Event Clustering & Event Series Characterization on Expected Frequency

Conrad M Albrecht, Marcus Freitag, Theodore G van Kessel, Siyuan Lu, Hendrik F Hamann

Physical Analytics
TJ Watson Research Center, IBM Research
Yorktown Heights, NY, U.S.A.
{*cmalbrec,mfreitag,tvk,lus,hendrikh*}*@us.ibm.com*

***Abstract*—We present an efficient clustering algorithm applicable to one-dimensional data such as e.g. a series of timestamps. Given an expected frequency ΔT^{-1}, we introduce an $\mathcal{O}(N)$–efficient method of characterizing N events represented by an ordered series of timestamps $t_1, t_2, \ldots, t_N$. In practice, the method proves useful to e.g. identify time intervals of *missing* data or to locate *isolated events*. Moreover, we define measures to quantify a series of events by varying ΔT to e.g. determine the quality of an Internet of Things service.**

***Keywords*-one-dimensional clustering; Internet of Things; network performance characterization;**

CONTENTS

I. MOTIVATION

The concept of *Internet of Things* (IoT) [1], [2] is intimately related to records of certain events, e.g. a network attached device capturing weather information to be broadcasted to other devices for processing. Given a) the transmitter frequently sends out such information every time interval ΔT, and b) the receiving device keeps track of the timestamps when data was transmitted/recorded, the time series $t = \{t_i\}_{i=1\ldots N}$ stores information on failure of recording/sending/transmission/receiving.

If we cluster the one-dimensional data t such that consecutive events are not more than ΔT apart, we can infer periods in time where data might be missing. Upon detection, corresponding action such as retransmission, data interpolation, etc. can be performed. Moreover, the characteristics of *intervals of no data* (relative frequency, duration, ...) might help to diagnose the sanity of the communication network.

Since general purpose, multi-dimensional clustering methods such as Fisher's discriminant [3], *k*–means [4] or more generally *EM* [5] do not exploit the special property of ordering in one dimension, we aim at a simpler approach that does not need knowledge of the number of clusters/intervals, and it avoids density estimation such as with DENCLUE [6].

Regarding cluster classification our approach is close to the conceptual notion DBSCAN [7] introduces: clusters of points and outliers/noise points. However, we exploit the fact that the sequence of timestamps is naturally ordered[1] and thus minimize computational complexity by a factor of $\mathcal{O}(\log N)$. Of course, if we would have to sort t first, e.g. using HEAPSORT [8], we are back to asymptotic runtime of $\mathcal{O}(N \log N)$.

Our main contribution here is to adapt the concept of DBSCAN to event clustering for application in IoT service quality characterization. We present an algorithm with linear runtime complexity which asymptotically outperforms native DBSCAN that operates at an overall average runtime complexity of $\mathcal{O}(N \log N)$. Of course, while DBSCAN can be applied to any number of spatial dimensions our approach is limited to the one-dimensional case.

II. ONE-DIMENSIONAL CLUSTERING

A. Problem Formulation

Given a set of N ordered timestamps $t = \{t_i\}_{i=1\ldots N}$, i.e.

$$i \leq j \quad \Rightarrow \quad t_i \leq t_j \tag{1}$$

and an *expected* time interval ΔT, provide time intervals τ_k such that

$$t_i, t_{i+1} \in \tau_k = [\tau_k^-, \tau_k^+] \quad \Rightarrow \quad \delta t_i = t_{i+1} - t_i \leq \Delta T \quad . \tag{2}$$

Note that ΔT might be an external parameter to the algorithm that provides the solution or it is defined by t itself, e.g. through $\langle \delta t \rangle = \frac{1}{N} \sum_i \delta t_i$. As we will discuss, the δt_i need

[1] the number of seconds passed since some defined event (for *UNIX epoch time* this is Jan 1, 1970 UTC) monotonically increases, thus records of consecutive events 1, 2, ... have ordered timestamps t_1, t_2, ...

10.1109/BigData.2017.8258495

to be computed and therefore $\langle\delta t\rangle$ is efficiently determined along the lines.

B. Central Idea

In order to fulfill eq. (2), of course, we need to compute at least $N-1$ time intervals

$$\delta t = \{\delta t_i = t_{i+1} - t_i\}_{i=1\ldots N-1} \quad . \tag{3}$$

Whenever a new time series point $t_{N+1} > t_N$ gets (randomly) added, there is no a priory way of determining whether ΔT got exceeded from the existing $t_{i\le N}$.

To classify the t_i as interval bounds $\tau_k^\pm$ we note that the binary sequence

$$b = \{b_i = int(\delta t_i > \Delta T)\}_{i=1\ldots N-1} \tag{4}$$

switches from `1` to `0` for an *opening* interval bound τ^-, and from `0` to `1` for a *closing* interval bound τ^+, only. $int(\cdot)$ denotes the function $int(True) \to 1$ and $int(False) \to 0$. Hence the quantity

$$B = \{B_i = b_i - b_{i-1}\}_{i=2\ldots N-1} \quad \text{with} \quad B_i \in \{-1, 0, 1\} \tag{5}$$

yields the desired association

$$B_i = \pm 1 \quad \Rightarrow \quad t_i \in \tau^\pm \quad . \tag{6}$$

Per requirement, a) eq. (1), the t_i are ordered, and b) the binary (discrete) function b_i implies the *alternating property*

$$\forall i < j : \quad 1 = B_i = B_j \quad \Rightarrow \quad \exists i < l < j : \quad B_l = -1 \quad . \tag{7}$$

Thus, linearly scanning through the t_i and their corresponding B_i results in the two sets

$$\tau^\pm = \{\tau_k^\pm : k < k' \Rightarrow \tau_k^\pm < \tau_{k'}^\pm\}_{k=1\ldots K^\pm \le N/2-1} \tag{8}$$

such that we can simply interleave these to obtain the corresponding time intervals as our solution, eq. (2).

C. Boundary Conditions

However, there is a couple of options how to exactly interleave the $\tau^\pm$ which depend on the boundary condition. More specifically, let us assume the sequence $t_1, t_2, \ldots$ starts e.g. with intervals that are smaller than ΔT. In this case, $\tau_1^- > \tau_1^+$, and one needs to manually add a τ_0^- to construct intervals

$$\tau_k = [\tau_{k-1}^-, \tau_k^+] \quad . \tag{9}$$

A corresponding issue might happen at the end of the time series $\{t_i\}$ depending on whether $K^+ = |\tau^+|$ is equal or not equal[2] to $K^- = |\tau^-|$.

[2] Note that by virtue of eq. (7) the difference $\left|K^+ - K^-\right|$ is at most 1. Actually, it is already obvious from the fact that $|B| = N-2 \neq |t| = N$ that one needs to manually add $\tau_k^\pm$ – imagine the case where each t_i is a boundary value, but we have two of the B_i missing to classify all t_i.

```
algorithm cluster_events is
    input:  list t of ordered timestamps t,
            float variable dT of expected inverse frequency
    ΔT
    output: list tau of cluster intervals τ,
            list x of isolated timestamps x

    define lists tauMinus, tauPlus, tau, x
    define lists dt, b, B

    N ← length of t
    b[0] ← 1
    b[N] ← 1

    for each i in 1,2,...,N-1 do
        dt[i] ← t[i] - t[i-1]
        if dt[i] > dT then b[i] ← 1
        else b[i] ← 0

    for each i in 0,1,...,N-1 do
        B[i] ← b[i+1] - b[i]
        if B[i]  = -1  then append t[i] to
    tauMinus else
        if B[i]  = 1   then append t[i] to
    tauPlus  else
        if b[i+1] = 1  then append t[i] to x

    for each i in 0,...,length of tauMinus do
        add interval [tauMinus[i], tauPlus[i]] to tau

    return tau, x
```

Listing 1. Sample implementation of our clustering procedure as pseudo-code.

In order to prevent manually dealing with all the (four) different boundary condition scenarios, we might want to (virtually) add the following timestamps from the outset:

$$t_0 = -\infty \quad \text{and} \quad t_{N+1} = +\infty \quad . \tag{10}$$

Hence, we obtain $\delta t_0 = \delta t_N = +\infty$, and therefore

$$b_0 = b_N = 1 \tag{11}$$

which yields N B_i that corresponding to N t_i for classification such that we always have

$$\tau = \left\{\tau_k = [\tau_k^-, \tau_k^+]\right\}_{k=1\ldots K} \tag{12}$$

from

$$\tau^\pm = \left\{\tau_k^\pm : k < k' \Rightarrow \tau_k^\pm < \tau_{k'}^\pm\right\}_{k=1\ldots K} \tag{13}$$

with $|\tau| = K \le N/2$.

D. Isolated Points

Given the solution eq. (12), due to the ordering of the t_i, we can simply form the open intervals

$$\bar\tau = \left\{\bar\tau_k = \left(\tau_k^+, \tau_{k+1}^-\right)\right\}_{k=1\ldots K-1} \tag{14}$$

that we associate with *time intervals of failure*. Note, that $[t_1, t_N] = \tau \cup \bar\tau$. However, these intervals do **not** imply

$$\forall i, k : t_i \notin \bar\tau_k \tag{15}$$

i.e., informally, it is not true that no event happens during the intervals $\bar{\tau}$, but we certainly have

$$t_i \in \bar{\tau}_k \quad \Rightarrow \quad |t_i - t_{i\pm 1}| > \Delta T \tag{16}$$

where we refer to t_i as an *isolated* event. In terms of DBSCAN these timestamps form the *noise*, while all $t_i \in \tau^{\pm}$ are *border points*.

Isolated events have $b_i = 1$ and since they are not interval boundary points they need to have $B_i = 0$. This way we can use b and B to classify isolated events according to

$$B_i = 0 \wedge b_i = 1 \quad \Rightarrow \quad t_i \in x \tag{17}$$

where x denotes the set of *isolated* timestamps. Likewise, we can define *clustered* timestamps as

$$B_i = 0 \wedge b_i = 0 \quad \Rightarrow \quad t_i \in \bar{x} \quad . \tag{18}$$

Since b is binary and eq. (5) holds for B, all t_i are uniquely classified, i.e. $t = \tau^+ \cup \tau^- \cup x \cup \bar{x}$. It is rather straightforward to convince oneself that there is the association $\overset{(-)}{x} \leftrightarrow \overset{(-)}{\tau}$ in the sense that all $t_i \in x$ are within an interval of $\bar{\tau}$ and $t_i \in \bar{x}$ within an interval of τ.

E. Implementation & Computational Complexity

Listing 1 provides an example implementation of the method from sections II-B and II-C in pseudo-code for demonstration purposes. E.g. the call of `cluster_events(t, dT)` on

`t=[-20,-18,1,2,2.9,10,11,100,200,202,202,203]`

given `dT` as

`-1, 0, 1, 10, 100`, and mean of the elements of `t`

returns output equivalent to

```
[], [-20, -18, 1, 2, 2.9, 10, 11, 100, 200, 202, 202, 203]
[[202, 202]], [-20, -18, 1, 2, 2.9, 10, 11, 100, 200, 203]
[(1, 2.9), (10, 11), (202, 203)], [-20, -18, 100, 200]
[(-20, -18), (1, 11), (200, 203)], [100]
[(-20, 203)], []
[(-20, 11), (200, 203)], [100]
```

respectively.

The procedure presented in sections II-B to II-D and listing 1 uses $N-1$ algebraic operations for δt, $N-2$ logical operations for b and again N algebraic operations for B which determines the interval boundary classification with a total of $3(N-1)$ operations. The final loop in listing 1 to interleave the `tauPlus` and `tauMinus` lists is just for the user's convenience.

The naive approach would compute two time intervals for each t_i and perform two logical operations of those against ΔT to determine the classification, hence $4N$ computations. Note that due to the given linear ordering in one-dimensional space, our algorithm's runtime $3N-3$ is exact. In particular, it is fully deterministic when the number of timestamps N is fixed.

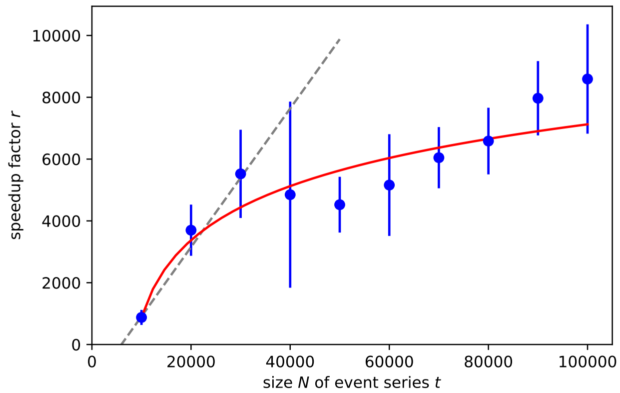

Figure 1. Numerical speedup analysis comparing a Python implementation of our algorithm using the module `numpy` [9], v1.13.0, to the vanilla implementation of DBSCAN, `DBSCAN()`, from the package `sklearn` [10], v0.18.1, module `cluster` with parameter settings `min_samples=2` and `metric='l1'`. We plot the quantity r (cf. blue dots) from eq. (19) versus the number of timestamps $N = |t|$ to cluster.
Timing measurements were performed with the standard Python module `timeit` on commodity hardware with sufficient RAM to prevent swapping. Experiments have been repeated 40 times to aggregate statistics for error estimation using error propagation by first order Taylor expansion.
The event series was generated by a white noise random distribution on $[0, 1)$ and has been rescaled by N. For the experiments, `dT` was set to two values: `1` and `1e-4`, corresponding to the parameter `eps` of `DBSCAN()`.
While we observed an approximately linear speedup for small N (cf. gray, dashed fit line $a \cdot r + b$), an overall logarithmic speedup (cf. red, solid fit line $c \log[a \cdot r + b]$) is plausible which supports the analytical result: $\mathcal{O}(N \log N)/\mathcal{O}(N) = \mathcal{O}(\log N)$.

Moreover, the required memory for our approach is linear in N. Only the event series list `t` of size N and the lists `x`, `tauMinus`, and `tauPlus` with a total size of at most N timestamps need to be stored. The lists `dt`, `b`, and `B` can be computed on the fly occupying storage $\mathcal{O}(1)$.

To confirm our analytical findings we performed a numerical experiment which is presented by fig. 1. It evaluates the speedup of our algorithm compared to a vanilla implementation of DBSCAN. Within the observed error boundaries, the scaling factor $\mathcal{O}(\log N)$ is plausible for large N wrt. the *speedup factor*

$$r(N) = R_{\text{DBSCAN}}(N)/R_{\text{lin}}(N) \tag{19}$$

with $R_{\dots}$ the individual runtime of DBSCAN and our linear approach, respectively.

F. Application

We observe that for a given, fixed event series t with total time interval $\Delta t = t_N - t_1$, the quantity

$$C_t^o(f) = \frac{1}{\Delta t} \sum_k |\tau_k| = \begin{cases} 1 & \Delta T \leq 0 \\ 0 \dots 1 & 0 < \Delta T < \Delta t \\ 0 & \Delta T \geq \Delta t \end{cases} \tag{20}$$

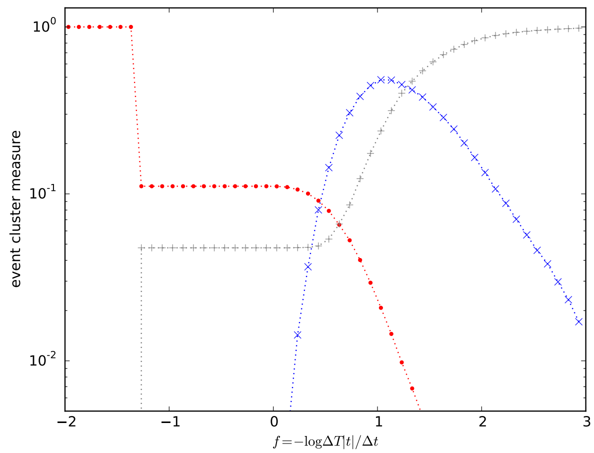

Figure 2. Sample plot of IoT service quality measures from event series t: C_t^o ($\cdot$, red), C_t^n ($\times$, blue), and C_t^s ($+$, gray) by varying ΔT. The series t consists of a burst of random events that covers approx. 10% of the total time range δt. During the rest of the time the events are periodic at a rate of about $1/50\Delta t$.

where

$$f = \log\left[\Delta T^{-1}/(\Delta t/|t|)^{-1}\right] = -\log \Delta T |t|/\Delta t \qquad (21)$$

computes the fraction of time with *no failure in operation*. ΔT is fixed by the *expected*, logarithmic, and normalized *event frequency* f, i.e. $f = 0$ represents the scale of frequency where all timestamps are equally spaced within the time series interval. $f > 0$ corresponds to smaller scales, $f < 0$ to larger ones.

Scanning C_t^o by varying f provides a characteristics that quantifies the reliability of e.g. an IoT service. It is rather straightforward to show that C_t^o is monoton decreasing with f increasing[3].

In case where the time series is generated by a single, periodic data stream, we get a unit step function $C_t^o(f) = \Theta(-f)$, i.e. 1 for $f < 0$ and 0 for $f > 0$. Nevertheless, similar information could be obtained by simply checking a histogram $n(\delta t)$, cf. eq. (3), that counts the number of δt_i in some binning interval (number density). In the case above we would observe a single peak in $n(\delta t)$. Note, that C_t^o contains similar information to $\frac{1}{\Delta t}\int_0^{\Delta T} n(\delta t) d\delta t$.

However, our clustering output (τ, x) provides information that $n(\delta t)$ is blind to, because it does not account for the ordering of the δt_i. In particular,

$$C_t^n(f) = 2\frac{|\tau| - \delta_{1|\tau|}}{|t|} = \begin{cases} 0 & \Delta T < 0 \\ 0 \ldots 1 & 0 \leq \Delta T < \Delta t \\ 0 & \Delta T \geq \Delta t \end{cases} \qquad (22)$$

provides a normed measure of the number of clusters[4]. While C_t^o just quantifies the total coverage of t by the clusters, C_t^n provides insight whether the coverage is established by a number of patches or a single/a few intervals with data frequency of at least ΔT^{-1}. This way we might draw conclusions on e.g. the reliability of an IoT service. Ideally we want $C_t^n \ll 1$.

Last but not least, we might consider the number of isolated events

$$C_t^s(\Delta T) = \frac{|x|}{|t|} = \begin{cases} 0 & \Delta T < 0 \\ 0 \ldots 1 & 0 \leq \Delta T < \Delta t \\ 1 & \Delta T \geq \Delta t \end{cases} \qquad (23)$$

as an additional indicator of reliability, since they are orthogonal to the information contained in τ. We might classify isolated events as indicator of loose IoT service quality and thus it should stay close to zero until it quickly increases to one for some $f > 0$.

Figure 2 illustrates these applications by plotting $C_t^{o,n,s}$ for an event series t generated from 10^4 uniformly random samples drawn from $[0, 1]$ joined by 10^3 equi-distant samples in $[1, 10]$. We observe that at $f = 0$ there is little variance in $C_t^{o,s}$, indicating that there is no single dominant event frequency $\nu_0 = |t|/\Delta t$. Moreover, there is a step in C_t^o at $f \approx -1$ that covers 90% of its range which refers to a dominant event frequency one order of magnitude lower than ν_0. Since $C_t^n \ll 1$ we conclude this frequency to be present along major time intervals within $[t_1, t_N]$. Also, C_t^s rapidly drops. Therefore, the existence of isolated events vanishes at time scales larger than $\sim \Delta T/10\nu_0$ such that we have a *clean signal*.

In contrast, $C_t^n \sim 1$ for $f \approx 1$. Thus, due to the randomness we introduced in our sample, for high-frequency events, increasing coverage of $[t_1, t_N]$ is achieved by a number of isolated clusters (random nature of the signal!). Finally, for frequencies 3 orders of magnitude larger than ν_0, $C_f^s \approx 1$, i.e. no more clustering of events is present.

Figure 3 depicts a sample data flow and processing pipeline where the discussed method can be employed to rate and monitor e.g. an IoT device or the data availability of satellite imagery in the big geo-spatial data platform *IBM PAIRS* [11], [12]. Given that this *information service* is expected to send data packages at frequency ΔT^{-1}, an *event cluster engine* records and stores the timestamps t_i for further analysis. At the same time a *frequency detector* might dynamically adjust ΔT, e.g. by computing the mean of the δt_i over a given time window. The event clustering engine is coupled to a *user interface* that might be interacted with by a *RESTful API* [13] served by e.g. *Python Flask* [14] to trigger the execution of listing 1 in order to return the sets τ and x. Once the clustering has been performed, the

[3] The larger ΔT, the more the clusters τ cover the whole time series. Due to eq. (2) clusters never shrink in size for increasing ΔT, they either grow or merge to bigger clusters, letting the overall cover increase.

[4] The Kronecker delta δ_{ij} is 1 for $i = j$, 0 else. It forces $|\tau| \in \{0, 1\}$ to result in $C_t^n = 0$.

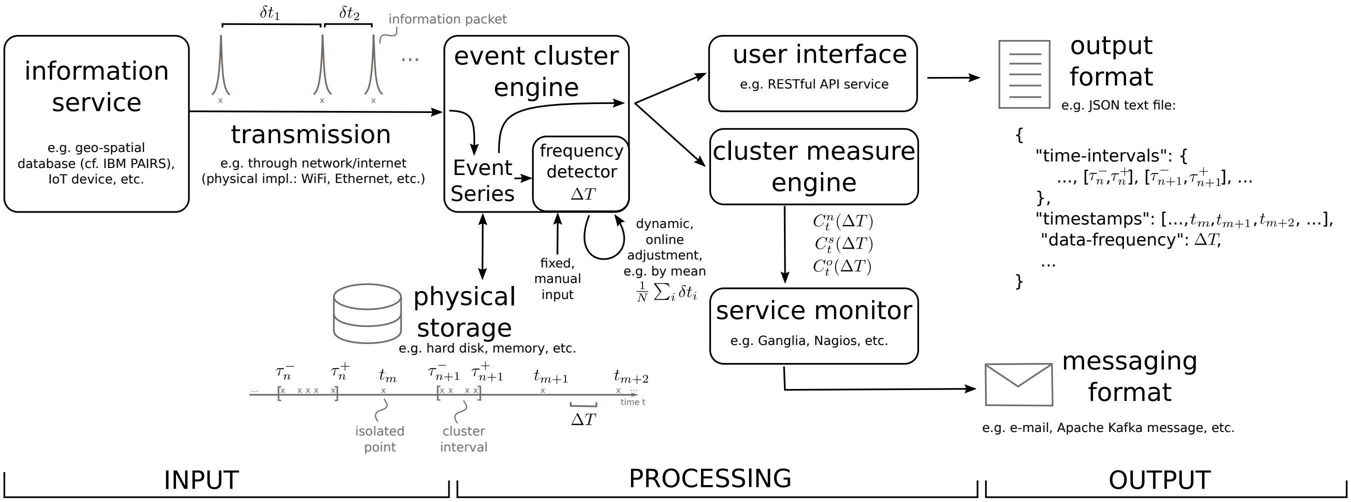

Figure 3. Sample data and processing flow of an implementation of the IoT quality service employing the procedure listing 1 as well as the measures eqs. (20), (22) and (23). The main text provides details. Note, that here the t_m, t_{m+1} and t_{m+2} reference isolated timestamps of the set x, not to be confused with all timestamps t_i of t indexed by i, i.e. $x \subseteq t$.

quantities $C_t^{o,n,s}$ can be computed and analyzed by a *cluster measure engine* which itself feeds derived service quality indicators to a monitoring system such as e.g. *Ganglia* [15] or *Nagios* [16]. These might then release alerts by an appropriate messaging service such as plain e-mail or employing a system such as *Apache Kafka* [17].

III. Conclusion

We discussed and implemented a one-dimensional, one-parameter clustering method with linear complexity on input and memory usage. It might be the preferred choice over the more general apporach DBSCAN takes when clustering ordered timestamps. Based on the algorithm's output we suggested measures that have useful application in the domain of IoT to quantify data availability or to indicate the reliability/stability of an IoT device connecting to the network. In particular, the presented approach is part of the data availability RESTful service of IBM's big geo-spatial database PAIRS.

The cluster method might be useful for other domains as well. Applications that have to characterize peaks of measurements can benefit. One of them is the problem of geo-locating leaks through a network of detector sensors [18] such as in the field of industrial pollution detection [19].

References

[1] ITU-T, "Overview of the Internet of things," 2012. [Online]. Available: http://handle.itu.int/11.1002/1000/11559

[2] M. Chiang and T. Zhang, "Fog and IoT: An Overview of Research Opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, p. 854, 2016. [Online]. Available: http://www.download-paper.com/wp-content/uploads/2017/01/2016-ieee-Fog-and-IoT-An-Overview-of-Research-Opportunities.pdf

[3] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, p. 179, 1936. [Online]. Available: http://www.comp.tmu.ac.jp/morbier/R/Fisher-1936-Ann._Eugen.pdf

[4] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, p. 129, 1982. [Online]. Available: http://www.cs.toronto.edu/~roweis/csc2515-2006/readings/lloyd57.pdf

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the royal statistical society. Series B (methodological)*, p. 1, 1977. [Online]. Available: http://www.eng.auburn.edu/~troppel/courses/7970%202015A%20AdvMobRob%20sp15/literature/paper%20W%20refs/dempster%20EM%201977.pdf

[6] A. Hinneburg and D. A. Keim, "An efficient approach to clustering in large multimedia databases with noise," in *KDD*, vol. 98, 1998, pp. 58–65. [Online]. Available: http://www.aaai.org/Papers/KDD/1998/KDD98-009.pdf

[7] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *KDD*, vol. 96, 1996, p. 226. [Online]. Available: http://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf

[8] I. Wegener, "The worst case complexity of McDiarmid and Reed's variant of BOTTOM-UP HEAPSORT is less than n log n+ 1.1 n," *Information and Computation*, vol. 97, no. 1, p. 86, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/089054019290005Z

[9] S. v. d. Walt, S. C. Colbert, and G. Varoquaux, "The NumPy array: a structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011. [Online]. Available: https://arxiv.org/pdf/1102.1523

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, and V. Dubourg, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011. [Online]. Available: http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

[11] L. J. Klein, F. J. Marianno, C. M. Albrecht, M. Freitag, S. Lu, N. Hinds, X. Shao, S. Bermudez Rodriguez, and H. F. Hamann, "PAIRS: A scalable geo-spatial data analytics platform," in *Big Data (Big Data), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1290–1298. [Online]. Available: http://researcher.watson.ibm.com/researcher/files/us-kleinl/IEEE_BigData_final_klein.pdf

[12] S. Lu, X. Shao, M. Freitag, L. J. Klein, J. Renwick, F. J. Marianno, C. Albrecht, and H. F. Hamann, "IBM PAIRS curated big data service for accelerated geospatial data analytics and discovery," in *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE, 2016, pp. 2672–2675. [Online]. Available: https://static.aminer.org/pdf/fa/bigdata2016/S09208.pdf

[13] Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, 2000. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[14] Wikipedia, "Flask (web framework)," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Flask_(web_framework)

[15] ——, "Ganglia (software)," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Ganglia_(software)

[16] ——, "Nagios," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Nagios

[17] ——, "Apache Kafka," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Apache_Kafka

[18] S. E. Haupt, "A demonstration of coupled receptor/dispersion modeling with a genetic algorithm," *Atmospheric Environment*, vol. 39, no. 37, pp. 7181–7189, Dec. 2005. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1352231005007685

[19] J. D. Albertson, T. Harvey, G. Foderaro, P. Zhu, X. Zhou, S. Ferrari, M. S. Amin, M. Modrak, H. Brantley, and E. D. Thoma, "A Mobile Sensing Approach for Regional Surveillance of Fugitive Methane Emissions in Oil and Gas Production," *Environmental Science & Technology*, vol. 50, no. 5, pp. 2487–2497, Mar. 2016. [Online]. Available: http://pubs.acs.org/doi/full/10.1021/acs.est.5b05059