Tasrif: processing wearable data in Python

Abdulaziz Al Homaid†, Syed Hashim, Fadhil Abubaker, Ummar Abbas†, Faisal Farooq, Joao Palotti†
Qatar Computing Research Institute, HBKU, Doha, Qatar
{abalhomaid,smoosavi,fabubaker,uabbas,fafarooq2,jpalotti}@hbku.edu.qa
† Correspondent Authors

***Abstract*—This paper introduces Tasrif, an open source Python framework that facilitates processing of wearable data. Based on the pipes and filter design pattern, Tasrif provides multiple functionalities from data readers of large wearable datasets to integration with popular ML frameworks. The vision of Tasrif is to obviate or at least significantly accelerate the wearable data wrangling step. The code is publicly available at https://github.com/qcri/tasrif/.**

***Index Terms*—Wearables, Lifestyle data, preprocessing, time-series data, open source**

I. Introduction

The last decade has seen a significant growth in developing and using new sensing technologies to ubiquitously and objectively monitor people's lifestyle [1]. Efforts on sensor miniaturization now allow devices to inexpensively and longitudinally monitor an extensive array of health variables, such as physical activity, heart rate variability, oxygen saturation and sleep. This quick technology evolution opened the doors to many research projects, such as the NIH All of Us research program[1], which will gather lifestyle data, among other types of data, from 1M or more people living in the US to enable new kinds of individualized health care. Another significant example is the MyHeartCounts project[2] [2], which has recently publicly released the anonymized physical activity, sleep and cardiovascular data of 50k participants [3].

Whether these projects aim to foster cardiovascular research [2], detect fall on elderly population [4], or measure how sleep quality correlates with physical activity [5], there is a non trivial amount of data processing that needs to be done for every new data collection. In fact, a 2020 report shows that 45% of a data scientist's time is spent on preparing the data, i.e., loading and cleaning for the task at hand [6]. These steps, commonly known as *data wrangling*, unfortunately, can even be an insurmountable entry barrier for research clinicians and data scientists without deep knowledge of time-series analysis.

To address this challenge, a few libraries and frameworks have been recently released to the research community. For example, pyActigraphy [7] and HypnosPy[3] help on preprocessing data extracted from actigraphy devices, which are wearable medical devices traditionally used in clinical studies. However, the scope of these libraries is very limited on performing specific tasks required by actigraphy data, such as reading actigraphy format and scoring sleep stages. These libraries do not generalize to *commercial* devices such as Fitbit, Huawei or Apple Watch that have become ubiquitous within this domain. To further accentuate this challenge, most of these devices use proprietary technology and data representation due to a lack of enforceable standards. In this paper we present **Tasrif**, a Python library to enable researchers and industry practitioners to preprocess wearable time series data captured with commercial devices, Tasrif is envisioned to enable reproducible experiments, while providing an intuitive user programming interface. Tasrif is built on Python's standard data science stack and is seamlessly compatible with several complementary libraries, such as Facebook's Kats[4] and TSFresh [8]. Our goal is to assist digital health researchers with reusable data pipelines that can accelerate preprocessing of wearable data for downstream tasks within machine learning (ML) frameworks like scikit-learn, pytorch, tensorflow, or tslearn.

II. System Description

Tasrif toolkit for lifestyle data processing is based on pipes and filters design pattern. Preprocessing data to make it consumable by a downstream ML task is complex, as there is a significant variability in data formats and a multitude of cleaning and wrangling steps that can be performed. The pipes and filters pattern allows a complex processing task to be decomposed into a collection of small tasks or filters that can be independently reused and tailored together to create new processing workflows. Filters are designed with an interface to receive data and return data after processing. A set of entities called pipes help to assemble such operators to facilitate defining processing workflows. Such an architecture affords an elegance to define new processing workflows by putting filters and pipes together as per the processing needs of the dataset or the task at hand. In this section, we will describe how Tasrif realizes such an architecture. A UML overview of the main modules implemented is shown in Figure 1.

A. Architecture

The architecture of Tasrif is based on two conceptual entities *pipes* and *filters*, implemented as what we called *Operator* classes. Pipes are modeled as *infrastructure* operators and filters are modeled as *functional* operators.

Infrastructure Operators represent pipes that allow users to assemble a group of functional operators. Tasrif contains many different types of infrastructure operators based on

[1] https://allofus.nih.gov/
[2] https://med.stanford.edu/myheartcounts.html
[3] https://github.com/HypnosPy/HypnosPy
[4] https://facebookresearch.github.io/Kats/

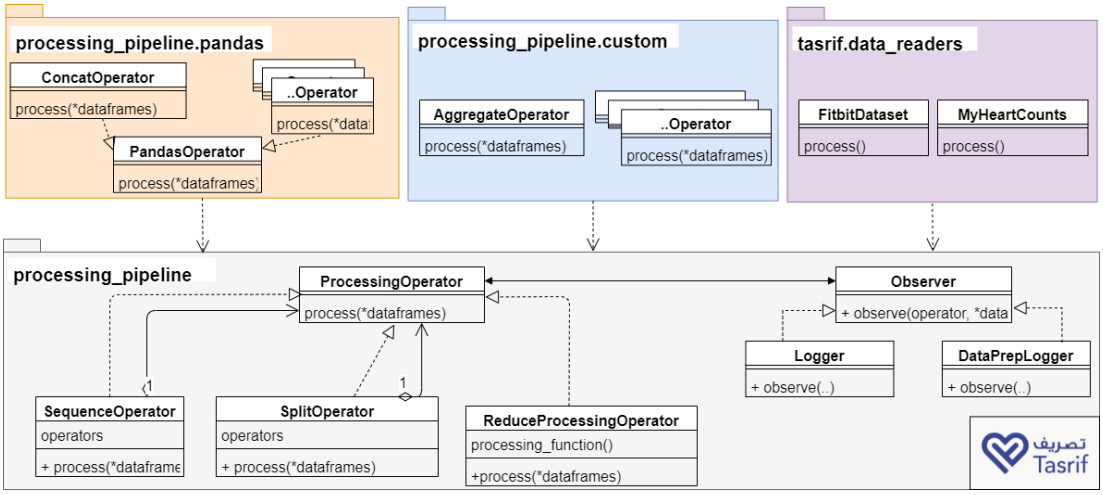

Fig. 1: Tasrif's UML schema with its main modules: infrastructure (bottom) and functional operators (top).

how the data is passed from one operator to another. These operators are summarized below:

- *SequenceOperator* executes an ordered sequence of functions or other infrastructural operators. The data flow is linear, with output of one operator feeding the next in a sequential manner.
- *ComposeOperator* consists of a collection of operators. Data received as input to the compose operator is passed to all the operators in the composition. Each operator then processes the input data independently and the final result is a composition of the results of all the operators.
- *SplitOperator* takes a list of input data frames and passes each dataframe to a different containing operator as per a user defined configuration.
- *MapProcessingOperator* is an abstract base class that receives as input a list of data frames. The resulting data frame is a combined output of applying the processing function (provided by the derived class) on each input element independently in a map-like fashion.
- *ReduceProcessingOperator* is an abstract base class that receives as input a list of data frames and applies a reduction function (provided by the derived class) successively on each input, resulting in a unique output data frame.

Besides the above, we have parallel versions of *Compose*, *Split* and *MapProcessingOperators* that uses multiprocessing Python packages to execute the composed operators in parallel.

Functional Operators represent filters that allow the user to define operations applied to incoming data:

- *Library specific operators* are operators that typically have a direct mapping with an existing Python data processing package. Tasrif is designed to contain all operators belonging to a specific library in a single package. For example, all operators related to *pandas* will be in the *tasrif.processingpipeline.pandas* package.
- *Domain-specific operators* are custom-designed operators that specifically work with timeseries data coming from wearable devices. All such operators are contained in the *tasrif.processingpipleine.custom* package

Besides this distinction, we also have *source* operators that are filters that typically occur at the very beginning of such a pipeline. As opposed to other functional operators, they do not receive their data from their predecessor. Rather, they load data either from disk (as a data reader) or other sources.

B. Functionalities

The main functionalities in Tasrif currently are:

- Read and preprocess public datasets available for research, e.g., MyHeartCounts [2] and SleepHealth [9];
- Read and preprocess data from commercial device, e.g., the default format of the exported data from Fitbit, Huawei and Apple Watch from the users account;
- Use the concept of pipelines, making the output easy to share and reproduce;
- Run pipelines sequentially or in parallel. Tasrif users can conveniently leverage multi-threading;
- Use the concept of observers to inspect and visualize how each operator process the data.

C. Code Example

We present an end-to-end example of Tasrif in action in Figure 2. In this short example, we illustrate how to read the data of a large collection, MyHeartCounts [2], for a simple machine learning use case: *given one hour of activity data per user, predict their activity in the following 15 minutes*. Note that with few lines of code, we execute a data cleaning procedure, create features and a machine learning model for the task. The bottom part of Figure 2 shows, for one random participant, the data before and after the cleaning procedure in which we filtered out days with less than 2,500 steps.

III. Discussion and Conclusion

This paper proposes a system to remove the entry barrier of wearable data wrangling faced by digital health researchers and practitioners globally. We developed Tasrif, a Python library focused on the pipeline design pattern. Tasrif is designed to prepare raw wearable data from various commercial devices into ready-to-be-used structures for downstream tasks

```
from tasrif.processing_pipeline import SequenceOperator
from tasrif.processing_pipeline.pandas import RenameOperator, ConvertToDatetimeOperator, DropNAOperator
from tasrif.processing_pipeline.custom import ReadCsvFolderOperator, AggregateOperator, FilterOperator,␣
↪SetStartHourOfDayOperator, SlidingWindowOperator
from tasrif.processing_pipeline.observers import VisualizeDaysObserver
from tasrif.data_readers.my_heart_counts import MyHeartCountsDataset
from tasrif.processing_pipeline.tsfresh import TSFreshFeatureExtractorOperator
import sklearn

# Read the physical activity data from 100 participants of the MyHeartCounts dataset
mhc = MyHeartCountsDataset(path_name='/path/to/MyHeartCounts/', table_name='healthkitdata', participants=100,
                           sources=['phone'], types=['HKQuantityTypeIdentifierDistanceWalkingRunning'])
df = mhc.process()

# Filter the data by a minimum number of steps in a day and plot the results
observer = VisualizeDaysObserver(date_feature_name='startTime', signals=['value'], participant_identifier='recordId',␣
↪start_hour_col='shifted_time_col', end_date_feature_name='endTime', figsize=(14, 7))

preprocess_pipeline = SequenceOperator([SetStartHourOfDayOperator(date_feature_name='startTime',␣
↪participant_identifier='recordId', shifted_date_feature_name='shifted_time_col', shift=9),
                                        FilterOperator(participant_identifier="recordId",␣
↪date_feature_name="shifted_time_col", filter_type="include", day_filter={"column": "value", "filter": lambda value:␣
↪value.sum() > 2500})],
                                        observers=[observer])
processed_df = preprocess_pipeline.process(*df)

# Prediction with standard Scikit-Learn after extracting features with a sliding window and a TSFresh operator
swo = SlidingWindowOperator(winsize="1h15t", time_col="startTime", label_col="value", participant_identifier="recordId")
df_timeseries, y, _, _ = swo.process(*processed_df)[0]

feature_extraction_pipeline = SequenceOperator([TSFreshFeatureExtractorOperator(date_feature_name="startTime",␣
↪value_col="value"), DropNAOperator(axis=1)])
X = feature_extraction_pipeline.process(df_timeseries)[0]
pipe = sklearn.pipeline.Pipeline([('scaler', sklearn.preprocessing.StandardScaler()), ('svc', sklearn.svm.
↪SVR(kernel="linear "))])
pipe.fit(X, y)
```

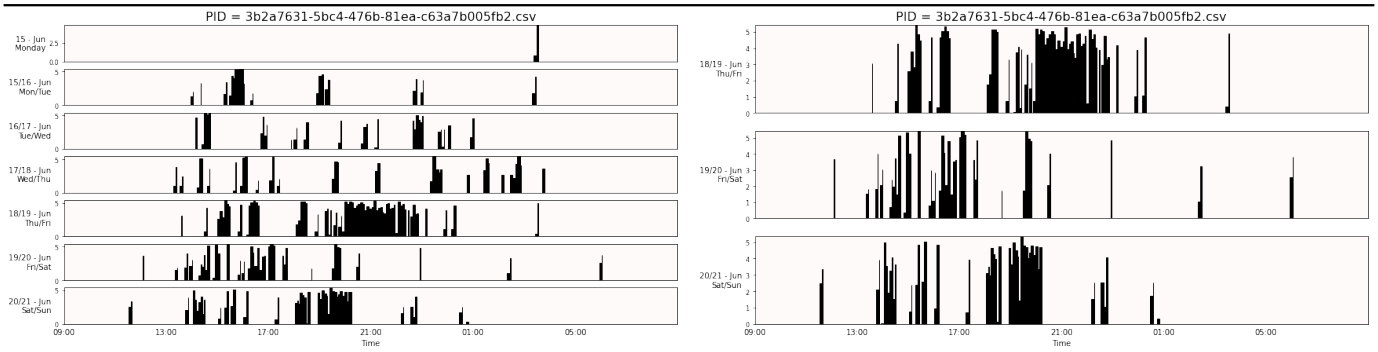

Fig. 2: **Code snippet demonstrating the usage of Tasrif on the MyHeartCounts [2].**

(such as ML). Tasrif is an ongoing effort and we plan to release further functionality in upcoming releases. In addition to building a more comprehensive library of data pipelines, upgrading core infrastructure and custom operators, we are also considering a visual interface to create pipelines without Python programming expertise. The aim is to make Tasrif easily accessible by clinical researchers. We are also hopeful that the community will contribute functionality to Tasrif based on common use cases making it more robust and valuable.

References

[1] I. Perez-Pozuelo *et al.*, "The future of sleep health: a data-driven revolution in sleep science and medicine," *NPJ digital medicine*, 2020.

[2] M. V. McConnell *et al.*, "Feasibility of obtaining measures of lifestyle from a smartphone app: the myheart counts cardiovascular health study," *JAMA cardiology*, 2017.

[3] S. G. Hershman *et al.*, "Physical activity, sleep and cardiovascular health data for 50,000 individuals from the myheart counts study," *Scientific data*, 2019.

[4] P. Pierleoni *et al.*, "A high reliability wearable device for elderly fall detection," *IEEE Sensors Journal*, 2015.

[5] J. Palotti *et al.*, "Predicting several sleep quality metrics based on same day physical activity," in *ICLR AIfPH*, 2021.

[6] Anaconda, "The state of data science," 2020. [Online]. Available: https://www.anaconda.com/state-of-data-science-2020

[7] G. Hammad *et al.*, "pyactigraphy: Open-source python package for actigraphy data visualization and analysis," *PLoS Computational Biology*, 2021.

[8] M. Christ *et al.*, "Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package)," *Neurocomputing*, vol. 307, pp. 72–77, 2018.

[9] S. Deering *et al.*, "Real-world longitudinal data collected from the sleephealth mobile app study," *Scientific Data*, 2020.