END-EVALUATION REPORT ON

DYNAMIC ADAPTATION OF DIGITAL WORKSPACE

ABSTRACT

To create an optimized digital workspace that undergoes adaptation based on the task a user is performing, two basic things are required:

1. Correct identification of the task- using Machine Learning Models
2. Implementation of Dynamic Adaptation- interaction with OS APIs using Python programs to change low-level system functionalities.

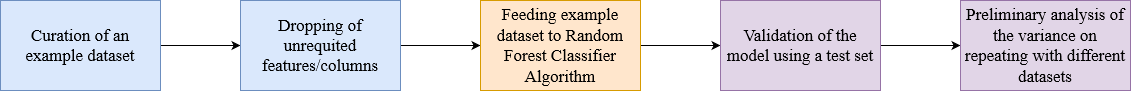
Accurate recognition of human activity by training an ML model allowed me to optimise computational resources by developing a Python script that modified functions such as Battery Plans and Display Settings.

METHODOLGY

Dataset: Human Activity Recognition on PC by analysis of recorded Raw Gaze Coordinates (linked to research paper posted on Kaggle along with the raw dataset I derived my final dataset from)

<https://www.kaggle.com/datasets/namratasri01/eye-movement-data-set-for-desktop-activities>

Machine Learning Model: Random Forest Classifier under Supervised Learning *Ensemble* models was chosen and the tuned to the requirements of my project.

Python Script: Developed in locally hosted Jupyter Notebook in order to access DLLs of the Windows API.

TECH STACKS USED

Frameworks:

* sklearn- provides algorithms and tools for data analysis and modelling; *used here* for training and testing of supported ML model: RandomForestClassifier
* imblearn- addresses challenges posed by imbalanced datasets in ML; *used here* to resample instances of target 'Activity' class while doing ML model training and testing

Libraries:

* pandas- for data manipulation and analysis by provision of data frames; *used here* for initial dataset loading and manipulation required for training of the ML model
* NumPy- generally provides mathematical support for multi-dimensional arrays; *used here* to ensure a specific array, used in GDI modification written to change colour hues, meets the memory layout requirement of being contiguous
* gdi32- DLL that handles all graphics-related tasks in Windows applications; *used here* to change colour hues during specific activities for user comfort
* user32- part of the Windows API that allows work with the user interface; *used here* for access and modification of Display Settings

Languages:

Python written onto Jupyter Notebook

Use of standard python modules such as datetime, time, random; *used* for applicability during various parts of code written in the adaptive\_workspace.ipynb program.

Other standard Python modules:

* sbc- installed using pip; *used here* to control the brightness of a Windows OS linked monitor
* ctypes- FFI providing a path to call functions written in C; *used here* to allow runtime loading of DLLs which will enable me to access system-level functionality under due control and support of the Windows OS
* winreg- interaction with Windows Registry; *used here* to access and modify the supported battery plans for subsequent resource optimisation
* subprocess- interaction with system shell and running external commands; *used here* to set a specific Battery Plan as the active one on my system.

CHALLENGES FACED

Finding a dataset that could aid my original intended aim was a difficult process since no datasets based on user preferences throughout the day or similar was found. Once I found the H.A.R. dataset based on eye-movement, the data pre-processing was an easier task with not a lot of null values to tackle and scaling to be done. Manual cleaning and removal of datapoints was a lengthy task. Choosing an ML model based on documentation and nature of my task proved to be a relatively simpler process. Further training and fine tuning of the model and comparison with other models was difficult and time consuming because of the imbalanced classes and size of the dataset. Resampling did not yield good results so instead of synthetic generation of datapoints, manual introduction of similar datapoints to balance out the classes has fixed issues with minority classes of BROWSE and DEBUG but code defaults to READ when subjected to WRITE targeting dataset due to similar nature of range and scatter of datapoints according to my estimate. Further developing a Python script that allowed me to not only access but modify the Windows API for me to be able to change Display settings and Battery plans was the biggest challenge since there are no documented codes for reference either. Initially I wished to be able to modify brightness settings, refresh rates and battery plans but in the iterative process of writing codes and undesirable outputs being generated due to underlying system configuration restrictions, I have now ended the BYOP phase by adaptation of brightness and battery plans only.

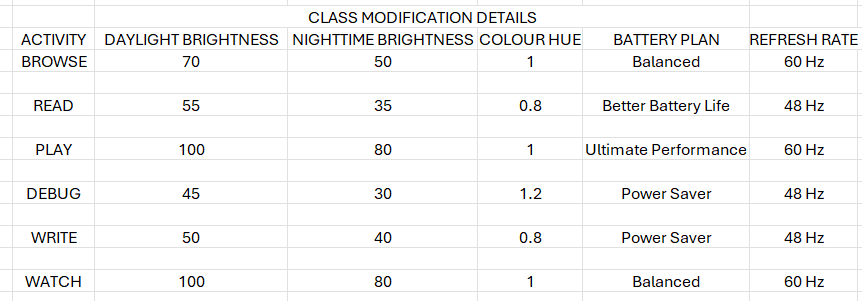
FINAL DELIVERABLE AND RESULTS

<https://github.com/advikasinha/BYOP_24_END>

Provided is a link to the GitHub repository containing the test datasets, executable.ipynb file and a README.MD that gives out detailed instruction and explanation of the code.

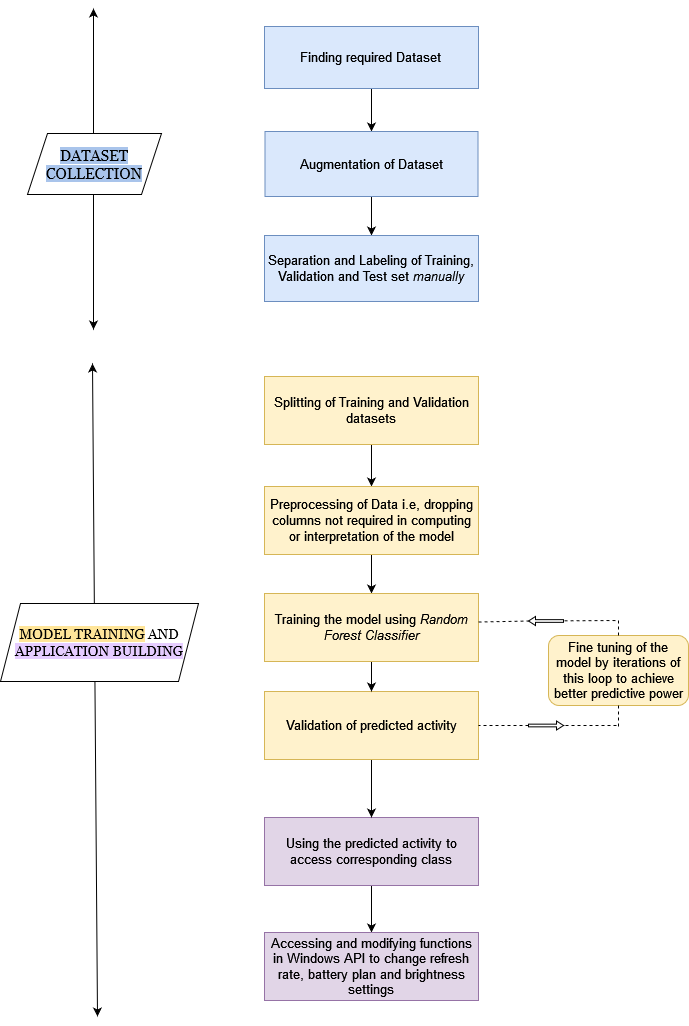
Since the program targets better battery health and performance triggered based on the activity being performed by the user, the underlying system functionalities are being accessed. Therefore, please grant administrative privilege to the IDE you work with. Also, for experiencing the varied changes brought in by use of this program, manual switching of datasets will have to be done since real time-datapoints cannot be fed to the model for the lack of technical hardware to support tracking of eye-movement.

The following changes are brought in to optimise computational resources depending on the activity predicted by my ML model. By adapting battery plan, many system settings like processor power management and system cooling policies are fine tuned for the device to be able to perform more efficiently. User comfort is also key while these optimisations are performed and so with time-regulated brightness settings, colour hues have also been adapted. For different Windows devices, the linked repository also contains code for change in refresh rate.



\*Following section of my report discusses the procedure undertaken and progress since the Mid-Evaluation. It will not delve into the difficulties encountered before Mid-Evaluation, only different things tried subsequently.

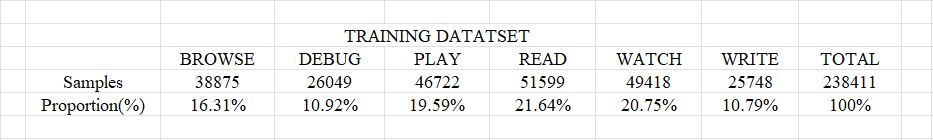
DATASET ALTERATION AND CHOICE OF MODEL:

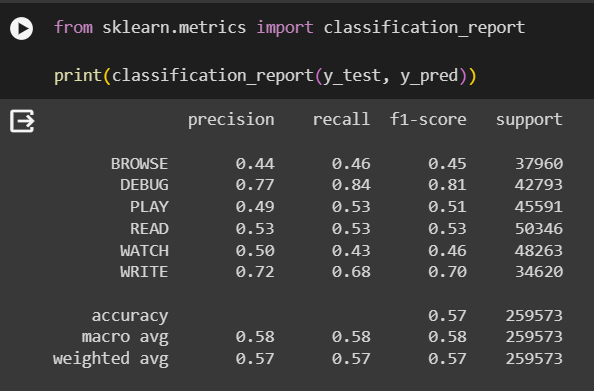
Part dealing with Dataset collection, as depicted by the project pipeline architecture, had been accomplished before the Mid-Evaluation. Initial submitted dataset was of 238411 datapoints. As I experimented with various classification-based models, the imbalance in target activity classes made it difficult to train models with higher accuracies or precision. The csv plots of each dataset corresponding to an activity had been adjusted for outliers and had varying scales too but all robust models still gave wrong outputs as the final predicted labels.

Using distance-based models like KNN for processing these raw-gaze coordinates was also futile because each activity has multiple points within the same concentrated range unless the model is trained on edge values which would make it highly sensitive to outliers.

As discussed previously, I had a large categorical dataset to work with. So choosing an ensemble learning model would helped optimise better predictive performance and decrease the variance of the single estimate. The ensemble models I worked on and comapred were: Decision Trees, Random Forests and XGBoost.

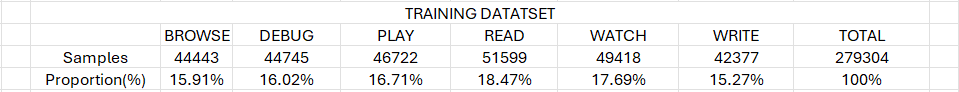
All three of these models, with the default hyperparameter, produced inaccurate results for the minority classes in my multi class classification task. Resampling by use of SMOTE, class weights and RandomUnderSampling proved ineffective for the minority classes. The models displayed high precision and recall for such classes but failed to predict accurate results.





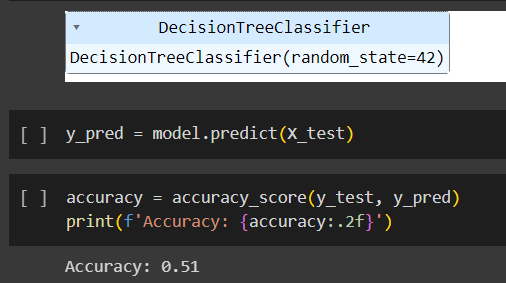
Evidently, classes like Debug and Write with overall lesser contribution to the training dataset had higher precision and recall compared to the majority classes but model failed to attribute a test dataset to them correctly.

My parent dataset contained datapoints for 24 participants corresponding to the 6 label activities. I had picked 6 participants for training the model on given their csv plots and so as to not add load onto my system. I manually subjected extra subsets of an n number of participants for the classes of BROWSE, DEBUG and WRITE to processing such that general trend of the activity plot was not lost and these data points could be added to the training dataset to make the classes more balanced and easier for the model to work with.



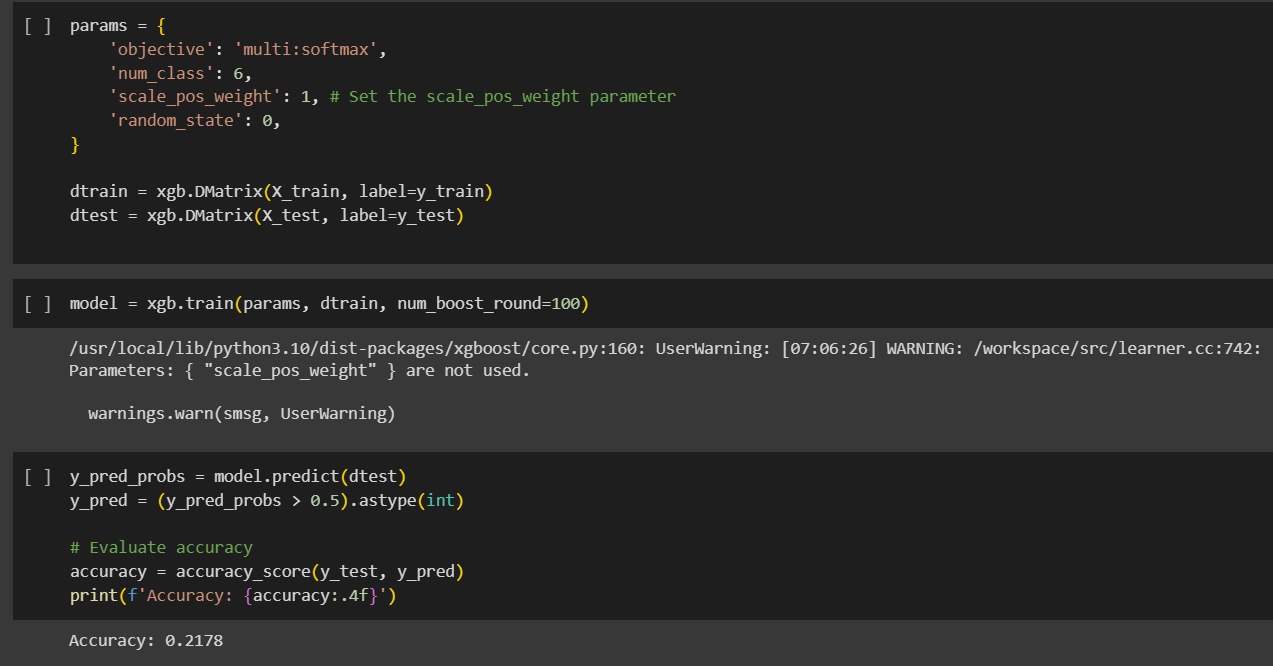
Classes were then more balanced with READ and WATCH still being the majority classes by a slight. I did not further attempt to manually modify the dataset. Instead, I applied the resampling techniques again, a combination of oversampling and under-sampling, to get accurate labels outputted by my chosen model when a minority activity corresponding dataset was subjected to it.

RANDOM FOREST CLASSIFIER:

Random Forest Classifier is robust ensemble ML algorithm that considers a random subset of features during the splits it makes. It also deals with numeric data better than it deals with categorical data.

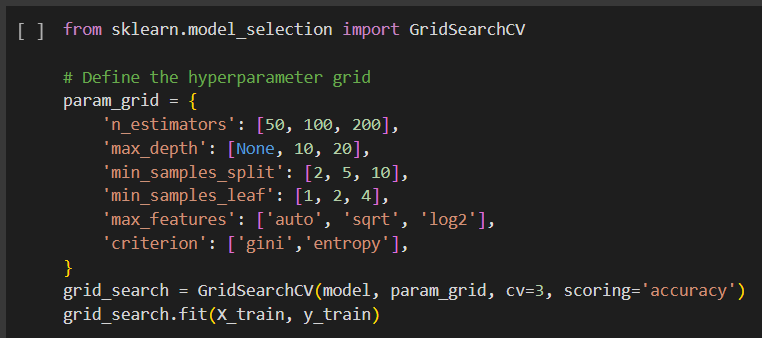
A singular Decision Tree would also make a random subset and given the nature, size and structuring of my preliminary dataset it would generally fail to predict target labels accurately.

 Knowing that I could not execute the final code on a Google Colab notebook, since I would need to access the system’s API and OS resources for overall implementation, XGBoost naturally became the less preferred choice. I still worked on it with my preliminary datasets and given the class weightage, I tried to resample the dataset the model trained on by using . I coupled this with a few other parameters that targeted the loss functions that was fed back into the model for an iterative process to produce the most accurate result (i.e., Boosting).

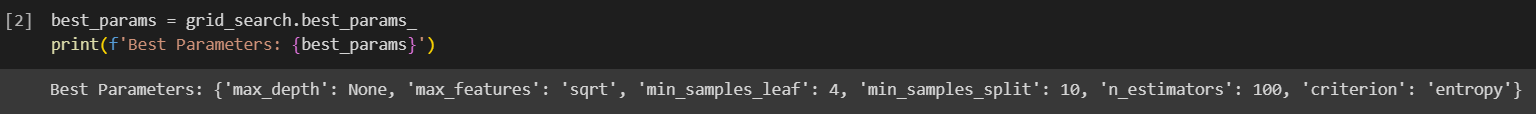


The model still failed to produce accurate results.

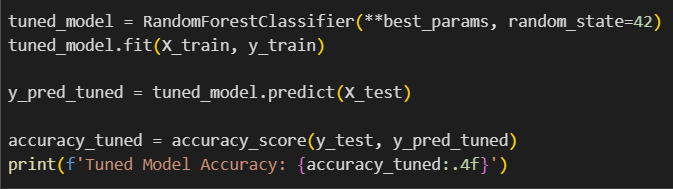
Random Forest Classifier on the other hand has a special parameter called class-weight that is equipped to deal with imbalanced classes. Immune to overfitting and easier to tune hyper parameters for, I chose this model and ran a grid search having the model be trained on the new dataset.

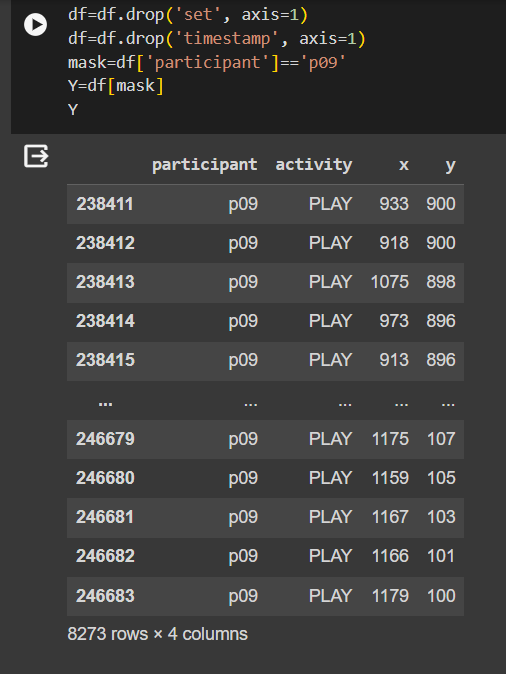


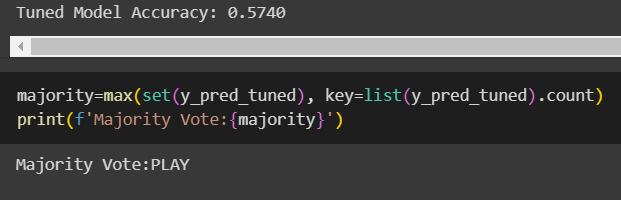
Best set of parameters:



Testing of tuned model on test split of the fed dataset:



Accuracy was up from 51% to 57.40% and the model predicted the correct label class as well.

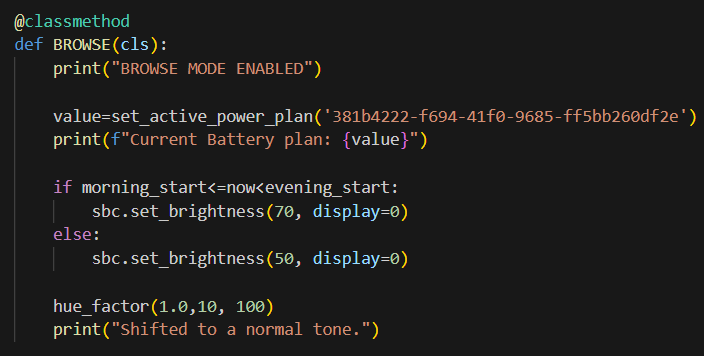


The final code shared also has the model be subjected to a combination of resampling in order to rid the final result of any inconsistencies based on the frequency of a single class picked by the random splits during model training.

This has fixed issues with minority classes of BROWSE and DEBUG but code defaults to READ when subjected to WRITE targeting dataset due to similar nature of range and scatter of datapoints according to my estimate.

PYTHON CODE SUBSEQUENTLY WRITTEN:

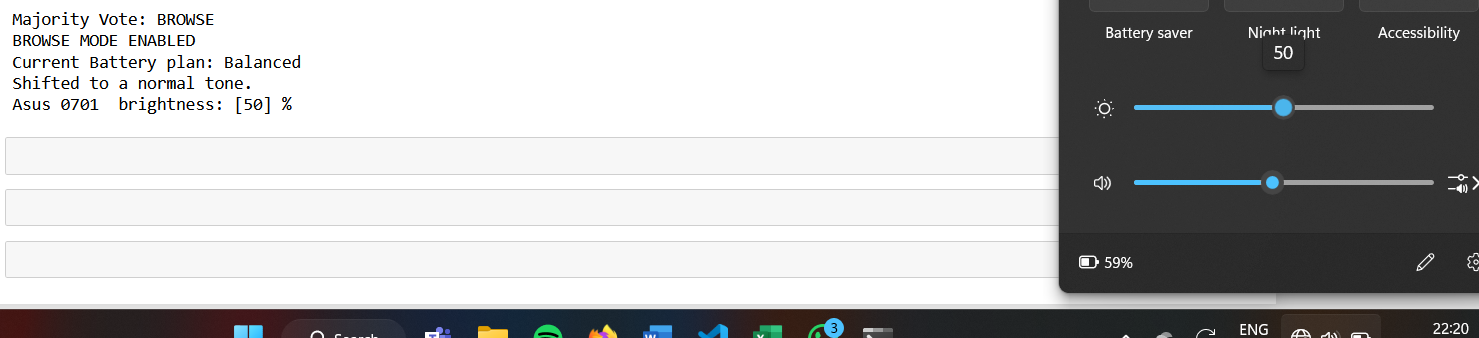
Class ActivityDecider in the adaptive\_workspace.ipynb notebook contains methods with names of the target activities. These methods are called and the snippets of code present inside them are executed.

The method contains the value of the battery plan\_id and a call to the set\_active\_power\_plan() defined previously.

It then contains the brightness level primary display of a system setup should have depending on the geographical sunlight hours and system time.

Lastly, all methods have a value of the hue\_factor which enables the display to shift between warm, normal and cooler tones to cater to User comfort.

EXAMPLE OF RESULTS:





CONCLUSION:

Making a context-aware system requires a dataset that was built on user preferences. My project has targeted certain activities based on the only dataset I found and has worked to optimise and improve resource allocation and utilization. By targeting features that such as battery plans and refresh rates, we can work more efficiently in our digital workspaces while not compromising on user comfort enabled by optimal display settings. Machine Learning Models are an integral part of this process because they add the element of dynamism without the human intervention.