data = pd.read\_csv('/kaggle/input/child-mind-institute-problematic-internet-use/test.csv')

data\_dict = pd.read\_csv('/kaggle/input/child-mind-institute-problematic-internet-use/data\_dictionary.csv')

sample\_submission = pd.read\_csv('/kaggle/input/child-mind-institute-problematic-internet-use/sample\_submission.csv')

* This loads the training and test data along with the data dictionary, which explains the dataset features.

**2. Data Preprocessing**

train\_data.dropna(axis=0, subset=['sii'], inplace=True)

y\_train\_full = train\_data.sii

X\_train\_full = train\_data.drop(columns=['sii', 'id'], axis=1)

X\_test = test\_data

# Find common columns between train and test data

common\_columns = X\_train\_full.columns.intersection(X\_test.columns)

X\_train\_full = X\_train\_full[common\_columns]

X\_test = X\_test[common\_columns]

* **Handling Missing Data**: Rows with missing target values (sii) are removed.
* **Feature Selection**: The target column sii is separated from features, and common columns between training and test data are selected.

**3. Splitting the Data**

from sklearn.model\_selection import train\_test\_split

X\_train, X\_valid, y\_train, y\_valid = train\_test\_split(X\_train\_full, y\_train\_full, train\_size=0.8, test\_size=0.2, random\_state=0)

* The dataset is split into a training and validation set (80%/20% split).

**4. Preprocessing Pipelines**

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder

# Preprocessing for numerical data: Impute missing values and scale features

numerical\_transformer = Pipeline(steps=[('imputer', SimpleImputer(strategy='mean'))])

# Preprocessing for categorical data: One-hot encoding

categorical\_transformer = Pipeline(steps=[('onehot', OneHotEncoder(handle\_unknown='ignore'))])

# Combining preprocessing pipelines

preprocessor = ColumnTransformer(transformers=[('num', numerical\_transformer, numerical\_cols),

('cat', categorical\_transformer, categorical\_cols)])

* **Numerical Data**: Missing values are imputed using the mean of each column.
* **Categorical Data**: One-hot encoding is applied to categorical features to convert them into numerical values.

**5. Hyperparameter Tuning**

from sklearn.model\_selection import GridSearchCV

param\_grid = {

'model\_\_hidden\_layer\_sizes': [(64, 32), (128, 64), (64, 64), (32, 32)],

'model\_\_activation': ['relu', 'tanh'],

'model\_\_solver': ['adam', 'sgd'],

'model\_\_learning\_rate\_init': [0.001, 0.01, 0.1],

'model\_\_alpha': [0.0001, 0.001, 0.01]

}

grid\_search = GridSearchCV(clf, param\_grid, cv=5, scoring='accuracy', n\_jobs=-1, verbose=1)

grid\_search.fit(X\_train, y\_train)

* **GridSearchCV**: A grid search is performed over several hyperparameters to find the best configuration for the model.

**6. Model Definition and Training**

from sklearn.neural\_network import MLPClassifier

model = MLPClassifier(hidden\_layer\_sizes=(64, 32), max\_iter=1000, random\_state=42, alpha=0.01, activation='relu', learning\_rate\_init=0.01, solver='adam')

clf = Pipeline(steps=[('preprocessor', preprocessor), ('model', model)])

clf.fit(X\_train, y\_train)

* **Model**: An MLPClassifier (Multi-layer Perceptron) is defined with two hidden layers (64 and 32 neurons).
* **Pipeline**: A pipeline is created combining preprocessing and model fitting.

**Modeling Pipeline**

The modeling pipeline consists of two main steps:

1. **Data Preprocessing**: The data is cleaned, transformed, and ready for use.
2. **Model Fitting**: A Multi-layer Perceptron (MLP) model is trained using the prepared data.

**Evaluation**

Once the model is trained and tuned, predictions are made on the test data, and the performance is assessed on a validation set.

**Conclusion**

This repository provides a comprehensive solution to the **Child Mind Institute's Problematic Internet Use** problem. By utilizing MLPClassifier in a well-structured pipeline, we can predict the severity of problematic internet use based on various features. The code also includes steps for hyperparameter tuning and model validation, making it a useful baseline for this problem.

By following this structure, your GitHub repository will be well-documented, easy to navigate, and professional, giving users clear instructions on how to understand and reproduce your work.