Statistical Outlier Curation Kernel Software (SOCKS): A Modern, Efficient Outlier Detection and Curation Suite

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

23-07-2021 / 28-07-2021

CITATION

Pal, Prasanta; Lutterveld, Remko Van; Quirós, Nancy; Taylor, Veronique; Brewer, Judson (2021): Statistical Outlier Curation Kernel Software (SOCKS): A Modern, Efficient Outlier Detection and Curation Suite. TechRxiv. Preprint. <https://doi.org/10.36227/techrxiv.15043695.v2>

DOI

[10.36227/techrxiv.15043695.v2](https://doi.org/10.36227/techrxiv.15043695.v2)

Statistical Outlier Curation Kernel Software (SOCKS): A Modern, Efficient Outlier Detection and Curation Suite

Prasanta Pal, *Member, IEEE*, Remko Van Lutterveld, Nancy Quirós, Veronique Taylor, and Judson A. Brewer

Abstract— Real world signal acquisition through sensors, is at the heart of modern digital revolution. However, almost every signal acquisition systems are contaminated with noise and outliers. Precise detection, and curation of data is an essential step to reveal the *true-nature* of the uncorrupted observations. With the exploding volumes of digital data sources, there is critical need for a robust but easy-to-operate, low-latency, generic yet highly customizable, *outlier-detection* and *curation* tool, easily accessible, adaptable to diverse types of data sources. Existing methods often boil down to data smoothing that inherently cause valuable information loss. We developed a *C++* based, software tool to decontaminate *time-series* and matrix like data sources, with the goal of recovering the *ground-truth*. The SOCKS tool would be made available as an open-source software for broader adoption in the scientific community. Our work calls for a philosophical shift in the design pipelines of real-world data processing tasks. We demonstrate that, raw data should be decontaminated first, through conditional flagging of outliers, schema-based curation of flagged data-points, followed recovery of asymptotically converged ground-truth. We claim, this method would immensely enhance accuracy of a wide range of traditional data processing tasks.

Index Terms—Applications of adaptive filters, Bio biomedical signal processing, Noise, Filter, Image processing, Data science, Image restoration, Machine learning, Outlier Detection, Outlier, C++, Algorithm, Time-series

INTRODUCTION

IN his book “*21 lessons for the 21st century*” Yuval Noah Harari stated, “*In a world deluged by irrelevant information, clarity is power*” [1]. The data-driven world of science, technology and medicine is so much deluged by countless engines of digital data collection processes that clarity is often profoundly blurred by the clout of over-information. It is more important than ever that, pursuit of information clarity is of far more importance than its mere volume. Clarity often comes through curation of the source level *raw-input-data* ($\mathcal{D}_{\text{raw}} \in \{\mathcal{F}, \mathcal{R}\}$)¹ originated at

P. Pal is with Brown Mindfulness Center & Department of Epidemiology, Brown University, Providence, RI, 02912 USA e-mail: (prasanta.pal@gmail.com).

P. Pal J. Brewer and V. Taylor are with Brown Mindfulness Center N. Quirós is with LEAD University, Manuscript received Jun 24, 2021;

¹data-source $\mathcal{D}_{\text{raw}}$ is often in the form a data-file ($\mathcal{F}$) or a *data-stream* ($\mathcal{R}$)

Fig. 1: Schematic representation of EEG headcap with 128 $\odot$ projected on a 2D surface using *Biosemi* cap coordinates. Each circle represents a $\odot$ and the number inside it represent the index of the $\odot$ as defined in *Biosemi* manual

some transducers from a set of digital sensors ($\odot$)². Data, collected from *natural-systems* are often contaminated by various artifacts, *background-noise* and extremely deviant outliers [2]–[4] that obfuscate the clarity of the underlying, presumably pristine, nature of the observed system. Noise is generally referred to as a systematic perturbation on observed measurements, whereas outliers or artifacts are considered as ad-hoc interruptions occurring at discrete time points, or cases that are significantly deviant from a signal’s central trends. Typically, outliers are defined by data points deviating by more than a set number of standard deviations (e.g., often 3.0 for normal distribution) from a signal’s central tendency measures (e.g. mean, median, mode etc.). A *real-world* example would be, the sound generated by the ongoing traffic on an otherwise quiet street can be considered as background noise, whereas the sounds emitted by honks at discrete time points would be considered outliers with respect to the otherwise natural state of quietness. By contrast, the constant beeping sound of a cardiac pulse monitor

²sensors and transducers are assumed to be equivalent

Fig. 2: Schematic representation of *EEG* data-points contaminated with outliers overlaid on the scalp geometry. The solid arrowheads represent the Cartesian coordinate system (x , y , z) and the tip of the dashed arrows indicate representative outlier data points. x (y) directions point towards the right (left) ear and z direction towards the top of the head

could be considered as a signal of interest (■), and any interruption of this rhythmic signal would be considered an outlier with respect to the natural rhythmic pulses. As a simple working definition, we'll consider steady state signals over a well defined period of time as ■ and any significant deviation from the central tendency measure as outliers. Outlier detection and curation is a centuries old classic problem, that pervades almost all disciplines of science, art and technology. In art, outliers most often consist of outstanding features that are in sharp contrast to the backdrop of an artistic creation. In science and technology, outliers appear in various contexts of data analysis, machine-learning, fault-tolerance testing and design of robust dynamic systems, such as aerodynamic pressure fluctuation on airplanes. Depending on the exact scenario, the significance of outlier detection and curation has minimal to profound effects on the accuracy of real-world systems' analysis, modeling and predictability. For example, the sound of a passing airplane over a quiet wooded area, however annoying to the ears, is not of any critical significance. On the other hand, a faulty pressure-sensor measurement feeding incorrect data to an automated airplane navigation software (*MCAS*) [5], led to multiple crash events of Boeing 737-MAX aeroplanes [6]–[8]. With the large scale adoption of machine-learning (*ML*) driven data modeling tools, it is crucial to note that, even with the most advanced *ML* tools, a barely simple prediction may completely fail if the training or test data sets are not perturbation free [9]. Also, it costs a lot more computational power to train outlier infused data sets than its relatively pristine counterparts due to the divergence of the convergence time to a stable model [10]–[13]. Ironically, a highly advanced *ML* model would also train the noise and outliers as valid data points. To harness the true strength

of core *ML* algorithms, decontamination of available data sets are of paramount importance [14]. In the common real-world situations, often outlier removal is erroneously equated to data smoothing [15]–[18], using various filters (e.g. Gaussian, Wiener etc.) or fitting of the data using generalized polynomial functions [19]–[22]. Also there are often simplistic assumption (e.g. Gaussian, Poisson etc.) [23]–[25] about the nature of the noise in the data that are often inaccurate in most realistic scenarios. The actual *time-scales* and *length-scales* of the noise models underlying the data, often do not conform to standardized models (e.g. Gaussian etc.). This issue calls for additional *data-discovery on-the-fly* during the data processing operations. Existing techniques, although, often make it visually more appealing by reduction of noise, however, it is at the cost of information loss and blurred clarity. Fitting and smoothing techniques are meaningful only when the data is optimally free from unwanted extreme values. As an example, the modern healthcare system heavily relies on various *medical-imaging* techniques [26], [27]. But, in most realistic scenarios, *medical images* are often plagued by various noise and artifact signals, that require rigorous noise-mitigation as part of pre-processing steps [28]–[30] to make accurate inferences.

On the other hand, real-time stream-based bio-medical applications such as *EKG* [31], [32] pulse or *EEG data-train* monitors [33]–[35], through which clinical decisions are to be made in *real-world* settings while a patient is hooked, it is critically important that the *data-stream* be outlier free in *real-time* to prevent inaccurate diagnostic outcomes on a *moment-by-moment* basis. Occasionally, *data-records* get contaminated at discrete *time-points* by fluctuations arising from various sources e.g. extraneous background noise, sensor-sensor coupling, movement-artifacts, thermal-noise, line-noise as well as electrical-interferences from equipment operating in the same electrical network. Another example includes, multi-dimensional *time-series* data (simultaneous recording of multiple sensors) for which sensors are physically constrained by underlying geometry of the sensor coordinates (e.g. *EEG* electrode placement positions on the scalp's surface). In this case, data contamination can result from coupling to neighboring channels' signal(s), and the extent of contamination can vary depending on sensors' relative position to each other in physical space. Fig. 2 demonstrates few outlier data-points (arrow-tips) clearly forming to a distinct group, when compared to the rest of the data-points.

In real-time bio-medical data processing applications like *bio-feedback*, where subjects are required to monitor or regulate biological signals (e.g. mental state) by taking cue from feedback measures (e.g. neuro-feedback), the efficiency and rigor of outlier curation is of paramount importance. Specifically, if $\mathcal{D}_{\text{out}}$ have not been rigorously de-contaminated from outliers in live clinical experiments involving neuro-feedback, subjects would receive real-time

bio-feedback signals from artifact-ridden signals falsely responding to the wrongly labeled data signatures. Thus, subjects would receive feedback derived from contaminated data and attempt to self-regulate using erroneous feedback measures. To make it even worse, the contaminated data is often orders of magnitude higher than that of pristine data. As an example of a situation that arises in real-time neuro-bio-feedback systems is when a participant undergoes mental training based on one's brain signal measured through sensors strategically placed on the exposed surface of subject's physical body. Often, the data-stream passes through source-localization or similar diagnostic and haptic [36], [37] or *audio-visual* feedback [38], [39] modules. In such situations, the outlier signal may deem to be the actual feedback signal unbeknownst to the subject. In training paradigms involving meditation or relaxation, a natural jaw-clenching, eye-blinking or swallowing may mislead the subject into believing this to be its actual neural signal or being in a state of higher awakening. Even if the source-localization algorithms are highly reliable, the presence of data contamination would induce the participant to train using contaminated information. Human subjective experience under normal conditions, being harmoniously continuous in nature, such disruption would only worsen any perceived outcomes as well as it may potentially raise the ethical question of making someone believe in something that is incorrectly grounded to its perceived origin.

On the other hand, even in the presence of an accurate outlier curation module, the temporal perception of processed data may be entirely inaccurate with respect to the neural signal of interest if the outlier curation process is not efficient enough to make it *real-time* and thus preventing processing delays. In high sampling rate (Ω), *real-time* and *high-density* sensor driven applications, minimization or complete elimination of processing delay is, therefore, an important processing step to ensure that participants receive signals that are temporally synchronized to their real-time physically and mentally grounded experience. In general practice cases of *multi-dimensional time-series* outlier curation problems the focus is often limited to single channel systems where each dimension is treated independently of each other. However, although easy to implement from a computational point of view, such simplistic modeling is likely to be prone to inaccurate curation because, often certain groups of sensors are geometrically, neurologically, dynamically or otherwise correlated. The consideration of geometrical constraints arising from the natural physical space would make the curation algorithm more reliable due to additional physically grounded constraints. As an example, complex systems like *high-density EEG* systems, multiple sensors typically placed on the scalp are utilized to collect *real-time high-density* data. The relative geometrical constraints manifested by the natural shape of the human scalp, should be taken into consideration to make our outlier curation process more robust and

customized.

Since, the nature of many *bio-physical* data e.g. *EEG*, *EKG* etc. are physically rooted both in spatial and temporal domains, we designed the software to address both of these orthogonal aspects of natural *data-sources*. Existing outlier removal methods *PCA* [40]–[42], *ICA* [43], [44] etc. although, well suited for offline tasks and targeted towards certain specialized kinds of data-sets, the available implementations are either not scalable or not efficient enough for adoption in real-time, multi-dimensional, multi-scale noise reduction tasks. For example, *EEG-LAB* [45] is an excellent toolbox for certain niche *EEG-data* processing tasks. However, its dependency on proprietary *MATLAB* software and being written in scripted language is best suited for offline tasks and can not be integrated into embedded devices like fitbit [46], Oura-ring [47] etc.

Moreover, there are often too simplistic assumptions about the nature of the underlying data e.g. linearly varying or under equilibrium conditions which in reality, may have nothing to do with those assumptions. Also, existing methods often remove outlier values at the cost of modification of non-outlier values which in turn leads to loss of information even for perfect data-points. Also such kind of tools should be portable enough to work on a wide range of devices including mobile phones, high-performance-computing, micro-controllers etc. with minimal memory footprint.

Finally, the world of open-source software is a major force behind the exponential growth of data science, *ML*, and scientific computing. Our open-source code initiative would encourage a thriving developer and research community³, that is of the likes of *PCL* [48], *armadillo* [49], *CGAL* [50], *Tensor flow* [51] etc. This initiative would open the gateway to better standardize outlier detection and curation techniques in the context of signal processing of *bio-physical signals*. As result it would help us answer universal questions like, “*What is the standard threshold for reliable EEG (or EKG) signal-fluctuations?*” or “*What is the correct data buffer-size to reliably identify cardiac-arrhythmia in real-time monitoring environments?*” Although, there are many *outlier-detection* standards available for generic signal processing [52]–[54], *source-localization* [55], [56], *audio-visual* processing [57]–[60] contexts, there is no such established standard-practices available for *outlier-detection* in large scale data driven *real-time* and *post-processing* software systems under complex non-trivial operating conditions. This work would help address those issues by creating a globally accessible platform dedicated towards that mission.

I. METHOD

The primary focus of this work is to build a generic *open-source* software tool such that, *outlier-detection* and raw-

³[SOCKScommunity](#)

Fig. 3: Overview of the design principles of *SOCKS* algorithms. Contaminated *time-series data* is taken as an input field having a matrix-like structure in *ASCII* format. The *SOCKS* kernel curates the data either on time domain or spatial domain (or image) to produce filtered curated data while separately storing the outlier data-points as well

data curation for a wide variety of data types are facilitated both offline as well as in *real-time* scenarios. There are two basic steps to achieve this goal. a) *Flagging* (or detection) of the outliers and b) *curation* of the flagged data-points. During regular *data-acquisition* scenarios, the recording is typically done with a specified Ω and in *EEG* like sensors (e.g. array or montage of sensors), representative values of Ω are 256, 512, 1024, 2048 etc. In our case-studies involving *time-series data* (e.g. *EEG*), it is set to $\Omega = 2048$. $\mathcal{D}^{\#}$ is read in a *block-wise* fashion and temporarily stored in a data-container ($\boxtimes$) using the *deque* [61], [62] data-container. For simplicity, a *data-block* is represented as $\#$ and $\mathcal{N}^{\#}$ as the number of instances of $\#$. The size of $\#$, is represented by $\mathcal{N}^{\#_{in}}$, and taken as an input parameter through a *configuration-file* ⁴. In practical situations, the parameter values would typically be set from the domain knowledge (e.g. spatial or temporal scales) of the system or the nature of the specific curation tasks. For example, if we want to curate a long signal train for very short *time-scale* chirps, $\mathcal{N}^{\#_{in}}$ would be only a few sample lengths and comparable to few times the sample-length of the chirp-size. On the other hand, for curation of a *long-epoch*, $\mathcal{N}^{\#_{in}}$ would scale with the length of the epoch.

We read the data from $\mathcal{D}^{\#}$ in a *block-by-block fashion* through the entire $\mathcal{D}^{\#}$. The total number of *data-blocks* ($\Sigma_{\#}$) is approximately proportional to the number of effective samples $\mathcal{N}^{\mathcal{D}^{\#}}$ present in the data-source. For

⁴*config.json* is the container for various parameter keys and vales

Fig. 4: Schematic representation of two consecutive curation cascades $\mathcal{C}_k$ & $\mathcal{C}_{(k+1)}$ applied in sequence

given values of $\mathcal{D}^{\#}$ and $\#^{in}$, the following relationship holds.

$$\Sigma_{\#} = \text{floor}\left(\frac{\mathcal{N}^{\mathcal{D}^{\#}}}{\mathcal{N}^{\#_{in}}}\right) + \min(1, \mathcal{N}^{\mathcal{D}^{\#}} \% \mathcal{N}^{\#_{in}}) \quad (1)$$

where, the k^{th} *data-block* is represented as $\#_{k \in \{1, 2, \dots, \Sigma_{\#}\}}$.

$\mathcal{N}^{\#_k}$ is effectively the interrogation *time-scale* δt_k for $\#_k$ upon which the median and *MAD* parameters calculated. The following relationship holds

$$\delta t_k = \frac{\mathcal{N}^{\#_k}}{\Omega} \quad (2)$$

Typically, an outlier curation routine is performed by choosing a value of δt_k and the curated output is obtained for that *time-scale*. For the purpose of simplicity and brevity we discuss only the temporal aspects of the operation, however, the spatial aspect of it follow a similar logic. A more sophisticated method would be to perform outlier curation at various scales in a sequential order. As a general recommendation, shorter *time-scale* outliers should be removed first followed by relatively longer ones. Often, the system underlying $\mathcal{D}^{\#}$ is continuous in nature and discrete, ad-hoc division of a continuous system into discrete-blocks may introduce cosmetic discontinuity [63], [64] ⁵ of trends in *outcome-measures* of consecutive *data-blocks*. To mitigate this kind of *processing-artifacts*, we introduce forward $\#^f$ and backward $\#^b$ *overlap windows*, *data-blocks*, ⁶ the size of which are usually a percentage (~25%-50%) of $\mathcal{N}^{\#_{in}}$. So, the effective *data-block* length, including the *overlap-windows* in forward and backward directions $\#_k \rightarrow \widehat{\#}_k$

$$\mathcal{N}^{\widehat{\#}_k} \approx \mathcal{N}^{\#_{in}} + \mathcal{N}^{\#^b} + \mathcal{N}^{\#^f} \quad (3)$$

All $\mathcal{N}^{\widehat{\#}_k}$ s are uniform by design choice except, near the start and end of the data stream due to natural nuances around data boundaries (e.g. the number of samples may not be integer multiples of $\#^{in}$).

⁵similar to Gibb's ringing

⁶forward means forward in time or southward in case of an image

Robustness is intrinsically built into the software by incorporating relatively robust central measure, e.g. *median* ($\hat{\mu}$), and *median absolute deviation* ($\hat{\sigma}$ or simply *MAD*) [65]–[68]. These measures are minimally influenced by presence of extreme outliers. We define modified z-score for value x_k as below.

$$\hat{Z}_k = \frac{(x_k - \hat{\mu})}{\hat{\sigma}} \quad (4)$$

In the flagging step, any if for a given data point k , ($abs(Z_k) \geq \lambda$) is satisfied, x_k (or its location) is flagged (or labeled) as an *outlier-candidate* and staged to be curated in the subsequent steps if additional constraints (specific to the curation) are satisfied. Curation is not mandatory⁷ and the program may halt at the flagging stage if necessitated by the user. Then, using non-*NaN*⁸ values we create a linear-interpolation (or any other order) model to *replace the NaN values* (flagged points) by constructing a model from the *data-points* in the neighborhood of the flagged points⁹. As a vanilla case, we take advantage of *armadillo* library's built-in *interp1* function that is optimized to fill in the *flagged values* from the un-flagged, neighborhood data points. *Interp1* does extrapolation of the values if it is outside of the domain on which the model is built.

Optionally, we can extend the default software settings *built-in with static initializers* into additional outlier detection procedures based on other statistical measures like arithmetic mean, harmonic mean, mode or custom criteria specific to the underlying data-curation task based on specific domain constraints. The kernel of the software consists of *multi-threaded data-io* mechanism, coupled with appropriate data type conversions, matrix transformations, flagging, curation and iteration process. Efficient availability of data from the source $\mathcal{D}^{\mathfrak{M}}$ is facilitated by the multi-threaded environment through *lock-based*, *mutex-enabled*, synchronization mechanisms. The above steps can be done recursively for certain number of iterations ($\mathbb{R}$). Once the targeted iterations are finished (or by any other runtime condition), the flagging and curation steps may continue by piping the data from the last curated output as input and potentially with a different *parameter tuple* e.g. $(\lambda, \#^{in}, \mathbb{R})$. We call it cascading with a *parameter tuple*. The software also allows, multiple processing cascades ($\mathcal{C}_k$) to be sequentially staged by cascading *flagging*, *curation* and *iteration* steps in a serial fashion. Fig. 4 describes the steps for a given cascade level and the number of cascades $\mathcal{N}^{\mathcal{C}_{k \in \{1, 2, \dots, p\}}}$ needed for a specific curation objective and can be defined through the *configuration file*.

A. Pre-post-processing and data source structure

ASCII data is read from $\mathcal{D}^{\mathfrak{M}}$ in floating point format. It is reasonable to presume that, the default data arrangement is *column-wise* in the sensor space and *row-wise*

in the temporal space. Each row is an individual record of the uniformly sampled data. Traditionally, the data may contain few meta-data header-lines at the beginning. Depending upon the circumstances, the header information may be utilized for calibration or may be discarded as well. In principle, there may be extraneous columns present in $\mathcal{D}^{\mathfrak{M}}$ than those deemed relevant for processing and may be discarded as well by setting appropriate boundaries of the data through the *configuration-file*. The curation is performed on the effective samples ($\mathcal{N}^{\mathcal{D}^{\mathfrak{M}}}$) that are candidates for actual processing. It is a sub-matrix of the original $\mathcal{D}^{\mathfrak{M}}$. The file or the stream often comes with auxiliary information, time-stamp, serial no. etc. channels and it is important to discard the indices of those channels. Also, often the first few lines of the data file may contain header information (e.g. file type, channel description, Ω etc.) and we discard these lines for actual processing. The zero time stamp starts at the first sample of the actual data after the header. If the input file contains few lines of header information, we skip those lines from processing (see footnote 4).

B. Design principles, data-io, transformations and dependencies

The underlying design principles are, *easy availability* (open-source), *flexibility*, *scalability* (can handle large data from diverse sources), *adaptability* (various input data formats, new filter implementations), *low-latency*, and *robustness* (works even in the presence of extreme values of data). It can be used, either as a *stand-alone* tool for offline-processing or as a plugin to real-time systems through file or streaming interfaces [69]. The flexibility is achieved by parametric curation of $\mathcal{D}^{\mathfrak{M}}$ where, important parameters (e.g., input file, config file etc.) are fed into the software through a combination of command-line options. Additional parameters are taken from a configuration file designed using JSON structure. The command-line options and JSON parsing are done using open-source *cxx-opt*¹⁰ and *json-cpp*¹¹ libraries respectively.

Highly efficient matrix-data processing mechanism facilitated by *armadillo* linear algebra library at the heart of the software. From the *input raw data matrix* ($\mathcal{M}$) it is trivial to create a reference submatrix *sub-matrix* $\mathfrak{M}$ containing only the absolutely necessary chunk of data to be curated by the *SOCKS* kernel. $\mathfrak{M}$ is constructed from $\mathcal{M}$ by taking into account appropriate row and column index ranges. The input data is presumed to be in a columnar manner separated by standard field separators (e.g. space, comma etc.) where each column represents *time-series data* from each individual sensor whereas each row is a single snapshot record of the measurement. As illustrated in Fig. 3, the operations on $\mathfrak{M}$ are divided into two primary categories based on the respective *operational domain* of the data a) time-domain $\mathcal{T}$ b) spatial-domain $\mathcal{S}$. Fig. 2 illustrates

⁷often it is enough to get the outlier points

⁸Not a number

⁹Actual details depend of the dimensionality, interpolation schema or evn keeping the *NaN* values

¹⁰[cxx-opt](#)

¹¹[jsoncpp](#)

the spatial arrangements of *EEG* sensors where the data points are generated from sensors placed on the head-model of standard *EEG* data-acquisition systems. Due to natural *hemi-spherical geometry* of human-head model, any individual sensor or a group of them may produce outlier data but it can be recovered in the context of $\mathcal{S}$, by constructing a *sensor proximity map* from the *inter-geodesic distances* between sensors. In principle, *inter-geodesic distances* can be replaced by any other suitable *distance measure* e.g. inter-sensor dynamic correlation, functional proximity (e.g. how functionally close two sensor locations are) etc. When spatial curation is under consideration, this map is optionally taken as an input from the *configuration-file* (see footnote 4). For the purpose of brevity, we would discuss the $\mathcal{T}$ domain operations only, however, $\mathcal{S}$ domain operations are done in an equivalent fashion except for few additional elements like *sensor proximity map* etc. and described in detail in the software documentation¹². The scalability of processing load depending on available resources was ensured by utilizing a *multi-threaded processing framework*, paired with *block-by-block* data-reading, processing and asynchronous storage protocols. At the initialization stage of the software a thread-pool ($\llbracket \llbracket \llbracket$) of size $\mathcal{N}^{\llbracket \llbracket \llbracket}$ is allocated so that computation can be delegated to various hardware threads when necessary, while avoiding expensive thread-creation and destruction operations. Generally, $\mathcal{N}^{\llbracket \llbracket \llbracket}$ scales with the number of *cpu-cores* of the *host-device* for processing. A run-time *data-block* container $\boxtimes$ internally having the *deque* data structure, is maintained as a container of $\#_k$ s throughout the lifetime of the *data-processing* task¹³, so that it acts as the interface between the $\mathcal{D}^{\boxtimes}$ and the *data-processing-module*¹⁴ while reducing the *data-loading* time from $\mathcal{D}^{\boxtimes}$ by making the *data-block* readily available in the hardware memory¹⁵. Before the *data-processing* operation starts, $\boxtimes$ initialized to be in a filled state $\boxtimes$ of size $\mathcal{N}^{\boxtimes} \geq 3$ whereas, the maximum size of $\mathcal{N}^{\boxtimes}$ is determined by the choice of *memory footprint* of the software as pre-defined in the *configuration-file* (see footnote 4). The front element of $\boxtimes$ is popped for processing while an asynchronous request is made to read the next data block from the $\mathcal{D}^{\boxtimes}$ to be pushed (or en-queued) in partially empty $\boxtimes$. Before the push or pop operation, $\boxtimes$ is locked with a mutex. This way the wait time to read data from the $\mathcal{D}^{\boxtimes}$ is avoided while the processing happens simultaneously on a separate thread of $\llbracket \llbracket \llbracket$. The following schematic sequence¹⁶ describes different states of the *data-block* container from the initial empty $\boxtimes$, followed by complete filling and consecutive *poping* and *pushing* of $\#_k$'s from *data-container* $\boxtimes$ until

all the data in $\mathcal{D}^{\boxtimes}$ is completely exhausted.

The output as well as outlier instances are *asynchronously* stored through an output data sink ($\mathcal{O}$) interface that can optionally be piped to a data file or a streamer *labstreaming layer* (*LSL*)¹⁷. The adaptability to variations of data formats is achieved by first reading each line of the input data as a fresh record in the format of a string. String being one of the most universal data types, any kind of *ASCII* data files can be read in a line by line fashion. Then the data parsing is done by utilizing the data-domain boundaries contained as the actual experimental details of $\mathcal{D}^{\boxtimes}$. As a concrete example, we used *EEG* data samples collected from *Biosemi* [70] systems where, the first column usually have time stamp information, mid columns have sensor data information, and the last few columns may have auxiliary or external trigger information. In most practical scenarios, only a subset of these data columns are relevant and *SOCKS* would parse it as per data-boundaries set through the *config file*. The low-latency and efficiency goal was achieved by using, highly optimized, *low-level*, *multi-threaded*, compiled *C++* language with modern standards ($\text{-std} \geq 11$) that intrinsically supports minimization of expensive data copy through the modern syntax introduced in standard 11. By move semantics [71], [72], reference passing and perfect forwarding [62] principles, the need for expensive data copying was partially if not completely eliminated and every unique piece of data is guaranteed to reside in a single memory location. It is to be noted that, the introduction of *move semantics* and *perfect forwarding*, native language level multi-threading, enabled a very powerful interface to build efficient, close to metal real-time modern softwares. Parallelized *OpenMP-loops* [73], [74] were used in appropriate *code-blocks* to make it even more efficient. Additionally, multi-purpose linear-algebra library *armadillo* [49] has been heavily utilized to facilitate a diverse variety of highly complex but efficient matrix as well as data transformation operations. One big advantage of the *armadillo library* is that, it creates a natural interface to form *sub-matrix views* using references such that any operation on a sub-matrix does not require expensive data copy operation. This has a significant impact on the efficiency of the software because the $\mathcal{D}^{\boxtimes}$ is often either imported from file on the hard drive or from a *live-stream* that in its raw form, belongs to a higher dimensional sensor space than the subspace of the matrix where $\llbracket \llbracket \llbracket$ of interest belongs. Without the reference based *sub-matrix* view, we have to

¹²socks-documentation

¹³all computations are referred to as *data-processing* task

¹⁴consists of flagging, curation and io-operations

¹⁵Reading from data storage is very expensive compared to processing time

¹⁶ $\mathcal{f} \Rightarrow \#_k$ pushed into $\boxtimes$, $\mathcal{f} \Rightarrow \#_k$ popped from $\boxtimes$, $\mathcal{O} \Rightarrow$ processing of $\#_k$

¹⁷labstreaminglayer

perform expensive and memory intensive copy operations. Functional operations/transformations are performed on a reference based sub-matrix structure of the imported $\mathcal{D}_{\text{ref}}$ in the form of matrix. In case of streaming (see footnote 17 for details) data we added the interface to stream the output to outgoing *data-stream* clients. In the current version of the software, we implemented only the outgoing *data-streaming*, identified by the *name* and *type* of the stream. These parameters can be trivially modified in the *configuration-file* as needed by specific situations. The incoming *data-stream* can be easily implemented but we skip it in the current version for the sake of simplicity.

It is to be noted that, output from *SOCKS* kernel can be further processed internally using traditional *auxiliary fileters* (e.g. Gaussian, Poisson etc.). For illustration purpose, we included an implementation of *half-Gaussian* filter that can be used optionally by turning on or off appropriate flags (for details see footnote 12).

Fig. 5: (a) An *EEG* data segment representing a “jaw-clench” event. The event epoch is approximately marked between two vertical dashed lines. (b) A magnified segment of a single channel (A1 of *Biosemi* cap) demonstrating the same *jaw-clench* event to highlight the start and end of the event.

C. Multiscale-modeling

Most, if not all, *time-series* data are intrinsically multiscale in nature [75], [76]. *EEG* or similar *data-sources*, under equilibrium measurement settings, may get intermittently polluted by outliers with varied *time-scales* as illustrated in Fig. 6 a), b) and c). One practical problem is, reliable detection of the various *time-scale* specific outliers, despite significant variations of its temporal length-scales

Fig. 6: Representative *time-series* to demonstrate various event *time-scales* during a controlled *EEG* recording experiment with a human subject. a) Short *time-scale time-series* chirp-like event of *time-scale time-series* τ_1 b) *jaw-clench* event τ_2 c) unspecified signal relaxation event at *time-scale* τ_3

(e.g. τ_1, τ_2 & τ_3). One way to address this problem is to pre-determine a set of *time-scales* ($\tau \in \{\tau_1, \tau_2 \text{ \& } \tau_3\}$) to isolate temporal spans of the outliers. The parameters of the curation task, are appropriately adjusted specific to those *time-scales*, followed by the flagging, curation, iteration and cascading steps. Depending the outcome objectives, users have complete freedom to determine how many *cascades* of the curation are appropriate. This can be performed by sequentially applying the flagging and

curation algorithms for each *time-scale*, followed by piping the output of the most recent *elemental curation task* as demonstrated in Fig. 4 (see footnote 16) to the input of the next curation step.

D. Software validation

Fig. 7: Demonstration of a series of jaw-clench segments for a *time-series* from 128 *EEG*-sensors. The black line represents the (.) and each colored lines represents corresponding curated data. The colored keys represent sensor indices from 128 of those on *Biosemi* head-cap.

1) *Measured data*: Validation is performed on data collected from high density (number of sensors = 128), high Ω *EEG* experiments using *Biosemi* device that include controlled outlier events like jaw-clenches. For the purpose of simplicity we show the examples with jaw-clenches but the software can handle any possible event. Fig. 5 a) shows the onset of a jaw-clench event followed by relaxation and b) highlights one such small segment for a particular sensor *C4* of *Biosemi* cap. Fig. 7 demonstrates the curated as well as the $\mathcal{D}_{\infty}$ containing a jaw-clench event approximately around the middle of the graph. (easymotion-prefix)

2) *Simulated data*: We created a software validation pipeline to emulate contaminated *time-series* data. Synthetic data has been created such that the use cases are intuitive, reproducible and closely resembles the primary outlier features of real *EEG* data. Using this guideline we performed validation on *simulated* data as well as the *measured* data. Fig. 8 demonstrates one such segment of a synthetically generated *data-instance* that was created to validate the software in a controlled fashion.

II. RESULTS

The results are divided into two major parts. a) Synthetic outlier infused data constructed in a controlled manner from pre-defined distribution functions (e.g. normal distribution) followed by *flagging* and *curation* steps. Fig. 8 demonstrates $\mathcal{D}_{\infty}$ as well as recovered data. Since, the data is simulated and the outlier locations are previously known, we verified that the *flagging* steps work with high accuracy as per user expectation and the curated data makes complete intuitive sense when observed carefully (e.g. converging to neighborhood data points), b) experimental data form *EEG* experiments with *human-subjects* obtained with *Biosemi EEG* device¹⁸. Fig. 10 demonstrates that the variability of the

Fig. 8: Recovery plot of a simulation based corrupt time-series. Filled squares (purple) represent simulated corrupt data and circles (green), represents recovered data. The horizontal axis represents time points as sample index

curated-data can be kept under parametric control by appropriate selection of λ . Variabilities of output data contains important information about the system and in practical scenarios, the variability for a certain objective should be set as a system requirement.

Fig. 9: Comparison of raw *time-series* data (black) and curated data (red) for a *jaw-clench* sequence for few representative values of $\#^{in}$ applied in sequence by piping the data from the output immediately preceding a given label e.g., input for b) is output from a) data from $\mathcal{D}_{\infty}$ is used at the start of the processing. a) $\#^{in} = 4$, b) $\#^{in} = 8$, c) $\#^{in} = 16$, d) $\#^{in} = 32$, e) $\#^{in} = 64$, f) $\#^{in} = 128$, g) $\#^{in} = 256$, h) $\#^{in} = 512$. The plots corresponding to capital letter labels *A-H* for each small letter labels *a-h* highlights a single instance of a jaw-clench event. The threshold $\lambda = 0.42$ is kept at the same value for all the plots *a-h*.

Fig. 9 demonstrates results with aggressive level of threshold parameter whereas Fig. 10 shows the same with a liberal value of λ . Fig. 11 is similar as Fig. 10 in parameter space except the curated data is fitted with

¹⁸the details of the device is irrelevant

Fig. 10: Comparison of raw *time-series data* (black) and curated data (red) for a jaw-clench sequence for various values of $\#^{in}$ applied in sequence by piping the data from the output immediately preceding a given label e.g. input for b) is output from a). $\mathcal{D}_{\text{raw}}$ is used as data source for a). The relevant parameter list is given below a) $\#^{in} = 4$, b) $\#^{in} = 8$, c) $\#^{in} = 16$, d) $\#^{in} = 32$, e) $\#^{in} = 64$, f) $\#^{in} = 128$, g) $\#^{in} = 256$, h) $\#^{in} = 512$. The plots corresponding to capital letter labels A-H for each small letter label highlights a single instance of a jaw-clench event. The threshold $\lambda = 3.0$ is kept at the same value for all the plots a-h

3rd degree polynomial. Fig. 11 demonstrates the effect of fitting and smoothing on the top of the curation steps. Fitting curated data with appropriate order polynomials preserves the local curvature of the data trends and that is often a clearer representations of the underlying system than the raw data itself even after outlier curation.

III. DISCUSSION, LIMITATIONS, CONCLUSION & FUTURE DIRECTION

We developed a low-latency, robust, open-source, tool to curate large volumes of *time-series data* without any practical limitations on the size of the input data source¹⁹. After operating on the *raw-data* followed by *curated-data* in an *iterative and cascaded* fashion, the output is separately stored in curated and outlier signal files for each iteration cycle within a given cascade.

The primary results are demonstrated in sec. II in Figs. 8, 10, 9, 10, 10 & 11 highlighting various curation-capabilities of the software. The actual details of the parameters are described in the figure captions. The key observation is that, the curated output can be parametrically controlled so that the tool can be adapted to a wide variety of real-world situations. The simulated data provides us with a basic pipeline for controlled validation of the software.

SOCKS is designed to be highly efficient, close to metal,

¹⁹since, at a given point of time, only a small chunk of the source data is loaded in the memory

Fig. 11: Comparison of raw *time-series data* (black) and curated data (red) for a jaw-clench sequence for various values of $\#^{in}$ applied in sequence by piping the data from the output immediately preceding a given label e.g. input for b) is output from a). Raw input data is used as data source for a) $\#^{in} = 4$, b) $\#^{in} = 8$, c) $\#^{in} = 16$, d) $\#^{in} = 32$, e) $\#^{in} = 64$, f) $\#^{in} = 128$, g) $\#^{in} = 256$, h) $\#^{in} = 512$. The plots corresponding to capital letter labels A-H highlights a single instance of a jaw-clench event. The threshold $\lambda = 3.0$ is kept at the same value for all the plots a-h. The data is fitted with a polynomial of degree 3 after the default curation step.

flexible and robust enough to be utilized in various data acquisition settings even in the presence of unprecedented levels of extreme outlier values. The primary *C++* base class *SOCKS* is structured such a way that, static variables and functions are designated for elements that preferentially need to be initialized along with the class object instantiation. To keep it simple, we put special emphasis on a single depth the class hierarchy. Also, class instances are not necessary for static functions so that we can call those elements through the *class-scope* without an actual instantiated *class-object*.

Although, the main context for building *SOCKS* was motivated by long, *high-dimensional, time-series data* from *EEG*, but the scope is far more expansive and can be easily extended to matrix or *image-processing* problems as well. It can be safely adapted to any generic *time-series* scenarios (*real-time* or *post-processing*) by adjusting appropriate configurational parameters. New family of filters can be constructed using the *flag, curate, iterate and cascade* protocols using the same *built-in data-io* facility. Appropriate usage of `|||` ensures that all the available hardware-threads of the *host device* are optimally utilized. Any parallelizable process is distributed on the independent cores of the *microprocessor* by virtue of the `|||` facility. Additionally, by utilizing *labstreaminglayer* (see footnote 17) in the streaming protocol, the process can be cloned to multiple hosts in parallel through *TCP-IP* [77] network protocol (e.g. WIFI,

ethernet etc.). For simplicity, we implemented streaming only for the curated *output-stream* but the *input-stream* can be trivially implemented in a similar fashion.

One of the known limitation is the pontential behavior of the software under sharp-edge *data-boundaries* (e.g. Heaviside function like edges in the data). Edge-preservation between *data feature-boundaries* is often, an important criteria for data processing outcomes. We'll address this aspect in future versions of the software. In the currently published version, we assume the data sources to be without sharp edges. Another known limitation is that, for image inputs, the software operates only on grayscale color planes, e.g. either the source itself is grayscale or the color image is converted to grayscale so that *SOCKS* can operate on each color plane independently. There can potentially be many ways of addressing the color image processing issues which is beyond the scope of the current article. To conclude, in the currently published version of the software, we have concretely demonstrated the median based *MAD-filter* and additional variants (e.g. mean, mode etc.) would be developed as an extension in future versions. The closest match with *MAD-filter* and popular *image-processing filters* is the median filter [68]. However, median filtes has some serious limitations like modifying even the un-corrupted data points. *SOCKS* is designed to be very generic in nature and can be used under diverse signal-prcoessing scenarios.

In a *time-series* segment, often valuable information is encoded in various orthogonal *frequency-bands*. For any kind of precise *time-series analysis*, bounds for the specific *frequency-band* must be specified. In the current version, we do not address the *frequency-band* specific implementation although, we are aware of the importance of it but beyond the scope of the current implementation goals. Future versions would address custom bandpass filters versions of the software using open-source DSPFilter²⁰ repository.

The software can be further extended for cloud support as well as support for non- *ASCII* file types (e.g. binary files like .edf, .bdf formats). In practice, a segment containing a lot of outliers in a *time-series data* does not necessarily mean that the segment is not *information-rich*. It may as well, contain valuable intrinsic information (sound of a honk) or effects of external influences on steady-state behavior of the system (sudden stop of traffic at red signal). Part of the design goal of the software is perform the *flagging* and *curation* steps sequentially. So, once we flag the outliers the curation is optional and we can simply store the flagged data records without any further curation. Although, there is some apparent similarity between outlier removal and *FFT* based *high-frequency* [78], [79] removal procedure but this similarity is only superficial. Any *FFT* based algorithm implicitly assume periodicity of underlying data and generally very sensitive to extreme outlier (e.g. Gibbs ringing

effect) however *SOCKS*, algorithm is grounded on feature discoverability *on-the-fly* and almost agnostic to any traditional noise models.

In future we'll add a separate repository of *SOCKS* compatible filters to create a *filter-bank* with most generic types of filter to address a wide variety of outlier scenarios (e.g. distributions of various categories). Since, the *source-coude* is written in *C++* language using modern standard (-std $\geq$ 11), it is extremely flexible to extend and customize the utilities under various *operating-systems* environments²¹, including *embedded-systems*, *HPC*, *mobile-OS*, *data-servers* as well as *desktop-applications*. Optionally, the matrix processing part of the computation can be delegated to *GPUs* [80]–[83] using *OpenCL*, *CUDA etc. frameworks* [84]–[86] to further augment the processing efficiency. Additional auxiliary filters (e.g. half Gaussian filter or any of the traditional filters) may optionally be applied further on the *curated-data* either *on-the-fly* or post storage on hard-drives.

In future, a user friendly *GUI* interface would be added as well as connectivity to databases, diverse variety of *data-streams* [87], [88] and direct interfaces with various devices, as well as integration with reactive libraries [89] for better intuitive *io-design*.

Although, the development of the software was inspired by *large-scale EEG* recordings, but the software is not limited to *EEG* data. The principles of time series processing are applicable to any other commonly available *time-series data* like data (e.g. air pressure fluctuations, audio signal processing, video processing, location tracking etc.) as long as the data is suitably transformed into *ASCII time-series* or metrices.

Documentation of the software was generated using the *Doxygen* [90] documentation tool and the documentation is hosted at the *project-repository* (see footnote 12). Finally, the *open-source* initiative would encourage a thriving developer and research community, and drive new innovations in the space of standardizing outlier detection and curation techniques to address questions like *what is the standard threshold for EEG outliers to detect certain category of neurological signals (e.g. alpha wave) or what is the definitive outlier value to know if the pressure-sensor measurement of a moving aeroplane is no-longer reliable?* Although, there are many standards available for *signal-processing*, *source-localization*, communications etc. but there is no such standardization for *outlier-detection* in *large-scale* data generation and storage systems. Building a forum towards the goal of *standardization of outliers* for various physical systems and normalization of its scales, under diverse operating conditions, would potentially be a revolutionary milestone to reach towards enhancing clarity of real-world data.

Philosophically, outlier curation is deeply linked to our perception of the *physical-world*. Our visual or auditory senses are heavily dependent on visual or audio signal

²⁰<https://github.com/vinniefalco/DSPFilters>

²¹a guarantee provided by the language itself!

intensities. Often, our *audio-visual* perception about common objects (e.g. image, audio) are strongly influenced by high-intensity spectrum of the signal. By curating those data points, containing high intensity signals, our perception would be relatively free from cognitive biases. The potential applications of *SOCKS* tool would span nearly all disciplines of *data-driven* systems with or without outliers (definition of outlier is often arbitrary). We strongly believe that, *SOCKS* would revolutionize the way we look at *real-world data*.

ACKNOWLEDGEMENT

The authors acknowledge logistical support from the Brown Mindfulness Center, Department of Epidemiology and School of Public Health, Brown University. We are thankful for financial support from Fetzer Memorial Trust Foundation. Part of this research was conducted using computational resources and services at the Center for Computation and Visualization, Brown University. PP personally acknowledges Megan Ranney, MD and Eric Loucks, PhD for helpful discussions and institutional support.

Prasanta Pal Prasanta Pal, PhD is an investigator of Epidemiology at the Brown Mindfulness Center, Brown School of Public Health. An applied physicist by training from Yale University School of Engineering, USA & IIT Kharagpur, India, Dr. Pal has been the chief director, designer and developer of several neuro-feedback, data-science and mobile-application technologies directed towards human health and wellbeing. He is closely involved with bio-medical engineering research

through first principle, minimal assumption methodos. Dr. Pal is deeply involved in building a suite of fundamental data science and medical imaging technologies to accelerate the field of data-driven diagnostic and healthcare interventions. Dr. Pal's work in medical technology is driven by the philosophy that mind and body works as one integrated system in a complimentary fashion. Holistic health outcomes are possible only when they are combined together through modern technology interfaces. He developed several technologies like MindView, MindScope etc. to help enhance various dimensions of human mind in a data driven, evidence based fashion. He serves in the editorial board of several biomedical engineering journals.

Remko Van Lutterveld Remko van Lutterveld received M.Sc. degrees in Cognitive Neuroscience and Fundamental Biomedical Science from Utrecht University, the Netherlands, and in 2012, his PhD at the faculty of medicine from Utrecht University. He held several positions in academia afterwards, including at the Center for Mindfulness of the University of Massachusetts from 2014 – 2018 and the Mindfulness Center of Brown University, Rhode Island, from 2018 – 2019. He currently holds

positions as senior research scientist at the Brain Research and Innovation Centre of the mental health department of the Dutch Ministry of Defence and investigator-teacher at the Department of Psychiatry of the University Medical Center, Utrecht, the Netherlands. His interests revolve around mental health, meditation and mindfulness. He has been awarded funding from NIH and NARSAD, and serves as a Review Editor for the journal *Frontiers in Psychiatry*.

Veronique Taylor Veronique Taylor completed her MSc. and obtained her PhD at University of Montreal in 2017 in cognitive neuroscience. The specific focus of her graduate work was investigating the neural and physiological bases of mindfulness and emotion regulation. She then held several academic positions, such as at Bishop's University (Psychology Department) where she was involved in teaching and conducted independent research.

She currently works as a postdoctoral research associate at the Mindfulness Center at Brown University, investigating the neural bases of mindfulness as well as its relationship with reward learning to treat addiction-related behaviors. She has been awarded research funding from several sources, including the Mind and Life Institute and from several Canadian research funding agencies.

Nancy Quirós Nancy Quiros obtained her PhD in Atomic Physics at the University of Nevada Reno (UNR). At Weinstein Lab in UNR, she studied the equilibrium thermodynamic properties of the van der Waals molecule Titanium-Helium (TiHe) at temperatures of 1K. Then she obtained a MSc in Digital Currency at the University of Nicosia (UNIC) and investigated the exchanges of Fiat money and Cryptocurrency. Currently, she teaches Cryptocurrency and Blockchain Technology at Lead University, Costa Rica. Her interests are analyzing the impact of emergent technologies in society, philosophy of science, mind and consciousness.

at Lead University, Costa Rica. Her interests are analyzing the impact of emergent technologies in society, philosophy of science, mind and consciousness.

Judson A. Brewer Jud Brewer MD PhD is the Director of Research and Innovation at the Mindfulness Center and associate professor in Behavioral and Social Sciences at the School of Public Health and Psychiatry at the Medical School at Brown University. He also is a research affiliate at MIT. A psychiatrist and internationally known expert in mindfulness training for addictions, Brewer has developed and tested novel mindfulness programs for behavior change, including both in-person and

app-based treatments for smoking, emotional eating, and anxiety. He has also studied the underlying neural mechanisms of mindfulness using standard and real-time fMRI, and source-estimated EEG, and is currently translating these findings into clinical use. He is the author of *The Craving Mind: from cigarettes to smartphones to love, why we get hooked and how we can break bad habits* (New Haven: Yale University Press, 2017) and the New York Times best-seller, *Unwinding Anxiety: New Science Shows How to Break the Cycles of Worry and Fear to Heal Your Mind* (Avery/Penguin Random House, 2021). Follow him on twitter @judbrewer.

REFERENCES

- [1] Y. N. Harari, *21 Lessons for the 21st Century*. Random House, 2018.
- [2] I. Ben-Gal, "Outlier detection," in *Data mining and knowledge discovery handbook*, pp. 131–146, Springer, 2005.
- [3] C. C. Aggarwal, "Outlier analysis," in *Data mining*, pp. 237–263, Springer, 2015.
- [4] C. C. Aggarwal and P. S. Yu, "Outlier detection for high dimensional data," in *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pp. 37–46, 2001.
- [5] M. Tran, "Midair collision and avoidance system (mcas)," Aug. 21 2001. US Patent 6,278,396.
- [6] P. Johnston and R. Harris, "The boeing 737 max saga: lessons for software organizations," *Software Quality Professional*, vol. 21, no. 3, pp. 4–12, 2019.
- [7] A. Pasztor, A. Tangel, R. Wall, and A. Sider, "How boeing's 737 max failed," *The Wall Street Journal*, 2019.
- [8] G. Travis, "How the boeing 737 max disaster looks to a software developer," *IEEE Spectrum*, vol. 18, 2019.
- [9] S. Gupta and A. Gupta, "Dealing with noise problem in machine learning data-sets: A systematic review," *Procedia Computer Science*, vol. 161, pp. 466–474, 2019. The Fifth Information Systems International Conference, 23-24 July 2019, Surabaya, Indonesia.
- [10] W. Li, W. Mo, X. Zhang, J. J. Squiers, Y. Lu, E. W. Sellke, W. Fan, J. M. DiMaio, and J. E. Thatcher, "Outlier detection and removal improves accuracy of machine learning approach to multispectral burn diagnostic imaging," *Journal of biomedical optics*, vol. 20, no. 12, p. 121305, 2015.
- [11] H. J. Escalante, "A comparison of outlier detection algorithms for machine learning," in *Proceedings of the International Conference on Communications in Computing*, pp. 228–237, 2005.
- [12] K. Zhang and M. Luo, "Outlier-robust extreme learning machine for regression problems," *Neurocomputing*, vol. 151, pp. 1519–1527, 2015.
- [13] P. Jeatrakul, K. W. Wong, and C. C. Fung, "Data cleaning for classification using misclassification analysis," *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 14, no. 3, pp. 297–302, 2010.
- [14] W. Siqi, L. Qiang, G. Xifeng, Z. En, and Y. Jianping, "Fast and unsupervised outlier removal by recurrent adaptive reconstruction extreme learning machine," *International Journal of Machine Learning and Cybernetics*, vol. 10, no. 12, pp. 3539–3556, 2019.
- [15] J. Meditch, "A survey of data smoothing for linear and nonlinear dynamic systems," *Automatica*, vol. 9, no. 2, pp. 151–162, 1973.
- [16] M. Tanaka, "Fuzzy data processing method and data smoothing filter," Mar. 14 1995. US Patent 5,398,303.
- [17] J. Kaiser and W. Reed, "Data smoothing using low-pass digital filters," *Review of Scientific Instruments*, vol. 48, no. 11, pp. 1447–1457, 1977.
- [18] D. G. Denison, "Applied smoothing techniques for data analysis: the kernel approach with s-plus illustrations," 1998.
- [19] P. C. Hansen, V. Pereyra, and G. Scherer, *Least squares data fitting with applications*. JHU Press, 2013.
- [20] T. Strutz, "Data fitting and uncertainty," *A practical introduction to weighted least squares and beyond*. Vieweg+ Teubner, 2010.
- [21] G. Kemmer and S. Keller, "Nonlinear least-squares data fitting in excel spreadsheets," *Nature protocols*, vol. 5, no. 2, p. 267, 2010.
- [22] K. Schittkowski, *Numerical data fitting in dynamical systems: a practical introduction with applications and software*, vol. 77. Springer Science & Business Media, 2013.
- [23] I. Rodrigues, J. Sanches, and J. Bioucas-Dias, "Denoising of medical images corrupted by poisson noise," in *2008 15th IEEE International Conference on Image Processing*, pp. 1756–1759, IEEE, 2008.
- [24] T. Le, R. Chartrand, and T. J. Asaki, "A variational approach to reconstructing images corrupted by poisson noise," *Journal of mathematical imaging and vision*, vol. 27, no. 3, pp. 257–263, 2007.
- [25] C. Boncelet, "Image noise models," in *The essential guide to image processing*, pp. 143–167, Elsevier, 2009.
- [26] J. A. Sáez, B. Krawczyk, and M. Woźniak, "On the influence of class noise in medical data classification: Treatment using noise filtering methods," *Applied Artificial Intelligence*, vol. 30, no. 6, pp. 590–609, 2016.
- [27] C. S. K. Dash, S. Dehuri, S.-B. Cho, and G.-N. Wang, "Towards crafting a smooth and accurate functional link artificial neural networks based on differential evolution and feature selection for noisy database," *International Journal of Computational Intelligence Systems*, vol. 8, no. 3, pp. 539–552, 2015.
- [28] D. Gamberger, N. Lavrac, and S. Dzeroski, "Noise detection and elimination in data preprocessing: experiments in medical domains," *Applied Artificial Intelligence*, vol. 14, no. 2, pp. 205–223, 2000.
- [29] U. Rebbapragada and C. E. Brodley, "Class noise mitigation through instance weighting," in *European conference on machine learning*, pp. 708–715, Springer, 2007.
- [30] X. Zhu, X. Wu, and Q. Chen, "Eliminating class noise in large datasets," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 920–927, 2003.
- [31] T. Y.-C. Lo and Y. S. Tsai, "Ekg based heart rate monitor with digital filter and enhancement signal processor," Mar. 2 1999. US Patent 5,876,350.
- [32] R. J. McLaughlin and W. J. Rogozinski, "Ekg cable monitoring system," Nov. 6 1979. US Patent 4,173,221.
- [33] J. Claassen, S. Mayer, R. Kowalski, R. Emerson, and L. Hirsch, "Detection of electrographic seizures with continuous eeg monitoring in critically ill patients," *Neurology*, vol. 62, no. 10, pp. 1743–1748, 2004.
- [34] K. G. Jordan, "Emergency eeg and continuous eeg monitoring in acute ischemic stroke," *Journal of clinical neurophysiology*, vol. 21, no. 5, pp. 341–352, 2004.
- [35] M. L. Scheuer, "Continuous eeg monitoring in the intensive care unit," *Epilepsia*, vol. 43, pp. 114–127, 2002.
- [36] M. D. Graham and C. J. Ulrich, "Biological sensing with haptic feedback," Apr. 2 2009. US Patent App. 11/862,639.
- [37] J. E. Colgate, M. Peshkin, J. Glassmire, and L. Winfield, "Haptic device with indirect haptic feedback," Mar. 26 2013. US Patent 8,405,618.
- [38] R. Van Lutterveld, S. D. Houlihan, P. Pal, M. D. Sacchet, C. McFarlane-Blake, P. R. Patel, J. S. Sullivan, A. Ossaditchi, S. Druker, C. Bauer, *et al.*, "Source-space eeg neurofeedback links subjective experience with brain activity during effortless awareness meditation," *NeuroImage*, vol. 151, pp. 117–127, 2017.
- [39] J. A. Brewer, P. D. Worhunsky, J. R. Gray, Y.-Y. Tang, J. Weber, and H. Kober, "Meditation experience is associated with differences in default mode network activity and connectivity," *Proceedings of the National Academy of Sciences*, vol. 108, no. 50, pp. 20254–20259, 2011.
- [40] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.
- [41] B. Schölkopf, A. Smola, and K.-R. Müller, "Kernel principal component analysis," in *International conference on artificial neural networks*, pp. 583–588, Springer, 1997.
- [42] R. Vidal, Y. Ma, and S. Sastry, "Generalized principal component analysis (gpca)," *IEEE transactions on pattern analysis and machine intelligence*, vol. 27, no. 12, pp. 1945–1959, 2005.
- [43] A. Hyvärinen and E. Oja, "Independent component analysis: algorithms and applications," *Neural networks*, vol. 13, no. 4-5, pp. 411–430, 2000.
- [44] F. R. Bach and M. I. Jordan, "Kernel independent component analysis," *Journal of machine learning research*, vol. 3, no. Jul, pp. 1–48, 2002.
- [45] A. Delorme and S. Makeig, "Eeglab: an open source toolbox for analysis of single-trial eeg dynamics including independent component analysis," *Journal of neuroscience methods*, vol. 134, no. 1, pp. 9–21, 2004.
- [46] L. A. Cadmus-Bertram, B. H. Marcus, R. E. Patterson, B. A. Parker, and B. L. Morey, "Randomized trial of a fitbit-based physical activity intervention for women," *American journal of preventive medicine*, vol. 49, no. 3, pp. 414–418, 2015.
- [47] H. Koskimäki, H. Kinnunen, T. Kurppa, and J. Röning, "How do we sleep: a case study of sleep duration and quality using data from oura ring," in *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on*

- Pervasive and Ubiquitous Computing and Wearable Computers*, pp. 714–717, 2018.
- [48] R. B. Rusu and S. Cousins, “3d is here: Point cloud library (pcl),” in *2011 IEEE international conference on robotics and automation*, pp. 1–4, IEEE, 2011.
 - [49] C. Sanderson and R. Curtin, “Armadillo: a template-based c++ library for linear algebra,” *Journal of Open Source Software*, vol. 1, no. 2, p. 26, 2016.
 - [50] C. Ggal, “Computational geometry algorithms library,” 2008.
 - [51] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, “Tensorflow: A system for large-scale machine learning,” in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pp. 265–283, 2016.
 - [52] D. Yu, G. Sheikholeslami, and A. Zhang, “Findout: finding outliers in very large datasets,” *Knowledge and information Systems*, vol. 4, no. 4, pp. 387–412, 2002.
 - [53] E. M. Knorr and R. T. Ng, “A unified notion of outliers: Properties and computation,” in *KDD*, vol. 97, pp. 219–222, 1997.
 - [54] K. Worden, G. Manson, and N. R. Fieller, “Damage detection using outlier analysis,” *Journal of Sound and Vibration*, vol. 229, no. 3, pp. 647–667, 2000.
 - [55] J. C. Chen, K. Yao, and R. E. Hudson, “Source localization and beamforming,” *IEEE Signal Processing Magazine*, vol. 19, no. 2, pp. 30–39, 2002.
 - [56] A. Beck, P. Stoica, and J. Li, “Exact and approximate solutions of source localization problems,” *IEEE Transactions on signal processing*, vol. 56, no. 5, pp. 1770–1778, 2008.
 - [57] R. Radhakrishnan, A. Divakaran, and A. Smaragdis, “Audio analysis for surveillance applications,” in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, 2005.*, pp. 158–161, IEEE, 2005.
 - [58] S. Brodinova, T. Ortner, P. Filzmoser, M. Zaharieva, and C. Breiteneder, “Evaluation of robust pca for supervised audio outlier detection,” in *Proceeding of 22nd international conference on computational statistics (COMPSTAT)*, 2015.
 - [59] S.-L. Lu, “Noise reduction system using multi-frame motion estimation, outlier rejection and trajectory correction,” Nov. 1 1994. US Patent 5,361,105.
 - [60] D. Hasler, L. Sbaiz, S. Susstrunk, and M. Vetterli, “Outlier modeling in image matching,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 3, pp. 301–315, 2003.
 - [61] B. Stroustrup, *The C++ programming language*. Pearson Education India, 2000.
 - [62] S. Meyers, *Effective modern C++: 42 specific ways to improve your use of C++ 11 and C++ 14.* O’Reilly Media, Inc., 2014.
 - [63] D. Gottlieb and C.-W. Shu, “On the gibbs phenomenon and its resolution,” *SIAM review*, vol. 39, no. 4, pp. 644–668, 1997.
 - [64] A. J. Jerri, *The Gibbs phenomenon in Fourier analysis, splines and wavelet approximations*, vol. 446. Springer Science & Business Media, 2013.
 - [65] B. Brown, “Statistical uses of the spatial median,” *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 45, no. 1, pp. 25–30, 1983.
 - [66] E. Ataman, V. Aatre, and K. Wong, “Some statistical properties of median filters,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 29, no. 5, pp. 1073–1075, 1981.
 - [67] K. K. V. Toh, H. Ibrahim, and M. N. Mahyuddin, “Salt-and-pepper noise detection and reduction using fuzzy switching median filter,” *IEEE Transactions on Consumer Electronics*, vol. 54, no. 4, pp. 1956–1961, 2008.
 - [68] H. Ibrahim, N. S. P. Kong, and T. F. Ng, “Simple adaptive median filter for the removal of impulse noise from highly corrupted images,” *IEEE Transactions on Consumer Electronics*, vol. 54, no. 4, pp. 1920–1927, 2008.
 - [69] C. Kothe, “Lab streaming layer (lsl),” <https://github.com/scn/labstreaminglayer>. Accessed on October, vol. 26, p. 2015, 2014.
 - [70] B. BioSemi, “Biosemi activetwo.[eeg system],” *Amsterdam: BioSemi*, 2011.
 - [71] A. Baráth and Z. Porkoláb, “Automatic checking of the usage of the c++ move semantics,” *Acta Cybernetica*, vol. 22, no. 1, pp. 5–20, 2015.
 - [72] N. M. Josuttis, *The C++ standard library: a tutorial and reference*. Addison-Wesley, 2012.
 - [73] R. Chandra, L. Dagum, D. Kohr, R. Menon, D. Maydan, and J. McDonald, *Parallel programming in OpenMP*. Morgan kaufmann, 2001.
 - [74] E. Ayguadé, N. Coptý, A. Duran, J. Hoeflinger, Y. Lin, F. Massaioli, X. Teruel, P. Unnikrishnan, and G. Zhang, “The design of openmp tasks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 3, pp. 404–418, 2008.
 - [75] M. A. Ferreira, A. Marco, and H. K. Lee, *Multiscale modeling: a Bayesian perspective*. Springer Science & Business Media, 2007.
 - [76] M. Feig, J. Karanicolas, and C. L. Brooks III, “Mmts tool set: enhanced sampling and multiscale modeling methods for applications in structural biology,” *Journal of Molecular Graphics and Modelling*, vol. 22, no. 5, pp. 377–395, 2004.
 - [77] W. R. Stevens, *TCP/IP illustrated vol. I: the protocols*. Pearson Education India, 1993.
 - [78] J. Wang and D. C. Liu, “2-d fft for periodic noise removal on brain image & &,” in *2010 4th International Conference on Bioinformatics and Biomedical Engineering*, pp. 1–4, IEEE, 2010.
 - [79] F. Shirbani and S. K. Setarehdan, “Ecg power line interference removal using combination of fft and adaptive non-linear noise estimator,” in *2013 21st Iranian Conference on Electrical Engineering (ICEE)*, pp. 1–5, IEEE, 2013.
 - [80] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, “Gpu computing,” *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.
 - [81] J. Nickolls and W. J. Dally, “The gpu computing era,” *IEEE micro*, vol. 30, no. 2, pp. 56–69, 2010.
 - [82] J. Sanders and E. Kandrot, *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.
 - [83] B. Daga, A. Bhute, and A. Ghatol, “Implementation of parallel image processing using nvidia gpu framework,” in *International Conference on Advances in Computing, Communication and Control*, pp. 457–464, Springer, 2011.
 - [84] J. E. Stone, D. Gohara, and G. Shi, “Opencl: A parallel programming standard for heterogeneous computing systems,” *Computing in science & engineering*, vol. 12, no. 3, p. 66, 2010.
 - [85] K. Karimi, N. G. Dickson, and F. Hamze, “A performance comparison of cuda and opencl,” *arXiv preprint arXiv:1005.2581*, 2010.
 - [86] M. Garland, S. Le Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang, and V. Volkov, “Parallel computing experiences with cuda,” *IEEE micro*, vol. 28, no. 4, pp. 13–27, 2008.
 - [87] M. Edenhill, “librdkafka—the apache kafka c/c++ client library,” *Software repository*. July, 2018.
 - [88] R. Rai, *Socket. IO Real-time Web Application Development*. Packt Publishing Ltd, 2013.
 - [89] P. Pai and P. Abraham, *C++ Reactive Programming: Design concurrent and asynchronous applications using the RxCpp library and Modern C++ 17*. Packt Publishing Ltd, 2018.
 - [90] D. Van Heesch, “Doxygen: Source code documentation generator tool,” [URL: http://www.doxygen.org](http://www.doxygen.org), 2008.
 - [91] K. Ito and K. Xiong, “Gaussian filters for nonlinear filtering problems,” *IEEE transactions on automatic control*, vol. 45, no. 5, pp. 910–927, 2000.
 - [92] M. A. King, P. W. Doherty, R. B. Schwinger, and B. C. Penney, “A wiener filter for nuclear medicine images,” *Medical physics*, vol. 10, no. 6, pp. 876–880, 1983.
 - [93] S. Kumar, “India spots fallen moon craft, fails to make radio contact,” 2019.
 - [94] A. Rovira, M. P. Wattjes, M. Tintore, C. Tur, T. A. Yousry, M. P. Sormani, N. De Stefano, M. Filippi, C. Auger, M. A. Rocca, *et al.*, “Magnims consensus guidelines on the use of mri in multiple sclerosis-clinical implementation in the diagnostic process (vol 11, pg 471, 2015),” *NATURE REVIEWS NEUROLOGY*, vol. 11, no. 8, 2015.
 - [95] O. Commowick, A. Istace, M. Kain, B. Laurent, F. Leray, M. Simon, S. C. Pop, P. Girard, R. Ameli, J.-C. Ferré, *et al.*, “Objective evaluation of multiple sclerosis lesion segmentation using a data management and processing infrastructure,” *Scientific reports*, vol. 8, no. 1, pp. 1–17, 2018.

APPENDIX A CASE STUDIES

As stated in sec. III, *SOCKS* can be used in multiple *data-contexts* beyond *EEG time-series data* including *image-processing* where *raw-image* is transformed into a *data-matrix* of grayscale *pixel-intensities*. In the context of *image-processing* it would be a very effective tool to perform both *flagging* and *curation* steps through iteration and cascading by choosing appropriate *kernel-radius* ($\mathbb{K}$)²² and λ -values. The details of *how it works* are available in the software documentation and beyond the scope this article. We shall demonstrate the *image-processing* capabilities of *SOCKS* through some *case-studies*. More diverse types of case-studies are available on *SOCKS* community website (see footnote 3) and future updates would be made available here.

A. Image processing

The basic algorithm described for *time-series data*, is easily adapted to *image-processing* problems for filtering [67], [68], [91], [92] and segmentation tasks. All that is needed is to transform a raw image into data-matrix using *OpenCV* based *imread* function followed by conversion to *armadillo* matrix. *armadillo* library has a very advanced *data-io* facility.

Fig. 12: Recovery of underlying geo-spatial features of *Vikram Lander* crash landing site and identification of potential fragments. a) Original image obtained by NASA at the Vikram crash site. Curation performed using threshold values of magnitude b) moderate c) low d) variable (low followed by high)

1) *Recovery of the underlying geo-texture of Vikram Mars Lander likely crash location:* In this example we identify the possible pixel points of India's lost lunar rover Vikram [93] from NASA images. We also give an estimate of the underlying terrain as demonstrated in Fig. 13. The impact of various threshold levels are tested. At very relatively higher threshold values, it is expected to show

Fig. 13: Recovery of Vikram Lander crash landing site and identification of potential fragments. a) Original image obtained by NASA at the Vikram crash site, b) using moderate threshold, c) using high threshold, d) using two separate threshold (high followed by low) values in sequence

a smoother underlying surface however at lower threshold values, more feature details are preserved although at the cost of higher level of variation. We used the curation algorithm in a recursive fashion, such that the output from a previous iteration is fed into the next iteration. After few iterations, it is observed that the plot converges to a stable one, with the number of outliers in each iteration steps to be greatly reduced from those at the start of the iteration.

Fig. 14: Segmentation of multiple sclerosis lesions using extended image processing module using recursions. Three regions of interests are highlighted for visual cues. A) Original image from Fig. 1d of ref. [94], B) curated regions where lesions were detected, C) segmented lesion.

2) *Segmentation of Multiple Sclerosis lesions:* Lesions observed in a multiple-sclerosis patient's MRI images is one of the neural bio-markers of the progress and recovery of the disease. Finding the exact coordinates of the lesions is a very important [95] but challenging task. We applied *SOCKS* kernel to recover the coordinates of the lesions along with an estimate of the underlying anatomic features with the hypothetical absence of the lesions. Fig. 14 demonstrates through 3 highlighted regions that the underlying regions with the lesions are appropriately reconstructed (as apparent visible) and the identified lesions are demonstrated in Fig. 14 C. The actual volume and span of the segmented lesions can be moderated through the threshold parameters as well as $\mathbb{K}$ and $\mathcal{N}^e$ ²³. This way we get access to various layers of the segmented regions.

²² $\mathbb{K}$ is equivalent to $\#^{in}$ for one dimensional case

²³This depends on the objective outcome-measure