HDCluster: An Accurate Clustering Using Brain-Inspired High-Dimensional Computing

Mohsen Imani, Yeseong Kim, Thomas Worley, Saransh Gupta, and Tajana Rosing
Computer Science and Engineering Department, UC San Diego, La Jolla, CA 92093, USA
{moimani, yek048, tworley, sgupta, tajana}@ucsd.edu

***Abstract*—Internet of things has increased the rate of data generation. Clustering is one of the most important tasks in this domain to find the latent correlation between data. However, performing today's clustering tasks is often inefficient due to the data movement cost between cores and memory. We propose HDCluster, a brain-inspired unsupervised learning algorithm which clusters input data in a high-dimensional space by fully mapping and processing in memory. Instead of clustering input data in either fixed-point or floating-point representation, HDCluster maps data to vectors with dimension in thousands, called *hypervectors*, to cluster them. Our evaluation shows that HDCluster provides better clustering quality for the tasks that involve a large amount of data while providing a potential for accelerating in a memory-centric architecture.**

***Index Terms*—Hyperdimension computing, Clustering, Brain-inspired computing**

I. Introduction

Internet of things (IoT) significantly increases the number of devices around the world. Recent studies report that more than 25 billion connected smart devices exist in 2015. [1] This number is expected to be doubled by 2020. [2] As the large network of connected devices generates a huge amount of data, machine learning gains popularity as an autonomous solution that extracts the useful information and learns patterns from the collected data [3]–[5]. However, the large amount of data dominates the processing capability of the current computing systems [6]–[8]. This inefficiency is mainly due to significant data movement costs between processing cores and memory. For example, to perform clustering tasks which are one of the most important unsupervised learning, [9], [10] the processors expensively compute the similarity between data points by fetching every point from the memory. A traditional solution is to migrate this issue by running the tasks on a cloud, but transferring large amount of data incurs significant congestion on the network. In addition, this may inherently lead to security and privacy issues in many applications. Thus, going toward IoT, it is crucial to have a light-weight clustering technique with adequate architectural supports which can efficiently run even on end-node devices.

Brain-inspired Hyperdimensional (HD) computing is based on understanding the fact that brains compute with *patterns of neural activity* which are not readily associated with numbers [11]. However, due to the very large size of the brain's circuits, such neural activity patterns can only be modeled with points of high-dimensional space (e.g., D=10,000). Operations on hypervectors can be combined into interesting computational behavior with unique features that make them robust and efficient. HD computing builds upon a well-defined set of operations with random HD vectors, is extremely robust in the presence of failures, and offers a complete computational paradigm that is easily applied to learning problems [11], [12]. Its main differentiation from other paradigms is that data are represented as approximate patterns, which can favorably scale for many learning applications.

TABLE I
Table of Notations

Symbol	Definition	Symbol	Definition
$\mathbf{v}$	Feature vector in original domain	$\mathbb{L}$	Level hypervectors $\in \{0,1\}^D$
n	# of features in original domain	Q	# of quanitized levels
D	Dimension of encoded data	ID_i	An ID hypervector $\in \{0,1\}^D$
K	# of clusters	C_k	A cluster center hypervector $\in \{0,1\}^D$
$\mathbf{h}$	A non-binary hypervector $\in \mathbb{N}^D$	N	# of data points
$\mathbb{E}^k$	Encoded data points of a cluster	I	# of executed iterations

In this paper, we present a new clustering algorithm which maps a large amount of original data to a hardware-friendly high-dimension space and performs the clustering tasks by using Hyperdimensional (HD) computing. HD computing is an alternative computational model which emulates cognition tasks by computing with vectors in high-dimensional space, called *hypervectors*. A hypervector has the dimensionality in thousands (e.g., D=10,000) to mimic neural activity [11]. Since the elements of hypervectors are independent in processing, we can design a fully-parallelized hardware architecture that handles the hypervectors. In this work, we show how the proposed HDCluster encodes the original data to the hypervectors without losing the necessary information and performs the clustering tasks with well-defined linear algebra for the data types in the high-dimension space. To the best of our knowledge, HDCluster is the first design which processes the clustering tasks with the hypervectors. Our evaluation shows that HDCluster provides better clustering quality for the tasks that involve a large amount of data while providing a potential for accelerating in a memory-centric architecture.

II. HDCluster Algorithm

A. HDCluster Overview

We propose HDCluster, a brain-inspired clustering algorithm which clusters input data into a high-dimensional space. Instead of clustering input data in either fixed-point or floating-point representations, HDCluster maps data to vectors with thousands of dimensions, called *hypervectors*, and then clusters them using concrete linear algebra. The well-defined set of HD operations is known to be extremely robust in the presence of failures, and offers a complete computational paradigm that is easily applied to learning problems such as analogy-based reasoning, sequence memory, language recognition, biosignal processing,

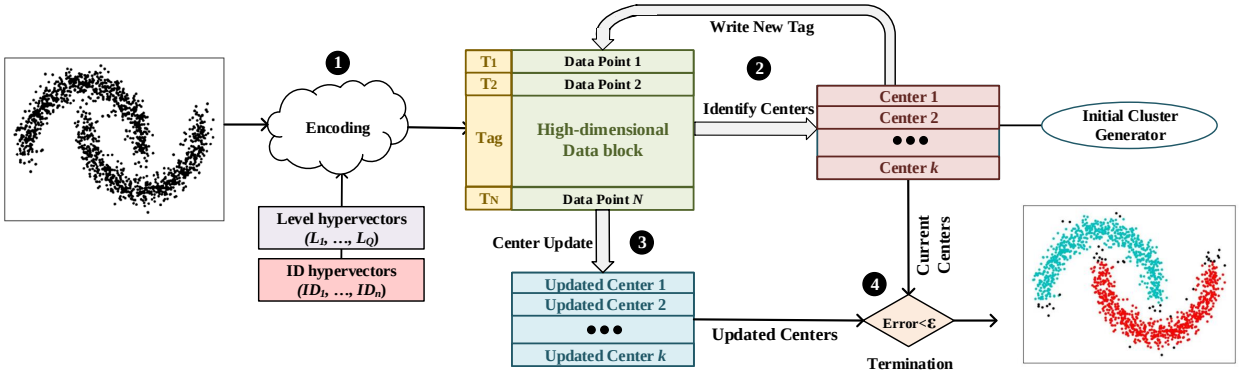

Fig. 1. The functionality of the proposed HDCluster algorithm.

speech recognition, and prediction from multimodal sensor fusion [13]–[20].

Figure 1 illustrates an overview of HDCluster. As the first step, HDCluster encodes data into high-dimensional space and then applies clustering tasks to the encoded data (❶). The clustering procedure starts with the initial centers of each cluster. For each iteration, HDCluster identifies which center is the most similar to each data point. The identified centers are stored as *tags* (❷). Then, HDCluster updates the centers by calculating the average of the data points whose tags are the same (❸). The iterations are repeated to obtain the converged cluster centers (❹). In the next section, we show the details of the clustering procedure.

B. Encoding into HD Space

The first step of HDCluster is to encode input data into the hypervectors, where an original data point has n features, i.e., $\mathbf{v} = \langle v_1, \dots v_n \rangle$ [21]. Table I summarizes all notations used in this paper. An encoded hypervector that corresponds to one data point has D dimensions (e.g. $D = 10,000$). We need to keep all information of a data point in the original space, i.e., the feature values and their indexes. As Figure 2a shows, we use two sets of pre-computed hypervectors: *level* and *ID* hypervectors. To create level hypervectors, we compute the minimum and maximum feature values among all data points, $\mathbf{v}_{min}$ and $\mathbf{v}_{max}$, and then quantize the range of $[\mathbf{v}_{min}, \mathbf{v}_{max}]$ linearly into Q levels, $\mathbb{L} = \{L_1, \cdots, L_Q\}$. Each level hypervector, L_i, is unique and has D binarized dimensions, i.e., $L_i \in \{0,1\}^D$. The level hypervectors need to have the spectrum of similarities, such that the neighbor levels get more similar hypervectors. We create the first level hypervector, L_1, by randomly selecting each element of a hypervector to be either 0 or 1 value. The second level hypervector, L_2, is created by flipping D/Q random dimensions of the L_1. This continues until creating the L_Q hypervector by flipping D/Q random dimensions of L_{Q-1}. Since we select and flip the dimensions randomly, there is a high probability that the L_1 and L_Q will have $D/2$ dimension difference. As a result, the level hypervectors have similar values if the corresponding original data are closer, while L_1 and L_Q will be nearly orthogonal.

The hypervector also needs to contain all the information that the original features have. To differentiate the impact of each feature index, we devise *ID hypervectors*, $\{ID_1, \cdots, ID_n\}$. An ID hypervector has the binarized dimensions, i.e., $ID_i \in \{0,1\}^D$. We create IDs with random binary values so that the ID hypervectors of different feature indexes are nearly orthogonal:

$$\delta(ID_i, ID_j) \simeq D/2 \qquad (i \neq j \ \& \ 0 < i, j \leqslant n)$$

where the similarity metric, $\delta(ID_i, ID_j)$, is the Hamming distance between the two ID hypervectors. The orthogonality of ID hypervectors is ensured as long as the hypervector dimension, D, is large enough compared to the number of features ($D >> n$) in the original data point.

Figure 2b shows how we map each data point, $\mathbf{v}$, to the high-dimensional space using the precomputed hypervectors. For each feature, we perform an element-wise XOR operation for the ID and level hypervector corresponding to the feature value. The different features are combined by adding each element. For example, when a feature value of an original data point, v_i, is quantized to $\overline{L}_i \in \mathbb{L}$, the following equation represents the calculated hypervector, $\mathbf{h}$:

$$\mathbf{h} \ = \ ID_1 \oplus \overline{L}_1 \ + \ ID_2 \oplus \ \overline{L}_2 + \dots \ + \ ID_n \oplus \overline{L}_n.$$

Note that the element-wise addition can make a hypervector that has integer elements, i.e., $H \in \mathbb{N}^D$. To perform all the other clustering procedures with binarized hypervectors, we apply a *majority* function for the calculated hypervector. For a given hypervector, $\mathbf{h} = \langle h_1, \cdots, h_D \rangle$, the majority function is defined as follows:

$$MAJ(\mathbf{h}, \tau) = \langle h'_1, \cdots, h'_D \rangle \text{ where } h'_i = \begin{cases} 0, & \text{if } h_i < \tau \\ 1, & \text{otherwise.} \end{cases}$$

Using the majority function, the final hypervector for each data point is encoded by $\mathbf{e} = MAJ(\mathbf{h}, n/2)$, and $\mathbf{e} \in \{0,1\}^D$.

C. Clustering in HD Space

HDCluster procedure identifies the cluster indexes (tags) through an iterative process as shown in Figure 2c and d, by using the encoded hypervectors. The proposed HD clustering algorithm is inspired by k-means. [22] In a similar way to the standard k-means algorithm, we initially choose K random hypervectors as cluster centers. We denote each hypervector for the centers as $\{C_1, C_2, \cdots, C_K\}$, where $C_k \in \{0,1\}^D$, as

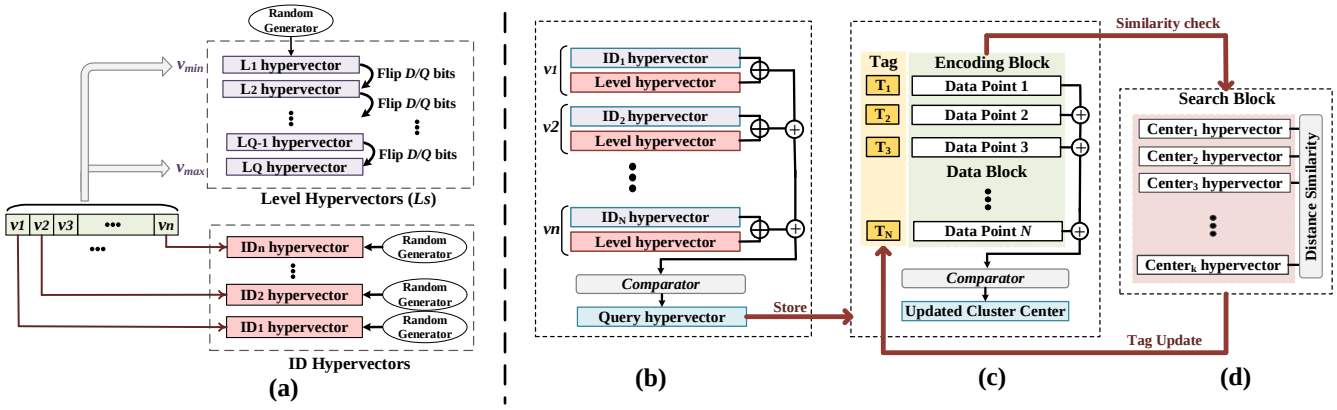

Fig. 2. The functionality of HDCluster algorithm using *encoding*, *data* and *search* blocks.

shown in Figure 2d. Randomness and high dimensionality of hypervectors ensure that the centers of clusters are orthogonal at the initial iteration. With the initial cluster centers, we perform an iterative process to find the best clusters. There are two steps in each iteration, *computing distance similarity* and *updating cluster centers*.

Computing distance similarity: In this step, HDCluster assigns a cluster center for each encoded hypervector among all the center candidates. HDCluster measures the Hamming distances between the hypervector of a data point and each k center and identifies the cluster center which has the highest similarity. For an encoded hypervector, $\mathbf{e}$, the index of the cluster center, called *tag*, k, is chosen as follows:

$$\underset{k}{\text{argmin}} \ \delta(C_k, \mathbf{e})$$

Updating cluster centers: After identifying tags of all data points, HDCluster updates the cluster centers using the data points which belong to each cluster. For a set of the data points in the kth cluster, $\mathbb{E}^k = \{\mathbf{e}_i^k\}$, we perform element-wise additions to produce the hypervector sum, $\mathbf{s}^k = \sum_i \mathbf{e}_i^k$. Then, HDCluster binarizes the hypervector sum so that it is mapped into the $\{0,1\}^D$ space. The following equation illustrates this procedure:

$$C_k = MAJ(\mathbf{s}^k, |\mathbb{E}^k|/2)$$

Termination of HDCluster: The algorithm converges when there is no significant change in the cluster centers. This iterative procedure continues until (i) the hypervectors representing the center of clusters has minor change during two consecutive iterations or (ii) the number of iterations exceeds a pre-defined parameter.

III. Experimental Results

A. Experimental Setup

We implemented full HDCluster functionality using C++ implementation. We test the clustering quality of the proposed HDCluster, with diverse datasets including: MNIST handwritten digit [23], DIM dataset [24], Glass Identification [25], Iris [26], Voice dataset (ISOLET) [27] and medical-related datasets such as Gene Expression Cancer RNA Sequence (RNA-Seq) [28],

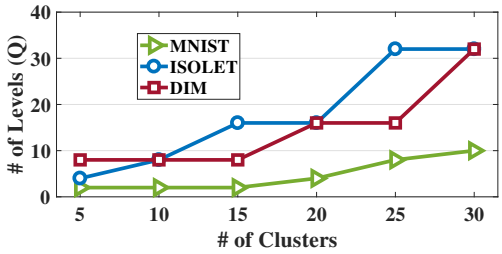

Fig. 3. Impact of number of clusters on the minimum quantization levels.

Breast Cancer dataset [29], Primary Tumor dataset [30], and Parkinsons dataset [31]. We evaluate the clustering quality for each dataset by using the provided ground truth.

B. Clustering Quality

Table II show the quality of clustering for HDCluster and k-means algorithms over different applications. The quality of clustering has been measured by comparing the result of clustering with the provided ground truth. For all the reported results, we use $Q = 16$ and $D = 10K$. To better understand when HDCluster provides higher quality and efficiency, we perform the evaluation for *DIM* which has multiple datasets with different feature sizes. Table III summarizes the evaluation results. As compared to the k-means++, HDCluster shows higher robustness with the increase in the dimension of the original data. For example, k-means++ and HDCluster exhibit the similar quality for DIM 32, while for DIM 1024, the HDCluster provides 10% better clustering quality. In addition, since the computation cost of HDCluster is independent of dimensions of a dataset after encoding the dataset, we achieve higher efficiency for larger dimension than k-means++ which does not scale.

C. Parameter Exploration

The precision of the encoding procedure is defined by how accurate it can map original data points to the high-dimensional space. For example, quantizing each feature to a small level (Q) may not keep all the required information of the original data points. With the relatively large number of clusters (K), the encoded hypervector may need to maintain more fine-grained information. Figure 3 shows the sufficient quantization

TABLE II
THE QUALITY OF CLUSTERING OF HDCLUSTER AS COMPARE TO K-MEANS ($Q = 16$).

Datasets	MNIST	ISOLET	IRIS	Glass	Unbalance	RNA-seq	Cancer	Ecoli	Parkinsons
# of Data Samples (N)	10,000	7,797	150	214	6,500	801	569	339	6,590
# of Features (n)	784	617	4	10	2	20,531	32	17	1
# of Clusters (K)	10	26	3	7	8	5	2	22	2
***k*-means**	48.8%	28.4%	88.7%	51.8%	93.8%	37.5%	94.1%	74.3%	75.3%
HDCluster	58.6%	33.1%	89.9%	67.5%	92.3%	37.5%	96.2%	78.5%	75.6%

TABLE III
THE QUALITY OF CLUSTERING OF HDCLUSTER ON DIM DATASET WITH DIFFERENT DIMENSIONS.

Datasets	DIM 32	DIM 128	DIM 256	DIM 512	DIM 1024
# of Features (n)	32	128	256	512	1024
***k*-means**	87.5%	81.2%	76.4%	68.6%	62.5%
HDCluster	87.5%	85.3%	81.2%	76.1%	72.6%

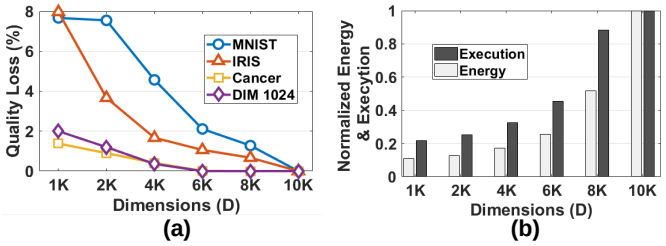

Fig. 4. (a) quality loss of clustering and (b) normalized Energy consumption and execution time of HDCluster using different dimensions.

levels, Q, when the number of clusters changes from 5 to 30, where the quality is evaluated in mean square error. HDCluster with a large number of cluster centers requires to quantize the input data into a higher number of levels. For instance, for ISOLET, HDCluster achieves the best quality with $Q = 8$ when clustering with 10 centers, while the quantization level of $Q = 32$ is required in the case of clustering with 30 centers.

D. Scalability with Hypervector Dimension

Since the hypervectors map the original information into a high-dimension space, the quality of clustering is directly related to the vector size. In the accelerator design, it also determines the system efficiency. Figure 4a shows the quality loss of clustering when HDCluster dimensionality changes from 1k to 10k. The results show that, for most of the applications, HDCluster can perform the clustering tasks with a relatively small dimensionality, while providing similar quality of clustering. When reducing the dimensionality of hypervectors to 4,000 and 2,000, HDCluster on average has 1.7% and 3.1% lower quality respectively as compared to HDCluster with the full dimension, i.e., $D = 10K$. Figure 4b also shows the normalized energy and execution time of HDCluster running. All results are normalized to HDCluster running with 10K hypervector. Our results shows that decreasing dimensionality from 10K to 4K can result in 2.1× speedup and 2.4× energy efficiency.

IV. CONCLUSION

We propose a novel clustering algorithm, HDCluster, that maps data points to the high-dimensional space and clusters them using concrete linear algebra for hypervectors. HDCluster provides high clustering quality for diverse and practical applications that involve a large number of samples and high complexity in feature domains. Our future work is to design an accelerator which processes the entire HDCluster operations in a memory-centric architecture.

ACKNOWLEDGEMENTS

This work was partially supported by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, and also NSF grants #1730158 and #1527034.

REFERENCES

[1] A. Aijaz *et al.*, "Cognitive machine-to-machine communications for internet-of-things: A protocol stack perspective," *IEEE IoT-J*, vol. 2, no. 2, pp. 103–112, 2015.
[2] A. Fehske *et al.*, "The global footprint of mobile communications: The ecological and economic perspective," *IEEE Communications Magazine*, vol. 49, no. 8, 2011.
[3] I. H. Witten *et al.*, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.
[4] S. Landset *et al.*, "A survey of open source tools for machine learning with big data in the hadoop ecosystem," *Journal of Big Data*, vol. 2, no. 1, p. 24, 2015.
[5] X. Wu *et al.*, "Data mining with big data," *IEEE TKDE*, vol. 26, no. 1, pp. 97–107, 2014.
[6] A. Botta *et al.*, "On the integration of cloud computing and internet of things," in *IEEE FiCloud*, pp. 23–30, IEEE, 2014.
[7] L. Mainetti *et al.*, "Evolution of wireless sensor networks towards the internet of things: A survey," in *IEEE SoftCOM*, pp. 1–6, IEEE, 2011.
[8] M. Imani *et al.*, "Bayesian control of large MDPs with unknown dynamics in data-poor environments," in *NIPS*, 2018.
[9] A. Fahad *et al.*, "A survey of clustering algorithms for big data: Taxonomy and empirical analysis," *IEEE TETC*, vol. 2, no. 3, pp. 267–279, 2014.
[10] Q. V. Le, "Building high-level features using large scale unsupervised learning," in *IEEE ICASSP*, pp. 8595–8598, IEEE, 2013.
[11] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, no. 2, pp. 139–159, 2009.
[12] S. Gupta *et al.*, "Felix: fast and energy-efficient logic in memory," in *ICCAD*, p. 55, ACM, 2018.
[13] M. Imani *et al.*, "Fach: Fpga-based acceleration of hyperdimensional computing by reducing computational complexity," in *ASP-DAC*, p. 55, IEEE, 2019.
[14] O. Rasanen and J. Saarinen, "Sequence prediction with sparse distributed hyperdimensional coding applied to the analysis of mobile phone use patterns," *IEEE Trans. Neural Netw. Learn. Syst*, vol. PP, no. 99, pp. 1–12, 2015.
[15] M. Imani *et al.*, "Voicehd: Hyperdimensional computing for efficient speech recognition," in *IEEE ICRC*, pp. 1–6, IEEE, 2017.
[16] S. Salamat *et al.*, "F5-hd: Fast flexible fpga-based framework for refreshing hyperdimensional computing," in *FPGA*, ACM, 2019.
[17] M. Imani *et al.*, "Hdna: Energy-efficient dna sequencing using hyperdimensional computing," in *BHI*, pp. 271–274, IEEE, 2018.
[18] Y. Kim *et al.*, "Efficient human activity recognition using hyperdimensional computing," in *IoT*, p. 38, ACM, 2018.
[19] M. Imani *et al.*, "Low-power sparse hyperdimensional encoder for language recognition," *IEEE Design & Test*, vol. 34, no. 6, pp. 94–101, 2017.
[20] M. Imani *et al.*, "Exploring hyperdimensional associative memory," in *IEEE HPCA*, pp. 445–456, IEEE, 2017.
[21] M. Imani *et al.*, "Hierarchical hyperdimensional computing for energy efficient classification," in *DAC*, pp. 1–6, IEEE, 2018.
[22] D. Arthur *et al.*, "k-means++: The advantages of careful seeding," in *ACM-SIAM*, pp. 1027–1035, Society for Industrial and Applied Mathematics, 2007.
[23] Y. LeCun, C. Cortes, and C. J. Burges, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2010.
[24] P. Fränti, O. Virmajoki, and V. Hautamäki, "Fast agglomerative clustering using a k-nearest neighbor graph," *TPAMI*, vol. 28, no. 11, pp. 1875–1881, 2006.
[25] "Uci machine learning repository." https://archive.ics.uci.edu/ml/datasets/glass+identification.
[26] "Uci machine learning repository." https://archive.ics.uci.edu/ml/datasets/iris.
[27] "Uci machine learning repository." http://archive.ics.uci.edu/ml/datasets/ISOLET.
[28] J. N. Weinstein *et al.*, "The cancer genome atlas pan-cancer analysis project," *Nature genetics*, vol. 45, no. 10, pp. 1113–1120, 2013.
[29] R. S. Michalski *et al.*, "The multi-purpose incremental learning system aq15 and its testing application to three medical domains," *Proc. AAAI 1986*, pp. 1–041, 1986.
[30] B. Cestnik *et al.*, "Assistant 86: A knowledge-elicitation tool for sophisticated users," in *EWSL*, pp. 31–45, Sigma Press, 1987.
[31] M. A. Little *et al.*, "Exploiting nonlinear recurrence and fractal scaling properties for voice disorder detection," *BioMedical Engineering OnLine*, vol. 6, no. 1, p. 23, 2007.